Avaliação da Qualidade de Especificações Orientadas a Objetos

Clifton Eduardo Clunie Beaufond

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profª. Ana Regina Cavalcanti da Rocha, D.Sc.
(Presidente)

Profª. Cláudia Maria Lima Werner, D.Sc.

Prof. Roberto da Silva Bigonha, Ph.D.

Prof. Paulo Cesar Masiero, D.Sc.

Prof. Jano Moreira de Sousa, Ph.D.

Prof. Marcos Roberto da Silva Borges, Ph.D.

Rio de Janeiro, RJ - Brasil
Março de 1997

CLUNIE, Clifton Eduardo Beaufond

Avaliação da Qualidade de Especificações Orientadas a Objetos. (Rio de Janeiro) 1997.

xi, 314 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 1997).

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1. **Especificações de Software**
2. **Orientação a Objetos**
3. **Qualidade de Software**

I. COPPE/UFRJ II. TíTULO (série)

à minha esposa Gisela,
aos meus filhos Grace e Clifton Jr.

Agradecimentos

À Professora Ana Regina, minha orientadora, uma pessoa muito especial, com quem muito tenho aprendido, a quem devo minha formação a nível de pós-graduação. Pela sua orientação, sugestões, críticas construtivas, e sobretudo, pela sua objetividade.

À Professora Cláudia Werner, minha co-orientadora, que me acompanhou ao longo de todo o processo de desenvolvimento deste trabalho, pelas sugestões apresentadas, apoio e palavras de estímulo.

Aos professores, Roberto Bigonha, Paulo Masiero, Marcos Borges e Jano Moreira de Souza por participarem da banca examinadora e pelas valiosas sugestões.

Agradeço aos colegas de projeto Lícia, Guilherme, Ana Cecília, Ronald, Silvério, Ricardo, Leonardo e Sérgio pela troca de experiências e pela grata convivência de três anos de trabalho numa experiência de projeto empresa-universidade.

Agradeço à equipe da IBM-Brasil, Antônio, Jefferson, Júlia e Sérgio que me dedicaram parte de seu valioso tempo para discutir algumas das propostas deste trabalho.

À Renata, Cristina, Belchior, Vera, Cleusa, Fernanda e Rosa pela amizade e incentivo.

Ao Sérgio pela amizade sincera e pelo computador, imprescindível ao desenvolvimento deste trabalho.

A Rodolfo e a Regina Lima pelo incentivo constante e ajuda no português.

A todos os funcionários e colegas do Programa de Sistemas que direta ou indiretamente me auxiliaram na realização deste trabalho.

À CNPq pelo apoio financeiro ao Projeto Memphis (Ambiente de Desenvolvimento de Software Baseado em Reutilização)

À CAPES pela ajuda financeira.

A Deus pela saúde e as graças recebidas.

Resumo da tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

Avaliação da Qualidade de Especificações Orientadas a Objetos

Clifton Eduardo Clunie Beaufond

Março de 1997

Orientador: Profª. Ana Regina C. da Rocha, D.Sc.
Co-Orientador: Profª. Cláudia Maria L.Werner, D.Sc.
Programa: Engenharia de Sistemas e Computação

O uso da tecnologia de orientação a objetos está em processo de expansão, sendo já utilizada em aplicações complexas e críticas. Conseqüentemente, a urgência pela qualidade torna-se um aspecto fundamental.

Embora a literatura disponível sobre orientação a objetos ofereça formas de como utilizar os métodos orientados a objetos, estes métodos não fornecem um tratamento direto dos aspectos relacionados à qualidade das especificações.

Este trabalho define atributos de qualidade e procedimentos para avaliação de software desenvolvido segundo o paradigma de orientação a objetos, considerando as fases de especificação de requisitos e especificação de projeto. A organização e discussão destes atributos é realizada de acordo com um método de avaliação da qualidade de software anteriormente proposto. É, também, apresentada uma experiência de avaliação de especificações, considerando os atributos propostos e os processos de avaliação definidos.

São, ainda, descritos os resultados de uma pesquisa de campo realizada com o objetivo de obter um perfil da qualidade desejável para especificações orientadas a objetos. A partir desta pesquisa, verificou-se a adequação dos atributos de qualidade propostos às especificações desenvolvidas utilizando este paradigma, estabelecendo-se uma hierarquização dos mesmos de acordo com o grau de importância e considerando a opinião de especialistas envolvidos em projetos de desenvolvimento orientado a objetos.

Abstract of thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Doctor of Sciences (D.Sc.)

Quality Evaluation for Object-Oriented Specifications

Clifton Eduardo Clunie Beaufond

March, 1997

Advisor: Prof[a]. Ana Regina Cavalcanti da Rocha, D.Sc.
Co-advisor: Prof[a]. Cláudia Maria Lima Werner, D.Sc.
Department: Systems and Computer Engineering

The scale of object-oriented technology has changed. Object-oriented technology use is expanding to large complex-critical applications being built by teams of developers with diverse skills. The urgency for quality assurance is heightened.

Al though many books and articles have been written on how do apply object-oriented methodologies, none directly address quality for object-oriented specifications.

This work defines quality attributes and evaluation processes for object-oriented software development considering the requirements and design phases. The attributes organization and discussion are fulfilled according to a software quality evaluation method previously proposed. It is also presented a specification evaluation experience considering the proposed attributes and the evaluation processes defined.

The results of a field research realized with the purpose of obtaining a profile about the desirable quality for object-oriented specifications are described. Based on this research the quality attributes adequacy to the object-oriented specifications development was verified, to set up an hierachy according to their importance grade, considering the specialists opinions.

Sumário

Índice de Figuras

Índice de Tabelas

Capítulo I

Introdução

"Não existe nada mais difícil de se ter em mãos, mais perigoso de se conduzir e mais incerto sobre seu sucesso do que a vantagem de introduzir uma nova ordem das coisas."

Nicolo Machiavelli

(O Príncipe)

I.1. Impacto da Tecnologia de Orientação a Objetos no Desenvolvimento e Qualidade de Software

Grandes esforços vêm sendo realizados na área de *Engenharia de Software* visando oferecer novas técnicas e métodos de desenvolvimento para atingir maior qualidade nos produtos e baixar o custo de desenvolvimento e manutenção do software.

A baixa qualidade e produtividade no processo de desenvolvimento é atribuída, principalmente, ao aumento constante do tamanho e complexidade do software, ao incremento na demanda por novos produtos, ao caráter dinâmico e iterativo ao longo de todo o ciclo de vida, à falta de procedimentos normalizados de avaliação da qualidade e à falta de métodos e ferramentas adequadas para construir novas aplicações (Basili et al. 1995).

As novas aplicações que surgem são de natureza diversas, em complexidade e domínios de aplicação. Tratar de representar e modelar o comportamento de tais aplicações, empregando métodos e linguagens de desenvolvimento tradicionais, às vezes, não é simples nem eficiente. Isto se deve, em parte, à dificuldade para abstrair o modelo da realidade num modelo computacional.

É comumente aceito considerar que a mente humana abstrai naturalmente o mundo da aplicação como um mundo de objetos que interagem entre si, cada um possuindo propriedades e comportamentos próprios, e onde cada objeto pode ter outros

objetos como componentes de sua estrutura (Wand em Kim e Lochovsky 1989). Partindo dessa idéia, a tecnologia de orientação a objetos constitui uma abordagem que minimiza a distância da representação entre os problemas no mundo real e as soluções no mundo computacional. Isso é possível implementando-se o conceito de abstrair naturalmente um modelo de objetos do mundo da aplicação num modelo computacional de objetos, diminuindo, portanto, a complexidade dos sistemas (Wirfs-Brock et al. 1990a).

Assim sendo, o aumento da complexidade é o acelerador paralelo para o uso do paradigma, pois fornece o melhor caminho no tratamento da complexidade dos sistemas (Henderson-Sellers 1996). Além disso, a tecnologia baseada em objetos oferece subsídios que atendem à natureza iterativa e dinâmica do processo de desenvolvimento. Para aumentar a qualidade e produtividade, fornece mecanismos básicos para reutilização (Karlsson 1995). Promove o desenvolvimento de software segundo a perspectiva de construção de sistemas físicos, isto é, como objetos complexos constituídos pela composição de diversos objetos pré-fabricados de uso geral (Cox 1991), (Winblad et al. 1993).

A introdução da tecnologia orientada a objetos pode apresentar problemas e alguns projetos baseados em objetos tem falhado (Martin 1994), (Capper et al. 1994). Portanto, como qualquer outra tecnologia de software, a tecnologia orientada a objetos não deve ser considerada uma panacéia.

Para ter sucesso no uso desta tecnologia e atingir a qualidade desejada nos produtos desenvolvidos é preciso treinamento intenso, um ambiente de desenvolvimento adequado, a utilização correta do método orientado a objetos escolhido, uma ferramenta CASE que apóie todo o processo de desenvolvimento e uma gerência que saiba o que está fazendo, controlando a qualidade dos produtos gerados e do processo de desenvolvimento (Rocha et al. 1994), (Clunie et al. 1995b; 1995c).

Firesmith (1993), Martin (1994) e Henderson-Sellers (1994;1996) identificam vantagens no uso da tecnologia de orientação a objetos. Apresentamos, aqui, as vantagens que consideramos relevantes do ponto de vista da qualidade dos sistemas, principalmente as relacionadas às fases iniciais do processo de desenvolvimento.

- Permite gerar uma especificação de requisitos e de projeto mais compreensível, através do uso de distintos tipos de abstração e de objetos que modelam diretamente as entidades do domínio da aplicação.

- Permite uma fácil transição entre as atividades de desenvolvimento de software, atingindo um alto grau de uniformidade, isto é, usando o mesmo paradigma, conceitos, modelos e notação ao longo de toda a atividade de especificação de requisitos e de projeto.

- Melhora a manutenibilidade, extensibilidade e gerência de configuração das especificações, projeto e código, pois permite-nos trabalhar com classes e objetos ao invés de operações (funções).

- Permite que as classes sejam projetadas de forma que possam ser reutilizadas em outras aplicações. Para minimizar este reaproveitamento, as classes podem ser construídas de forma que possam ser adaptadas (customizadas) a cada situação. Além disso, as classes projetadas para reutilização repetida tornam-se estáveis, aumentando a estabilidade do sistema como um todo.

- Permite aplicar melhor os princípios de engenharia de software relacionados a modularidade e ocultamento da informação (encapsulamento), através da geração de módulos[1] com alto grau de coesão e com melhores interfaces abstratas.

- Facilita a transição para a implementação do sistema com o uso de uma linguagem orientada a objetos (LOO) e um gerenciador de banco de dados orientado a objetos (GBOO).

- Aumenta a qualidade das especificações de análise e projeto, assim como a produtividade, pois permite um processo de desenvolvimento iterativo e recursivo, facilitando a verificação e validação dos produtos que estão sendo gerados. Além disso, o software construído a partir de classes estáveis, bem testadas e aprovadas, é menos suscetível a erros do que o software construído a partir do nada. Cada método em uma classe é, por si só, relativamente simples e pode ser projetado para ser confiável.

A grande compensação da tecnologia baseada em objetos vem quando ela é utilizada em larga escala, o que maximiza o reaproveitamento e minimiza os custos de manutenção, permitindo a construção de sistemas com mais qualidade (Melo et al. 1995).

Estas vantagens fazem com que, atualmente, existam poucas dúvidas de que a tecnologia baseada em objetos vá, aos poucos, permear quase todo o desenvolvimento de software (Martin 1994).

[1] O conceito de módulo em orientação a objetos pode ser entendido como classe-objetos ou *clusters*

I.2. Motivação

Usuários demandam, constantemente, novos e melhores produtos de software. Para satisfazer esta demanda, gerentes de desenvolvimento de software procuram utilizar novos métodos e técnicas com o intuito de melhorar a qualidade dos produtos e a produtividade no processo de desenvolvimento.

O desenvolvimento de software é uma atividade que consome tempo e recursos. Apesar do incremento na automatização das atividades de desenvolvimento de software, os recursos ainda são insuficientes. Torna-se necessário oferecer informações precisas que ajudem os gerentes a tomar decisões, elaborar planos e cronogramas, assim como, destinar recursos para as diferentes atividades que são realizadas durante o processo de desenvolvimento de software.

Neste sentido, métricas de software são de fundamental importância, pois, além de constituírem uma fonte de informações na tomada de decisões, ajudam os gerentes a identificar onde os recursos devem ser destinados (Basili et al. 1995), (Fenton 1994), (Möller e Paulish 1993). Assim sendo, resulta necessária a existência de métricas, principalmente quando as organizações estão adotando uma nova tecnologia.

No final da década passada, algumas empresas americanas introduziram a tecnologia de orientação a objetos em seus ambientes de desenvolvimento de software. Atualmente, métodos de análise e projeto orientados a objetos e linguagens de programação orientadas a objetos estão sendo utilizados em grandes e pequenas organizações dedicadas ao desenvolvimento de software.

O processo de desenvolvimento de software orientado a objetos e seus produtos gerados apresentam características diferentes, se comparados com o desenvolvimento tradicional, que nos induzem a pensar que as métricas existentes para o paradigma tradicional são inadequadas para este paradigma. Portanto, métricas propostas para o paradigma de orientação a objetos devem ser definidas e validadas de maneira a tornarem-se úteis na indústria (Brooks 1993), (Chidamber e Kemerer 1991;1994), (Abreu e Carapuça 1994), (Lorenz e Kidds 1994), (Basili et al. 1995; 1996), (Li e Henry 1995), (Henderson-Sellers 1996).

Existem relativamente poucas experiências descritas na literatura com relação à definição e validação de atributos de qualidade aplicáveis ao paradigma de orientação a objetos. A maioria das propostas são baseadas nos trabalhos descritos por Chidamber e Kemerer (1991;1994). Contudo, existem algumas experiências que se preocupam por aspectos relacionados com a manutenção (Li e Henry 1993), (Lattanzi e Henry 1994),

(Li et al. 1995) e complexidade (Chen e Lu 1993), (Henderson-Seller 1996) dos sistemas orientados a objetos.

A maioria das experiências, entretanto, propõe atributos de qualidade relacionados a aspectos estruturais aplicáveis ao projeto detalhado e código. Não são discutidos atributos de qualidade relativos à modelagem e à sua documentação. Além disso, a confiabilidade conceitual, como por exemplo aspectos relacionados à fidedignidade e suficiência do modelo, não são abordados. Pode-se, portanto, dizer que, na literatura, as experiências relatadas não se preocupam em avaliar completamente a qualidade dos produtos gerados nas fases iniciais do processo de desenvolvimento.

Considerando esta problemática, três aspectos nortearam nossa proposta de identificar e definir atributos de qualidade para as fases iniciais do processo de desenvolvimento orientado a objetos. Uma primeira motivação está vinculada à nossa experiência a nível de tese de mestrado e o desejo de dar continuidade a este trabalho desenvolvido na área de qualidade de especificações de requisitos (Clunie 1987), (Clunie e Rocha 1987a; 1987b).

Uma outra motivação está relacionada com a nossa participação num projeto de desenvolvimento de um grande sistema orientado a objetos, onde está envolvida a *Fundação COPPETEC* e uma empresa estatal. Durante o desenvolvimento pudemos constatar as dificuldades encontradas e a necessidade de se ter atributos de qualidade e formas de avaliação para os produtos gerados nas fases iniciais do processo de desenvolvimento (Rocha et al. 1994), (Clunie et al. 1995b; 1995c).

Ao identificarmos as dificuldades, propomos, como uma forma de minimizá-las, a elaboração de um manual completo e sistemático, dirigido ao grupo responsável pela avaliação da qualidade das especificações orientadas a objetos. Neste manual são definidas características de qualidade, processos de avaliação, comentários e sugestões.

A linha de pesquisa de Engenharia de Software do Programa de Engenharia de Sistemas da *COPPE/UFRJ* tem estado envolvida em atividades de pesquisa na área de qualidade de software desde 1985, com o objetivo de propor técnicas e métodos de avaliação da qualidade de software para diferentes produtos, domínios de aplicação e paradigmas de desenvolvimento. Os resultados destas pesquisas estão descritos em Clunie (1987), Palermo (1989), Andrade (1991), Passos, (1991), Bahia (1992), Comerlato (1994), Campos G. (1994), Campos F. (1994), Oliveira (1995), Blaschek (1995) e Braga (1995) entre outros. Dando continuidade aos objetivos da linha de pesquisa, nossa proposta visa responder a uma necessidade no que se refere à qualidade de produtos de software desenvolvidos segundo o paradigma de orientação a objetos.

I.3. Objetivo da Tese

Esta tese tem como objetivo elaborar uma proposta para *Avaliação da Qualidade de Especificações Orientadas a Objetos.* Esta proposta foi desenvolvida como resultado da experiência na participação de um grande projeto de desenvolvimento orientado a objetos e da revisão da literatura sobre o desenvolvimento e qualidade em orientação a objetos.

O trabalho insere-se no contexto das pesquisas realizadas em qualidade de software pela linha de Engenharia de Software da *COPPE/UFRJ,* tendo sido, também, um dos projetos do *Programa Brasileiro da Qualidade e Produtividade, no Subprograma Setorial da Qualidade e Produtividade em Software - SSQP/SW* (Weber 1996).

I.4. Metodologia de Pesquisa

A metodologia de pesquisa adotada neste trabalho está baseada no método proposto por Basili (Basili e Rombach 1988), (Basili et al. 1995) - *Goal/Question/Metric (GQM)* - já utilizado em muitos projetos de pesquisa em Engenharia de Software. O método facilita a organização e orientação na determinação dos objetivos gerais e específicos e o refinamento de cada objetivo em um conjunto de questões de interesse nas áreas correlatas da pesquisa. A partir daí, são definidas as atividades gerais que foram realizadas (Figura I.1.).

De modo geral, a pesquisa foi desenvolvida realizando as seguintes etapas:

Etapa 1. Definição dos objetivos gerais e específicos da pesquisa (Clunie et al. 1995)

Etapa 2. Pesquisa bibliográfica centrada principalmente nas seguintes questões de interesse:

- Métodos de Desenvolvimento de Software Orientado a Objetos
- Métricas aplicáveis ao Paradigma de Orientação a Objetos
- Avaliação da Qualidade de Especificações

Etapa 3. Análise de métodos de desenvolvimento de software orientado a objetos (Clunie e Werner 1994), (Clunie et al. 1995c)

Etapa 4. Definição do processo de geração/avaliação de especificações orientadas a objetos

Etapa 5. Identificação e discussão de atributos de qualidade (Clunie et al. 1994a; 1994b; 1994c; 1995a), (Clunie e Xexeo 1996)

Etapa 6. Hierarquização dos atributos de qualidade

Etapa 7. Avaliação de especificações orientadas a objetos

Etapa 8. Elaboração de um Manual para Controle da Qualidade de Especificações Orientadas a Objetos (Clunie et al. 1994a; 1997)

I.5. Organização da Tese

Esta tese está dividida em oito capítulos e quatro apêndices. O primeiro deles define o objetivo do trabalho, suas motivações e sua organização.

No Capítulo II - ***Qualidade de Software*** - são definidos aspectos referentes a qualidade de software. São apresentados os modelos de avaliação da qualidade mais representativos e discutidos os problemas das medições em software. Por fim, é feita uma recompilação de vários projetos visando a pesquisa aplicada e a divulgação de métricas.

No Capítulo III - ***Desenvolvimento de Software Orientado a Objetos*** - são apresentados os princípios e conceitos do paradigma de orientação a objetos, discutindo o conceito de modelagem, e são caracterizados os métodos de desenvolvimento de software orientado a objetos. São descritos os métodos de desenvolvimento orientado a objetos mais relevantes segundo a literatura especializada e, por fim, são apresentadas algumas considerações sobre os métodos e a qualidade de software.

No Capítulo IV - ***Qualidade de Software e o Paradigma de Orientação a Objetos*** - são discutidas as propostas mais representativas sobre métricas aplicadas ao paradigma de orientação a objetos.

No Capítulo V - ***Avaliação da Qualidade de Especificações Orientadas a Objetos*** - iniciamos com uma discussão sobre o desenvolvimento de especificações orientadas a objetos. Em seguida, é definido o *Processo de Geração/Avaliação de Especificações Orientadas a Objetos* e é discutido, em detalhe, um conjunto de atributos de qualidade para avaliar especificações orientadas a objetos. A organização da discussão dos atributos é realizada de acordo com o Método de Avaliação da Qualidade de Software proposto por Rocha (1983).

No Capítulo VI - ***Hierarquização dos Atributos de Qualidade de Especificações Orientadas a Objetos*** - descrevemos os resultados de uma pesquisa de campo realizada com o objetivo de obter um perfil da qualidade desejável para especificações orientadas a objetos. A partir desta pesquisa, verificamos que os atributos de qualidade propostos adequam-se às especificações desenvolvidas utilizando este paradigma e estabelecemos uma hierarquização dos mesmos, de acordo com o seu grau de importância. A pesquisa foi realizada consultando a opinião de profissionais com experiência no uso do paradigma de orientação a objetos no desenvolvimento de projetos reais.

No Capítulo VII - ***Experiência de Avaliação de Especificações Orientadas a Objetos*** - descrevemos uma experiência de avaliação de especificações orientadas a objetos. O capítulo se inicia com a descrição, em linhas gerais, do projeto que serviu de base para a realização desta experiência de avaliação. Em seguida apresentamos a experiência de avaliação, segundo os atributos propostos e os processos de avaliação definidos.

No Capítulo VIII - ***Conclusões e Perspectivas Futuras*** - são apresentadas as conclusões do trabalho, destacando-se suas contribuições e os trabalhos futuros que podem ser realizados dentro do contexto da tese.

O Apêndice I - ***Manual para Controle da Qualidade de Especificações Orientadas a Objetos*** - contém um guia completo e sistemático, para ser utilizado durante o processo de desenvolvimento e avaliação de especificações orientadas a objetos. Neste manual são definidas características de qualidade, processos de avaliação e incluídos comentários e sugestões para correção quando isto for pertinente .

O Apêndice II - ***Formulários de Avaliação de Especificações Orientadas a Objetos*** - contém um conjunto de formulários a serem utilizados para avaliação de especificações orientadas a objetos.

O Apêndice III - ***Instrumento para Hierarquizar Critérios de Qualidade para Especificações Orientadas a Objetos*** - contém o formulário utilizado na pesquisa de campo para obter o grau de importância dos atributos de qualidade para especificações de requisitos e de projeto, segundo as visões de analistas/projetistas e programadores.

O Apêndice IV - ***Formulários de Coleta de Dados*** - são apresentados os formulários de coleta de dados para a realização da avaliação das especificações.

Figura I.1. - Metodologia de Pesquisa

Capítulo II

Qualidade de Software

"A importância da qualidade teve um crescimento extraordinário na sociedade em geral. Muito desse crescimento deve-se ao fenômeno chamado de "a vida por trás das barreiras da qualidade". A sociedade industrial proporciona à coletividade os incríveis benefícios da tecnologia. Entretanto, a continuidade desse estilo de vida depende exclusivamente da qualidade dos bens e serviços, que constituem a base de uma sociedade industrial"

J. M. Juran

II.1. Introdução

Neste capítulo são discutidos diversos aspectos relacionados a qualidade de software com o intuito de proporcionar uma visão geral sobre o assunto. Para atingir este objetivo, o capítulo foi organizado em oito seções. A segunda delas introduz uma discussão sobre a necessidade de qualidade em software, para posteriormente, na terceira, se chegar à definição de qualidade de software. Na quarta parte, apresenta-se o problema das medições de software, onde discutimos o conceito de métrica em software, suas propriedades e aspectos de confiabilidade das medidas. Na quinta seção são discutidas algumas técnicas de controle da qualidade de software. Na sexta parte, são comentados os modelos mais relevantes de avaliação da qualidade. Seguidamente são apresentadas, de forma resumida, algumas experiências de projetos visando à investigação aplicada e à divulgação de métricas. Por fim, apresentamos as conclusões do capítulo.

II.2. A Necessidade de Qualidade em Software

Existe uma tendência mundial no sentido do aumento das expectativas do usuário em relação à qualidade. Acompanhando essa tendência, houve uma crescente tomada de consciência de que melhorias contínuas na qualidade são freqüentemente necessárias para atingir e assegurar um bom desempenho econômico.

Por outro lado, o crescimento da competição pela qualidade tornou-se intenso, pois as indústrias vêm respondendo a essa necessidade crescente de qualidade dentro de seu processo produtivo. Essa preocupação se manifesta através da ampliação dos objetivos de qualidade nas empresas, incluindo atividades de apoio à produção, processos comerciais e necessidades do usuário; planejamento da qualidade utilizando uma abordagem estruturada e formal; administração da qualidade e treinamento em administração da qualidade. Este comportamento da indústria obriga a trabalhar em direção da obtenção de qualidade, considerando seu planejamento, controle e aperfeiçoamento na produção de bens, serviços e *produtos de software.*

A crescente dependência da sociedade por sistemas computacionais e, em especial, por produtos de software, tem direcionado o foco central do processo de desenvolvimento de software para a qualidade. Como conseqüência, isso tem levado ao desenvolvimento de iniciativas para elaboração de padrões internacionais que estabeleçam procedimentos para assegurar e avaliar a qualidade do processo de software, e, conseqüentemente, do produto. A série de normas *ISO*[1] surge como resposta a essa necessidade.

As normas *ISO* 9000 estabelecem orientações/recomendações/diretrizes e usos destas normas. Estão compostas pelas normas *ISO* 9001, 9002, 9003, 9004 relacionadas com gestão e garantia da qualidade. Elas foram elaboradas pelo *ISO Technical Committee 176* (*ISO TC* 176) (*ISO* 9000 1990) e são compostas de:

- ***ISO 9001*** - compreende um modelo para garantia da qualidade em projeto, desenvolvimento, produção, instalação e assistência técnica.
- ***ISO 9002*** - compreende um modelo para garantia da qualidade em produção e instalação.
- ***ISO 9003*** - compreende um modelo para garantia da qualidade em inspeção e ensaios finais.
- ***ISO 9004*** - compreende a norma que estabelece como deve ser a gestão da qualidade na empresa. É um guia geral para todas as organizações.

Para o caso do desenvolvimento de software tem-se a norma *ISO 9000-3,* uma particularização da *ISO 9001* para software e que compreende procedimentos para a garantia da qualidade de software em relação ao processo de desenvolvimento.

Esta última norma está composta de três partes: (ISO 9000-3 1990)

1- *ISO* - International Organization for Standarization

- ***Estrutura*** - descreve os aspectos organizacionais a serem considerados na produção de software, o que inclui: responsabilidade da administração, sistemas de qualidade, auditorias internas do sistema e ações corretivas.
- ***Atividades do ciclo de vida*** - define as ações necessárias para conduzir cada fase do processo de desenvolvimento, o que inclui: revisão de contrato, especificação de requisitos do cliente, planejamento do desenvolvimento, planejamento da qualidade, projeto e implementação, testes e validação, aceitação, cópia, entrega e manutenção.
- ***Atividades de apoio*** - define as atividades de suporte à produção, entrega e manutenção de software, o que inclui: gerenciamento de configuração, controle de documentos, registros da qualidade, medições, regras práticas e convenções, ferramentas e técnicas, compras, aquisição, produto de software incluído e treinamento.

O processo de garantia da qualidade de software está relacionado à qualidade de seu processo de desenvolvimento e às características de qualidade do próprio produto. Ou seja, produto e processo estão fortemente relacionados e não devem ser tratados de forma separada quando se analisa qualidade. Assim sendo, para estabelecer um programa de garantia da qualidade de software, é necessário ter procedimentos para garantia da qualidade a nível de produto e a nível do processo de desenvolvimento (Bazzana et al. 1993).

Tusukmo et al. (1995) estabeleceu que a relação entre processo e produto pode ser examinada respondendo às seguintes questões:

- Como os requisitos de qualidade do produto podem orientar o processo de desenvolvimento?
- De que forma os documentos gerados durante o processo de desenvolvimento podem ser utilizados na definição dos requisitos de qualidade do produto?
- Como a avaliação dos produtos intermediários e final deve realimentar o processo de desenvolvimento?
- De que forma as atividades realizadas durante o desenvolvimento podem ser modificadas para permitir realizar avaliações de produtos intermediários?

Na mesma linha de preocupação, o Modelo de Maturidade de Capacidade de Processo de Software (*Capability Maturity Model - CMM*) foi desenvolvido pelo Instituto de Engenharia de Software (*Software Engineering Institute- SEI*) na *Universidade de Carnegie Mellon.* Este modelo busca quantificar a capacidade de uma organização produzir, de forma previsível e consistente, produtos de software de alta qualidade (Saideian e Kuzara 1995).

O *CMM* incorpora à infra-estrutura de maturidade de processo do *SEI* um questionário e dois métodos de avaliação de processo de software. Estes são compostos por 18 processos-chave, que representam 343 práticas-chave que têm mostrado ser eficazes no aumento da capacidade de desenvolvimento e manutenção de software (Humphrey 1989), (Paulk et al. 1993). Esta infra-estrutura é um caminho evolutivo dos processos e possui cinco níveis: *inicial* (caos), *repetível, definido, gerenciado* e *otimizado*, onde as capacidades nos níveis mais baixo fornecem bases progressivamente mais sólidas para os níveis superiores. Cada estágio de desenvolvimento, ou nível de maturidade, distingue uma capacidade de processo de software na organização.

II.3. Conceituação de Qualidade de Software

O conceito de qualidade é utilizado no quotidiano para descrever o grau de excelência de um produto ou serviço, sendo uma palavra genérica e não de uso exclusivo do software. Por isso, o termo qualidade têm distintos significados, pois são consideradas visões diferentes das pertencentes às pessoas diretamente envolvidas com a avaliação de um produto ou serviço. Uma característica comum às várias formas de definir o conceito é a influência de percepções e expectativas, que, inegavelmente, evidenciam a dimensão psicológica associada ao conceito qualidade (Mosley 1993). O que demonstra que qualidade é um conceito complexo, de várias facetas, e que pode ser descrito atendendo a cinco perspectivas identificadas por Garvin em Kitchenham e Pfleeger (1996). São elas: *perspectiva transcendental, perspectiva do usuário, perspectiva do fabricante, perspectiva do produto* e *perspectiva baseada no valor.*

É consenso que qualidade é uma meta a ser perseguida e que o software exige, para seu desenvolvimento, um processo sistemático e controlado. Portanto, um produto de software com qualidade requer um processo de desenvolvimento orientado a qualidade (Basili e Musa 1990). Vários autores vêm discutindo o conceito de qualidade de software. De modo geral, todos concordam em considerar a qualidade como um conceito multidimensional cujas dimensões incluem a entidade de interesse, o ponto de vista sobre a entidade e as características de qualidade pertinentes a esta entidade. Rocha (1987) sintetiza esse conceito da seguinte maneira:

> *"Qualidade é um atributo associado a alguma coisa. Assim sendo, qualidade não pode ser definida universalmente mas deve ser definida para o item em questão (qualidade de especificações, qualidade de projeto, qualidade de programas etc.). Qualidade é, também, um conceito multidimensional que se realiza através de um conjunto de atributos ou características. Qualidade de software é, portanto, um conjunto de propriedades a serem satisfeitas em determinado grau, de modo que o software satisfaça às necessidades de seus usuários".*
>
> **Rocha - 1987**

A qualidade de software é tão necessária quanto não é *absoluta.* É um conceito multidimensional cujas definições estão baseadas num conjunto de propriedades a maioria das vezes de difícil quantificação. Para cada domínio de aplicação, a qualidade tem conceitos e significados diferentes. Desta forma, a busca pela qualidade depende do contexto da aplicação, pois, para cada tipo de aplicação, um conjunto diferente de propriedades será utilizado e quantificado de forma independente. Porém, existirão sempre propriedades que podem ser medidas absolutamente e que, na maioria dos casos, terão maior importância na qualidade do produto final.

A experiência tem demonstrado que a qualidade de software não surge espontaneamente, e sim através de um conjunto de procedimentos, cuidadosamente observados ao longo do desenvolvimento. Portanto, a qualidade do software é uma responsabilidade de todos os que participam na sua construção e deve ser construída em cada fase do ciclo de desenvolvimento. Não se pode adicionar qualidade em datas posteriores, por pessoal que não participou do desenvolvimento, a menos que sejam utilizadas adequadamente técnicas de re-engenharia e sejam considerados os custos envolvidos. Collins et al. (1994) propõem que as pessoas envolvidas no processo de desenvolvimento tenham responsabilidades éticas para minimizar os riscos encontrados e derivados de suas participações nesse processo, e com isso contribuir para melhorar a qualidade do produto.

Para que uma organização possa construir software com a qualidade desejada e se possa medir se o software produto possui esta qualidade, é necessário que se defina a qualidade de software de acordo com seu contexto. Para isto, deve-se analisar a natureza do produto e identificar quais são as características consideradas como desejáveis. Estas características devem ser possíveis de serem avaliadas, existindo um grau mínimo a ser atingido para que um software seja considerado de boa qualidade.

II.4. O Problema das Medições em Software

A determinação da qualidade de qualquer produto é um fator importante no nosso viver diário, pois convivemos com isso. A qualidade é avaliada de uma forma quase direta, por comparação de objetos sob idênticas condições, considerando-se atributos pré-determinados. Os objetos serão julgados de acordo com suas características próprias. Entretanto, este tipo de julgamento é muito subjetivo e para ter algum valor é necessário que seja conduzido por especialistas.

Portanto, para a avaliação da qualidade do software, devem ser utilizados parâmetros objetivos que fundamentem julgamentos e forneçam argumentos para futuras

avaliações. Além disso, a qualidade só pode ser atingida quando são utilizados métodos e técnicas que tornam o processo de desenvolvimento organizado e metódico.

A qualidade de software é traduzida através de características de qualidade. A obtenção de dados quantitativos relativos a essas características é, assim, fundamental para introduzir melhorias no processo de desenvolvimento. Por isso a medição deve fazer parte do processo de desenvolvimento de software, tal como é corrente na generalidade dos processos industriais, no sentido de permitir a previsão e o monitoramento dos custos e cronogramas, controlar a qualidade e, por outro lado, melhorar a compreensão e validação do próprio processo de desenvolvimento. Sem a obtenção desses dados quantitativos, torna-se difícil, senão impossível, estabelecer de forma racional condições contratuais - *prazos e orçamentos* -, tomar decisões de atribuição de recursos humanos e financeiros, ou tentar atingir objetivos definidos. Por isso, a medição de software é uma chave vital para a escalada de excelência no software. Sem ela, a avaliação e melhora do processo e produto do software é impossível.

Qualidade de software é o resultado de muitos aspectos, não, somente, da qualidade do código. O produto da *especificação de requisitos e projeto* de um determinado software é uma representação que, como qualquer outro produto, também possui padrões de qualidade que precisam ser quantificados. Contudo, o software é abstrato e fisicamente inexistente, o que dificulta sua avaliação.

A subjetividade e a experiência também são aplicadas à determinação da qualidade de software. Para ajudar a minimizar o problema, precisamos de uma *estrutura de avaliação da qualidade* que permita que o processo de controle da qualidade seja de fato viável. Esta estrutura deve estar composta, principalmente, por um *método de avaliação da qualidade, técnicas de controle da qualidade, guias de avaliação* e *ferramentas de suporte*. Desta maneira é possível realizar medições quantitativas baseadas em uma análise objetiva. É óbvio que este controle da qualidade deve ser conduzido através de uma gestão eficaz.

Qualquer medição possui um certo grau de imprecisão. Daí ser necessário estabelecer o nível de tolerância segundo o qual se confia ou não no resultado da medição. Assim sendo, parte do objetivo da medição é a determinação dessa área de tolerância e de como ela pode afetar o uso e a interpretação do resultado da medição.

Segundo Kitchenham e Mellor em Fenton (1991), um dos maiores problemas das medições de software está na habilidade para coletar e analisar dados confiáveis que possam ser utilizados durante o desenvolvimento para avaliação e compreensão de

processos e produtos. Para minimizar estes problemas, os autores sugerem diretrizes de como coletar dados, quando coletar dados, como armazenar e recuperar dados e como analisar os dados.[2]

Podemos concluir, então, que a engenharia de software deve estar marcada pela disponibilidade de parâmetros de medidas, precisas e bem entendidas. No entanto, medidas de software não podem ser obtidas como nas demais disciplinas de engenharia, baseadas em ciências físicas. Isto torna necessário que a abordagem dada à medição de software seja marcada pelo paradigma científico tradicional de hipótese, avaliação, crítica e revisão. Além disso, a engenharia de software, da mesma forma que em outras disciplinas, deve aderir à ciência da medição, para atingir uma total aceitação e validade (Fenton 1994). Assim sendo, a engenharia de software deve caminhar nessa direção para se tornar uma real disciplina de engenharia e para satisfazer as demandas futuras de desenvolvimento de software (Basili e Musa 1990).

II.4.1. Métricas de Software

O conceito de medidas em engenharia de software ou o que, geralmente, é chamado de *métricas de software,* não é novo. O crescente interesse pelas métricas de software baseia-se no controle que precisamos ter sobre o produto e o processo de desenvolvimento; isto é, para que se tenha garantia da qualidade do software, temos que ter mecanismos para medir esta qualidade. Outra forma de abordar a difusão do interesse por métricas é a necessidade de prever, ainda nas fases iniciais, problemas que poderão advir nas fases de construção e testes. Portanto, o principal objetivo do estudo das métricas de software não é só a determinação estática das propriedades do software, mas sim uma previsão científica durante todo o ciclo de vida. O uso de métricas reduz a subjetividade na avaliação da qualidade do software, pois fornece bases quantitativas para a tomada de decisões sobre a qualidade do software. Entretanto, o uso de métricas não elimina a necessidade de um julgamento humano na avaliação da qualidade do software (Schneidewind 1992).

Ghezzi et al. (1991) e Pressman (1995) assinalam que o objetivo do estudo das métricas em software é ajudar na escolha da melhor alternativa nos vários pontos do ciclo de vida. Através das métricas, é fornecida uma indicação de quão adequado é o software às solicitações implícitas e explícitas do usuário. Além disso, é possível prever, ainda nas fases iniciais, problemas que poderão advir nas fases de construção e testes.

---

[2]- A abordagem completa sobre análise e coleta de dados é encontrada em Fenton (1991), pp. 89 - 110

Portanto, métricas de software é uma área destinada à designação numérica ou simbólica a determinados atributos de entidades (produto, processo ou recurso), usadas para quantificá-los, conservando suas relações empíricas. Consistem de uma escala, medidas e métodos de medição (Ince 1990), (Shepperd 1992), (Fenton 1994), (Miyoshi et al. 1993) e (Möller e Paulish 1993).

As métricas são utilizadas ao longo do desenvolvimento. Portanto, têm como objetivo ajudar a responder às seguintes perguntas, que vão surgindo à medida em que se vai progredindo no desenvolvimento.

- É o momento adequado para passar para a etapa seguinte?
- Será necessário retroceder a uma etapa anterior?

As respostas a estas perguntas devem possuir um certo grau de exatidão, para que se possa continuar com o controle efetivo do processo de desenvolvimento. Com o auxílio de métricas, pretende-se procurar as respostas às duas perguntas acima. Entretanto, ainda, não se consegue obter respostas com o grau de exatidão desejado.

As métricas, contudo, nem sempre têm sido utilizadas de uma forma sistemática e metódica. Com efeito, o estado da arte nesta área caracteriza-se por uma certa profusão de métricas propostas, mas poucas têm sido validadas. Mesmo no caso das métricas mais estudadas, como Linhas de Código Fonte - *LOC* (Jones 1991), métricas de Halstead (1977) e McCabe (1976), não há um consenso generalizado sobre aquilo que pretendem quantificar. Quando são utilizadas, os resultados em ambientes diversificados nem sempre são corroborantes, o que, acrescentando-se à falta de fundamentação teórica da maioria das métricas, levou a que apenas algumas tivessem aceitação generalizada (Sheppard 1995).

Embora as métricas não sejam a panacéia para todos os males no processo de desenvolvimento de software, quando utilizadas em conjunto com outras atividades, tais como um recrutamento cuidadoso dos membros das equipes de desenvolvimento e de controle de qualidade, a sua utilização tem sido apontada como um fator catalisador da qualidade e produtividade. Devido a seu potencial preditivo, elas podem ser utilizadas para (DeMarco 1989), (Ejiogu 1991;1993), (Abreu 1992), (Möller e Paulish 1993), (Chidamber e Kemerer 1994), (Pfleeger 1995), (Basili et al. 1995), (Henderson- Sellers 1996):

- Melhorar a compreensão da atividade de gerência, dado que a obtenção de dados quantitativos faz contornar uma certa *imprevisibilidade* associada ao

desenvolvimento de software, isto é, reduzir a subjetividade em favor da objetividade na atividade de gerência do projeto.

- Identificar problemas mais cedo, permitindo ações corretivas em tempo útil, com um menor impacto nos prazos e custos previamente estabelecidos para o projeto.
- Avaliar métodos de desenvolvimento, estruturas organizacionais, linguagem utilizada e o desempenho da equipe desenvolvedora (individual e coletivamente).
- Ajudar a gerência a ter uma estimativa quantitativa da qualidade de seu trabalho.
- Manter a credibilidade da entidade responsável pelo projeto, dado que a confiança dos clientes aumenta ao evidenciarem ações concretas, no sentido de melhorar a qualidade dos produtos e serviços fornecidos.
- Realizar *feedback* para o pessoal envolvido no projeto, pois a atitude dos integrantes da equipe é motivada pelo fato de se estarem a medir parâmetros do produto e processo que possam realmente provar que se está evoluindo positivamente.

Os benefícios acima mencionados constituem uma vantagem para as organizações que trabalham com um *programa de métricas*[3]. Embora os benefícios atingidos através da implementação de um programa de métricas sejam inegáveis, muitas organizações o fazem apenas com o objetivo de cumprirem com os requisitos necessários para sua certificação (Linkman e Walker 1991).

Mais recentemente, Pfleeger (1995) propõe um programa de métricas combinando três enfoques: O modelo *GQM - Goal Question Metrics* - (Basili e Rombach 1988), o processo de maturidade do *SEI* (Paulik et al. 1993) e o modelo de processo. Segundo a autora, a combinação destes três enfoques permite que a organização possa coletar e analisar métricas adequadas de acordo com suas necessidades e considerando as características do projeto e da organização.

Alguns estudos detalhados no contexto do projeto *ESPRIT* indicam que a implementação de um programa de métricas ajuda na obtenção de melhores resultados do ponto de vista da gerência, melhorando a estimativa, a produtividade e a qualidade. Com efeito, um número cada vez maior de organizações tem obtido resultados promissores com a implementação de programas de métricas. Estes indicadores positivos têm alertado a comunidade de gestores de projetos, pelo que se espera que, também, nas empresas com projetos de menor dimensão se possa vir a colher idênticos benefícios.

[3]- Refere-se ao conjunto integrado de atividades que inclui: estabelecimento de requisitos de qualidade de software, identificação das métricas, implementação das métricas, análise dos resultados e validação das métricas (Schneidewind 1992).

No caso brasileiro, constatou-se que a maioria das empresas nacionais ainda não apresentam maturidade no tratamento da qualidade. Numa pesquisa realizada em 1995 pelo *Programa Brasileiro de Qualidade e Produtividade em Software - SSQP/ SW - PBPQ ob*servou-se que apenas 22% das empresas elaboram e mantêm sistematicamente atualizado, com periodicidade fixa, plano estratégico ou plano de metas, 39% incluem de forma sistemática metas ou diretrizes para qualidade nestes planos e 25% coletam indicadores da qualidade de seus produtos e serviços. Programas da qualidade total estão implantados em 11% das empresas e a contabilidade de custos da qualidade e da não qualidade é mantida de forma sistemática em 4%. Um número elevado de empresas, 86%, desconhece o modelo *CMM - Capability Maturity Model.* No entanto, 35% mantêm procedimentos específicos de garantia da qualidade do produto de software, sendo menos de 2% baseados nas normas *ISO 9126* para auto-avaliação, enquanto 15% das empresas possuem equipe específica dedicada à garantia da qualidade (SEPIN/MCT 1995).

Quanto ao uso de métodos, ferramentas e procedimentos de Engenharia de Software, foi possível observar, nesta pesquisa, o pouco uso de procedimentos de revisão de software (por exemplo, inspeção e *walkthrough*) e de ferramentas CASE. Os resultados obtidos apontam que há uma média de utilização por empresa de 6,2 diferentes técnicas de engenharia de software e de 3,3 ferramentas automatizadas (SEPIN/MCT 1995).

II.4.2. Propriedades e Taxonomia

Uma métrica é um indicador ou escala que permite quantificar um atributo de um produto ou do próprio processo de desenvolvimento. O termo produto pode ser aplicado a especificações de requisitos, especificações de projeto, código fonte ou código executável.

Uma métrica é útil se auxilia no desenvolvimento de software e é capaz de estimar os parâmetros de um processo ou produto e não apenas descrevê-lo. Deve fornecer um retorno para o usuário da medida; isto é, a medida deve fornecer informação sobre o atributo ou característica de interesse, que permita ao usuário ter um melhor entendimento para melhorar seu processo ou produto (Ejiogu 1993). Para que isto aconteça, as métricas deverão ter as seguintes propriedades desejáveis: (DeMarco 1989), (Möller e Paulish 1993), (Abreu e Carapuça 1994).

- *Propriedade 1 - **As métricas devem ser definidas formalmente***
  Diferentes pessoas, em momentos e lugares diferentes, devem coletar os mesmos valores quando realizam a medição do mesmo sistema. A subjetividade na avaliação da qualidade de software impede que isto possa se realizar com

confiabilidade, pois escalas subjetivas são freqüentemente utilizadas (exemplo: *Muito Baixo, Baixo, Alto e Muito Alto*).

- *Propriedade 2 - **As métricas devem ser independentes do tamanho do sistema***
  Para que as métricas sejam úteis, elas devem ser coletadas e analisadas ao longo do tempo e em diferentes projetos, quando possível. Desta forma podem ser realizadas comparações e derivadas conclusões. No entanto, estes projetos variam em tamanho. Assim sendo, a métrica deve ser aplicável a qualquer outro sistema similar, independente de seu tamanho.

- *Propriedade 3 - **As métricas devem poder ser dimensionadas e expressas numa unidade de medida consistente***
  Métricas são utilizadas para dimensionar um determinado produto ou processo; portanto, elas devem ter uma unidade de medida. Unidades de medida subjetivas ocasionam problemas de entendimento do resultado da medição e restringem sua utilização em outras situações.

- *Propriedade 4 - **As métricas devem ser coletadas nas fases iniciais do ciclo de vida***
  Os custos de recuperação de erros aumenta, exponencialmente, à medida que o projeto progride. Portanto, as métricas devem ser coletadas nas fases iniciais do processo de desenvolvimento, pois desta forma são identificados possíveis defeitos no produto e/ou inadequações no processo de desenvolvimento.

- *Propriedade 5 - **As métricas devem ser aplicáveis a diferentes níveis de abstração***
  Sistemas de software são construídos por uma equipe de desenvolvimento. Freqüentemente, os sistemas são organizados em subsistemas, módulos e programas. Cada grupo de trabalho na equipe de desenvolvimento pode ser responsável por uma determinada parte do sistema (subsistema, módulo). Por isso, métricas são necessariamente aplicáveis não só a nível de sistema mas também a nível de seus módulos ou subsistemas.

- *Propriedade 6 - **As métricas devem ser facilmente coletadas e calculadas***
  Coletar dados de métricas é uma tarefa repetitiva que toma tempo e requer investimento. Por isso, é apropriado dispor de uma ferramenta automatizada que permita extrair as informações, permitindo computar as métricas. O esforço para construir este tipo de ferramenta é considerável, mas necessário.

- *Propriedade 7 - **As métricas devem ser independentes da influência da equipe do projeto***
  A métrica, para ser útil, tem que ser independente da influência consciente da equipe do projeto. Não pode haver nenhum modo de se alterar o valor medido, a menos que se alterem as qualidades latentes que o valor reflete.

- *Propriedade 8 - **As métricas devem ser precisas***
  O ponto chave sobre a precisão da métrica não é o fato de maximizá-la, mas sim torná-la pormenorizadamente notada e registrada como parte de cada dado coletado. A precisão é função da exatidão.

Dada a concepção de que o software é uma abstração e pela imaturidade da ciência de medição em engenharia de software, é difícil identificar métricas dirigidas a processos ou produtos, nos diferentes paradigmas de desenvolvimento e para diferentes domínios de aplicação, que possuam estas propriedades na sua totalidade.

A literatura apresenta várias propostas de classificação de métricas. Estas propostas variam dependendo do ponto de vista em que são analisadas. Por exemplo, em (Abreu 1992) encontramos uma proposta de classificação de métricas baseada em três perspectivas: quanto ao *objeto, critério* e *método*. Quanto ao *objeto,* as métricas podem ser classificadas em duas categorias: *métricas de produto e métricas de processo* (Ince 1990), (Fenton 1991).

Métricas de produto são aplicáveis aos produtos de qualquer das fases de desenvolvimento, quantificando, por exemplo, a sua dimensão, complexidade ou qualidade (confiabilidade, facilidade de utilização, manutenibilidade etc). *Métricas de processo,* devem refletir a eficácia da execução do processo, o grau de atendimento aos objetivos de qualidade do processo dentro dos prazos, e a eficiência na redução da probabilidade de falhas serem introduzidas ou permanecerem não detectadas.

Quanto ao *critério,* as métricas podem ser definidas igualmente em duas categorias: métricas objetivas e métricas subjetivas (Möller e Paulish 1993). *Métricas objetivas* são geralmente obtidas através de regras bem definidas, única forma de possibilitar comparações posteriores consistentes. Na prática, isso significa que os valores a que se chega deveriam ser sempre os mesmos, independentemente do instante, condições ou indivíduo que os determina. A obtenção destas métricas é passível de automação. *Métricas subjetivas* são baseadas em atributos cuja medição não pode ser

feita senão de uma forma subjetiva, sendo muitas vezes derivada de resultados de questionários e entrevistas, e por vezes classificada numa escala[4].

Quanto ao *método* de obtenção, podem ser classificadas como em (Fenton 1991), (Ghezzi et al. 1991) e (Pressman 1995): *Métricas elementares, primitivas ou diretas* são a expressão de um atributo único. Neste caso não dependem da medida de outro atributo, permitindo a quantificação de uma característica observada no produto. Por exemplo, inclui linhas de código, velocidade de execução e falhas observadas ao longo de um determinado período de tempo. *Métricas compostas, computadas ou indiretas* são calculadas com base em outras métricas, ou seja, envolvem as medidas de um ou mais atributos relacionados a este. Por exemplo: funcionalidade, complexidade, eficiência, manutenibilidade etc.

Por outro lado, Mann (1988), Ghezzi et al. (1991) e Pressman (1995) afirmam que as métricas de qualidade podem ser observadas dentro de duas categorias: *métricas preditivas* e *métricas de testes. Métricas preditivas* são aquelas que buscam predizer a qualidade do software. São aplicadas nos primeiros estágios do desenvolvimento e visam indicar como o produto final exibirá as características de qualidade. *Métricas de testes* são aquelas utilizadas após o processo, na implementação.

Independentemente das propostas de classificação apresentadas, a qualidade de software é caracterizada por duas grandes categorias de métricas: *métricas quantitativas* e *métricas qualitativas.* As métricas quantitativas, apesar de indicarem quantificação, não escapam do fato de poderem estar baseadas em critérios subjetivos. Por outro lado, as métricas qualitativas são difíceis de serem avaliadas e verificadas, sendo de extrema importância a experiência da equipe desenvolvedora.

II.5. Técnicas para Controle da Qualidade de Software

As avaliações devem ser realizadas utilizando-se técnicas que permitam fazer revisões dos produtos e subprodutos que estão sendo gerados ao longo do desenvolvimento. Desta forma é controlada a qualidade.

Uma avaliação é simplesmente uma revisão de um produto qualquer feita por um grupo de pessoas. O produto pode ser: listagem de código, especificação de requisitos, especificação de projeto e/ou outros produtos associados ao desenvolvimento de um sistema de software.

[4]- Em (*ISO/IEC 9126*, 1991) propõe-se a escala: excelente/bom/razoável/pobre

De maneira geral, as técnicas para controle da qualidade de software são agrupadas em duas grandes categorias: *técnicas de certificação* e *técnicas de testes*. Pela importância na avaliação de especificações, apresentaremos nesta seção somente técnicas de certificação.

- **Técnicas de Certificação**

Refere-se às técnicas de revisão que são realizadas sem submeter o programa a execução. Inclui: *walkthrough* e inspeções

Walkthrough e inspeções envolve atividades de planejamento, preparação e uma reunião da qual participam três a cinco pessoas, devendo ter uma duração em torno de duas horas. Na reunião participam um moderador, representantes da equipe de desenvolvimento, usuários e avaliadores externos ao projeto.

De maneira geral, estes tipos de revisões têm com objetivos: verificar se o software sob avaliação atinge os requisitos, identificar os requisitos que não podem ser alcançados, detectar erros em qualquer representação de software, assegurar que foram obedecidos normas e padrões, assegurar que o software é desenvolvido de forma uniforme, tornar o produto gerenciável e treinar a equipe (Presman 1995).

A diferença entre ambas as técnicas de revisão é que na técnica de inspeção o produto será examinado segundo uma série de critérios previamente estabelecidos. Assim sendo, ao aplicar estas técnicas os possíveis resultados da revisão são: aceitar o produto sem modificações, aceitar com correções, ou rejeitar o produto.

Mais recentemente, foi proposta uma variação à técnica de inspeção denominada *inspeção por fases* (Knigh 1993), onde a avaliação do produto é feita através de uma série de avaliações parciais (fases), que podem ser realizadas com *inspetores individuais* ou com *múltiplos inspetores*. Cada avaliação parcial tem por objetivo verificar se o produto possui uma ou mais propriedades. As propriedades são organizadas na inspeção por fases, de forma que cada avaliação parcial pode assumir a existência das propriedades verificadas ou validadas nas avaliações parciais anteriores.

Uma avaliação parcial, com inspetores individuais, é um processo rigoroso e controlado por uma lista de questões a serem respondidas pelo avaliador. Por outro lado, uma avaliação parcial, com múltiplos inspetores, é realizada para analisar questões subjetivas e onde se busca, através de uma reunião, obter o consenso dos avaliadores. Consiste de duas etapas: uma avaliação individual e uma avaliação em grupo.

II.6. Modelos para Avaliação da Qualidade de Software

A qualidade do software é atingida através da avaliação de um conjunto de atributos ou características claramente definidas. É necessário ter, através de um modelo, uma forma explícita de organizar e avaliar estas características, de modo a obter medidas que nos permitam controlar o processo de desenvolvimento de software.

A literatura técnica sobre qualidade de software apresenta a descrição e discussão de diversos modelos para a especificação, avaliação e controle da qualidade de software (Roche e Jackson 1994). A partir destes modelos, pretende-se determinar exatamente o escopo do que deve ser avaliado para a obtenção da qualidade. Entre os modelos de qualidade de software mais citados e que serviram de base na geração de outros modelos de qualidade, destacam-se os modelos definidos por *McCall et al.* (1977) e *Boehm et al.* (1978).

- ***McCall et al.*** (1977) define um modelo hierárquico de características de qualidade, observando quatro níveis: *orientações, fatores, critérios e métricas.* O *nível mais alto - orientações -* está relacionado com as atividades do ciclo de vida associadas ao produto final. É composto por *operação do produto, revisão do produto e transição do produto.* O segundo nível, dos fatores, é agrupado de acordo com as orientações. O terceiro nível, dos critérios, é constituído por características de qualidade que têm mais relação com o próprio software que está sendo desenvolvido. O último nível é o das métricas de qualidade. Segundo o autor, podem-se obter medidas quantitativas dos atributos representados pelos critérios. McCall considera, também, que as métricas podem ser aplicadas durante todas as fases do desenvolvimento e que a estrutura hierárquica de qualidade suporta formulação matemática e a validação das relações entre as métricas e os fatores de qualidade.

- ***Boehm et al.*** *(1978)* desenvolveu uma árvore hierárquica de características de qualidade de software para sistemas programados em Fortran. O modelo é hierárquico e os critérios de qualidade são organizados, a fim de identificar áreas a serem consideradas. Esta hierarquia é importante na medida em que são consideradas características em vários níveis. O modelo agrupa características com pontos em comum, que não poderiam ser unificadas em um único critério, devido à sua individualidade intrínseca. A partir daí, atribuem-se pesos para cada uma das características, de acordo com os objetivos de medição de qualidade da aplicação.

A partir do estudo e utilização destes modelos, e com o intuito de reformulá-los e melhorá-los, outras propostas, descritas a seguir, foram surgindo observando o problema da qualidade de software sob diferentes óticas. Estas propostas são o resultado das experiências em projetos desenvolvidos em universidades, centros de pesquisa e organizações preocupadas com a normalização na área de qualidade.

- ***QFD - Quality Function Deployment*** (Kogure e Akao 1983), (Brown 1991) Enfoca a qualidade observando as experiências realizadas na indústria tradicional de fabricação, sendo utilizado em grandes empresas como a *Ford* e a *General Motors* e na área de desenvolvimento de hardware, em organizações como a *IBM, Hewlett-Packard e DEC.* Recentemente está sendo utilizado para avaliar a qualidade de software. *QFD* permite que, ao longo do desenvolvimento, possa ser especificado o relacionamento entre os requisitos de qualidade e as características técnicas do produto. O método não apóia de maneira adequada a qualidade do processo.

- ***Goal - Question- Metric - GQM*** (Basili e Weiss 1984), (Basili e Rombach 1988), (Basili et al. 1995) Foi desenvolvido para ser utilizado na avaliação e melhoria do processo de desenvolvimento de software, dos produtos gerados e dos métodos utilizados. A proposta se preocupa em especificar um sistema de medição em três níveis: *conceitual (GOAL), opcional(QUESTION) e quantitativo (METRIC).* O *GQM* é uma abordagem baseada em objetivos, fornecendo flexibilidade, suportando definições de métricas em vários escopos e propósitos de medição. Permite especificar o planejamento das etapas, isto é, indica que ações têm que ser realizadas para definir e coletar métricas. No entanto, não apresenta guias detalhadas para o uso das medidas, de modo a estabelecer relacionamentos previsíveis para avaliar e melhorar o processo de desenvolvimento.

- ***COQUAMO - COnstructive QUAlity MOdel*** - (Kitchenhman 1987) Foi desenvolvido como parte do projeto *REQUEST* do programa *ESPRIT.* Utiliza uma abordagem do desenvolvimento da qualidade através do uso de um método de construção baseado em três facetas: métricas de garantia da qualidade, procedimentos e padrões. O modelo observa a qualidade de acordo com as cinco perspectivas propostas por Garvin (1984): *transcendente, baseada no produto, baseado no fabricante* e *baseada no valor.* O modelo tem por objetivo determinar a previsão da qualidade final do produto, monitorar o progresso no desenvolvimento do produto e analisar os resultados a fim de aperfeiçoar as previsões para o próximo projeto. Para atingir estes objetivos, o modelo está composto por um conjunto de ferramentas (*COQUAMO1, COQUAMO2* e

COQUAMO3) de previsão e medição que refletem a visão de qualidade do usuário, enquanto que a de monitoração reflete a visão do desenvolvedor.

- ***LOQUM - LOcally defined QUality Modelling*** - (Gillies 1992) Surge como resposta às deficiências encontradas no estudo dos modelos de avaliação da qualidade existentes, tais como Boehm et al. (1978), McCall et al. (1979) e *COQUAMO* (Kitchenhman 1987). Segundo este autor, estes modelos carecem de critérios para satisfazer as necessidades do usuário, métricas adequadas relacionadas às características propostas, uma visão geral de qualidade, critérios para facilitar a comunicação entre desenvolvedores e usuários e diretivas para a própria utilização dos modelos. Portanto, o modelo fornece um conjunto de ferramentas *(LOCRIT, LOCREL, LOCPRO)* e procedimentos para guiar o desenvolvedor na construção de seu próprio modelo, mais adequado às suas necessidades e à sua aplicação.

- ***CSA - Computing Service Association*** - (Möller e Paulish 1993) Tem por objetivo apresentar um enfoque prático que permita descrever características de qualidade de software para produtos e processos, através do processo de seleção e classificação de métricas. Apresenta três categorias de métricas (*tamanho, qualidade do produto, qualidade do processo*), que podem ser utilizadas ao longo do processo de desenvolvimento de software e que são independentes do ciclo de vida utilizado, de padrões e da metodologia de desenvolvimento utilizada.

- ***ISO/IEC*** - (ISO/IEC 9126 1991) Propõe um modelo que está sendo considerado como uma norma, que permite a avaliação objetiva e quantitativa de produtos de software. Define um conjunto de características e subcaracterísticas de qualidade que descrevem a qualidade do produto final. Estes atributos de qualidade não permitem uma medição direta. Assim sendo, estão sendo definidas métricas para correlacionar estas características ao produto de software propriamente dito. As diretrizes apresentadas pela *ISO 9126* estão organizadas de acordo com as áreas: *utilização, visões de qualidade* e *modelo do processo de avaliação.* O modelo carece de fundamentos para determinar que fatores de qualidade devem ser considerados na definição da qualidade, e de formas de decidir que critérios se relacionam com determinado fator. Além disso, não apresenta como as métricas são tratadas para avaliar as características de qualidade de nível mais alto.

- ***Dromey*** (1995; 1996) Segundo este autor, qualidade é um conceito utilizado de forma indiscriminada, ora com referência ao processo, ora ao produto. Além disso, é dada muita atenção aos atributos de qualidade de alto nível e quase

nenhuma às propriedades concretas do produto que podem influenciar aqueles atributos. Portanto, propõe que um modelo de qualidade de software deve identificar os atributos internos do produto (mensuráveis) que tenham maior impacto sobre os atributos externos (ou de alto nível). Seu modelo fornece uma estrutura que particulariza especificações, projeto e implementação. A proposta tenta resolver alguns dos problemas encontrados nos modelos hierárquicos. O enfoque permite verificar o modelo, ou seja, fornece critérios para a inclusão de propriedades particulares que o modelo deve possuir, e sugere uma forma de verificar quando o modelo está completo, através da avaliação de categorias de propriedades, tais como: *correção, propriedades internas, propriedades contextuais* e *propriedades descritivas.*

II.6.1. Modelo de Rocha

No contexto deste trabalho consideramos, para definição e organização das características de qualidade discutidas na seção V.1 desta tese, o modelo proposto por Rocha (1987), que descrevemos a seguir (Figura II.1).

- ***Objetivos de qualidade***: são propriedades gerais que o produto deve possuir;
- ***Fatores de qualidade do produto***: determinam a qualidade do produto sob o ponto de vista dos diferentes usuários (usuário final, mantenedores, etc.);
- ***Critérios:*** são atributos primitivos passíveis de serem avaliados;
- ***Processos de avaliação***: determinam o processo e os instrumentos a serem utilizados para se medir o grau de presença, no produto, de um determinado critério;
- ***Medidas***: indicam o grau de presença, no produto, de um determinado critério;
- ***Medidas agregadas***: são o resultado da agregação das medidas obtidas na avaliação, segundo os critérios que quantificam os fatores.

Os objetivos de qualidade são atingidos através dos fatores de qualidade, que podem ser compostos por outros fatores e são avaliados através de critérios. Os critérios definem atributos de qualidade para os fatores. Medidas são valores resultantes da avaliação de um produto segundo um critério específico. Objetivos e fatores não são diretamente mensuráveis e só podem ser avaliados através de critérios. Um critério é um atributo primitivo, isto é, um atributo independente de todos os outros atributos. Nenhum critério isolado é uma descrição completa de um determinado fator ou sub-fator. Da mesma forma, nenhum fator define completamente um objetivo.

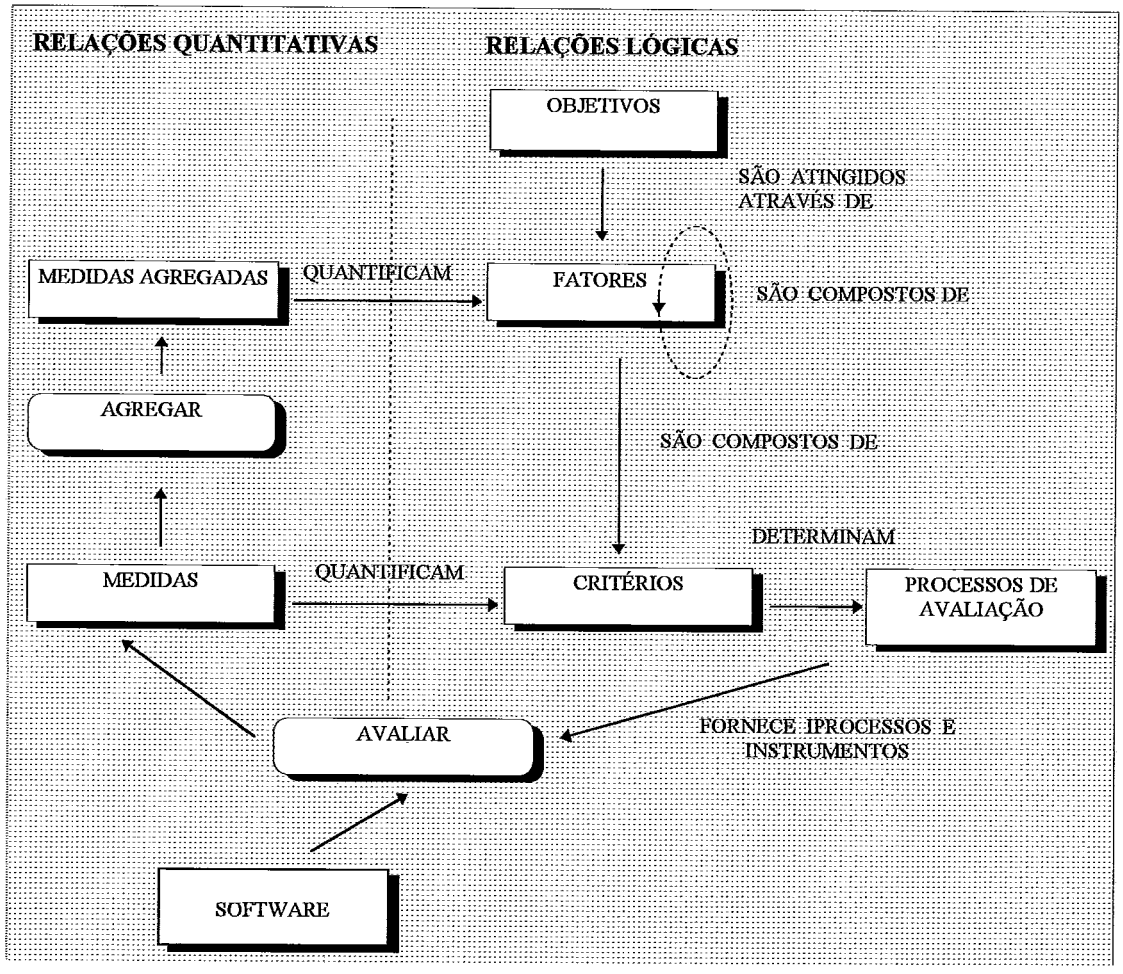


Figura II.1. - Modelo de Avaliação da Qualidade de Software

A utilização deste modelo deve-se ao fato de esta tese estar inserida no contexto das pesquisas que estão sendo desenvolvidas na *COPPE/UFRJ* na área de *Qualidade de Software*. Entre seus objetivos estão a definição de atributos de qualidade e processos de avaliação para diferentes domínios de aplicação e paradigmas de desenvolvimento.

O modelo foi particularizado e aplicado para definir objetivos, fatores, sub-fatores e critérios de qualidade para diferentes produtos gerados ao longo do desenvolvimento, tais como: *especificação de requisitos* Clunie (1987), *especificações de projeto* Passos (1991), *código* Comerlato (1994); e para vários domínios de aplicação específicos, como são: *software científico* Bahia (1992), Palermo e Rocha (1993), *financeiro* Belchior (1992), *educacional* (Campos G. 1994), *multimídia* (Campos F. 1994), *sistemas especialistas* Oliveira (1995), *sistemas de informação* Blaschek (1995), *software musical* Braga et al. (1995) e *software médico* Oliveira et al. (1995).

II.7. Experiências

A importância dada, atualmente, à qualidade do produto e do processo de desenvolvimento é evidenciada pela proposição de vários modelos de qualidade encontrados na literatura e pelos esforços realizados nas universidades, centros de pesquisa e instituições dedicadas à padronização. Além das iniciativas com o Modelo de Maturidade de Capacidade para Software - *CMM* e as normas *ISO 9000* (apresentadas na seção item II.2.), apresentamos uma recompilação de alguns outros projetos no âmbito internacional, visando à investigação aplicada e à divulgação de métricas. Dentre deles, destacam-se as experiências no plano europeu do programa *ESPRIT* (Abreu 1992), (Oivo e Basili 1992), (Müllerburg 1993), (Fenton 1991), (Möller e Paulish 1993), (Ashley 1993a), (Ashley 1993b), (Hausen e Welzel 1993), (Dorling 1993), (Murine e Murine 1995). São elas:

- ***MUSE - "Software Quality and Reliability Metrics for Selected Domains"***,cujo objetivo é propor métricas de qualidade e confiabilidade, demonstráveis no âmbito de vários domínios de aplicação selecionados. O consórcio, liderado pela Brameur Ltd (Reino Unido), inclui empresas da França, Alemanha e Grécia.

- ***METKIT - "Metrics Education ToolKIT"***, cujo objetivo é aumentar a divulgação e fomentar a utilização de métricas de software na indústria européia, através da produção de material educacional dirigido à comunidade industrial e acadêmica. O consórcio inclui instituições do Reino Unido, Alemanha, Dinamarca, França é Itália e liderado pela Brameur Ltd (Reino Unido).

- ***PYRAMID - "Promotion of Metrics"***, cujo objetivo é ajudar a melhorar a qualidade e produtividade nas atividades de desenvolvimento e manutenção de software na Europa, através da utilização de métodos quantitativos. Os seus promotores esperam que os resultados concorram para uma utilização crescente de métricas de software, reduzindo o atraso com relação aos EUA e Japão. O *"primer contractor"* do projeto PYRAMID é a Veridatas (França), reunindo o consórcio outras instituições do Reino Unido, Alemanha, Grécia, Itália e França.

- ***MUSIC - "Metrics for Usability Standards In Computing"*** cujo objetivo é o desenvolvimento de uma metodologia modular que possa ser utilizada para especificar os requisitos mínimos para a utilização de um produto e para certificar se este produto cumpre com os requisitos especificados. Nesse sentido, estão sendo desenvolvidos um modelo de referência, métricas, procedimentos e um curso de formação. Os participantes pretendem, também, disseminar os conceitos e o método *MUSIC* em comitês de normalização e na comunidade industrial

européia. O consórcio é formado por universidades e outras instituições do Reino Unido, Itália, Espanha, Alemanha, Irlanda e Holanda, sendo liderado pela Brameur Ltd (Reino Unido).

- *AMI - "Application of Metrics in Industry"*, cujo objetivo é demonstrar a maturidade da aplicabilidade da tecnologia de métricas à comunidade da indústria de software. Como produto principal, os promotores indicam o *Metrics User's Handbook*, que é um guia para a progressiva adaptação, instalação e aplicação do modelo de métricas proposto pelo *AMI*, o qual esperam venha a tornar-se um padrão *de facto* no seio da comunidade. O consórcio, coordenado pela *GEC Marconi Ltd* (Reino Unido), integra várias instituições da França, Áustria, Alemanha, Espanha, Itália e Reino Unido.

- ***SCOPE - "Software Certification in Europe"***, cujo objetivo é desenvolver um guia explícito para a realização de procedimentos de certificação para produtos de software na comunidade européia. Um aspecto importante dos procedimentos propostos é a avaliação de vários ciclos de vida, utilizando várias classes de métricas, tais como: tamanho, estrutura de fluxo de controle, modularidade e fluxo de informação, estrutura de dados, eficiência e complexidade de algoritmos e medidas gerais de complexidade.

- ***BOOTSTRAP*** - Projeto conduzido como parte do *Projeto ESPRIT*, cujo objetivo é desenvolver um método para avaliação, medição quantitativa e melhoria do processo de software. O método avalia tanto unidades produtoras de software (*UPS*) como seus projetos. Determina o nível de maturidade de uma organização, identifica méritos e deficiências (capacidade) e fornece guias para aperfeiçoamento (planos de ação).

- ***SPICE - "Software Process Improvement and Capability Determination"***, cujo objetivo é a geração de um padrão internacional para avaliação de processo de software, visando à melhoria contínua do processo e à determinação de sua capacitação. Baseia-se nas melhores características dos modelos de avaliação de processo existentes, como: *CMM, TRILLIUM, Software Techonology Diagnostic-STD e Bootstrap*. O *SPICE* pode ser utilizado por organizações envolvidas em planejar, monitorar, controlar e melhorar a aquisição, fornecimento, desenvolvimento, operação, evolução e suporte de software.

- ***TAME*** - Tem por objetivo definir um processo iterativo baseado em medições, para o desenvolvimento e manutenção de software. Para isto, o projeto *TAME* é baseado em dois paradigmas: o primeiro está relacionado com a melhoria da

qualidade evolutiva adaptada ao desenvolvimento de software, chamado de Paradigma de Melhoria da Qualidade (*Quality Improvement Paradigm - QIP*); e o segundo é o paradigma para o estabelecimento de metas corporativas e de projeto e para o mecanismo para medição chamado *GQM* (Goal/ Question/Metric).

Cada vez mais, torna-se imprescindível o uso de uma abordagem consistente e sistemática para a melhoria da qualidade de software. Cada projeto tenta contribuir de alguma forma para este objetivo. Quanto a experiências bem sucedidas, existem variadas referências, por todo o mundo (Möller e Paulish 1993), como, por exemplo nas seguintes organizações: *Siemens Nixdorf* (Alemanha), *Data Logic* (Reino Unido), *ETNOTEAM* (Italia) e *Veridata* (França). Fora do plano europeu, o Modelo CMM e a *ISO 9000* tem sido bastante utilizado na prática, como é possível constatar pelo elevado número de artigos relatando experiências, tais como (Kitson e Masters 1993), (Johnson 1994) e (Winston 1993).

II.8. Conclusão

Apesar de cunhada no final da década de 60, a Engenharia de Software ainda não atingiu a sua maturidade. Se comparada com as engenharias tradicionais, todas apoiadas em ciências maduras, com leis amplamente comprovadas através de experimentação controlada, a Engenharia de Software possui características incipientes.

O processo de desenvolvimento de software e o próprio software ainda carecem de definições consensuais no que se relaciona com seus parâmetros mais básicos, o que torna extremamente difícil fazer avaliações comparativas da qualidade do produto final, bem como de custos e prazos de execução das diferentes etapas do processo de desenvolvimento. Assim, cada projeto de software tende a constituir uma experiência única, que não garante o sucesso de projetos subseqüentes.

Hetzel (1995) aponta algumas questões visando minimizar os problemas decorrentes da prática da engenharia de software e especificamente relacionados à avaliação da qualidade. Propõe um programa de qualidade para a avaliação de trabalhos práticos, consistindo de três componentes chave. O primeiro componente é uma base de dados que serve como referência e que contém estudos e resultados de trabalhos em avaliação da qualidade. Um segundo componente deve conter critérios objetivos que indicam padrões de qualidade que ajudam a melhoria da visibilidade e validade de avaliações que estão sendo realizadas. O terceiro componente consiste de um programa de revisões independentes que será aderido ao componente de padrões de qualidade.

Qualidade, inegavelmente, é uma característica fundamental tanto no processo de desenvolvimento como no seu produto final. A medida desta qualidade é a chave vital para a excelência no software.

Este capítulo abordou o tema da qualidade de software. Inicialmente buscou-se definir o termo qualidade. Mostrou-se a problemática da medição em software, ressaltando a dificuldade de se obterem medidas para o software, por este ser um produto lógico e não físico. Foram comentados os modelos mais relevantes de avaliação da qualidade de software, assim como algumas experiências no âmbito internacional visando à investigação aplicada e à divulgação de métricas.

A avaliação da qualidade de software não deve ser vista como um procedimento especial e intangível, mas sim como um processo que deve ser definido caso a caso, considerando as especificidades da aplicação, da tecnologia a ser adotada na construção do produto, da organização, da equipe de desenvolvimento e dos futuros usuários do produto (Rocha et al. 1993).

O embasamento científico da *Engenharia de Software* só terá sido obtido quando essas duas realidades, métricas e experimentação, permearem todos os níveis do processo de desenvolvimento de software.

Capítulo III

Desenvolvimento de Software Orientado a Objetos

"Não há dúvida que o desenvolvimento de software orientado a objetos é fundamentalmente diferente do enfoque tradicional. Este requer formas distintas de pensamento para a decomposição, produzindo uma arquitetura de software que se afasta da cultura do enfoque estruturado"

Edward Yourdon

III.1. Introdução

Os enfoques tradicionais de desenvolvimento de software têm sido discutidos e aplicados por desenvolvedores de software há mais de uma década, algumas vezes com resultados bem sucedidos e outras vezes com falhas. O enfoque de orientação a objetos surge como uma alternativa de solução de alguns problemas encontrados no desenvolvimento tradicional. Várias vantagens têm sido apontadas como conseqüência da utilização deste enfoque. O fato de poder modelar o problema em termos de objetos torna possível diminuir o *gap semântico* entre o problema no mundo real e sua abstração. Além disso, é vantajoso desenvolver software sem isolar os dados de seus procedimentos, pois estes tendem a facilitar significativamente as atividades de avaliação, reutilização, testes e manutenção (Bordoloi e Lee 1994).

Neste capítulo introduz-se o tema do desenvolvimento de software orientado a objetos, iniciando com uma discussão sobre os princípios e conceitos básicos que caracterizam uma aplicação desenvolvida utilizando este paradigma. É apresentado o conceito de modelo de sistemas, particularizando para o caso do paradigma de orientação a objetos. São, ainda, definidos os requisitos desejáveis para métodos de desenvolvimento orientado a objetos. Em seguida são descritos, de forma sucinta, os métodos de desenvolvimento orientado a objetos mais representativos. Apresentamos algumas considerações sobre qualidade de software e os métodos de desenvolvimento

orientado a objetos para então finalizar com nossas conclusões e comentários sobre o tema tratado.

III.2. Princípios e Conceitos Chave em Orientação a Objetos

A engenharia de software tem como objetivo principal a construção de produtos de alta qualidade. Isto quer dizer que o software produto deve atender a um conjunto de características. Por isso, para atingir o nível de qualidade desejado, o software deve ser construído obedecendo aos *princípios* e *conceitos* do paradigma de desenvolvimento utilizado. De acordo com a definição dos termos *princípios* e *conceito*, segundo o Webster's New International Dictionary (Webster's 1977),

- Princípios: "*Proposições diretoras ou leis que regem um paradigma às quais todo o desenvolvimento posterior desse paradigma deve estar subordinado*".
- Conceito: "*Representação ou descrição de um objeto (paradigma) por meio de suas características gerais*",

e particularizando para o caso do desenvolvimento de software orientado a objetos, é necessário conhecer e aplicar seus *princípios* e *conceitos* para atingir o objetivo da engenharia de software.

A literatura especializada sobre orientação a objetos apresenta uma diversidade de terminologias, mas não estabelece de modo geral um consenso em relação aos princípios a serem obedecidos e aos conceitos a serem utilizados em um desenvolvimento considerado orientado a objetos. Da mesma forma como são colocados de modo geral (i.e., princípios chave na engenharia de software), é apropriado particularizar para o caso do paradigma de orientação a objetos, seus *princípios* e *conceitos* chave que devem estar presentes em qualquer desenvolvimento orientado a objetos. Nesta seção abordamos estes *princípios* e *conceitos*, baseados em discussões sobre o assunto encontradas em Meyer (1988), Robson (1990), Ingalls (1990), MacLennan (1990), Rine (1991), Martin e Odell (1992), Shlaer e Mellor (1992), Coad e Yourdon (1992), Snyder (1993), Winblad et al. (1993), Rumbaugh et al. (1994), Sullo (1994), Yourdon (1994), Martin (1994) e Booch (1994).

III.2.1. Princípios

Os princípios que serão discutidos são os seguintes: *abstração, encapsulamento, modularidade, decomposição hierárquica e comunicação via mensagem.*

- **Abstração**

Abstração é o princípio de ignorar os aspectos de um assunto não relevantes para o propósito em questão, tornando possível uma concentração maior nos assuntos principais (Coad e Yourdon 1992). Por exemplo, no desenvolvimento de um sistema orientado a objetos, durante a fase de análise, o analista deve concentrar-se no que um objeto é e o que ele faz, antes de decidir como ele deve ser implementado. O uso da abstração preserva a liberdade de se tomar decisões evitando, tanto quanto possível, um comprometimento prematuro com detalhes (Rumbaugh et al. 1994).

Segundo Dijkstra, em Booch (1994), *"abstrações surgem como uma maneira de reconhecer as similaridades entre objetos, situações ou processos no mundo real, e a decisão de concentrar-se nas similaridades e ignorar por um momento a existência das diferenças"*. Por outro lado, Shaw em Booch(1994) define abstração como *"uma descrição simplificada, ou especificação de um sistema, que enfatiza alguns de seus detalhes ou propriedades enquanto omite outros"*. Ou seja, uma boa abstração é aquela que dá ênfase aos detalhes que são de interesse ao leitor ou usuário e suprime detalhes, pelo menos num certo momento, considerados irrelevantes.

Combinando diferentes visões, Booch (1994) define abstração da seguinte maneira:

"Uma abstração denota as características essenciais de um objeto que o diferencia dos outros objetos e que mostra seus limites conceituais, relativos à perspectiva do observador."

Este autor apresenta uma classificação de tipos de abstração, com o intuito de orientar o desenvolvedor sobre onde concentrar sua atenção durante o processo de abstração. Estes são: abstração de entidade, abstração de ação, abstração da máquina virtual e abstração por coincidência.

Por outro lado, em Firesmith (1993), observamos uma outra classificação de tipos de abstrações, dependendo do que está sendo modelado: abstrações de objeto, classes, dados, funções e processos.

Segundo Snyder (1993), todos os objetos representam uma abstração que tem significado para o usuário. Embora um objeto tenha dados associados a ele, um objeto não é um dado, pois o objetivo dos dados é representar informações associadas com a abstração.

Abstração também é utilizada para decompor (*particionar*) um sistema complexo em suas partes componentes e para compor um sistema complexo fora de suas partes componentes. Tanto a decomposição como a composição geram arquiteturas (de requisitos, projeto e código) que apresentam uma hierarquia através de *níveis de abstração*. Ou seja, as abstrações aparecem para cada nível de sistema. Um alto nível de abstração é implementado em termos de níveis inferiores (Firesmith 1993).

Berzin, Gray e Naumann, em Booch (1994), recomendam que, para compreender um sistema orientado a objetos e minimizar erros, cada abstração deve ser entendida como um todo, sem a necessidade de observar seus níveis inferiores de abstração, independentemente do mecanismo eventualmente utilizado para realizar isto.

- **Encapsulamento**

Uma vez abstraídos as classes e objetos do problema, tanto ao nível de atributos quanto ao de operações, há a necessidade de escondê-los na sua forma em uma mesma entidade (objeto). À propriedade de se implementarem atributos e operações correlacionados em uma mesma entidade dá-se o nome de *encapsulamento*.

O encapsulamento consiste na separação entre os aspectos externos de um objeto, acessíveis por outros objetos, e os detalhes internos da implementação daquele objeto, que ficam ocultos dos demais objetos (Coad e Yourdon 1992), (Rumbaugh 1994), (Sullo 1994), (Booch 1994). A implementação de um objeto pode ser modificada sem que isso afete as aplicações que o utilizam. Desta forma, é garantida a integridade dos dados no objeto (Snyder 1993). A idéia por trás do encapsulamento é que a utilização de um sistema orientado a objetos não deve depender de sua implementação interna, e sim de sua interface. Isto ajuda o tratamento da complexidade através da ocultação da informações das abstrações modeladas. Por outro lado, o encapsulamento satisfaz alguns objetivos fundamentais da engenharia de software (*modularidade, localizabilidade e ocultamento da informação*), características fundamentais para o desenvolvimento de um sistema confiável, robusto e manutenível (Firesmith 1993).

Particularizando para o processo de modelagem de requisitos, o encapsulamento diminui o trabalho no desenvolvimento de um novo sistema, e esta é sua maior vantagem. Desta forma, é possível reunir os aspectos mais instáveis no modelo, facilitando a alteração dos requisitos. Localizar a instabilidade é essencial, pois vivemos em um ambiente sujeito a contínuas mudanças. O encapsulamento permite agrupar os aspectos relacionados no modelo, minimiza o fluxo entre os objetos no modelo e separa requisitos específicos que outras partes da especificação podem usar (Coad e Yourdon 1992).

- **Modularidade**

O mundo real está composto por sistemas, cada um dos quais contendo um conjunto de entidades que interagem entre si para atingir um objetivo específico. O princípio de modularidade, de modo geral, é entendido como a propriedade de um sistema estar organizado em um conjunto de entidades (módulos) com alta coesão (abstrações logicamente relacionadas) e baixo acoplamento (minimização das dependências entre elas) (Meyer 1988).

Um modelo de objetos pode estar formado por centenas de classes e objetos. Por isso, uma das características de um sistema orientado a objetos é a sua capacidade de agrupamento de classes e objetos em módulos[1], onde as classes fortemente acopladas são agrupadas incorporando um subconjunto lógico do modelo completo. Cada um destes módulos apresentam uma independência de funcionamento. Isto significa que podem ser facilmente entendidos, construídos, testados, corrigidos e estendidos.

Sistemas complexos serão formados por conjunto de módulos independentes, a cada um dos quais está associada uma abstração. Pois, para aplicações na qual existem centenas de classes, o uso de módulos é essencial para o tratamento da complexidade (Booch 1994).

- **Decomposição Hierárquica**

O encapsulamento ajuda no tratamento da complexidade através da ocultação da informação entre as abstrações. Por outro lado, a modularidade oferece uma forma de agrupar logicamente um conjunto de abstrações que estão relacionadas. Isto, contudo, não é suficiente. Algumas das abstrações modeladas são decompostas em *hierarquias*. Identificar estas hierarquias ajuda, significativamente, no entendimento do problema.

De modo geral, são identificadas duas importantes hierarquias em um sistema orientado a objetos, estrutura de classe (hierarquia *"é um"* - herança; hierarquia *"parte de"* - agregação). Além disso, propostas mais recentes discutem *hierarquia de clusters* (Firesmith 1994).

Os princípios de *abstração, encapsulamento, modularidade* e *decomposição hierárquica* apresentam um comportamento sinérgico em relação ao *tratamento da complexidade.*

[1]Os métodos de desenvolvimento orientados a objetos referem-se a este conceito de formas diversas: *Assembly* (Firesmith 1993), Categorias de Classes (Booch 1994), *Cluster* (Eifel, Firesmith 1993, Lorenz 1994, Meyer 1988, Shaler/Mellor 1992), Dominio (Shlaer/Mellor 1992), Módulo (Booch 1994, Rumbaugh 1993), Assunto (Coad/Yourdon 1992), Subsistema (Booch 1994, Jacobson 1992, Lorenz e Kidd 1994, Shaler/Mellor 1992, Wirfs-Brock 1990).

- **Comunicação via mensagem**

A mensagem é o mecanismo de comunicação entre entidades (objetos), através do qual se desencadeia a execução de uma operação específica. A solicitação indica o objeto destinatário, o seletor da operação e, opcionalmente, um conjunto de informações adicionais. O objeto receptor determina a operação a ser executada, que por sua vez responde ao objeto solicitante, podendo também enviar mensagens a outros objetos. O objeto solicitante não tem conhecimento de onde e de que maneira o objeto receptor realiza a operação. Assim como as *caixas pretas de engenharia*, a estrutura interna de um objeto é escondida (através do encapsulamento) e *as mensagens que o objeto recebe* são os únicos condutos que conectam o objeto ao mundo exterior. Um princípio para a administração da complexidade do sistema (notadamente para interfaces) é a comunicação via mensagens (Coad e Yourdon 1992).

III.2.2. Conceitos

Os conceitos chave que serão discutidos são os seguintes: *objeto, classe, atributo, herança, relacionamentos, operação, interface e polimorfismo.*

- **Objeto**

Um enfoque tradicional de desenvolvimento de software consiste em procedimentos e dados. Um desenvolvimento orientado a objetos consiste somente em objetos, que encapsulam dados e procedimentos. Em outras palavras, objetos são entidades que contêm dados e operações que manipulam estes dados (Robson 1990), (Winblad et al. 1993), (Sullo 1994), (Booch 1994), (Rumbaugh et al. 1994), (Snyder 1993), (Martin 1994). Assim sendo, um objeto é simplesmente alguma entidade que faz sentido no contexto de uma aplicação, com limites nítidos e significado em relação ao problema em questão. Todos os objetos têm estados, comportamento e identidade e são distinguíveis, servindo a dois objetivos: facilitam a compreensão do mundo real e oferecem uma base para a implementação em computador (Rumbaugh 1994), (Booch 1994), (Yourdon 1994), (Snyder 1993).

Um objeto é, portanto, uma entidade independente que possui um estado interno e um comportamento. O estado interno consiste de uma memória interna onde se pode armazenar e modificar informação ao longo da vida do objeto. O comportamento é caracterizado por um conjunto de ações através das quais o objeto responde à demanda de processamento por parte de outros objetos e das ações que requer de outros objetos. O estado interno é expresso através de atributos específicos (dados). O comportamento é expresso através de um conjunto de operações. Isto é, um objeto pode ser instanciado, modificado, mudar de estado ou ser compartilhado (MacLennan 1990). Objeto é a entidade principal numa aplicação orientada a objetos e consiste, portanto, de um

conjunto de atributos, mensagens e operações. Um objeto pode ser composto de outros objetos. Esses objetos, por sua vez, podem ser compostos de outros objetos. Esta estrutura dá margem para que sejam definidos objetos bastante complexos.

- **Classe**

Objetos podem ser agrupados em coleções de objetos similares, denominadas de classes. Nas classes são descritas as estruturas de dados e o comportamento de um ou mais objetos, quase idênticos, que têm seus dados estruturados da mesma forma e são manipulados pelas mesmas operações. Cada objeto é uma instância da classe. Os objetos representados por uma determinada classe diferenciam-se entre si pelo seu estado, isto é, pelos valores de seus dados. A habilidade em abstrair descrições de operações e dados comuns a um conjunto de objetos e armazená-los em uma classe é a centralização da capacidade da orientação a objetos (Winblad et al. 1993). Definir classes significa posicionar uma codificação *reutilizável* em um depósito comum em vez de expressá-la várias vezes.

Em um sistema uniformemente orientado a objetos, uma classe é também um objeto, denominado de *template*, que possui atributos e operações, pertencentes a ela mesma e não a suas instâncias. Desta forma, pode receber mensagens que ativam suas operações (Firesmith 1993). A diferença fundamental entre classes e objetos está no fato de um objeto constituir uma entidade concreta com tempo e espaço de existência, enquanto a classe é tão somente uma abstração (Booch 1994).

- **Atributo**

Um atributo é considerado um dado discreto, que representa uma característica inerente, propriedade, quantidade ou qualidade de um objeto ou de uma classe e tem seu próprio valor (constante ou variável). Atributos são utilizados para identificar, descrever, prover o estado ou servir como uma referência explícita a outro objeto ou classe.

- **Herança**

A herança é uma hierarquia do tipo *"é um"* e representa um mecanismo de compartilhar, de modo automático, atributos e operações entre classes e objetos. Ou seja, é um mecanismo que permite expressar as similaridades entre classes, simplificando a definição de classes iguais a outras que já foram definidas. A herança permite criar novas classes, especificando somente a diferença entre elas e a classe ancestral, tornando explícitos os atributos e operações comuns em uma hierarquia de classes.

A herança pode ser simples ou múltipla. Como herança simples, temos que uma subclasse pode herdar atributos e operações de uma única classe ancestral, bem como acrescentar ou redefinir comportamentos para si própria. Por outro lado, a herança múltipla permite a uma subclasse herdar atributos e operações de várias classes

ancestrais e, como na herança simples, adicionar operações e/ou atributos, assim como redefinir serviços.

Segundo Firesmith (1993), a herança é um aspecto crítico em qualquer desenvolvimento orientado a objetos, pois fornece benefícios significativos. Através da fatoração de aspectos comuns, as classes são simplificadas e suas diferenças são claramente localizadas e enfatizadas. As declarações de aspectos comuns podem ser reutilizadas, não só pelas subclasses que pertencem à hierarquia, mas em outras aplicações que precisam das mesmas declarações. A herança aumenta a reutilização, bem como minimiza a especificação e a documentação do sistema. A *herança* também apresenta alguns riscos e limitações que não devem ser ignorados. A herança limita os princípios de engenharia de software relacionados à localizabilidade e encapsulamento. Ao invés de localizar toda a declaração da classe na própria classe e encapsular todos os seus dados, estas declarações aparecem dispersas ao longo de toda a hierarquia de herança. Desta forma, a compreensibilidade é afetada porque todas as características da classe não podem ser coletadas em uma mesma localização. Além disso, a atividade de testes das classes é afetada (Queiroz e Ribeiro 1994), (Travassos et al. 1995) e o problema do *Iolô* (Taenzer em Queiroz e Ribeiro 1995), produto do tratamento das mensagens numa hierarquia, deve ser considerado. No caso de *herança múltipla* estes problemas tendem a se agravar.

- **Relacionamentos**

Objetos e classes não são totalmente independentes, pois eles estão relacionados de várias maneiras. Numa hierarquia de abstrações, objetos e classes interagem e dependem uns dos outros. Grupos de objetos colaboram para implementar alguma capacidade do sistema. Objetos normalmente dependem de outros objetos para realizar seus serviços (via mensagens) ou possivelmente para obter algum dado (atributo). O relacionamento existe entre duas entidades (objetos, classes) que de alguma forma estão associadas.

Rumbaugh et al. (1994) estabelece uma diferença entre *ligação e associação* para denotar os relacionamentos. A primeira corresponde a uma conexão física ou conceitual entre instâncias de objetos. Por outro lado, associação descreve um conjunto de potenciais ligações com estrutura e semântica comuns, da mesma maneira que uma classe descreve um conjunto de potenciais objetos.

Segundo Coad e Yourdon (1992), os relacionamentos formam um modelo de mapeamento (s) de domínio de problemas que um objeto precisa ter com outros objetos, para cumprir suas responsabilidades.

- **Operação**

Uma operação *(método, serviço)* é uma atividade discreta, ação ou comportamento executado por um objeto ou por uma classe (Firesmith 1993). Elas determinam como o objeto atuará quando receber uma mensagem. Trabalham em resposta às mensagens e manipulam os valores de seus dados. De fato, as operações detêm o único mecanismo para alteração dos valores de seus dados. Além disso, também podem enviar mensagens a outros objetos, requisitando ação ou informação.

- **Interface**

A interface descreve as formas em que um objeto pode ser usado, isto é, descreve um conjunto potencial de mensagens que identifica o objeto e seus parâmetros. Um objeto satisfaz uma interface se ela tem um significado com relação a cada uso potencial descrito. O conjunto de mensagens que um objeto pode responder é, também, chamado de *protocolo*.

- **Polimorfismo**

Os objetos agem e produzem diferentes comportamentos observáveis em resposta às mensagens que recebem. Uma mesma mensagem pode resultar em ações completamente diferentes quando recebida por objetos diferentes. Com o polimorfismo, um usuário pode enviar uma mensagem genérica e deixar os detalhes de implementação para o objeto receptor. Polimorfismo significa *"o que possui várias formas"*. Isto é, em orientação a objetos, uma mesma mensagem poderia ser enviada a diferentes objetos, pertencentes a diferentes classes, e estes implementariam essa solicitação de formas ligeramente diferentes. Ou seja, o polimorfismo é a habilidade de dois ou mais objetos responderem à mesma mensagem, cada um a seu modo (Winblad et al. 1993), (Firesmith 1993), (Booch 1994), (Sullo 1994), (Martin 1994), (Rumbaugh et al. 1994). Particularizando para análise orientada a objetos, o polimorfismo é utilizado para indicar a propriedade de se usar o mesmo nome para serviços implementados em diferentes níveis de uma hierarquia de classes. Neste caso, para cada classe tem-se um comportamento específico para o serviço. O polimorfismo é um dos responsáveis pela extensibilidade em um sistema orientado a objetos (Firesmith 1993), (Martin 1994) (Booch 1994).

III.3. Modelo Orientado a Objetos

Um modelo é uma abstração de um sistema *real*, isto é, resulta de um processo intelectual de seleção de certas características do sistema, consideradas relevantes para explicar um dado comportamento do mesmo (McMenamin e Palmer 1984). O fato de muitas características concretas do sistema serem desconsideradas no modelo, como resultado consciente do processo de abstração, não é um defeito, mas uma qualidade

necessária para que o modelo seja um instrumento eficaz ao estudo e compreensão do sistema. Assim sendo, um *modelo* pode ser definido como (DeMarco 1989):

"Uma representação abstrata que permite descrever e/ou prever comportamentos específicos de uma realidade complexa, através de características específicas do sistema que ele quer representar."

Modelos são ferramentas extremamente úteis durante o processo de desenvolvimento, servindo para diversos propósitos (DeMarco 1989), (Rumbaugh et al. 1994), tais como: *comunicação entre os participantes do desenvolvimento, visualização de um problema ou entidade, auxílio no tratamento da complexidade,* indo da visão mais abstrata de um sistema até o seu detalhamento, e *auxilio na divisão do projeto em tarefas com produtos bem definidos* com uma dependência mínima entre elas.

Pode-se, portanto, dizer que a modelagem de sistemas é um instrumento essencial para a apreensão intelectual de uma realidade intrinsecamente complexa. Um modelo constitui a forma mais efetiva para documentar, comunicar e analisar sistemas (Spriet e Vanstenkiste 1982).

Modelos são considerados ferramentas críticas em todas as disciplinas de engenharia, pois nos permitem representar o domínio da aplicação. Em orientação a objetos, esta representação se dá através da construção de um modelo de objetos composto por um conjunto de *classes e objetos relevantes* ao domínio em questão. O uso da *abstração* como ferramenta mental para o tratamento da complexidade é fundamental no entendimento das características e comportamento destas classes e objetos, assim como a aplicação dos princípios de *decomposição hierárquica, encapsulamento, modularidade e comunicação via mensagens.*

Firesmith (1993) , Rumbaugh et al. (1994) e Booch (1994) consideram que os modelos de objetos são abstrações que incluem todas as capacidades essenciais, propriedades ou aspectos que estão sendo modelados sem detalhes irrelevantes. Eles são construídos para que um problema seja compreendido antes da implementação de uma solução e representam um subconjunto da realidade, selecionado para um determinado objetivo. Assim sendo, classes e objetos representam abstrações do domínio da aplicação, encapsulando seus atributos essenciais (*características*) e operações (*comportamentos*).

A arquitetura básica de um modelo de objetos requer a especificação de:

- *uma linguagem gráfica de representação rigorosa*, com sintaxe e semântica definidas e suficientemente ricas para comunicar a complexidade requerida pelo idealizador do modelo, bem como formal o bastante para evitar ambigüidades nesse processo de comunicação. Deve ser um facilitador da comunicação, possuir apoio automatizado e não requerer demasiado esforço para o seu aprendizado;
- *critérios para realizar a partição e organização de um sistema de grande porte e a abstração de detalhes excessivos de um sistema intrinsecamente complexo;*
- *todas as entidades (objetos)* relevantes ao domínio do problema;
- *todos os seus atributos e operações*;
- *todos os relacionamentos (interações)* existentes entre as entidades do modelo, relevantes para sua construção; e os
- *comportamentos* do sistema, adequados ao nível de abstração no qual o modelo está construído.

Um bom modelo orientado a objetos deve ainda ser construído atendendo aos seguintes princípios e objetivos da engenharia de software (Firesmith 1993):

- *abstração*, para representar todas as características essenciais e de comportamento das entidades modeladas, através de um adequado conjunto de classes;
- *completitude*, para garantir que todas as características essenciais e de comportamento das entidades estão sendo modeladas. Isto pode implicar no uso de diferentes tipos de modelos;
- *validabilidade*, para garantir que o modelo que resolve o problema está sendo construído corretamente;
- *independência*, garantindo que a dependência entre objetos e classes seja mínima;
- *uniformidade de detalhe*, para garantir que o modelo apresente o nível de detalhe adequado para seu entendimento por parte do usuário;
- *modularidade*, garantindo que o modelo não seja muito grande e complexo; e
- *consistência*, garantindo que o modelo não apresente aspectos conflitantes.

Os diversos métodos de desenvolvimento orientado a objetos (descritos na seção III.5.), produzem *modelos de objetos*, mas por caminhos diferentes, utilizando terminologia, notações e simbologia distintas. Esses modelos se caracterizam por minimizar o *gap semântico* entre os problemas do mundo real e as soluções do mundo computacional, mapeando de maneira bastante natural o problema do mundo real num modelo computacional de objetos. O modelo resultante consiste numa organização e

uma notação gráfica apropriada que descreve e define a representação e conceituação do domínio da aplicação; focaliza o mundo real sob estudo; identifica, classifica e sumariza o que está no problema, organizando as informações em estruturas de objetos. Desta maneira, é *formalizado o conhecimento.* O registro correto e completo das informações garante uma adequada implementação.

III.4. Caracterização de Métodos de Desenvolvimento de Software Orientado a Objetos

Os métodos de desenvolvimento de software orientado a objetos propostos na literatura técnica são diversos. Apresentam características variadas e, de modo geral, estão influenciados por um determinado enfoque de desenvolvimento (*abordagem orientado a funções, dados*) ou por uma *linguagem de programação*. Em geral, os métodos partilham as seguintes atividades (Clunie e Werner 1994):

- *compreender os requisitos do sistema;*
- *identificar e classificar as classes e objetos;*
- *definir os objetos em termos de visões estáticas e dinâmicas;*
- *identificar as relações estáticas e dinâmicas entre objetos;*
- *implementar as classes e objetos; e*
- *gerar uma documentação.*

Para *compreender os requisitos* do sistema, devemos ler o documento de requisitos e consultar qualquer outra fonte de informação onde o problema, ou parte dele, seja descrito. Além disso, temos que entrevistar os utilizadores do sistema.

Para *identificar e classificar classes e objetos*, alguns métodos propõem diversas estratégias, como procurar substantivos, pronomes, adjetivos e advérbios no documento de requisitos inicial, por exemplo em Wirfs-Brocks (1990); outros oferecem diretrizes para procurar objetos como em Coad e Yourdon (1992), Shlaer e Mellor (1990), Booch (1994); enquanto outros sugerem que a melhor forma de identificar classes e objetos é identificar o seu comportamento no sistema (Rubin e Goldberg 1992).

Um objeto é definido em termos das visões *estática e dinâmica.* A visão estática inclui a representação das classes e objetos, atributos, classificação ("*é um*" e "*parte de*"), métodos, relacionamentos e agrupamento de classes e objetos (*clusters).*

Por outro lado, a visão dinâmica é geralmente descrita usando técnicas de modelagem dinâmica, como são: *Diagramas de Comunicação de Objetos* e *Diagrama de Ciclo de Vida dos Objetos* (Yourdon 1994), *Diagramas de Fluxos de Eventos* e *Diagrama de*

Estado (Rumbaugh et al 1994), *Diagrama de Estado - Transição* e *Diagrama de Eventos* (Martin e Odell 1993), *Modelo de Estado* e *Modelo de Comunicação de Objetos* (Shlaer e Mellor 1992) e *Modelo de Interação de Objetos* (Embley et al. 1992).

As *relações* entre objetos podem ser *estáticas ou dinâmicas*. As relações estáticas são representadas por seu nome e cardinalidade. As relações dinâmicas constituem as mensagens de ligação entre objetos.

A *etapa de implementação* é realizada uma vez concluídos os passos anteriores. É escolhida uma linguagem orientada a objetos para a implementação do modelo de objetos. Esta etapa não deve ser considerada a última, pois é possível ter-se que reiniciar parte do processo, dada a natureza *incremental - iterativa* do processo de desenvolvimento, utilizando-se este paradigma.

A *documentação* desempenha um papel crucial no desenvolvimento de software. Muitos métodos incluem explicitamente em que momento e de que forma deve ser gerada a documentação, enquanto em outros isso é apenas implícito.

Baseado nos trabalhos de Capretz e Lee (1992), Walker (1992), Monarchi e Putchen (1992), Champeaux e Faure (1992), Winblad et al. (1993), Gilpin (1993) e Henderson-Sellers (1994), e de nossa experiência no uso de um método de desenvolvimento orientado a objetos (Clunie et al. 1995b), (Clunie et al. 1995c), apresentamos algumas considerações sobre os requisitos desejáveis para métodos de desenvolvimento de software orientado a objetos.

- **Com relação aos Princípios e Conceitos**

Os métodos de desenvolvimento de software orientado a objetos de modo geral partilham as mesmas atividades. No entanto, o *modelo de objetos*, formado por um *conjunto de abstrações,* é o produto principal destas atividades que deve ser gerado atendendo aos *princípios* e *conceitos*, discutidos na seção III.2.

- **Com relação à Simbologia e Notação**

Do ponto de vista sintático os métodos de desenvolvimento devem oferecer um conjunto de simbologia e notação simples, que facilitem sua compreensão e viabilizem a comunicação com o usuário e os membros da equipe de desenvolvimento. Além disso, do ponto de vista semântico, essas representações devem possibilitar a modelagem de uma variedade de situações.

- **Com relação às atividades de Verificação e Validação**

Para obter qualidade é necessário que o método de desenvolvimento forneça mecanismos para que o processo de desenvolvimento seja orientado à qualidade. Ou seja, ao se utilizar o método, devem estar definidos, ao longo do todo o ciclo de vida, os marcos e pontos de controle, indicando onde e quando devem ser realizadas as avaliações. Além disso, devem estar definidos os critérios de qualidade a serem utilizados.

- **Com relação à Independência da Linguagem**

A independência da linguagem de programação durante o processo de desenvolvimento nas fases de análise e projeto é um assunto de muita controvérsia. Alguns autores (Meyer 1988), (Wirfs-Brock et al. 1990), (Winblad et al. 1993), (Booch 1994) consideram que esta característica é desejável; no entanto, aceitam o fato de que em alguma etapa do processo de desenvolvimento se tenha conhecimento da linguagem de implementação do sistema. O problema está na distinção que deve ser feita entre a etapa de *arquitetura do sistema* e *implementação.* Ou seja, o momento em que a demarcação acontece não é perfeitamente controlado pela maioria dos métodos.

- **Com relação ao Tratamento da Complexidade**

A maioria dos métodos de desenvolvimento orientado a objetos coincidem, em maior ou menor grau, quanto à necessidade de contar com estratégias para decompor o domínio do problema, isto é, organizar e estruturar o conjunto de classes e objetos (*cluster)* em diferentes níveis de abstração para controlar a complexidade. No entanto, estes processos de *partição e agregação* apresentam um tratamento bastante restrito em alguns métodos, e em outros não são nem mesmo considerados. Além disso, por ser o processo de desenvolvimento de software orientado a objetos um processo *iterativo - incremental - recursivo,* onde as atividades e as fases estão sobrepostas, os métodos de desenvolvimento devem permitir ao desenvolvedor *compreender* a tarefa que está realizando, *focalizar* sua atenção de uma maneira relativamente simples numa determinada atividade e *integrá-la* ao processo de desenvolvimento como um todo.

- **Com relação ao Gerenciamento do Processo de Desenvolvimento**

O método de desenvolvimento, além de fornecer estratégias para realizar particões e agregação que permitam uma divisão efetiva das tarefas a serem realizadas pela equipe de desenvolvimento, deve oferecer guias e mecanismos para uma melhor utilização dos conhecimentos, recursos disponíveis para o projeto e suporte para o controle e gerência do processo, verificando o andamento do trabalho e controlando a execução e encadeamento de atividades.

- **Com relação à Ferramenta CASE**

O êxito da aplicação de um método de desenvolvimento num projeto depende da disponibilidade e possibilidade de utilização de uma ferramenta automatizada para seu

suporte. Esta deve oferecer suporte completo durante o processo de desenvolvimento, no que se refere ao apoio na geração, manutenção, documentação, avaliação e gerência do projeto. Assim sendo, deve fornecer aos engenheiros de software os mecanismos para coleta de métricas, para avaliar produtos e o processo de desenvolvimento e para tornar mais fácil a alteração das definições de processo.

- **Com relação à Reutilização**

Os métodos de desenvolvimento devem fornecer algumas diretrizes a serem seguidas quando se trata de reutilização. No *desenvolvimento com reutilização,* atividades específicas relacionadas à *avaliação, adaptação e integração* de componentes existentes no novo contexto devem ser inseridas nas fases do ciclo de vida adotado. Estas atividades incluem: identificação dos requisitos do componente, procura de componentes, compreensão do componente, seleção do componente, adaptação do componente e documentação (Karlsson 1995). Por outro lado, no *desenvolvimento para reutilização,* devem incluir atividades planejadas para construir componentes com possibilidades de reutilização que envolve uma série de decisões de projeto (Karlsson 1995).

Dimensões	Critérios
Abstração	• Heurísticas para identificar classes e objetos, estados, operações e interfaces e para encapsular classes e objetos. • Heurísticas para criar classes abstratas. • Heurísticas para agregar e decompor um conjunto de classes objetos em diferentes níveis de abstração.
Simbologia e Notação	• Notação para representar mensagens enviadas, solicitadas e passagem de parâmetros entre objetos numa hierarquia de classes. • Suporte para representar concorrência, restrições e exeções.
Independência da Linguagem	• Estabelecer os limites entre a arquitetura do sistema e a implementação.
Verificação/Validação	• Regras de transformação entre alternativas de representação de projeto. • Heurísticas para geração de testes. • Métricas para testes. • Métricas para verificação e validação do modelo.
Reutilização	• Heurísticas para identificar, buscar, compreender, selecionar, adaptar, integrar e documentar a componente. • Heurísticas de projeto.
Feramenta CASE	• Possuir suporte automatizado para todo o desenvolvimento.
Gerência	• Definir métricas para o controle do processo de desenvolvimento. • Definir métricas para estimativas de custos e cronogramas. • Estabelecer divisão de tarefas na equipe de desenvolvimento.

Tabela III.1. - Requisitos dos Métodos de Desenvolvimento Orientado a Objetos

As discussões apresentadas podem ser sumarizadas na Tabela III.1, através do conjunto de critérios a se considerar num método de desenvolvimento orientado a objetos, agrupados nas dimensões de *abstração, notação e simbologia, independência da linguagem, verificação* e *validação, reutilização, ferramenta CASE* e *gerência.*

III.5. Métodos de Desenvolvimento de Software Orientado a Objetos

A abordagem de orientação a objetos considera que a mente humana abstrai naturalmente o mundo da aplicação como um mundo de objetos que interagem entre si, cada um possuindo propriedades e comportamentos próprios, e onde cada objeto pode ter outros objetos como componentes de sua estrutura. Partindo dessa assertiva, o desenvolvimento de software orientado a objetos constitui uma abordagem que minimiza, como o uso da abstração, a distância entre os problemas do mundo real e as soluções no mundo computacional, diminuindo, portanto, a complexidade dos sistemas.

Existem na literatura inúmeras propostas de métodos de desenvolvimento orientado a objetos. Algumas propostas são feitas para apoiar a fase de análise, outras para a fase de projeto e poucas cobrem adequadamente todo o processo de desenvolvimento. De maneira geral, as propostas ainda não estão consolidadas e a maioria está no processo de maturação.

Por este motivo, para efeito desta tese, foram considerados quatro critérios de seleção dos métodos de desenvolvimento de software orientado a objetos que serão apresentados: *disponibilidade de uma documentação do método através da existência de um livro texto, referências citadas em publicações especializadas na área de orientação a objetos, utilização do método, pelo menos no meio acadêmico, e evolução do método através da existência de versões anteriores.*

Baseado nestes critérios e considerando que nosso trabalho insere-se no contexto de avaliação da qualidade do produto nas fases iniciais do desenvolvimento, apresentamos de maneira sucinta os seguintes métodos de desenvolvimento orientado a objetos para as fases de análise e projeto: *Booch, Coad* e *Yourdon, Jacobson, Rumbaugh, Wirfs-Brock, Coleman, Martin e Odell, Shlaer e Mellor, Rubin, Firesmith* e *Lano e Haughton.* Uma descrição detalhada e um estudo comparativo de alguns destes métodos podem ser encontrados em Clunie e Werner (1994).

III.5.1. Método de Booch (Booch 1994), (White 1994).

Grady Booch é o responsável pelo desenvolvimento de um método orientado a objetos para as fases de análise e projeto, denominado *Object-Oriented Analysis and Design.* O objetivo do método é prover um processo e uma notação para o desenvolvimento e comunicação nas fases de análise e projeto de um sistema orientado a objetos. Isto é, criar um modelo de sistema que possa ser implementado. O método se baseia no modelo de desenvolvimento em *espiral*, onde os enfoques *incremental e iterativo* são utilizados, dando ênfase durante o processo de desenvolvimento à construção de *protótipos*. Consiste das etapas de *análise de requisitos, análise de domínio* e *projeto de sistema.* O método utiliza um conjunto de diagramas que permitem modelar as estratégias adotadas nas fases de análise e projeto. Destacam-se os *diagramas de classes, transição de estados, objetos, processos, categorias, módulos* e *subsistemas.*

III.5.2. Método de Coad e Yourdon (Coad e Yourdon 1992), (Coad e Yourdon 1993), (Coad e Nicola 1993), (Coad et al. 1995).

Coad, Yourdon e Nicola são os responsáveis pelo desenvolvimento de um método orientado a objetos para apoiar as fases de análise (*OOA - Object Oriented Analysis*), projeto (*OOD - Object Oriented Design*) e programação (*OOP - Object Oriented Programming).* Os conceitos estabelecidos pelo método para o tratamento da complexidade *(abstração, encapsulamento, generalização - especialização, associação, comunicação através de mensagem* e *escala*) são evidenciados em sua notação, arquitetura, assuntos, padrões e qualidade dos requisitos, com o desenvolvimento de um modelo de multiníveis. Estes conceitos são aplicados de maneira consistente ao longo de todo o processo de desenvolvimento. O método permite um desenvolvimento orientado a objetos bastante abrangente que inclui notação, atividades e estratégias. Isto é realizado através de um desenvolvimento concorrente (*baseball model*) das seguintes atividades: análise orientada a objetos, projeto orientado a objetos e programação orientada a objetos.

III.5.3. Método de Jacobson (Jacobson et al. 1992).

Ivar Jacobson é responsável pelo desenvolvimento de um método de engenharia de software orientado a objetos denominado OOSE - *Object Oriented Software Engineering.* O método parte da premissa de que a complexidade pode ser tratada e controlada através do uso de uma seqüência de modelos: *modelo de requisitos, modelo de análise, modelo de projeto, modelo de implementação* e *modelo de testes.* Cada modelo tem a responsabilidade de captar um determinado aspecto do sistema que está sendo construído. Estes modelos são produto das fases de *análise de requisitos,*

construção e *testes*, como mostra a Figura III.1. Os autores estabelecem que a análise deve produzir uma descrição do sistema independente da realidade de implementação. A definição de atributos e métodos é adiada para o projeto, pois a prioridade da análise é definir papéis e responsabilidades dos elementos do sistema. O principal conceito do método é o *use case*. *Use cases* são as situações a que o sistema pode ser submetido, ou seja, as situações de processamento a que o sistema deve responder.

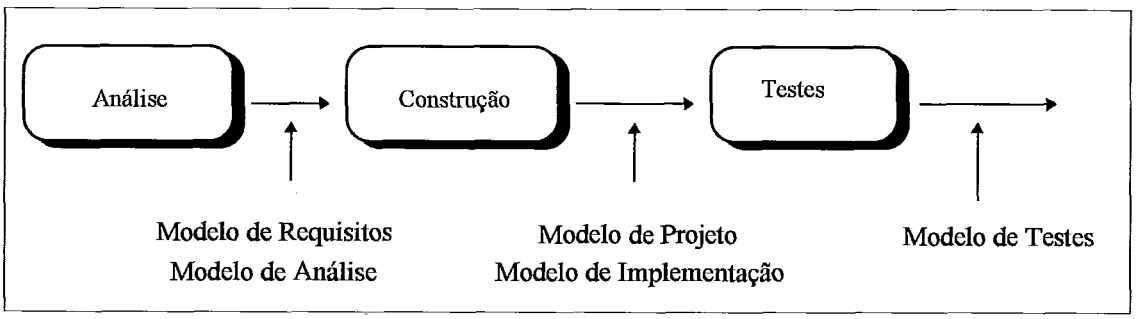


Figura III. 1. - Modelos associados às Fases de Desenvolvimento OOSE (Jacobson et al. 1992)

III.5.4. Método de Rumbaugh (Rumbaugh et al. 1994).

Rumbaugh, Blaha, Premerlani, Eddy e Lorensen, da General Electric Research and Development Center, são os responsáveis pelo desenvolvimento de um método de modelagem e projeto orientado a objetos (OMT- *Object-Oriented Modeling and Design Technique*). O método está subdivido em quatro etapas: *análise, projeto de sistema, projeto de objeto e implementação*, fazendo uma distinção entre as atividades de análise e projeto de modelagem. O modelo da análise é segmentando em três submodelos: o *modelo de objeto,* que descreve a estrutura estática do sistema, ou seja, as classes e suas associações; o *modelo dinâmico,* que descreve a seqüência de interações entre os objetos do sistema e o *modelo funcional,* que descreve as transformações de valores dos dados procedidas pelo sistema. A etapa de projeto é dividida em duas subetapas: o *projeto do sistema* e o *projeto dos objetos.* Na etapa de *projeto do sistema* são definidos os subsistemas. Cada subsistema engloba classes do sistema que compartilham algumas propriedades comuns. No *projeto dos objetos* é realizado o processo de otimização e refinamento do *modelo de objetos,* para facilitar sua transição a uma linguagem de programação.

III.5.5. Método de Wirfs-Brock (Wirfs-Brock et al. 1989), (Wirfs-Brock et al. 1990a), (Wirfs-Brock et al. 1990b), (Gilpin 1993).

Wirfs-Brock, Wilkerson e Wienir são os responsáveis pelo desenvolvimento de um método de projeto orientado a objetos baseado em responsabilidades *(RDD* -

Responsabilities Driven Design). O RDD é um método baseado no modelo *cliente-servidor*, onde os sistemas são considerados compostos por uma coleção de servidores, que apresentam suas próprias responsabilidades e que oferecem serviços a seus clientes através de *contratos.* O método fornece guias para o desenvolvimento de software reutilizável (*componentes, classes abstratas, frameworks* e *aplicações)* e focaliza sua atenção a aspectos de *extensibilidade, coesão e acoplamento.* O método está baseado nas seguintes fases: *inicial /exploratória* e *análise detalhada*, como indica a Figura III.2.

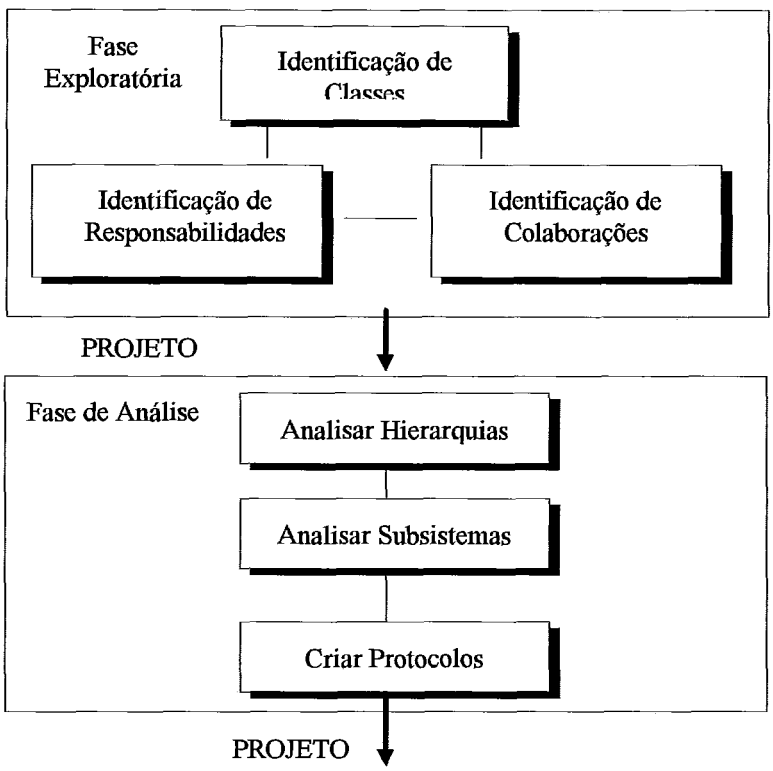


Figura III.2. - Fases do Projeto: Exploração e Análise (Wirfs-Brock et al. 1990a)

III.5.6. Método de Coleman (Coleman et al. 1993).

D. Coleman, C. Dollin e P. Jeremaes, pertencentes ao *Object-Oriented Design Group* da Hewlett-Packard Laboratories, são os responsáveis pelo desenvolvimento de um método orientado a objetos denominado *Fusion.* O método se propõe a integrar o melhor dos aspectos descritos nos métodos formais de desenvolvimento orientado a objetos, como Object-Z (Duke et al. 1991) e Z++ (Lano e Haughton 1993) e dos métodos de Booch (Booch 1991), OMT (Rumbaugh et al. 1991) e CRC (Beck e Cunningham 1989), (Wirfs-Brock et al. 1990), como mostra a Figura III.3. Como ponto de partida o método utiliza o documento de requisitos escrito em linguagem natural, pois o método não contempla explicitamente o processo de redigir o documento de requisitos. O método trabalha com três etapas no desenvolvimento: *análise, projeto* e *implementação.*

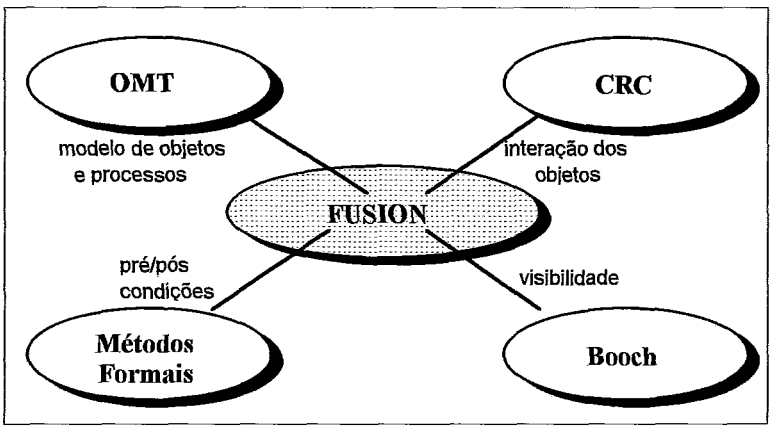


Figura III.3. - Métodos que influenciam ao Método Fusion (Coleman et al. 1993)

Na análise é construído o modelo de objetos e o modelo de interface. O modelo de objetos de Fusion é muito semelhante ao modelo de objetos de OMT, ou seja, com uma semântica muito próxima aos diagramas de entidade-relacionamento. O modelo de interface contém a modelagem dinâmica desenvolvida na etapa de análise. Durante a análise não existe a nocão de método de classe, mas de operação do sistema. São identificadas as possíveis interações do sistema com meio externo - as operações - de uma forma semelhante à construção do modelo de requisitos de OOSE. Após isto, as operações são descritas. Na modelagem das operações durante análise, o sistema é sempre visto como um todo (ao invés de um conjunto de objetos que interagem). Durante o projeto é produzida uma descrição do conjunto de classe-objetos - são definidos novos atributos e o conjunto de métodos. O projeto utiliza modelos diferentes dos modelos da análise. Para implementação, Fusion prevê o uso de uma linguagem orientada a objetos.

III.5.7. Método de Martin e Odell (Martin e Odell 1992), (Martin 1994).

A análise e projeto orientado a objetos proposta por Martin e Odell é considerada uma extensão da *Engenharia de Informação.* Seu objetivo é melhorar a modelagem de objetos através do uso de uma simbologia familiar aos praticantes da Engenharia de Informação. O método parece com o modelo de entidade-relacionamento; entretanto, define um conjunto de notações para representar qualquer aspecto do objeto (*herança, composição e comunicação*). Isto permite que seja gerado ao longo do processo de *Engenharia de Informação Orientada a Objetos* (EIOO) um modelo de objetos.

O método é caracterizado por dois aspectos: um primeiro aspecto refere-se aos *tipos de objetos, relacionamentos entre objetos e herança.* Isto se realiza através das

atividades de *análise e projeto da estrutura de objeto*. Um segundo aspecto refere-se ao *comportamento dos objetos* e as suas mudanças no tempo. Isto se realiza através das atividades de *análise e projeto de comportamento de objetos*. Sendo assim, o desenvolvimento do sistema utilizando este método consta de quatro atividades, cada uma das quais está associada a uma ou mais subatividades orientadas a objetos.

III.5.8. Método de Shlaer e Mellor (Shlaer e Mellor 1990), (Shlaer e Mellor 1992).

Shlaer e Mellor são os responsáveis pelo desenvolvimento de um método de análise orientado a objetos - OOA - *Object Oriented Analysis*. O método pertence à categoria de técnicas de modelagem semântica de dados, onde existe uma preparação típica para a implementação em um banco de dados relacional. O método sugere que o analista elabore três modelos: *modelo de informação, um conjunto de modelos de estado* e *um conjunto de modelos de processos*. Assim sendo, o método oferece um método de análise, um sistema de notação e um padrão de documentação para projetos de sistemas orientados a objetos. Adota o conceito de objeto como um registro em um banco de dados e fornece estratégias para modelar os dados

III.5.9. Método de Rubin e Goldberg (Rubin e Goldberg 1992).

K. S. Rubin e Goldberg da ParcPlace Systems são os responsáveis pelo desenvolvimento de um método de análise orientado a objetos denominado - OBA - *Object Behavior Analysis*. Este método enfatiza primeiramente o que acontece no sistema, isto é, identificam-se os comportamentos do sistema. Em seguida, assinalam-se esses comportamentos a partes do sistema e tenta-se compreender quem *inicia* e quem *participa* desses comportamentos. Assim sendo, o método facilita a compreensão das múltiplas interações entre objetos, pois a partir daí é possível identificar todas as responsabilidades necessárias para suportar o comportamento do sistema. O método está na categoria daqueles que definem inicialmente os objetos a partir de seu comportamento, mais que de seus atributos. Neste sentido, o método utiliza o enfoque puro de orientação a objetos, em oposição aos métodos de orientação a objetos considerados híbridos.

III.5.10. Método de Firesmith (Firesmith 1993).

Firesmith é o responsável pelo desenvolvimento de um método de análise de requisitos e projeto lógico denominado *OORALD - Object-Oriented Requeriments Analysis and Logic Design*. De maneira geral, o método fornece um adequado entendimento dos conceitos de desenvolvimento orientado a objetos, uma descrição detalhada de sua notação gráfico - textual, um método de desenvolvimento orientado a

objetos da terceira geração, denominado *ADM3 (ASTS Development Method 3),* um tratamento dos aspectos estáticos e dinâmicos e um guia para o gerenciamento do processo de desenvolvimento. O método utiliza o enfoque iterativo-recursivo para o desenvolvimento do sistema, trabalhado com seis modelos diferentes, *modelo de sistema (assembly model), modelo de objeto, modelo de classe, modelo de estado, modelo de controle* e *modelo de tempo.*

III.5.11. Método de Lano e Haughton (Lano e Haughton 1993).

Lano e Haughton são os responsáveis pelo desenvolvimento de método orientado a objetos denominado Z++. É um método formal que exige o desenvolvimento rigoroso de um sistema utilizando técnicas de orientação a objetos. O método trabalha com uma extensão da linguagem de especificação formal Z e combina alguma notação do método OMT. Z é uma notação baseada em lógica de predicados e teoria de conjuntos para modelagem rigorosa e construtiva de sistemas. Uma especificação em Z é formada por vários tipos de declarações para introduzir nomes, expressões que denotam termos, predicados que expressam propriedades dos valores dos nomes declarados e esquemas que impõem uma estrutura sobre as declarações e predicados. O método permite a formalização dos requisitos e o desenvolvimento de uma especificação sem ambigüidades, que possa ser mapeada diretamente para o código. Portanto, uma característica do método é o uso consistente de sua notação formal e diagramática ao longo do desenvolvimento do sistema.

III.6. Considerações sobre os Métodos de Desenvolvimento Orientado a Objetos e a Qualidade de Software

A comunidade de software tem apresentado diversos métodos de desenvolvimento de software orientado a objetos que tem evoluído rapidamente e aumentado em número. De maneira geral, os métodos propostos na literatura estão influenciados por enfoques de desenvolvimento tradicional: *orientado a dados, funções* e *modelagem semântica* ou por uma *linguagem de programação.* Por isso, não é simples identificar, entre as propostas existentes, quais são os aspectos efetivamente relevantes dos métodos que nos permita determinar as similaridades e diferenças entre as propostas.

A literatura técnica sobre orientação a objetos apresenta diversos trabalhos relativos à comparação e/ou classificação de métodos de desenvolvimento segundo este paradigma. Estes trabalhos se caracterizam pela sua diversidade com relação aos aspectos considerados na discussão de similaridades e diferenças entre os métodos de desenvolvimento (Arnold et al. 1991), (Sutcliffe 1991), (Champeaux e Faure 1992), (Fichman e Kemerer 1992), (Monarchi e Putchen 1992), (Capretz e Lee 1992), (Walker

1992), (Karam e Casselman 1993), (Gilpin 1993), (Sharble e Cohen 1993) e (Clunie e Werner 1994).

O grupo OMG - *Object Management Group* (Hutt 1994) relata os resultados de suas pesquisas com relação ao uso de métodos orientados a objetos em organizações dedicadas ao desenvolvimento de sistemas. Como resultado deste estudo, propõe uma *estrutura técnica* para comparar métodos de análise e projeto orientado a objetos. A estrutura é organizada considerando os componentes mais representativos nos métodos de desenvolvimento orientado a objetos. Estes componentes incluem: *modelo de ciclo de vida, estratégias de modelagem, modelagem da análise, modelagem do projeto, modelagem da implementação, construção, produtos resultantes* e *reutilização.* De acordo com o método empregado, estes componentes podem ser estudados de forma separada.

É importante salientar que os componentes de modelagem relativos a análise e o projeto apresentam preocupação pela qualidade, sugerindo a utilização de técnicas de controle da qualidade (*walkthrough*) e a realização de verificações de consistência e completitude nos modelos. No entanto, não fornecem atributos de qualidade específicos a serem utilizados, nem indica a maneira como devem ser feitas as avaliações. Isto é evidenciado pela pouca maturidade com que são abordados os aspectos de qualidade nos métodos de desenvolvimento orientado a objetos. A pesquisa realizada pelo grupo *OMG* (Hutt 1994), sobre o uso de *conceitos de qualidade* nos métodos, conclui da seguinte maneira:

"*A desigualdade das respostas obtidas, com relação aos conceitos de qualidade nos métodos de desenvolvimento orientado a objetos, torna difícil chegar a ter uma opinião conclusiva a respeito. Isto indica a existência de diferenças de estilo, maturidade e definição de processos de desenvolvimento*".

Isto nos confirma que, apesar das diversas propostas existentes na literatura (descritas no Capítulo V), a área de qualidade em orientação a objetos é incipiente (Henderson-Sellers 1996). Ainda que algumas características de qualidade de software como *manutenibilidade, extensibilidade, testabilidade* e *reutilizabilidade* sejam freqüentemente enfatizadas nos projetos desenvolvidos utilizando este paradigma, as mesmas não são especificadas por nenhum método de desenvolvimento orientado a objetos.

III.7. Conclusão

Este capítulo apresentou uma discussão sobre os princípios e conceitos em orientação a objetos, caracterizou modelos e métodos de desenvolvimento orientado a objetos, mostrou as propostas mais representativas de métodos de desenvolvimento

orientado a objetos e conclíu com uma discussão sobre aspectos de qualidade nos métodos de desenvolvimento orientado a objetos.

Infelizmente, os métodos de desenvolvimento orientado a objetos não têm maturidade suficiente, pois não existe nenhum método orientado a objetos que tenha atingido um status reconhecido, como é o caso dos métodos estruturados do paradigma tradicional. A seguir são apresentadas nossas considerações gerais sobre o desenvolvimento de software orientado a objetos:

- Percebe-se a necessidade de consenso por parte da comunidade de engenharia de software sobre os *princípios, conceitos e terminologia* utilizados no desenvolvimento orientado a objetos.

- As atividades de análise e projeto, nos métodos descritos, de certa forma são semelhantes. De maneira geral, cada método possui uma maneira particular de identificar e representar os objetos e seus relacionamentos, utilizando suas próprias estratégias, notação, simbologia e terminologia.

- Uma padronização para as fases de análise e projeto não foi identificada. Observa-se, por exemplo, que algumas atividades definidas na fase de análise de um método são incluídas na fase de projeto de outro, não sendo claro o limite e distinção entre as fases.

- De modo geral os enfoques de desenvolvimento utilizados são o *incremental, espiral, iterativo* e *concorrente.*

- Existe pouca maturidade com relação à avaliação da qualidade do produto e a qualidade do processo nos métodos de desenvolvimento orientado a objetos.

- Alguns métodos fornecem maneiras de tratar a complexidade com a partição (*decomposição*) do sistema em subsistemas ou por agrupamento de objetos (*clusters*). No entanto, critérios mais explícitos e práticos que permitam agregar classes ou decompor sistemas devem ser definidos.

- Com relação à *modelagem dos aspectos dinâmicos do sistema,* também se evidencia uma relativa imaturidade, pois as técnicas para modelar estes aspectos têm sido adaptadas de fora do paradigma, trazendo com isto algumas dificuldades. Os aspectos estáticos e dinâmicos são tratados com relativa independência. A maioria dos métodos modelam os aspectos dinâmicos, baseados na MEF (Máquina de Estado Finito) através de diagramas de transição de estados.

Não é possível afirmar qual o melhor método orientado a objetos. Com a continuidade das experiências, novas técnicas de análise e projeto surgirão para suportar a modelagem e implementação de sistemas desenvolvidos segundo este paradigma. Por outro lado, os métodos de desenvolvimento orientado a objetos existentes seguirão sofrendo mudanças de estratégias que permitam um adequado desenvolvimento dos sistemas. Até então, os projetos desenvolvidos utilizando este enfoque necessitarão da *habilidade* e *experiência* por parte do grupo desenvolvedor e de uma cuidadosa *gerência do processo de desenvolvimento*.

No próximo capítulo abordaremos aspectos relativos à qualidade de software e ao paradigma de orientação a objetos, isto é, o estado da arte com relação à qualidade de sistemas orientados a objetos.

Capítulo IV

Qualidade de Software e o Paradigma de Orientação a Objetos

"Os defeitos não são de graça. Alguém os faz e é pago para fazê-los"

W. Edwards Deming

IV.1. Introdução

O *estado da arte* das métricas de qualidade de software é um problema geral da indústria, e não apenas do paradigma de orientação a objetos. Pesquisadores têm desenvolvido propostas com a tentativa de estabelecer métricas quantitativas para o desenvolvimento de software, com pouco sucesso (Martin W. 1996). Por exemplo, nos últimos 20 anos, é reconhecida a importância de minimizar o acoplamento. No entanto, não têm sido publicados estudos científicos que comprovem que o acoplamento é um aspecto negativo. Estes estudos poderiam medir o acoplamento entre sistemas e obter valores estatísticos representativos que relacionassem o acoplamento com o esforço de manutenção ou com outros fatores de qualidade. Isto induz a pensar que os desenvolvedores de software acreditam que o acoplamento é prejudicial. Contudo, este assunto é mais complexo do que aparenta. Isto é, o que é mais prejudicial? A quantidade de acoplamento ou a natureza do acoplamento? (Fowler 1993).

Dadas as diferenças inerentes entre as duas visões - *orientação a objetos* e *funções* - não é de se surpreender ou constatar que as métricas de software desenvolvidas para serem aplicadas aos métodos tradicionais de desenvolvimento não são facilmente mapeadas para os novos conceitos de orientação a objetos, tais como *classes, herança, encapsulamento e comunicação via mensagens* (Wilde e Huitt 1992). Por este motivo, dado que as métricas de software existentes estão sendo questionadas

pela falta de fundamentação teórica e rigor matemático (Fenton 1994), (Chidamber e Kemerer 1994) e por não suportar os conceitos do paradigma de orientação a objetos, parece apropriado o desenvolvimento de novas métricas especialmente dirigidas a medir a qualidade dos produtos gerados durante o desenvolvimento orientado a objetos.

As métricas orientadas a objetos estão muito distantes de serem consideradas regras de controle do processo de desenvolvimento de software, devido principalmente a que o paradigma de orientação a objetos é relativamente novo, apresentando pouquíssimos relatos de experiências de coleta de métricas na indústria (Williams 1993), (Basili et al. 1995; 1996), (Henderson-Sellers 1996). Além disso, os métodos de desenvolvimento orientado a objetos atualmente disponíveis não tratam com maturidade esta questão.

As métricas desenvolvidas para serem utilizadas no paradigma tradicional são inadequadas para avaliar sistemas orientados a objetos, embora algumas ainda sejam utilizadas, particularmente para avaliar os métodos nas classes (Chidamber e Kemerer 1991; 1993; 1994), (Tegarden et al. 1992), (Li e Henry 1993a;1993b), (Abreu e Carapuça 1994a; 1994b), (Li et al. 1995), (Stiglic et al. 1995). Portanto, novas métricas devem ser propostas pela comunidade de orientação a objetos, pois os engenheiros de software precisam obter informações sobre o sistema em desenvolvimento.

No Capítulo II foram apresentados, de modo geral, aspectos chave que ajudam a compreender o conceito de *qualidade de software*. Os principais modelos de qualidade de software foram descritos. Foram discutidos os problemas das medições em software, definindo propriedades e taxonomias das métricas e foram discutidos aspectos relacionados à confiabilidade das mesmas.

No Capítulo III foram estabelecidos os princípios e conceitos do paradigma de orientação a objetos. Em seguida, foram descritos de forma sucinta os métodos de desenvolvimento orientado a objetos mais representativos. Ao final, concluiu-se que o desenvolvimento orientado a objetos apresenta sinais evidentes de imaturidade, devendo ser incorporados nos métodos mecanismos que permitam avaliar os produtos gerados ao longo do processo de desenvolvimento através de um conjunto de métricas de qualidade.

Neste capítulo, apresentamos os estudos encontrados na literatura e considerados os mais representativos na área de métricas aplicadas a sistemas desenvolvidos segundo o paradigma de orientação a objetos. São também apresentados critérios e diretrizes sugeridas por alguns autores com relação a este tema.

IV.2. Características de Qualidade para o Paradigma de Orientação a Objetos

Através da revisão da literatura, pudemos constatar que a área de qualidade de software para o caso do paradigma de orientação a objetos é relativamente escassa. Embora se vislumbre um aumento significativo de trabalhos sobre métricas orientadas a objetos aplicadas a diversas áreas desta tecnologia (Figura IV.1), ainda não estão disponíveis trabalhos considerados completos pela comunidade de orientação a objetos, com relação à fundamentação teórica e validação experimental de métricas. No entanto, existem algumas experiências e propostas de grande aceitação pela comunidade, destacam-se as pesquisas realizadas pelo grupo de *MIT* (Chidamber e Kemerer 1991; 1993;1994), o grupo de *Virginia Tech Metrics* (Li e Henry 1993a; 1993b), (Lattanzi e Henry 1994) e (Li et al. 1995), os trabalhos de Lorenz e Kidd (1994) e Abreu e Carapuça (1994a; 1994b).

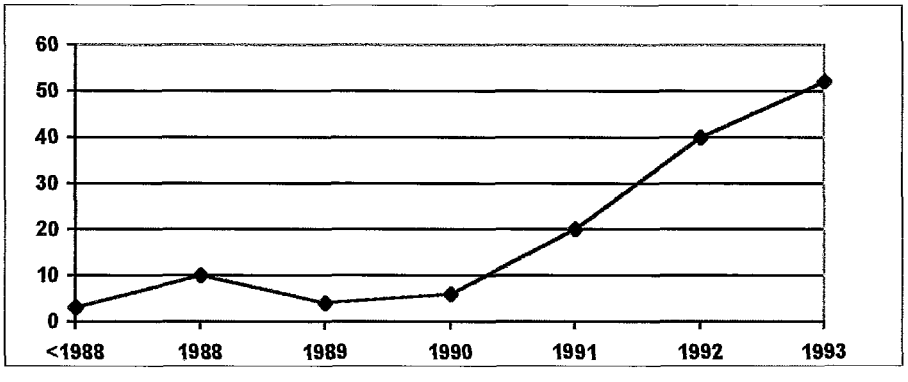


Figura -IV.1. Número de trabalhos publicados sobre Métricas Orientadas a Objetos (Whitty 1996)

Duas justificativas podem ser consideradas em relação a esta realidade. Em primeiro lugar, a imaturidade do paradigma e, em segundo lugar, a falta de reconhecimento generalizado nas organizações pelo controle da qualidade de seus produtos e processo de desenvolvimento.

A seguir são apresentadas as propostas mais discutidas na literatura sobre qualidade em orientação a objetos. Outras propostas, centradas em aspectos relativos à complexidade em sistemas orientados a objetos, podem ser encontradas em Tegarden et al. (1992; 1995) e Henderson-Sellers (1996).

IV.2.1. Proposta de Chidamber e Kemerer (Chidamber e Kemerer 1991; 1993; 1994).

Entre os trabalhos citados na literatura referentes à área de qualidade do software desenvolvido utilizando o paradigma de orientação a objetos, a proposta de Chidamber e Kemerer, do *Massachusetts Institute of Technology - MIT,* é considerada a mais relevante. Estes autores consideram que a qualidade de um projeto orientado a objetos pode ser avaliado analiticamente, através de um conjunto de seis métricas formais. Estas métricas foram estabelecidas a partir dos princípios *ontológicos*[1] descritos em Bunge (1977) e das atividades de desenvolvimento orientado a objetos descritas em Booch (1991). Os princípios ontológicos são interessantes para os pesquisadores teóricos da orientação a objetos porque tratam com o significado e definição das representações no mundo real, o qual é precisamente o objetivo e a base fundamental do paradigma da orientação a objetos.

Para estes autores, um projeto orientado a objetos pode ser visto como um sistema relacional, sendo definido como uma tupla que consiste de um conjunto de elementos, um conjunto de relações e um conjunto de operações binárias. Mais especificamente, um projeto orientado a objetos *P* é representado como um sistema relacional consistindo de classes - objetos, relacionamentos empíricos e operações binárias que são realizadas por estas classes - objetos. Isto pode ser expresso formalmente da seguinte maneira:

$\mathbf{P = (A, R_1.. R_n, O_1..O_m)}$, onde
A é o conjunto de classes - objetos;
$\mathbf{R_1.. R_n}$ são relacionamentos entre as classes - objetos; e
$\mathbf{O_1..O_m}$ são operações binárias.

Para realizar qualquer medida do projeto de objetos, é necessário partir do relacionamento empírico e realizar uma transformação para um relacionamento formal. As métricas propostas são as seguintes:

- **Ponderação dos Métodos na Classe**

Refere-se à quantidade e à complexidade dos métodos declarados na classe. Esta métrica está diretamente relacionada com a definição de Bunge (1977) sobre a complexidade dos objetos no mundo real. Uma vez que os métodos sejam considerados como *propriedades*[2] das classe-objetos, a complexidade é determinada pela cardinalidade

[1] Ontologia é parte da metafísica que estuda o ser em geral e suas propriedades transcendentais.
[2] De acordo com a ontologia, o mundo real é visto como sendo composto de objetos, referidos como substâncias individuais e conceitos. Estas substâncias individuais possuem propriedades. As propriedades são características inerentes que possuem as substâncias individuais. O observador pode designar outras características, mas elas são consideradas como atributos e não propriedades.

do conjunto de suas propriedades. O número de métodos e a complexidade dos métodos são indicadores do requisito de tempo e esforço para desenvolver e manter o objeto. Portanto, um grande número de métodos em um determinado objeto é um indicador do potencial do impacto sobre as especializações imediatas na hierarquia, uma vez que as especializações herdam os métodos definidos no pai.

- **Profundidade da Árvore de Herança**

Refere-se à longitude máxima a partir do nó (classe), em particular até o nó raiz da árvore. Na estrutura hierárquica é visualizada a profundidade da classe, e o número de métodos com possibilidades de serem herdados representam um indicador do grau de complexidade da estrutura. Isto é, árvores de hierarquia muito profundas produzem um projeto mais complexo.

- **Número de Filhos**

Refere-se ao número de subclasses imediatas subordinadas a uma determinada classe em uma hierarquia de classes. Segundo estes autores, é melhor ter profundidade (*nível de profundidade da herança*) que largura *(número de especializações)* numa hierarquia de classes, sempre e quando esta promova a reutilização de métodos através da herança.

- **Acoplamento entre Objetos**

Refere-se à contagem do número de relacionamentos entre classes, sem incluir os relacionamentos por herança. Os autores assinalam que um acoplamento excessivo entre objetos afeta a modularidade do projeto e impede a reutilização. Um modelo de objetos com um alto acoplamento entre objetos é altamente sensível a mudanças nas outras partes do modelo e a atividade de manutenção é mais difícil. Quanto maior independência tiver um objeto, mais fácil será sua reutilização em uma outra aplicação.

- **Falta de Coesão entre Métodos**

Refere-se ao grau de similaridade dos métodos na classe, através dos conjuntos disjuntos formados pelas variáveis de instâncias utilizadas pelos métodos. Se não existem variáveis de instâncias em comum, o grau de similaridade é zero (0). O número de conjuntos disjuntos fornece uma medida para indicar o grau de desigualdade dos métodos na classe. Menos conjuntos disjuntos implicam numa maior similaridade de métodos.

- **Resposta da Classe**

Refere-se ao conjunto de métodos que, potencialmente, são executados como resposta a uma mensagem recebida por um determinado objeto.

A contribuição da proposta de Chidamber e Kemerer refere-se a três aspectos:

a) Proposta de um conjunto de métricas que são geradas baseadas em conceitos teóricos relacionados à medição de software e ontologia dos objetos, incorporando a experiência profissional dos desenvolvedores de software;
b) Avaliação das métricas propostas com relação ao subconjunto de seis propriedades proposta por Weyuker (1988)[3]; e
c) Apresentação de dados empíricos, a partir de projetos desenvolvidos em empresas comerciais americanas, para demonstrar a viabilidade de coletar estas métricas e sugerir a forma como estas são utilizadas.

Estas métricas estão num processo de validação e reformulação pela comunidade de qualidade em orientação a objetos. Entre os trabalhos destacam-se os descritos em Chen e Lu (1993), Sharble e Cohen (1993), Lattanzi e Henry (1994), Zuze em Henderson-Sellers (1996), Churcher e Shepperd (1995), Stiglic et al. (1995) e Basili et al. (1995, 1996).

IV.2.2. Proposta de Li, Henry e Lattanzi (Li e Henry 1993a; 1993b), (Lattanzi e Henry 1994), (Li et al. 1995).

Estes autores formam parte do grupo de qualidade em orientação a objetos - *Virginia Tech Metrics* - da Universidade de Virginia que realiza pesquisas na área de validação de métricas orientadas a objetos relacionadas ao *esforço de manutenção* do software. Propõem um conjunto de métricas as quais são classificadas em dois grupos: *métricas de código* e *métricas de estruturas.* As *métricas de código* orientadas a objetos são geradas observando-se os componentes sintáticos do código. São identificadas oito métricas deste tipo, a maioria das quais já propostas por Chidamber e Kemerer (1991;1993, 1994). Estas métricas são as seguintes (Tabela IV.1.):

Métricas	Descrição
1. Profundidade da hierarquia	Posição da classe na hierarquia da herança.
2. Número de filhos	Quantidade de subclasses diretamente derivadas a partir de uma classe
3. Resposta da classe	Cardinalidade do conjunto de respostas dadas pela classe.
4. Coesão da classe	Correlação entre os métodos e as variáveis de instâncias da classe
5. Número de métodos locais	Quantidade de métodos locais na classe
6. Tamanho - 1	Número de pontos e vírgulas
7. Tamanho - 2	Número de métodos locais mais o número de dados dos atributos locais
8. Complexidade do método	Complexidade estática de todos os métodos locais de uma classe

Tabela IV.1. - Métricas de Código (Li e Henry 1993a, 1993b), (Lattanzi e Henry 1994), (Li et al. 1995)

[3]Propõe nove axiomas para avaliar a complexidade de programas e validar métricas desenvolvidas para o paradigma tradicional.

O segundo tipo de métricas orientadas a objetos são as *métricas de estrutura* (acoplamento). Elas são medidas a partir das interações entre as classes no sistema (Tabela IV.2.).

Métricas	Descrição
1. Acoplamento de mensagens	Mede a complexidade das mensagens entre as classes
2. Acoplamento de abstração de dados	Mede o número de instâncias de outras classes dentro de uma classe, ou seja, se uma classe tem uma variável local que é uma instância (objeto) de uma outra classe, este tipo de acoplamento acontece.

Tabela IV.2. - Métricas de Estruturas (Li e Henry 1993a;1993b), (Lattanzi e Henry 1994), (Li et al. 1995)

As métricas propostas por estes autores tentam determinar o relacionamento entre as métricas de software coletadas a partir do documento de projeto e código e a manutenibilidade dos sistemas orientados a objetos.

Para realizar a validação, foram coletados, durante três anos, dados relativos ao esforço de manutenção, em termos de linhas de código modificadas por classe, de dois sistemas comerciais (projeto e implementação) desenvolvidos utilizando a linguagem *Classic-Ada.* Os dois sistemas são *(UIMS - User Interface Management Systems)* e *QUES (Quality Evaluation Systems).* Através, de um análise estatística dos dados concluiu-se que as métricas propostas servem como indicadores para predizer o esforço de manutenção dos sistemas orientados a objetos.

Além disso, para dar suporte a estas experiências, a equipe desenvolveu uma ferramenta chamada de *Metrics Generator* que, através de uma interface interativa, permite aos desenvolvedores de software gerar várias métricas de software durante o desenvolvimento. Este software pode gerar métricas a partir do código fonte escrito em *Ada, C, Classic-Ada* ou *C++*.

IV.2.3. Proposta de Lorenz e Kidd (Lorenz e Kidd 1994).

Lorenz e Kidd (1994) propõem um conjunto de métricas classificadas em duas categorias: *métricas de processo* e *métricas de projeto (design)*. As primeiras são utilizadas para ajudar a predizer as necessidades do sistema em desenvolvimento (pessoal, esforço, cronograma, etc) e capturam a dinâmica do projeto em desenvolvimento. Entretanto, não medem a qualidade do produto que está sendo construído. Apresentam um alto nível de abstração, sendo menos prescritivas e mais

imprecisas, embora sejam mais importantes do que as métricas para avaliar produtos, na perspectiva do projeto como um todo. Por este motivo, são identificadas como *métricas globais* (Tabela IV.3.).

MÉTRICAS DE PROCESSO	
Critérios	**Métricas**
1. Tamanho da Aplicação	Número de cenários *scripts*
	Número de classes chave
	Número de classes de suporte
	Média de classes chave por classe de suporte
	Número de subsistemas
2. **Tamanho da Equipe**	Média homem-dias por classe
	Média de classes por desenvolvedor
3. **Cronograma**	Número das principais iterações
	Número de contratos terminados

Tabela IV.3. - Métricas de Processo (Lorenz e Kidd 1994)

Por outro lado, as métricas orientadas ao projeto *(design)* (Tabela IV.4.) são mais específicas e permitem avaliar características estáticas do projeto, caracterizando-se por serem métricas mais localizadas e de natureza mais precisa, avaliando a qualidade do ponto de vista da construção dos componentes de software durante o ciclo de vida.

A proposta é o resultado de estudos feitos em vários projetos desenvolvidos utilizando os métodos orientados a objetos de Wirfs-Brock et al. (1990), Jacobson (1992) e Lorenz (1993) e implementados na *linguagem C++* e *Smalltalk*. Os projetos estudados contêm em média 60 a 700 classes e o período de duração dos projetos foi entre seis meses a dois anos e meio, com equipes de desenvolvimento compostas de 2 a 25 pessoas.

METRICAS DE PROJETO	
Critérios	**Métricas**
1. Tamanho do Método	Número de mensagens enviadas
	Número de linhas de código
	Tamanho médio do método
2. Métodos Internos	Complexidade do método
	Parâmetros enviados nas mensagens
	Número de parâmetros
3. Tamanho da Classe	Número de métodos públicos na classe
	Número de métodos na classe
	Média de métodos por classe
	Número de variáveis de instâncias da classe
	Número de métodos de classe na classe
	Número de variáveis de classe na classe
4. Herança da Classe	Níveis de hierarquia na estrutura de classes
	Uso de herança múltipla
5. Herança de Método	Número de métodos redefinidos pela subclasse
	Número de métodos herdados por subclasse
	Número de métodos agregados por subclasse
	Índice de especializações
6. Classes Internas	Coesão da classe
	Uso global
	Média de parâmetros por método
	Uso de funções *friends*
	Percentagem de funções orientadas a código
	Número de linhas de comentários por classe/método
	Número de problemas reportados por classe ou contrato
7. Classes Externas	Acoplamento da classe
	Número de vezes que a classe é reutilizada
	Número de classes/métodos descartados
	Número de colaborações entre classes
8. Acoplamento de Subsistemas	Número de relacionamentos entre subsistemas
	Número de relacionamentos entre classes

Tabela IV.4. - Métricas de Projeto (Lorenz e Kidd 1994)

IV.2.4. Proposta de Abreu e Carapuça (Abreu e Carapuça 1994a; 1994b).

Abreu e Carapuça propõem uma estrutura taxonômica, denominada *TAPROOT* (*TA*xonomy *Pr*écis for *O*bject *O*riented me*T*rics), baseada em dois vetores

independentes: *categoria* e *nível de granularidade* (Tabela IV.5). Segundo estes autores, a idéia de trabalhar com o vetor *categoria* baseia-se no fato de se obter, assim, uma correspondência entre as propriedades do software produto e do processo de desenvolvimento. Desta forma, as métricas podem ser utilizadas efetivamente pela equipe que gerencia o processo de desenvolvimento.

Por outro lado, o vetor *nível de granularidade* foi considerado pelo fato de que, em qualquer sistema desenvolvido segundo o paradigma de orientação a objetos, existem interações entre objetos que são instâncias de um grupo de *classes* definidas, e que cada objeto se comporta de acordo com os *métodos* definidos dentro da classe. Então, estes três níveis de abstração (*sistema, classes* e *métodos*) são apropriados para representar os níveis de granularidade considerados na avaliação.

Categorias	Granularidade	Métricas
1. PROJETO	Método	Percentagem de variáveis de instância
		Densidade de comentários
	Classe	Coesão
		Profundidade da herança
		Resposta da classe
	Sistema	Média da dimensão do método
		Média do número de métodos por classe
		Média do número de variáveis de instância
2. TAMANHO	Método	Número de sentenças executáveis
		Número de operadores e operandos
		Número de variáveis de instâncias utilizadas
	Classe	Número de métodos
		Total de número de variáveis de instância
		Tamanho da interface da classe
	Sistema	Número de classes
		Total de número de métodos
		Total de número de variáveis de instância
3. COMPLEXIDADE	Método	Complexidade ciclomática (McCabe 1976)
		Métrica de Volume (Halstead 1977)
		Fluxo de Informação (Henry e Kafura 1981)
	Classe	Número de filhos
		Número de pais
		Número de descendentes
	Sistema	Comprimento total da herança
		Acoplamento entre objetos

Tabela IV.5 - Métricas Orientadas a Objetos organizadas de acordo com a Estrutura de Classificação *TAPROOT* (Abreu e Carapuça 1994a)

Categorias	Granularidade	Métricas
4. REUTILIZAÇÃO	Método	Número de vezes que o método é herdado (n1)
		Número de vezes que o método é sobrecarregado (n2)
		Eficiência na reutilização do método (n1 - n2) / n1
	Classe	Percentagem de métodos herdados sobrecarregados
		Número de vezes que a livraria de classes é reutilizada
		Número de vezes que a biblioteca de classes é reutilizada
	Sistema	Percentagem de reutilização das classes sem adaptações
		Percentagem de reutilização das classes com adaptações
		Fator de qualidade da biblioteca[4]
5. PRODUTIVIDADE	Método	Esforço para desenvolver um método
		Novos métodos desenvolvidos por unidade de esforço[5]
	Classe	Esforço para desenvolver uma classe
		Novas classes desenvolvidas por unidade de esforço
	Sistema	Média do esforço para desenvolver uma classe
		Classes reutilizadas e adaptadas por unidade de esforço
6. QUALIDADE	Método	Confiabilidade do método
		Número de defeitos por unidade de tempo
		Média do tempo para identificar e corrigir defeitos
	Classe	Confiabilidade da classe
		Média de defeitos por método
		Média de número de falhas por método
	Sistema	Efetividade dos testes
		Média do tempo entre falhas
		Média do tempo de aprendizagem em horas

Tabela IV.5. - Métricas Orientadas a Objetos organizadas de acordo com a Estrutura de Classificação *TAPROOT* (Abreu e Carapuça 1994a) (cont).

A proposta apresenta seis categorias de métricas associadas a três níveis de granularidade, onde são identificados dezoito tipos diferentes de métricas. Desta forma, é possível avaliar um sistema orientado a objetos. Algumas das métricas já foram propostas por outros autores: Chidamber e Kemerer (1991), Bieman (1992) e Karunanithi e Bieman (1992). Outras são adaptadas ou são métricas novas.

Na estrutura *TAPROOT* alguma sobreposição pode ocorrer entre as categorias propostas. Por exemplo, a distinção entre métricas de tamanho e complexidade algumas vezes não fica muito clara. No entanto, estas duas categorias também podem ser combinadas com as métricas de projeto.

[4] Calculado a partir de $\Sigma\rho = ni / p$, onde p é o total de classes na biblioteca e ni é o número de vezes que a classe i foi reutilizada.

[5] Unidade de esforço significa homem-mês, homem-ano, etc.

Assim sendo, os autores assinalam que a estrutura definida não deve ser considerada como uma proposta final e sim como um ponto de partida, sendo necessário um processo de maturação e melhoramentos através da experimentação em situações reais.

IV.2.5. Diretrizes de Qualidade Aplicadas ao Paradigma de Orientação a Objetos

Como resultado da experiência no desenvolvimento de sistemas orientados a objetos, alguns autores, entre eles, Coad e Yourdon (1992; 1993), Love (1991; 1993), Firesmith (1993), Sullo (1994) e Rumbaugh et al. (1994), sugerem *diretrizes* e *critérios de qualidade* que devem ser seguidos no desenvolvimento orientado a objetos. Além disso, assinalam a necessidade de utilizar estes critérios e diretrizes, pois tornam-se necessários, principalmente à medida em que o tamanho da aplicação vai aumentando. Portanto, o grupo desenvolvedor deve conhecer e aplicar estes critérios de avaliação para ajudar na escolha entre as diversas alternativas. Os critérios e diretrizes são os seguintes:

- Minimizar o acoplamento entre operações que pertencem a objetos diferentes;
- Maximizar a coesão entre operações que pertencem ao mesmo objeto;
- Maximizar a coesão entre objetos que pertencem à mesma estrutura hierárquica;
- Definir completa e precisamente cada operação;
- Aplicar restrições de tamanho aos objetos;
- Aplicar restrições de tamanho aos componentes do projeto;
- Os objetos não devem acessar dados definidos nos supertipos;
- Os objetos perto da raiz da árvore hierárquica não devem depender de objetos inferiores;
- Métodos com o mesmo nome, em diferentes objetos, devem significar a mesma coisa;
- Métodos com nomes diferentes devem ter significados diferentes;
- Métodos devem cumprir alguma função e não, apenas, passar o controle para outro método;
- Uma árvore hierárquica deve ter processamento especializado nas folhas;
- Um código orientado a objetos deve ter um número mínimo de IFs e CASEs; e
- Os objetos devem ser fáceis de se descrever e visualizar.

IV.3. Conclusão

O desenvolvimento de software utilizando o paradigma de orientação a objetos é algo que ainda está em *fase de amadurecimento*, necessitando de mais conhecimento, ferramentas de suporte e sobretudo de experiência sobre o próprio processo de

desenvolvimento. Métricas orientadas a objetos ainda estão na sua infância. De modo geral, as experiências relatadas na literatura não têm atingido um status de reconhecimento pela comunidade de pesquisa em orientação a objetos nem da indústria (Henderson-Sellers 1996).

Neste capítulo, foram descritas as propostas mais relevantes, presentes na literatura, de métricas aplicadas ao paradigma de orientação a objetos. Observa-se, que existe uma carência em relação ao relato de experiências de validação das métricas. Além disso, percebe-se uma falta de discussão das métricas propostas, assim como de sugestões para sua aplicabilidade no processo de desenvolvimento.

De modo geral, o conjunto de métricas pesquisados pode ser classificado em três categorias: *métricas de processo, métricas de projeto (design)* e *métricas de código*. As primeiras estão relacionadas ao controle do processo de desenvolvimento do produto. O segundo e o terceiro tipo de métricas preocupam-se, respectivamente, com a qualidade do projeto detalhado *(design)* e do produto final respectivamente.

As categorias de métricas apresentadas capturam aspectos *estáticos de projeto* relacionados às *classes* e *métodos*, misturando aspectos relativos à qualidade do código. Assim sendo, as propostas não se preocupam em avaliar aspectos relacionados a especificações orientadas a objetos que inclui modelagem em orientação a objetos. Além disso, não são consideradas métricas para avaliar aspectos semânticos, nem aspectos referentes à fidedignidade e suficiência dos produtos de software orientados a objetos.

Capítulo V

Avaliação da Qualidade de Especificações Orientadas a Objetos

"A formulação e especificação de um problema é freqüentemente mais essencial que sua solução, a qual pode ser apenas uma questão matemática ou um conhecimento experimental".

Albert Einstein e Leopold Infeld
The Evolution of Physics

V.1. Introdução

A literatura sobre qualidade de software, conforme visto no capítulo anterior, apresenta vários trabalhos que tentam identificar características de qualidade de código para produtos desenvolvidos segundo o paradigma da orientação a objetos. No entanto, o código representa um dos produtos gerados durante o desenvolvimento e, além disso, sua disponibilidade está restrita às fases finais do processo de desenvolvimento. É amplamente reconhecido pela comunidade de software que a geração de melhores produtos de software requer melhorias nas *fases iniciais do processo de desenvolvimento*. Por isso, faz-se necessária uma contínua avaliação das especificações que estão sendo geradas.

A geração de *especificações com qualidade* minimiza a necessidade de rever, modificar e refazer o código e as próprias especificações de software. Desta forma, as organizações desenvolvedoras reduzem seus tempos e custos de desenvolvimento e aumentam a qualidade do produto final.

Por este motivo, a disponibilidade de métricas nas fases iniciais do processo de desenvolvimento garante um melhor gerenciamento das fases posteriores. A *avaliação da qualidade das especificações* tem um grande impacto no processo de desenvolvimento de software pelo fato de tornar possível a execução de ações

preventivas e/ou corretivas que implicam em menores custos para o processo de desenvolvimento como um todo.

O desenvolvimento de software utilizando o paradigma de orientação a objetos surge como alternativa para a melhoria da qualidade e da produtividade no desenvolvimento de software. Embora exista literatura disponível sobre o assunto, que tenta ensinar como aplicar os métodos orientados a objetos, nenhum método trata com profundidade aspectos relativos a qualidade. A razão é simples. O desenvolvimento de produtos com o enfoque orientado a objetos ainda não possui maturidade suficiente, para dispor de medidas precisas e bem entendidas que possam ser utilizadas, para avaliar produtos e processos de desenvolvimento orientados a objetos, conforme foi discutido no Capítulo IV. Valores para medidas de atributos e formas de prevenção e correção de defeitos ainda não estão estabelecidos.

A tecnologia de orientação a objetos não garante que os sistemas desenvolvidos utilizando este paradigma de desenvolvimento sejam melhores, nem que os desenvolvedores utilizaram as facilidades disponíveis da melhor forma possível (Stiglic et al. 1995). Portanto, devemos estar conscientes de que esta tecnologia não é uma solução mágica para os problemas do desenvolvimento de software. *Métodos e linguagens*, especialmente conhecidos como *híbridos*, não garantem o uso correto dos conceitos do paradigma, embora forneçam mecanismos que permitem suportá-los e implementá-los.

No Capítulo IV, foram discutidas as propostas mais representativas sobre métricas aplicadas ao paradigma de orientação a objetos. Observa-se, neste estudo, que a maioria das métricas propostas são aplicáveis ao código orientado a objetos e baseadas nos trabalhos desenvolvidos pelo grupo do *MIT* (Chidamber e Kemerer 1991; 1993; 1994).

Neste capítulo, iniciamos com uma discussão sobre o processo de geração da especificação. Em seguida são apresentados comentários sobre o desenvolvimento de especificações orientadas a objetos, destacando-se alguns aspectos considerados relevantes, como resultado da revisão da literatura sobre desenvolvimento e qualidade em orientação a objetos e da nossa experiência de participação em um grande projeto de desenvolvimento orientado a objetos, conforme é descrito no Capítulo VI. É discutido um conjunto de características de qualidade, proposto para avaliar especificações orientadas a objetos. Para organizar a discussão dos atributos de qualidade, utilizamos o método de avaliação da qualidade de software proposto por Rocha (1983), conforme é descrito no Capítulo II. São apresentados ao final da discussão de cada objetivo de qualidade, as Tabelas V.3 e V.4, onde estão relacionados os atributos de qualidade

proposto organizados segundo *especificações em geral* e *especificações orientadas a objetos.*

V.2. O Processo de Geração/Avaliação da Especificação

A especificação descreve como o software irá ao encontro dos requisitos do problema e constitui a base sobre a qual se realiza todo o processo de desenvolvimento do software. Por isso, a geração de uma especificação completa e com qualidade é essencial para o sucesso do esforço de desenvolvimento do software. Assim sendo, o processo de geração da especificação envolve as etapas de ***levantamento de requisitos, construção, avaliação*** e ***correção da especificação*** (Figura V.1).

Na etapa de ***levantamento de requisitos*** é realizado o processo de *elicitação de requisitos.* Este processo tem por objetivo tornar explícitos os desejos, as intenções e necessidades dos usuários em relação ao software que vai ser construído, ou seja, capturar os *requisitos do software.* Segundo Leite (1990) este processo compreende os subprocessos de coleta, validação e comunicação. A coleta procura identificar o conhecimento sobre o problema em questão. A validação é a forma com que se confirma ou não a precisão e efetividade do processo de coleta. O subprocesso de comunicação permeia a coleta e a validação, já que é fundamental o diálogo entre os vários atores presentes na elicitação.

O processo de *elicitação* busca *conhecer o comportamento de sistemas existentes,* o *conhecimento da parte específica do domínio da aplicação,* que está relacionado com o sistema de software a ser implementado, e os *objetivos e limitações específicas dos usuários.* Este processo de elicitação é geralmente realizado através de entrevistas, revisão de documentos que descrevem os objetivos e desejos da organização e revisão de relatos da experiência no desenvolvimento, caso existam. Por ser esta uma atividade de constante aprendizado, o engenheiro de requisitos precisa se envolver com o trabalho do usuário, conhecer seu ambiente, observar, aprender e questionar.

Na etapa de ***construção da especificação*** é realizado o processo de *modelagem,* que é responsável pela formalização do conhecimento obtido na elicitação através de uma correta representação, organização e documentação, utilizando um método de desenvolvimento específico e obedecendo aos princípios e conceitos do paradigma adotado. Nesta etapa são produzidos modelos conceituais, que descrevem estaticamente e dinamicamente aspectos sobre o problema. São modeladas as propriedades do domínio da aplicação de interesse do usuário e ignorados os detalhes e relações importantes para outros propósitos ou contextos.

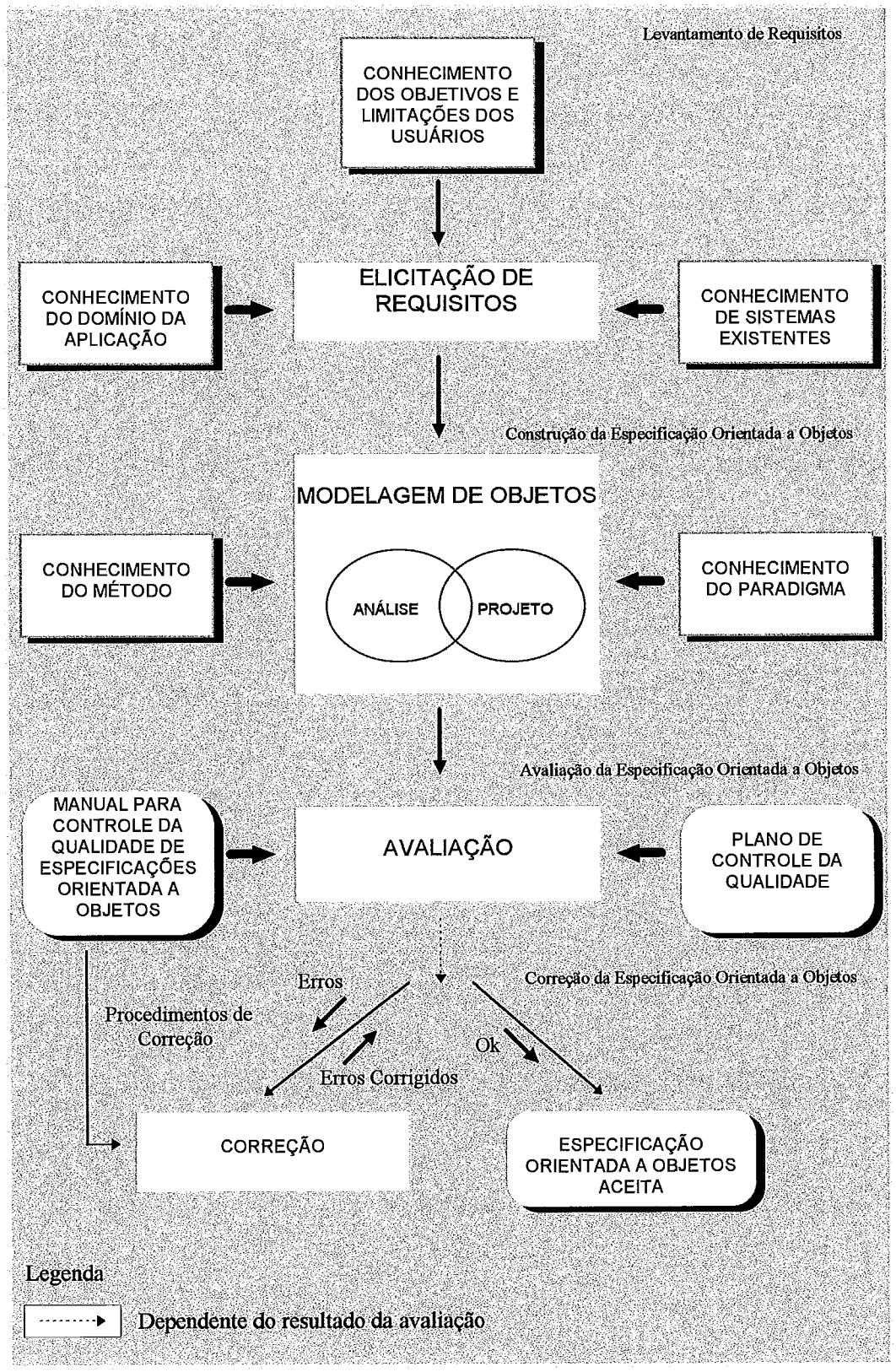


Figura V.1. - Processo de Geração /Avaliação de Especificações Orientadas a Objetos

Na etapa de ***avaliação da especificação*** é utilizado o *Plano de Controle da Qualidade*. Este plano contém a definição dos *requisitos de qualidade do produto* que se pretende desenvolver, que variam de acordo com o domínio da aplicação, projeto específico e da tecnologia a ser utilizada. A partir do processo de desenvolvimento adotado para o projeto, são definidos para cada etapa do processo, os marcos e pontos de controle, os atributos de qualidade a serem avaliados, o pessoal envolvido e as técnicas de avaliação a serem utilizadas.

Nesta etapa é utilizado, também, o *Manual para o Controle da Qualidade de Especificações Orientadas a Objetos.* Este manual atua como um instrumento guia que garante que o processo de avaliação da qualidade seja feito de uma forma organizada e disciplinada, atendendo aos marcos e pontos de controle definidos previamente no *Plano de Controle da Qualidade*. Este manual é organizado de acordo com o Método de Avaliação da Qualidade escolhido.

Na etapa de ***correção da especificação,*** são realizadas as modificações necessárias para corrigir os erros identificados na ***avaliação.*** Para isso podem ser utilizados os procedimentos de correção, definidos no *Manual para o Controle da Qualidade de Especificações* que indicam as medidas a serem seguidas para a correção de problemas específicos.

V.2.1. Desenvolvimento de Especificações Orientadas a Objetos

Os métodos de desenvolvimento orientado a objetos estabelecem um conjunto de atividades a serem realizadas nas fases de análise e projeto. No entanto, a *construção da especificação do sistema* acontece simultaneamente em diferentes dimensões. Ou seja, a *especificação* é *desenvolvida iterativamente.* Primeiro constrói-se um subconjunto do modelo, que depois é ampliado até que todo o problema seja compreendido. Conseqüentemente, o processo de desenvolvimento de software orientado a objetos é um processo *iterativo/recursivo/incremental,* não sendo claras as fronteiras entre as fases do processo de desenvolvimento.

Por outro lado, como resultado desta forma de desenvolvimento e pela falta de uma demarcação das fases de análise e projeto, são gerados produtos ao longo do desenvolvimento - *modelos de classes-objetos* - com diferentes *níveis de abstração.* Entretanto, é necessária uma padronização de atividades[1] a serem realizadas nas fases de

[1] As atividades realizadas durante o processo de desenvolvimento orientado a objetos são: identificação de classe-objetos, identificação de atributos, identificação de serviços, definição de herança, definição de agregação, identificação da colaboração entre objetos, definição de agrupamento de classe-objetos.

análise e projeto, mesmo que cada método de desenvolvimento orientado a objetos defina-as e execute-as de maneira particular.

Na realidade, algumas diferenças podem ser observadas na geração da especificação do sistema, se *objetivos específicos* são definidos para cada fase de desenvolvimento. Por exemplo, no método de Coad e Yourdon (1992; 1993), na fase de análise é gerada uma primeira aproximação do modelo de classes-objetos e a preocupação está em transmitir adequadamente ao usuário suas necessidades, através da modelagem adequada de um conjunto de classes-objetos. Isto é, ao iniciarmos o processo de desenvolvimento de software, estamos preocupados com a capacidade de comunicação do modelo para refletir adequadamente o problema.

Por outro lado, na fase de projeto, são realizadas as mesmas atividades da fase anterior. No entanto, nessa fase a preocupação está em gerar uma arquitetura adequada que viabilize a sua implementação e facilite o processo de *manutenção, evolução, testes* e *garanta possibilidades de reutilização.*

Assim sendo, vários aspectos podem ser comentados com relação ao desenvolvimento de uma especificação orientada a objetos e às fases de análise e projeto, no que se refere aos *níveis de abstração das classes-objetos, hierarquia de classes-objetos, instâncias das classes-objetos, nível de detalhe dos serviços, atributos nas classes-objeto e ao agrupamento das classes-objetos.* Estes comentários são os seguintes:

- **Com relação aos Níveis de Abstração das Classes-Objetos**

A identificação e documentação das principais *classes-objetos do domínio*[2] da aplicação são realizadas através da revisão e discussão do documento de descrição dos requisitos do sistema e da interação constante com os especialistas do domínio. Os critérios de seleção destas classes-objetos, durante a fase de análise, devem permitir identificar abstrações que representam entidades do mundo real no contexto do domínio do problema.

Por outro lado, as classes-objetos de projeto, de modo geral, são abstrações que representam classes-objetos de implementação. Por isso, podemos afirmar que o *nível de abstração* do modelo de análise é maior e é *independente de decisões de implementação.* Ou seja, o modelo gerado na fase de análise se caracteriza por

[2] Classe-objetos do domínio representam classe-objetos que são conceitos do domínio da aplicação. Os conceitos de implementação computacional são representados através de abstrações de classe-objetos tipo janelas, relatórios, gerenciadores ou banco de dados.

identificar um conjunto de classes-objetos que representam *conceitos do domínio da aplicação* e não *conceitos de implementação computacional*. Isto é, os modelos de análise e de projeto em orientação a objetos representam produtos diferentes, sendo cada um deles construído de acordo com *objetivos específicos* e observando um determinado nível de abstração.

- **Com relação às Hierarquias de Classes-Objetos**

Na fase de análise, são identificados os relacionamentos do tipo generalização - especialização com o objetivo de facilitar o entendimento do modelo, observando as semelhanças e diferenças entre as classes-objetos e formando uma hierarquia. No entanto, na fase de projeto, estes relacionamentos - *que inclui herança simples e múltipla* - são tratados considerando a sua influência sobre as decisões de implementação, definição de testes, possíveis problemas de manutenção e possibilidades de reutilização.

- **Com relação às Instâncias das Classes-Objetos**

Quando especificamos uma classe-objetos no modelo de análise, assumimos de maneira natural que a classe-objetos possuirá múltiplas instâncias. No entanto, na fase de projeto, é crítico ressaltar e especificar condições para criar e destruir instâncias de uma classe-objetos. Conseqüentemente, é necessário detalhar algoritmos para a criação e destruição de classes-objetos, bem como serviços para o tratamento das associações e desassociações dos relacionamentos modelados.

- **Com relação ao Nível de Detalhe dos Serviços**

Na especificação do modelo de análise é, comumente, aceito não especificar em detalhes os serviços nas classes-objetos. O que interessa, nesse momento, é entender o que a classe-objetos faz no contexto do domínio do problema e não investir tempo em detalhes desnecessários relativos à especificação detalhada de seus serviços. Os serviços de projeto, no entanto, devem ser especificados em detalhes, utilizando uma *linguagem de projeto* ou uma linguagem próxima à própria linguagem de programação que será utilizada na implementação do sistema.

- **Com relação aos Atributos nas Classes-Objetos**

Na fase de análise, estamos preocupados em representar, da melhor forma possível, através de um subconjunto de classes-objetos, o domínio da aplicação. Os atributos identificados nessa etapa, geralmente, estão incompletos e não representam propriedades da classe-objetos relacionadas à implementação. Por outro lado, na fase de projeto, atributos são acrescentados para caracterizar melhor as classes-objetos e implementar os relacionamentos entre elas.

- **Com relação ao Agrupamento de Classes-Objetos**

O agrupamento das classes-objetos, na fase de análise, é feita para melhorar o entendimento do modelo, oferecendo uma organização de classes-objetos numa perspectiva superior, mostrando claramente sub-domínios da aplicação. Na fase de projeto, são representadas as classes-objetos de implementação, acrescentado o nível de detalhe dos serviços, e especificados os atributos de implementação. Por fim, o modelo cresce em tamanho e quantidade de informações, permitindo fazer agrupamentos de classes-objetos de acordo com diferentes critérios. Por exemplo, as classes-objetos - *domínio e implementação* - podem ser agrupadas atendendo às funções ou sub-funções específicas que o sistema realiza. Neste momento, estamos trabalhando com a *arquitetura* do sistema e, conseqüentemente, um dos objetivos a ser alcançado pode ser um adequado nível de desempenho.

V.3. Características de Qualidade de Especificações Orientadas a Objetos

O processo de desenvolvimento de software orientado a objetos apresenta sinais evidentes de ainda não ter atingido a sua maturidade. Os métodos descritos na literatura propõem o desenvolvimento de software utilizando estratégias, notações, simbologias e terminologias próprias, gerando produtos com diferentes níveis de abstração e modelando os sistemas com variados níveis de riqueza sintática e semântica. O objetivo da fase de análise, conforme dito anteriormente, é especificar o domínio do problema, sem introduzir restrições para uma determinada implementação. Conseqüentemente, durante a fase de projeto são realizadas refinamentos, o que exige uma continua avaliação das especificações que são produzidas.

O processo de desenvolvimento de software deve realizar-se de uma forma controlada. Para que isto aconteça, é necessário que o controle da qualidade do produto se dê ao longo de todo o ciclo de vida. Deve, portanto, ter início com a avaliação das especificações do produto, cuja qualidade tem forte impacto na qualidade do produto final. Se isto não é feito, os erros se propagarão ao longo de todo o ciclo de desenvolvimento e os custos de correção aumentarão drasticamente a medida que o desenvolvimento evolui.

Conforme visto no Capítulo IV, a literatura especializada sobre qualidade de software não apresenta trabalhos relacionados à *qualidade de especificações orientadas a objetos,* mas apenas sobre a qualidade referente a *projeto detalhado e código.* É necessário, portanto, identificar atributos de qualidade que permitam realizar avaliações de especificações. Estes atributos podem ser organizados e discutidos considerando características gerais de qualquer especificação e características particulares de

especificações orientadas a objetos. Somente de posse destes atributos será possível comprovar, através de avaliações, que o software especificado está sendo construído corretamente de acordo com o paradigma da orientação a objetos, com o método de desenvolvimento utilizado e que satisfaz as necessidades dos usuários.

Nesta seção definimos um conjunto de atributos de qualidade que devem ser considerados para avaliar especificações de modo geral e para o caso de especificações orientadas a objetos, no que se refere a *modelos de classes-objetos* e *sua documentação associada.* Estes atributos de qualidade foram definidos a partir de trabalhos sobre qualidade de especificações descritos em Clunie (1987), Clunie e Rocha (1987a; 1987b) e Davis (1993), da discussão sobre os princípios e conceitos do paradigma orientado a objetos, descritos na seção III. 2, da literatura técnica sobre qualidade em orientação a objetos Tegarden et al. (1992), Coppick e Thomas (1992), Korson e McGregor (1992), Sharble e Cohen (1993), Kolewe (1993), Abreu e Carapuça (1994a; 1994b), Basili et al. (1995; 1996), Chidamber e Kemerer (1991; 1993; 1994), Chen e Lu (1993), Stiglic et al. (1995), Coad e Yourdon (1993), Yourdon (1994), Love (1991), Li e Henry (1993a; 1993b), Binder (1994), Lattanzi e Henry (1994), Lorenz e Kidd (1994), Churcher e Sheppard (1995: 1995a), Wang e Wang (1995), Li et al. (1995), McGregor (1995), Karlsson (1995), Martin C. (1995), Archer e Stinson (1995), Henderson - Sellers (1995; 1996) e Montazeri e Hitz (1996), dos métodos de desenvolvimento de software orientado a objetos e da experiência de participação em um projeto de desenvolvimento orientado a objetos (Rocha et al. 1994), (Clunie et al. 1995b, 1995c). A organização da discussão destes atributos é realizada de acordo com o modelo proposto por Rocha (1983). Resultados iniciais estão publicados em Clunie et al. (1994; 1994a; 1994b; 1994c; 1995; 1995a; 1996, 1996a, 1996b, 1997).

Segundo Rocha (1987), produtos de software são desenvolvidos para atenderem a determinadas necessidades de seus usuários. Após serem colocados em operação, espera-se que tenham uma vida útil, longa e produtiva. Para que isto se concretize, devem ser atingidos os seguintes objetivos de qualidade: ***Confiabilidade da Representação, Confiabilidade Conceitual*** e ***Utilizabilidade.***

V.3.1. Objetivo: Confiabilidade da Representação

O objetivo ***Confiabilidade da Representação*** refere-se às características que tornam a especificação confiável para seus usuários, considerando-se aspectos relativos à sua forma e que tornam possível a sua compreensão e manipulação, tendo em conta que a especificação evolui gradativamente ao longo do desenvolvimento e deve ser modificada durante a vida útil do produto ao serem realizadas manutenções.

Este objetivo se realiza através dos fatores de qualidade ***Comunicabilidade*** e ***Manipulabilidade.***

V.3.1.1. Fator: Comunicabilidade - *conjunto de atributos de qualidade que avaliam a capacidade da especificação de comunicar o seu conteúdo.*

Especificações são documentos lidos por diferentes pessoas durante o desenvolvimento e a vida operacional do software produto. Esses usuários podem ser pessoas que não participaram no desenvolvimento e muitas vezes não têm possibilidade de contato com os desenvolvedores para esclarecimentos. Além disso, para que seja possível avaliar se a especificação reflete o que o usuário deseja para o produto, é necessário que esta seja facilmente compreendida. A comunicabilidade torna-se, então, uma característica fundamental que viabiliza as atividades de avaliação, implementação, manutenção e reutilização.

Especificações orientadas a objetos incluem modelos de classes-objetos, onde são representadas e documentadas as entidades que caracterizam o domínio da aplicação e resolvem o problema. Estes modelos de classes-objetos servem a diversos objetivos: proporcionam uma base de concordância entre cliente e desenvolvedor, tornam-se a base para a implementação e permitem avaliar se os requisitos descritos estão sendo implementados.

Para atingir estes objetivos é necessário que as especificações sejam geradas de acordo com as regras de construção fornecidas pelo método e os padrões definidos pela organização, obedeçam aos princípios e conceitos do paradigma da orientação a objetos e estejam expressas utilizando um vocabulário pertencente ao domínio da aplicação a que se destina o software. O fator comunicabilidade é avaliado através dos subfatores ***Correção no uso do Método, Uniformidade de Terminologia, Uniformidade no Nível de Abstração, Modularidade da Documentação, Correção da Arquitetura, Concisão*** e ***Simplicidade.*** A seguir discutimos em detalhes cada um destes subfatores, destacando e dando maior ênfase aos aspectos relacionados diretamente a especificações orientadas a objetos. Atributos de qualidade de especificações em geral, sem considerar o método utilizado em seu desenvolvimento, já foram objetos de trabalho anterior (Clunie 1987) e por isso não serão detalhados neste trabalho.

Subfator: Correção no Uso do Método - *conjunto de atributos de qualidade que avaliam a correção de uma especificação do ponto de vista do uso do método de desenvolvimento.*

A comunicabilidade de uma especificação depende da correção no uso do método escolhido para a sua construção. Esta correção deve ser avaliada no que se refere à notação, sintaxe, semântica e formato de documentação caso este esteja definido no método. Assim sendo, independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devem ser considerados os critérios ***Correção da Notação, Correção Sintática, Correção Semântica*** e ***Correção no Uso do Formato de Documentação.*** A definição destes critérios está na Tabela V.3. Uma discussão detalhada pode ser encontrada em Clunie (1987).

OBJETIVO: CONFIABILIDADE DA REPRESENTAÇÃO	
COMUNICABILIDADE	Conjunto de atributos de qualidade que avaliam a capacidade da especificação de comunicar seu conteúdo.
Correção no Uso do Método	Conjunto de atributos de qualidade que avaliam a correção da especificação do ponto de vista do uso do método de desenvolvimento.
Correção da Notação	Característica que avalia se a notação do método foi usada de forma correta.
Correção Sintática	Característica que avalia se o conjunto de regras sintáticas definidas no método foi usado de forma correta.
Correção Semântica	Característica que avalia se o conjunto de regras semânticas definidas no método foi usado de forma correta.
Correção no Uso do Formato de Documentação	Característica que avalia se o formato de documentação definido no método foi usado de forma correta.
Uniformidade de Terminologia	Conjunto de atributos de qualidade que avaliam se a especificação foi escrita com uniformidade de termos e notação.
Uniformidade de Termos	Característica que avalia se os termos técnicos foram utilizados de forma uniforme ao longo do texto e de acordo a definições pré-estabelecidas.
Uniformidade de Notação	Característica que avalia se a notação foi utilizada de forma uniforme ao longo do texto.
Uniformidade no Nível de Abstração	Conjunto de atributos de qualidade que avaliam se a especificação possui um nível de detalhe uniforme, considerando-se um determinado estágio do desenvolvimento.
Uniformidade de Detalhes da Documentação	Característica que avalia se todos os aspectos estão descritos na especificação com o mesmo nível de detalhamento, considerando-se um determinado estágio de desenvolvimento.
Independência de Restrições de Projeto	Característica que avalia na especificação de requisitos a existência de restrições com relação à escolha de alternativas de solução próprias da fase de projeto.
Uniformidade de Abstração das Classes-Objetos	Característica que avalia o nível de abstração de cada classe-objetos, considerando-se um determinado estágio de desenvolvimento.
Uniformidade de Abstração dos Atributos	Característica que avalia o nível de abstração dos atributos em cada classe-objetos, considerando-se um determinado estágio de desenvolvimento.
Uniformidade de Abstração dos Serviços	Característica que avalia o nível de abstração dos serviços em cada classe-objetos, considerando-se um determinado estágio de desenvolvimento.
Uniformidade de Abstração dos *Clusters*	Característica que avalia o nível de abstração dos componentes modelados nos *clusters,* considerando-se um determinado estágio de desenvolvimento.

Tabela V.3. - Características de Qualidade Relacionadas ao Objetivo Confiabilidade da Representação

OBJETIVO: CONFIABILIDADE DA REPRESENTAÇÃO	
Modularidade da Documentação	Conjunto de atributos de qualidade que avaliam se as partes da especificação podem ser entendidas e modificadas de forma independente.
Coesão de Informações	Característica que avalia o grau de associação das informações de um capítulo e, dentro deste, seções.
Acoplamento entre as Seções	Característica que avalia o grau de interdependência entre os módulos - *capítulos ou seções* - .
Estrutura da Documentação	Característica que avalia se a estrutura, composta de capítulos e, dentre destes, seções, está organizada numa seqüência lógica.
Correção da Arquitetura	Conjunto de atributos de qualidade que avaliam a correção da modelagem considerando a disposição, composição e relacionamento de seus componentes.
Nível de Fatoração das Classes-Objetos	Característica que avalia a quantidade de descendentes imediatos de uma classe-objetos.
Nível de Profundidade da Hierarquia de Classes-objetos	Característica que avalia a quantidade média de níveis em uma hierarquia de classes-objetos.
Coesão Estrutural das Classes-Objetos	Característica que avalia o grau de relacionamento dos serviços especificados nas classes-objetos.
Coesão Estrutural dos *Clusters*	Característica que avalia o grau de relacionamento entre as classes-objetos de um determinado *cluster*.
Acoplamento de Relacionamento das Classes-Objetos	Característica que avalia o grau de dependência estática das classes-objetos.
Acoplamento de Relacionamento dos *Clusters*	Característica que avalia o grau de dependência estática dos *clusters*.
Acoplamento de Interação das Classes-objetos	Característica que avalia o grau de comunicação via mensagens das classes-objetos.
Acoplamento de Interação dos *Clusters*	Característica que avalia o grau de comunicação via mensagens dos *clusters*.
Concisão	Conjunto de atributos de qualidade que avaliam se a especificação contém um volume mínimo de texto por ter-se maximizado o volume de informação por unidade de texto.
Complementabilidade	Característica que avalia se a especificação faz uso de documentos auxiliares - *referências, glossários, dicionários de dados* - que facilitam seu entendimento.
Tamanho da Classe-Objetos	Característica que avalia a quantidade de atributos e serviços definidos na classe-objetos.
Tamanho do Serviço	Característica que avalia a quantidade de linhas de pseudocódigo definidos no serviço.
Tamanho do *Cluster*	Característica que avalia a quantidade de classes-objetos definidas num *cluster*.
Tamanho das Interfaces	Característica que avalia a quantidade de argumentos definidos nas mensagens.

Tabela V.3. - Características de Qualidade Relacionadas ao Objetivo Confiabilidade da Representação (cont.)

OBJETIVO: CONFIABILIDADE DA REPRESENTAÇÃO		
Simplicidade		Conjunto de atributos de qualidade que avaliam em cada classe-objetos o grau de simplicidade dos seus serviços e atributos.
	Simplicidade dos Serviços	Característica que avalia o grau de simplicidade dos serviços na classe-objetos.
	Simplicidade dos Atributos	Característica que avalia o grau de simplicidade dos atributos na classe-objetos.
Conformidade		Conjunto de atributos de qualidade que avaliam se a especificação foi gerada considerando as normas estabelecidas para o desenvolvimento do produto.
	Aderência a Normas da Organização Desenvolvedora	Característica que avalia se a especificação foi gerada obedecendo as normas estabelecidas pela organização desenvolvedora.
	Aderência a Normas estabelecidas pelo Contratante	Característica que avalia se a especificação foi gerada considerando as normas estabelecidas pelo contratante.
MANIPULABILIDADE		Conjunto de atributos de qualidade que avaliam a facilidade de manipulação da especificação para diversas formas de uso.
Disponiblidade		Conjunto de atributos de qualidade que avaliam a facilidade de acesso de uma especificação para seus usuários autorizados, na sua versão mais atualizada.
	Acessibilidade	Característica que avalia se qualquer usuário autorizado pode facilmente consultar a especificação e/ou obter uma cópia da mesma.
	Estar Atualizada	Característica que avalia se a especificação reflete a informação mais recente.
Rastreabilidade		Conjunto de atributos de qualidade que avaliam a facilidade de se percorrer as especificações de requisitos e de projeto, identificando a agregação de detalhes a um determinado aspecto, desde sua visão mais global até a mais detalhada e vice-versa.
	Organização da Documentação	Característica que avalia se o conjunto de especificações está organizado, de forma a facilitar sua manipulação.
	Localizabilidade Interna	Característica que avalia se existem facilidades para se localizar todos os elementos, dentro de uma especificação, relacionados com determinado aspecto ou assunto.
	Localizabilidade Externa	Característica que avalia se existem facilidades para localizar todas as especificações e demais documentos relacionados a um determinado aspecto ou assunto.

Legenda

Fatores de Qualidade — Critérios de Qualidade para Especificações em Geral

SubFatores de Qualidade — Critérios de Qualidade para Especificações O.O.

Tabela V.3. - Características de Qualidade Relacionadas ao Objetivo Confiabilidade da Representação (cont.)

Subfator: Uniformidade de Terminologia - *conjunto de atributos de qualidade que avaliam se a especificação foi escrita com uniformidade de termos e notação.*

A especificação constitui o documento base em torno do qual deve se apoiar todo o processo de desenvolvimento e manutenção do software. Para que possa ser facilmente comprendida, por seus diferentes leitores, devem ser evitadas nomenclaturas despadronizadas que geram incompreensões e ilegibilidade. Torna-se, portanto, necessária a padronização da terminologia utilizada, principalmente no que se refere a termos técnicos. Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator devem ser, sempre, considerados os critérios ***Uniformidade de Termos*** e ***Uniformidade de Notação.*** A definição destes critérios está na Tabela V.3. Uma discussão detalhada pode ser encontrada em Clunie (1987).

Subfator: Uniformidade no Nível de Abstração - *conjunto de atributos que avaliam se a especificação possui um nível de detalhe uniforme, considerando-se um determinado estágio do desenvolvimento.*

A construção de um sistema por refinamentos sucessivos implica que durante o ciclo de desenvolvimento, a cada passo, tenhamos um detalhamento maior do sistema do que no passo anterior. Desta forma visualiza-se o todo e, a partir daí, vai-se detalhando as partes, particularizando cada vez mais. Entretanto, é importante ter-se o controle sobre a uniformidade do nível de abstração introduzido neste processo. Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devem ser considerados os critérios ***Uniformidade de Detalhes da Documentação*** e ***Independência de Restrições de Projeto.*** A definição destes critérios está na Tabela V.3. Uma discussão detalhada pode ser, também, encontrada em Clunie (1987).

No caso de especificações orientadas a objetos, a característica ***Uniformidade no Nível de Abstração*** é, particularmente, importante pelo fato do desenvolvimento orientado a objetos ser um processo recursivo/iterativo/incremental, o que implica na existência de diferentes níveis de abstração, ao longo do desenvolvimento, para classes-objetos, atributos, serviços e *clusters*[3]. A avaliação de uma especificação orientada a objetos, no que se refere a este subfator, deve considerar, também, os critérios ***Uniformidade de Abstração das Classes-objetos, Uniformidade de Abstração dos Atributos, Uniformidade de Abstração dos Serviços*** e ***Uniformidade de Abstração dos***

[3]Um *cluster* deve ser entendido como o conjunto de classes-objetos relacionadas para atingir um objetivo específico.

Clusters. A definição destes critérios está na Tabela V.3. A seguir apresentamos uma discussão sobre os mesmos.

- **Uniformidade de Abstração das Classes-Objetos**

Um bom modelo de classes-objetos incorpora, os aspectos fundamentais de um problema e omite os demais porque, se este contém detalhes em excesso, limita desnecessariamente a escolha de decisões próprias de outras fases, desviando a atenção dos problemas reais. Por exemplo, na modelagem relativa à etapa de análise, de modo geral, as classes-objetos representam entidades conceituais do domínio da aplicação e não entidades de implementação, como classes de interfaces, base de dados, relatórios, etc. Caso contrário, isto pode introduzir restrições com relação à escolha de alternativas de solução próprias das fases de projeto e/ou implementação.

Classes-objetos que descrevem de maneira precisa e uniforme o domínio do problema e as responsabilidades do sistema dentro desse domínio são mais estáveis. Por maior que seja o sistema, sua essência - *classes-objetos do domínio do problema, no escopo das responsabilidades do sistema* - permanecerá através de quaisquer alterações potencialmente importantes. Desta forma, o nível de abstração adequado das classes-objetos, considerando-se o estágio de desenvolvimento, possibilita que os componentes do modelo de análise ou de projeto possam ser reutilizados em outras aplicações.

- **Uniformidade de Abstração dos Atributos**

A uniformidade de abstração dos atributos de uma determinada classe-objetos permite caracterizá-la dentro do contexto do estágio do processo de desenvolvimento em questão. A não uniformidade de abstração dos atributos pode trazer problemas de entendimento da classe-objetos. Por exemplo, na fase de análise, o objetivo é a compreensão da classe-objetos e seus relacionamentos entre classes-objetos no nível conceitual. Portanto, não devem ser modelados atributos de implementação, pois este tipo de tratamento será considerado nas fases posteriores. Isto é, classes-objetos com atributos que representam diferentes níveis de abstração são difíceis de serem reutilizados.

- **Uniformidade de Abstração dos Serviços**

Um serviço é uma função ou transformação que pode ser aplicada a objetos ou, por estes, a uma classe. Os serviços descritos na fase de análise apresentam, de modo geral, um nível de detalhe menor do que os descritos na fase de projeto. Na fase de projeto são especificados ou detalhados os serviços relacionados à própria implementação da classe-objetos, situação esta que não acontece na fase de análise. O importante, na fase de análise, é atingir o entendimento do modelo, sem detalhes

desnecessários de projeto e implementação. Portanto, os serviços especificados na classe-objetos devem ter o nível de abstração correspondente ao estágio em questão.

- **Uniformidade de Abstração dos *Clusters***

No caso de sistemas muito grandes, compostos por centenas de classes-objetos, o desenvolvimento do software pode ser realizado por uma equipe de desenvolvimento igualmente grande. Esta equipe é subdividida em equipes menores responsáveis por determinada parte do sistema. Estas partes, compostas por um conjunto de classes-objetos - *cluster* -, devem manter o mesmo nível de abstração, que corresponda à atividade do desenvolvimento que se está realizando. Caso contrário, tornam-se mais difíceis de serem entendidas e de serem feitas modificações e/ou evoluções, pois o grupo encarregado pelo seu desenvolvimento necessita lidar com diferentes graus de abstração. Além disso, torna-se difícil a comunicação entre os diferentes membros da equipe que trabalham com classes-objetos que estão relacionadas mas que pertencem a *clusters* diferentes.

Subfator: Modularidade da Documentação - *conjunto de atributos de qualidade que avaliam se as partes de uma especificação podem ser entendidas e modificadas de forma independente.*

Especificações não modulares são difíceis de entender e modificar, o que além de dificultar a comunicação, muitas vezes, produz problemas de inconsistência. A modularidade é, portanto, um atributo de qualidade da representação que facilita o entendimento dos diversos aspectos da especificação de forma independente, isto é, sem se ter a necessidade de ler toda a especificação. A documentação tem uma organização modular se seus componentes são escritos em uma disposição, formada por capítulos e, dentro deste, seções. Cada capítulo ou seção está composto por elementos de informação. Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devem ser sempre considerados os critérios ***Coesão de Informações, Acoplamento entre Seções*** e ***Estrutura da Documentação.*** A definição destes critérios está na Tabela V.3. Uma descrição destes critérios pode ser encontrada em Clunie (1987).

Subfator: Correção da Arquitetura - *conjunto de atributos de qualidade que avaliam a correção da modelagem considerando a disposição, composição e relacionamentos de seus componentes.*

Um dos objetivos do processo de modelagem é fornecer uma adequada disposição, composição e relacionamentos entre componentes que formam o sistema, com o intuito de ter-se um modelo de classes-objetos de fácil entendimento, avaliação,

implementação, manutenção e com possibilidades de reutilização em outros contextos. Para que isto seja atingido, é necessária uma avaliação dos componentes do modelo segundo uma perspectiva local *(i.e a nível de classes-objetos)* e uma perspectiva de conjunto *(i.e a nível de clusters)*. Assim sendo, para avaliarmos segundo este subfator, devem ser considerados os seguintes critérios: ***Nível de Fatoração das Classes-objetos, Nível de Profundidade da Hierarquia de Classes-Objetos, Coesão Estrutural das Classes-objetos, Coesão Estrutural dos Cluster, Acoplamento de Relacionamento das Classes-objetos, Acoplamento de Interação das Classes-objetos, Acoplamento de Relacionamento dos Clusters*** e ***Acoplamento de Interação dos Clusters***. A definição de cada um destes critérios está na Tabela V.3. A seguir apresentamos uma discussão sobre os mesmos.

- **Nível de Fatoração das Classes-Objetos**

A estrutura hierárquica de classes-objetos indica em que medida as especializações herdam os atributos e serviços de sua classe ancestral. Desta forma, o número de especializações associadas a uma classe-objetos é um indicador do potencial de influência que determinada classe-objetos tem sobre seus descendentes. Assim sendo, o nível de fatoração da classe-objetos (***NFC***) é entendido como a quantidade de subclasses imediatas de uma classe ancestral C_i . O valor pode ser obtido da seguinte maneira:

NFC (C_i) = *Número de classes-objetos imediatas da classe* C_i

Se a classe-objetos apresenta uma grande quantidade de descendentes imediatos, é uma indicação de quantas classes de especializações vão herdar os atributos e serviços da classe ancestral. Embora isto seja considerado positivo, sob a ótica de reutilização - *sendo a herança vista como uma forma de reutilização* -, uma classe-objetos com muitos descendentes poderá requerer um maior esforço durante o processo de testes dos serviços das classes-objetos envolvidas, porque os serviços da classe-objetos ancestral precisam ser novamente testados nas classes-objetos descendentes (Binder 1994). Além disso, isto pode indicar uma modelagem inadequada das especializações ou uma classe ancestral inapropriada.

Segundo Johnson e Foote (1988), uma forma de diminuir a quantidade de especializações é através da criação de novos níveis de hierarquia, através da identificação de mensagens e serviços comuns, migrando-os para uma nova superclasse. Isto pode criar a necessidade de decompor serviços em serviços menores e dividi-los entre superclasses e subclasses. Além disso, devem ser eliminados os serviços da superclasse que são freqüentemente redefinidos em vez de diretamente utilizados por suas subclasses. Isto torna a superclasse mais abstrata e conseqüentemente mais útil sobre o ponto de vista de reutilização.

O fato de uma classe-objetos ter uma quantidade adequada de descendentes imediatos, depende das características da classe-objetos no contexto do domínio da aplicação. No entanto, para facilitar o entendimento da classe-objetos e minimizar erros, Firesmith (1993;1994) recomenda que seja considerada a proposta de Miller (1956). Isto é, cada classe-objetos deve ter em média não mais do que 7 ± 2 descendentes imediatos.

Segundo Taylor (1993), um bom projeto orientado a objetos se caracteriza por ter subclasses que, de modo geral, são intercambiáveis com suas superclasses. As subclasses não devem ser modeladas com a única intenção de simplificar ou reutilizar código. Embora simplicidade e reutilizabilidade sejam características importantes, na modelagem, deve-se assegurar que cada subclasse seja um caso especial na hierarquia.

- **Nível de Profundidade da Hierarquia de Classes-Objetos**

Uma estrutura generalização-especialização representa uma hierarquia de classes-objetos formada por vários níveis. Através de uma análise desta hierarquia, é possível identificar em que medida as classes ancestrais afetam as demais classes, visualizando-se a profundidade das classes na hierarquia e o número de serviços herdados. Estas informações são indicadores do grau de complexidade da estrutura (Chidamber e Kemerer 1994). O nível de profundidade da hierarquia ***NPH*** é obtido da seguinte maneira (Chidamber e Kemerer 1994), (Henderson-Sellers 1996):

Sejam $C_1, \ldots, C_n$ as classes-objetos e $NPC(C_i)$ o nível da classe-objetos na hierarquia, o nível de profundidade ***NPH*** é dada por:

$$NPH = \frac{\sum_{i=1}^{n} NPC(C_i)}{n}$$

O aumento dos níveis de profundidade da hierarquia implica em um maior esforço para o entendimento e manutenção das classes-objetos. Isto se deve ao fato de que a herança afeta a localizabilidade e o princípio de encapsulamento. Ao invés de localizar todas as características da classe-objetos na própria classe-objetos e encapsular todos os seus dados, estas declarações aparecem dispersas ao longo de toda a hierarquia. No caso de *herança múltipla,* estes problemas tendem a se agravar ainda mais.

Os testes das classes-objetos também são afetados, pois ao se realizar os testes dos serviços nas classes-objetos ancestrais há a necessidade de testá-los nas classes-objetos descendentes. As razões que justificam esta preocupação são discutidas em Binder (1994) e Perry e Kaiser (1990), citados por Queiroz e Ribeiro (1995):

a) ***Redefinições de métodos,*** isto é, uma classe-objetos descendente modifica os serviços herdados em função de suas necessidades específicas. Neste caso, um novo conjunto de testes deve ser aplicado.

b) ***Acréscimo de novas classes-objetos descendentes***, pode demandar o teste de todos os serviços herdados, já que se trata de um novo contexto para execução dos serviços.

Outra dificuldade advinda de se ter uma hierarquia de classes-objetos com uma grande quantidade de níveis é o problema do *IoIô*, isto é, torna-se necessário percorrer a hierarquia *(para cima e para baixo)* para compreender as conseqüências de uma mensagem que é enviada para o objeto. Portanto, toda vez que uma mensagem é enviada, tem que se examinar vários níveis da hierarquia para determinar como ela será tratada pelo objeto receptor (Travasssos et al. 1995), (Taenzer em Queiroz e Ribeiro 1995) (Wilde et al. 1993).

Na prática, o fato de uma hierarquia de classes-objetos ter uma profundidade considerada adequada depende de detalhes específicos do problema, complexidade da herança múltipla, da total funcionalidade que a hierarquia fornece, da experiência do projetista e do método de desenvolvimento utilizado (Wang e Wang 1995). No entanto, experiências descritas em Coad e Yourdon (1992), Rumbaugh (1993), Chidamber e Kemerer (1994), Firesmith (1994), Martin (1994), Lorenz e Kidd (1994), Yourdon (1994) e Henderson-Sellers (1996) sugerem que as estruturas de hierarquia de classes-objetos devem ter em média não mais do que seis níveis. Em geral, com alguma reestruturação, pode-se reduzir a profundidade de uma estrutura de generalização-especialização estendida em demasia.

- **Coesão Estrutural das Classes-Objetos**

Se uma classe-objetos tem diferentes serviços, realizando funções diferentes sobre um mesmo conjunto de argumentos, então a classe é *coesa*. Para realizar esta verificação, é possível determinar o grau de similaridade[4] entre os serviços especificados na classe-objetos, observando o conjunto de argumentos que participam nos serviços. Serviços que trabalham com os mesmos argumentos pode ser uma indicação de que os serviços estão relacionados.

Seja a classe-objetos C_i, com n serviços $S_1, \ldots, S_n$ e $\{A\}$ o conjunto de argumentos declarado no serviço S_i, onde $1 \leq i \leq n$. Então, existem N conjuntos $\{A_1\}, \ldots, \{A_n\}$, tal que M (número de conjuntos disjuntos) é gerado a partir da interseção entre os conjuntos de argumentos $A_i \cap A_j$.

[4]Segundo Bunge (1977) em Chidamber e Kemerer (1994), a similaridade entre duas entidades é determinada a partir da comparação de suas propriedades.

Baseado nos trabalhos de Chidamber e Kemerer (1994) e Chen e Lu (1993) o indicador da ***Coesão Estrutural*** (*CEC*) pode ser calculado pela relação entre o número de conjuntos disjuntos M e o conjunto total de argumentos N da seguinte forma:

$$CEC(C_i) = (1 - M/N)$$

Um valor alto de *CEC* é indicativo de que existe uma alto relacionamento entre os serviços especificados na classe-objetos. Por outro lado, um valor pequeno pode indicar que a classe-objetos deva ser dividida em duas ou mais classes-objetos, pois ela provavelmente realiza serviços não relacionados, tornando a classe-objetos mais complexa e aumentando as possibilidades de erros durante o desenvolvimento. Isto também incide nos processos de manutenção da classe-objetos e as possibilidades de reutilização da classe-objetos são reduzidas. Para ter-se um indicador mais preciso da *coesão da classe-objetos*, deve-se considerar também o aspecto semântico envolvido que é avaliado através do critério *Coesão Semântica da Classe-Objetos* do objetivo confiabilidade conceitual, subfator *Correção Semântica da Arquitetura.*

- **Coesão Estrutural dos *Clusters***

O aspecto da coesão de um conjunto de classes-objetos pode ser avaliado através da média do número de relacionamentos internos entre as classes-objetos. Seja *CL* um *cluster* específico e r o número de relacionamentos que são internos ao *cluster* (*i.e. que não estão conectados com classes-objetos fora do cluster*). Seja n o número de classes-objetos no *cluster*. Então, o indicador da ***Coesão Estrutural do Cluster (CCL)*** pode ser calculado da seguinte maneira:

$$CCL(CL_i) = \frac{(r + 1)}{n}$$ (O valor de 1 na equação evita que $CCL = 0$, quando $n=1$)

Um valor alto de *CCL* é uma indicação de que existe um alto relacionamento entre as classes-objetos no *cluster*. Caso contrário, o objetivo definido para o *cluster* deve ser revisto. Para ter-se um indicador mais preciso da *coesão do cluster*, deve-se considerar também o aspecto semântico envolvido, que é avaliado através do critério *Coesão Semântica do Cluster* do objetivo confiabilidade conceitual no subfator *Correção Semântica da Arquitetura.*

- **Acoplamento de Relacionamento das Classes-Objetos**

O ***Acoplamento de Relacionamento (ACR)*** é definido pelo número de relacionamentos - *dependência estática* - que uma determinada classe-objetos tem com outras classes-objetos no modelo, excluindo os relacionamentos por herança. Seja C_i uma classe-objetos do modelo e r a indicação do número de relacionamentos com outras classes-objetos. Então *ACR* é dada por:

$$ACR(C_i) = \sum_{i=1}^{n} r_i$$

Um modelo com classes-objetos com um alto *Acoplamento de Relacionamento* pode tornar o modelo difícil de entender e levar a erros durante o desenvolvimento. Uma mudança - *ou uma falha* - em uma classe-objetos pode ter um efeito de propagação em outras classes-objetos no modelo. Isto dificulta a atividade de manutenção e a realização de testes. Os serviços que implementam os relacionamentos - *construção, destruição, associação* e *desassociação* - entre classes-objetos podem tornar-se complexos. Um baixo acoplamento deste tipo maximiza o encapsulamento e aumenta as possibilidades de reutilização das classes-objetos, pois os objetos com poucas dependências são mais facilmente reutilizados em outras aplicações. Além disso, o tempo de desenvolvimento é reduzido, pois o projetista precisa de menos tempo para compreender detalhes de outras classes-objetos. Desta forma, as atividades de extensão e/ou customização são realizadas com maior facilidade.

- **Acoplamento de Relacionamento dos *Clusters***

Um *cluster* específico, além de estar composto por um conjunto de classes-objetos relacionadas, deve ter uma dependência mínima em relação a outros *clusters.* Assim sendo, o acoplamento entre clusters ***(ARCL)*** é definido pelo número de relacionamentos entre classes-objetos do *cluster* e classes-objetos que pertencem a *clusters* diferentes. Seja $CL_i, \ldots, CL_n$ um conjunto de *clusters* e r a indicação do número de relacionamentos entre classes-objetos que pertencem a *clusters* diferentes. Então, ***ARCL*** é dada por:

$$ARCL(CL_i) = \sum_{i=1}^{n} r_i$$

Um valor baixo para ***ARCL*** é desejável, na medida em que os relacionamentos entre as classes-objetos ficam confinadas aos *clusters,* sem atravessar suas fronteiras. Isto permite que o conjunto de classes-objetos possa ser implementado, testado e reutilizado de modo quase independente, sem afetar os demais *clusters.*

- **Acoplamento de Interação das Classes-Objetos**

Uma classe-objetos tem *Acoplamento de Interação* ***(ACI)*** com outra classe-objetos se uma delas atua sobre a outra via mensagens, ou seja, se os seus serviços utilizam os serviços ou atributos da outra classe-objetos. A partir dos serviços especificados na classe-objetos, é possível determinar o grau de interação que uma classe-objetos tem com as outras. Seja C_i uma classe-objetos com $S_1, \ldots, S_n$ os serviços declarados e R_i o conjunto de serviços chamados por $S_i = \{R_{ij}\}$, então o indicador que mede o grau de comunicação da classe-objetos é (Chidamber e Kemerer 1994), (Henderson-Sellers 1996):

$$ACI(C_i) = S_i \cup \{R_{ij}\}$$

Um valor alto de ***ACI*** é uma indicação de que a classe-objetos se comunica via mensagem com uma grande quantidade de classes-objetos, o que pode significar uma alta complexidade da classe-objetos. Além disso, através deste indicador é possível determinar a complexidade dos testes nas várias partes do modelo e possíveis problemas de desempenho. Idealmente, para maximizar o encapsulamento e facilitar a manutenção, espera-se que os valores de ***ACI*** sejam pequenos.

- **Acoplamento de Interação dos *Clusters***

Um *cluster* tem *Acoplamento de Interação (AICL)* se existe comunicação via mensagens entre classes-objetos do *cluster* e classes-objetos que pertencem a *clusters* diferentes. Seja CL_i um cluster formado pelas classes-objetos $C_1, \ldots, C_n$ e $ACI(C_i)$ o acoplamento de interação da classe-objetos C_i. O indicador do grau de ***Acoplamento de Interação do Cluster*** é calculado a partir de:

$$AICL(CL_i) = \frac{\sum_{i=1}^{n} ACI(C_i)}{n}$$

Um valor baixo para ***AICL*** é desejável, pois indica que a maioria das interações entre as classes-objetos se realizam dentro dos limites do *cluster*. Isto permite que o conjunto de classes-objetos possa ser implementado, reutilizado e testado com maior facilidade. Para o caso específico dos testes do *cluster*, estes tornam-se mais complexos, pois devem ser planejados casos de testes considerando as classes-objetos que estão dentro e fora do *cluster*.

Subfator: Concisão - *conjunto de atributos de qualidade que avaliam se a especificação contém um volume mínimo de texto por ter-se maximizado o volume de informação por unidade de texto.*

Pessoas resistem à leitura de documentos volumosos. Um texto grande e repetitivo prejudica o entendimento da especificação. O tempo e esforço mental para a leitura da especificação é maior, podendo provocar a perda de seu sentido global.

Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devemos sempre considerar o critério ***Complementabilidade***. A definição deste critério está na Tabela V.3 e uma descrição detalhada pode ser encontrada em Clunie (1987).

Ao se considerar o subfator *concisão*, para o caso de especificações orientadas a objetos, devem ser considerados, ainda, os seguintes critérios: ***Tamanho da Classe-Objetos, Tamanho do Serviço, Tamanho do Cluster*** e ***Tamanho das Interfaces***. A

definição destes critérios está na Tabela V.3. A seguir apresentamos uma discussão sobre os mesmos.

- **Tamanho da Classe-Objetos**

O tamanho adequado para uma classe-objetos varia dependendo da complexidade da aplicação e de sua responsabilidade no contexto do problema. O ***Tamanho da Classe-Objetos (TC)*** é definido como a quantidade de atributos e serviços declarados na classe-objetos. Seja a classe-objetos C_i com NA atributos e NS serviços. O tamanho da classe-objetos pode ser calculado a partir de:

$$TC(C_i) = NA + NS$$

Uma classe-objetos com um valor de TC muito grande - *se comparado com valores para outras classes-objetos* -, pode ser uma indicação da necessidade de divisão em classes-objetos menores. Caso isto seja necessário, deve-se preservar a coesão dos serviços definidos nela. O número exato de atributos e serviços desejável para uma classe-objetos é difícil de se definir, pois este depende das particularidades da classe-objetos no contexto da aplicação.

Do ponto de vista da reutilização, o tamanho da classe-objetos é um dilema. Por um lado, considerando-se o custo/benefício, uma classe-objetos com uma grande quantidade de atributos e serviços é desejável, na medida em que suas especializações possam herdar e utilizar uma grande quantidade de atributos e serviços. Entretanto, o tamanho de uma classe-objetos também pode ser uma indicação de que a classe-objetos foi projetada para uma aplicação específica, reduzindo suas possibilidades de reutilização em outras aplicações (Biggerstaff e Ritcher 1987).

A extensibilidade, também, é afetada por esta característica, pelo fato de que uma classe-objetos com uma grande quantidade de atributos e serviços tende a ser complexa, limitando as capacidades do projetista de realizar modificações e desenvolver novas extensões para a aplicação. Os testes tornam-se mais complexos de se realizar, pois uma maior quantidade de atributos e serviços precisam ser verificados e validados.

- **Tamanho do Serviço**

Embora não existam, ainda, pesquisas que determinem valores desejáveis para o tamanho dos serviços de uma classe-objetos, principalmente pelo fato deste ser altamente dependente da linguagem utilizada e do estilo de codificação do desenvolvedor é evidente que esta característica afeta o entendimento da solução modelada. Serviços muito grandes devem ser subdivididos em serviços menores, sem prejudicar a coesão da própria classe-objetos. Quando um serviço cresce muito, aumenta a chance de incluir "*bagagem extra*". O ***Tamanho do Serviço (TS)*** pode ser obtido a partir do número de

linhas de pseudocódigo declaradas no servico. Seja $S_1 \ldots S_n$ os serviços definidos numa classe-objetos C_i. O tamando do serviço pode ser obtido a partir de:

TS (S_i) = número de linhas de pseudocódigo do serviço S_i

Podemos identificar algumas conseqüências, não desejáveis, da existência de *TS* muito grandes:

a) São difíceis de se ler e entender;

b) Podem ser um indicador de que o código que vai ser gerado é mais orientado a funções do que orientado a objetos (Love 1990;1991), (Yourdon 1994), (Lorenz e Kidd 1994), (Henderson-Sellers 1996);

c) São, provavelmente, mais complexos e mais voltados para aplicações específicas, minimizando as possibilidades de reutilização e dificultando a realização dos testes e o desenvolvimento de futuras modificações e/ou extensões; e

d) Apresentam, provavelmente, uma baixa coesão.

- **Tamanho do *Cluster***

O *Tamanho do Cluster (TCL)* é entendido como a quantidade de classes-objetos definidas no *cluster*. Seja CL_i um cluster composto pelas classes-objetos $C_1, \ldots, C_n$. Seu tamanho pode ser obtido através de:

TCL (CL_i) = número de classes-objetos do cluster CL_i

Através do emprego de um mecanismo de identificação incremental das classes-objetos (*i.e. classes-objetos que vão surgindo ao longo do desenvolvimento e obedecendo a critérios pré-estabelecidos, com relação ao número de classes-objetos em cada cluster),* é possível ter-se um melhor controle, minimizar erros e aumentar a produtividade (Firesmith 1995). Valores exatos da quantidade de classes-objetos para um determinado *cluster* não são, ainda, estabelecidos. No entanto, este autor sugere que um *cluster* adequado contém em média cinco a vinte classes-objetos. Ao estabelecer restrições de tamanho aos *clusters,* pretende-se garantir que as classes-objetos que o compõem possam ser analisadas, projetadas, codificadas e testadas com facilidade por uma equipe de desenvolvimento composta por duas a cinco pessoas. As restrições de tamanho são estabelecidas em função do tamanho da equipe de desenvolvimento e da complexidade da aplicação.

- **Tamanho das Interfaces**

Segundo Jonhson e Foote (1988), uma forma de gerar projetos de alta qualidade, com classes reutilizáveis, é através do aperfeiçoamento da qualidade dos protocolos padrões (*i.e. novos comportamentos das mensagens e métodos).* Isto pode ser realizado através da redução do número de argumentos que participam nas mensagens, dividindo uma mensagem em várias. Isto aumenta o número de mensagens com argumentos similares, que poderiam ter o mesmo nome. Além disso, é possível considerar a criação

de uma nova classe-objetos que representa um pequeno grupo de argumentos. Assim sendo, o ***Tamanho das Interfaces (TI)*** é entendido como a quantidade de argumentos definidos em uma mensagem. Seja M_i uma mensagem composta pelos argumentos A_1,... A_n. Seu tamanho pode ser calculado através de:

$TI(M_i)$ = *número de argumentos da mensagem* M_i

Um bom número para ***TI*** é seis ou perto disto (Jonhson e Foote 1988). Por outro lado, Yourdon (1994) recomenda que o protocolo de mensagem deve ser tão simples quanto possível e conter poucos parâmetros (em média não mais do que três parâmetros).

Subfator: Simplicidade - *conjunto de atributos de qualidade que avaliam o grau de simplicidade dos serviços e atributos definidos nas classes-objetos.*

Manutenções em especificações acontecerão devido a que, freqüentemente, encontramos falhas nas especificações ou devido a mudanças nos requisitos por alterações no meio ambiente. Portanto, com o intuito de minimizar os impactos causados por modificações e/ou possíveis extensões na especificação, a *simplicidade* deve ser uma característica a ser considerada na geração da classe-objetos. Classes-objetos com atributos e serviços complexos são mais difíceis de serem entendidos e conseqüentemente implementados, reutilizados e testados.

Ao se considerar o subfator *simplicidade*, para o caso específico de especificações orientadas a objetos, devem ser considerados os critérios: ***Simplicidade dos Serviços*** e ***Simplicidade dos Atributos.*** A definição destes critérios está na Tabela V.3. A seguir apresentamos uma discussão sobre os mesmos.

- **Simplicidade dos Serviços**

O critério simplicidade dos serviços ***(SIS)*** avalia o grau de simplicidade de cada classe-objetos através da análise de cada um de seus serviços. Desta forma, é conhecido o esforço necessário para desenvolver, testar e manter a classe-objetos. *Seja* C_i uma classe, com n serviços $S_1, S_2, ..., S_n$ e sejam $s_1, s_2, ..., s_n$ os valores de 0 - 1 associados ao grau de simplicidade de cada serviço, segundo a Tabela V.1.

Grau de Simplicidade dos Serviços	Significado
0	Serviços com instruções CASE, IF's , Ciclos
0,25	Serviços com instruções IF's, Ciclos
0,50	Serviços com instruções IF's
0,75	Serviços com instruções Ciclos
1	Serviços sem instruções IF's, CASE, Ciclos

Tabela V.1 - Escala de valores para simplicidade dos serviços de acordo com as estruturas de controle

O valor da ***Simplicidade dos Serviços*** da classe-objetos é dada por:

$$SIS(C_i) = \sum_{i=1}^{n} s_i \ / \ \mathrm{n}$$

Um valor baixo de ***SIS,*** para uma dada classe-objetos, pode significar um grande impacto em suas subclasses, devido à herança de serviços. De fato, quanto maior o valor ***SIS,*** provavelmente, mais fácil será a reutilização, manutenção e teste da classe-objetos, pois se torna menos complexa. Espera-se, portanto, um valor de ***SIS*** o maior possível.

- **Simplicidade dos Atributos**

O critério simplicidade dos atributos ***(SIA)*** avalia o grau de simplicidade dos atributos definidos em cada classe-objetos. Desta forma, é conhecido o esforço necessário para desenvolver e manter a classe-objetos. *Seja* Ci uma classe, com ***n*** atributos $A_1, A_2, .., A_n$, e sejam $a_1, a_2, ..., a_n$ os valores de 0 - 1 associados ao grau de simplicidade de cada atributo, segundo a Tabela V.2.

Grau de Simplicidade dos Atributos	Significado
0	Listas/Vetores
0,25	Referências
0,50	Registro/Estrutura
0,75	Caracter
1	Inteiro, Real, Booleano

Tabela V.2 - Escala de valores para simplicidade dos atributos de acordo com sua estrutura

O valor da simplicidade dos atributos na classe-objetos é dada por:

$$SIA(C_i) = \sum_{i=1}^{n} a_i \ / \ \mathrm{n}$$

Um valor alto para ***SIA*** é um indicador da existência de uma classe-objetos menos complexa, provavelmente mais fácil de ser entendida, modificada, estendida e reutilizada.

Subfator: Conformidade - *conjunto de atributos de qualidade que avaliam se a especificação foi gerada considerando as normas estabelecidas para o desenvolvimento do produto.*

Uma especificação deve estar elaborada obedecendo a uma padronização pré-estabelecida, como consequência de normas da organização desenvolvedora da aplicação e/ou de sua contratante. Uma característica importante deste subfator é garantir a homogeneização para que o leitor da especificação possa entender o especificado e para que os desenvolvedores possam desempenhar a sua função na construção do produto.

Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devemos sempre considerar os critérios ***Aderência a Normas da Organização Desenvolvedora*** e ***Aderência a Normas estabelecidas pelo Contratante.*** A definição destes critérios está na Tabela V.3. e uma descrição detalhada dos mesmos pode ser encontrada em Clunie (1987).

V.3.1.2. Fator: Manipulabilidade - *conjunto de atributos de qualidade que avaliam a facilidade de manipulação da especificação para diversas formas de uso.*

Especificações são utilizadas, constantemente, ao longo do desenvolvimento e uso do produto, sempre que é necessário realizar consultas, avaliações e manutenções. No caso de especificações orientadas a objetos compostas por centenas de classes-objetos, esta característica é, particularmente, importante pelo fato de cada classe-objetos ser documentada em relação a vários aspectos - *atributos, serviços, relacionamentos, mensagens* - o que, para um sistema grande, produz centenas de páginas de documentação. Torna-se, portanto, necessária uma organização adequada da especificação que permita ao leitor percorrê-la, com facilidade, e localizar as informações de seu interesse.

O fator manipulabilidade deve ser avaliado através dos subfatores ***Disponibilidade*** e ***Rastreabilidade***.

Subfator: Disponibilidade - *conjunto de atributos de qualidade que avaliam a facilidade de acesso de uma especificação para seus usuários autorizados, na sua versão mais atualizada.*

A disponibilidade da especificação diz respeito ao fato de que ela deve estar pronta para ser utilizada quando se fizer necessário. Por seu caracter dinâmico, especificações devem ser constantemente atualizadas para, quando necessário, ser possível o acesso à sua última versão. Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devem ser considerados dois critérios: ***Estar Atualizada*** e ***Acessibilidade.*** A definição destes critérios está na Tabela V.3. e uma descrição detalhada pode ser encontrada em Clunie (1987).

Subfator: Rastreabilidade - *conjunto de atributos de qualidade que avaliam a facilidade de se percorrer as especificações de requisitos e de projeto identificando a agregação de detalhes a um determinado aspecto, desde sua visão mais global até a mais detalhada e vice-versa.*

As especificações devem fornecer mecanismos que facilitem o acesso a seus elementos de informação. Desta forma, podem ser percorridos caminhos ao longo das

especificações, permitindo encontrar o nível de detalhe desejado. Isto é fundamental para a realização de manutenções nas especificações, fornecendo facilidades para a localização do item que vai ser alterado e de todos os demais elementos afetados pela mudança.

Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devem ser considerados os critérios: ***Organização da Documentação, Localizabilidade Interna*** e ***Localizabilidade Externa.*** A definição destes critérios está na Tabela V.3. Uma descrição detalhada dos mesmos pode ser encontrada em Clunie (1987).

V.3.2. Objetivo: Confiabilidade Conceitual

Uma especificação deve registrar todas as características que o software deve conter, com o principal objetivo de satisfazer as necessidades dos seus usuários. Portanto, a especificação deve estar correta com relação a seu conteúdo, isto é, ao problema que descreve. Assim sendo, o objetivo ***Confiabilidade Conceitual*** refere-se às características de qualidade que avaliam se uma especificação é confiável para seus usuários, do ponto de vista de seu conteúdo, satisfazendo os requisitos que motivaram a sua construção.

Este objetivo se realiza através dos fatores de qualidade ***Fidedignidade*** e ***Suficiência.***

V.3.2.1. Fator: Fidedignidade - *conjunto de atributos de qualidade que avaliam se a especificação representa o que é entendido como sendo as necessidades e expectativas dos usuários do produto.*

O fator fidedignidade é avaliado através dos subfatores: ***Consistência, Não Ambigüidade, Correção das Abstrações, Correção Semântica da Arquitetura*** e ***Correção Comportamental.***

SubFator: Consistência - *conjunto de atributos de qualidade que avaliam se a especificação está isenta de contradições entre os aspectos especificados.*

Muitos aspectos da especificação evoluem através das diversas especificações produzidas ao longo de uma seqüência de refinamentos. É perfeitamente natural que, como conseqüência da ampliação e amadurecimento do conhecimento relativo ao domínio do problema, um mesmo aspecto seja descrito ou modelado de forma diferente

em diferentes lugares da mesma especificação. Inconsistências podem, ainda, estar relacionadas a entidades externas, tais como outros sistemas existentes ou em desenvolvimento na organização, especificações de software para o tratamento das interfaces com o usuário e pacotes de software. Para que seja possível dar uma solução correta ao problema, é necessário que todas as referências a um determinado aspecto se completem, sem estabelecer contradições.

Uma especificação é, portanto, consistente, se não contém descrições conflitantes. Existem vários tipos prováveis de conflito em especificações (Davis 1993):

a) ***Conflitos de comportamentos,*** ocorre quando são especificadas respostas diferentes com relação a um mesmo estímulo e condição.
b) ***Conflito de termos,*** ocorre quando dois ou mais termos de igual significado, são utilizados para descrever um mesmo objeto, que é tratado em contextos e lugares diferentes.
c) ***Conflito entre características,*** ocorre quando existem contradições entre características específicas de objetos descritos na especificação.
d) ***Conflito temporal,*** ocorre quando existe conflito lógico ou temporal entre duas ações específicas que dependem do tempo.

Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devem ser considerados os critérios: ***Consistência Interna*** e ***Consistência Externa.*** A definição destes critérios está na Tabela V.4. Uma descrição detalhada pode ser encontrada em Clunie (1987).

SubFator: Não Ambigüidade - *conjunto de atributos de qualidade que avaliam se o conteúdo da especificação está expresso, de forma a evitar a possibilidade de diferentes interpretações para qualquer aspecto ou assunto.*

Uma especificação é não ambígua, se e somente se, todos os aspectos nela descritos permitem apenas uma interpretação. O grau de formalidade da linguagem de especificação utilizada é um componente facilitador para a não ambigüidade da especificação. Entretanto, mesmo que a linguagem da especificação utilizada não seja formal, todos os aspectos devem ser especificados com uma redação explícita e precisa. Isto significa que palavras com contéudo vago, devem ser evitadas, buscando-se quantificar precisamente os requisitos.

Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devem ser considerados os critérios: ***Ser Explícita*** e ***Precisão.*** A definição destes critérios está na Tabela V.4. Uma descrição detalhada pode ser encontrada em Clunie (1987).

OBJETIVO: CONFIABILIDADE CONCEITUAL	
FIDEDIGNIDADE	Conjunto de atributos de qualidade que avaliam se a especificação representa o que é entendido como sendo as necessidades e expectativas dos usuários do produto.
Consistência	Conjunto de atributos de qualidade que avaliam se a especificação está isenta de contradições entre os aspectos especificados.
Consistência Interna	Característica que avalia se existem conflitos entre aspectos especificados na mesma especificação.
Consistência Externa	Característica que avalia se existem conflitos entre aspectos especificados em outras especificações ou entidades externas.
Não Ambigüidade	Conjunto de atributos de qualidade que avaliam seu o conteúdo da especificação está expresso, de forma a evitar a possibilidade de diferentes interpretações para qualquer aspecto ou assunto.
Ser Explícita	Característica que avalia se na especificação existem condições, hipóteses e/ou restrições definidas por contexto.
Precisão	Característica que avalia se os aspectos especificados estão descritos de forma precisa e, sempre que possível quantificada.
Correção Semântica da Arquitetura	Conjunto de atributos de qualidade que avaliam a correção das abstrações modeladas de acordo com o produto que especifica.
Correção das Classes-objetos	Característica que avalia a correta definição das classes-objetos.
Correção dos Atributos	Característica que avalia a correta definição dos atributos especificados na classe-objetos.
Correção dos Serviços	Característica que avalia a correta definição dos serviços especificados na classe-objetos.
Correção dos Relacionamentos de Associação	Característica que avalia a correta definição dos relacionamentos de associação especificados.
Correção dos Relacionamentos de Composição	Característica que avalia a correta definição dos relacionamentos de composição especificados.
Correção da Hierarquia de Classes-objetos	Característica que avalia a correta definição das estruturas de hierarquia de classes-objetos especificadas.
Coesão Semântica das Classes-objetos	Característica que avalia o grau de relacionamento semântico entre os conceitos especificados na classe-objetos.
Coesão Semântica dos Serviços	Característica que avalia o grau de relacionamento semântico entre os elementos de informação definidos no serviço.
Coesão Semântica dos *Cluster*	Característica que avalia o grau de relacionamento semântico entre os serviços definidos nas classes-objetos que compõem o *cluster.*
Acoplamento de Herança	Característica que avalia o grau em que são utilizados os atributos e serviços herdados.

Legenda

Fatores de Qualidade — Critérios de Qualidade para Especificações em Geral

SubFatores de Qualidade — Critérios de Qualidade para Especificações O.O.

Tabela V.4. - Características de Qualidade Relacionadas ao Objetivo Confiabilidade Conceitual

OBJETIVO: CONFIABILIDADE CONCEITUAL	
Correção Comportamental	Conjunto de atributos de qualidade que avaliam a correção da especificação do ponto de vista dinâmico.
Correção dos Cenários	Característica que avalia se na especificação está definido de correta a seqüência de interação entre as classes-objetos.
Correção do Comportamento das Classes-Objetos	Característica que avalia se na especificação está definido de forma correta o comportamento de cada classe-objetos.
Correção do Comportamento dos *Clusters*	Característica que avalia se na especificação está definido de forma correta o comportamento de cada *clusters*.
SUFICIÊNCIA	Conjunto de atributos de qualidade que avaliam se estão presentes na especificação todos os aspectos necessários e somente estes.
Necessidade	Conjunto de atributos de qualidade que avaliam se todos os aspectos considerados na especificação são imprescindíveis.
Necessidade dos Requisitos	Característica que avalia se na especificação estão descritos os requisitos considerados imprescindíveis.
Necessidade das Classes-objetos	Característica que avalia se na especificação estão definidas as classes-objetos consideradas imprescindíveis.
Necessidade dos Atributos	Característica que avalia se em cada classe-objetos estão definidos os atributos considerados imprescindíveis.
Necessidade dos Serviços	Característica que avalia se em cada classe-objetos estão definidos os serviços considerados imprescindíveis.
Necessidade dos Relacionamentos	Característica que avalia se na especificação estão definidos os relacionamento considerados imprescindíveis.
Não Redundância	Conjunto de atributos de qualidade que avaliam se existem aspectos repetitivos na especificação.
Não Redundância de Informações	Característica que avalia se um mesmo aspecto é descrito em mais de um lugar da especificação.
Não Redundância de Classes-objetos	Característica que avalia a duplicidade de classes-objetos na especificação.
Completitude	Conjunto de atributos de qualidade que avaliam se todos os aspectos que devem ser especificados estão presentes na especificação.
Completitude com Relação ao Roteiro definido pela Organiz.	Característica que avalia se o roteiro definido pela organização desenvolvedora foi totalmente coberto pela especificação.
Completitude com Relação ao Método de Desenvolvimento	Característica que avalia se foram utilizados todos os recursos previstos no método de desenvolvimento.
Completitude com Relação aos Requisitos	Característica que avalia se na especificação estão definidos todos os requisitos estabelecidos para o produto.

Legenda

Fatores de Qualidade — Critérios de Qualidade para Especificações em Geral

SubFatores de Qualidade — Critérios de Qualidade para Especificações O.O.

Tabela V.4. - Características de Qualidade Relacionadas ao Objetivo Confiabilidade Conceitual (cont.)

SubFator: Correção Semântica da Arquitetura - *conjunto de atributos de qualidade que avaliam a correção das abstrações modeladas de acordo com o produto que especifica.*

Este subfator é específico de especificações orientadas a objetos. Durante o processo de desenvolvimento, são gerados modelos de objetos que apresentam soluções do problema em diversos graus de abstração. Estes modelos são compostos de abstrações - *classe-objetos, atributos, serviços e relacionamentos* - cujo significado é determinado pelo domínio da aplicação. A construção de modelos de objetos requer múltiplas iterações, gerando cada vez mais um modelo refinado e estável. Este processo de refinamento é realizado acrescentando-se gradativamente detalhes às abstrações modeladas e estas evoluções devem ser feitas obedecendo-se aos padrões pré-estabelecidos pela organização e ao método de desenvolvimento utilizado. A não obediência a estes padrões pode resultar em ambigüidades e, conseqüentemente, em mal entendidos e numa implementação inadequada do produto. Por serem abstrações imprescindíveis para o entendimento e implementação da solução modelada, torna-se, necessária, sua avaliação de acordo com o problema que descreve.

Ao avaliarmos uma especificação orientada a objetos, segundo este subfator, devem ser considerados os critérios: ***Correção de Classes-Objetos, Correção de Atributos, Correção de Serviços, Correção dos Relacionamentos de Associação, Correção dos Relacionamentos de Composição, Correção da Hierarquia de Classes-objetos, Coesão Semântica das Classes-Objetos, Coesão Semântica dos Serviços, Coesão Semântica dos Clusters*** e ***Acoplamento de Herança.*** A definição destes critérios está na Tabela V.4. A seguir apresentamos uma discussão sobre os mesmos.

- **Correção das Classes-Objetos**

Definir uma classe-objetos como uma abstração do mundo real é essencial para o entendimento da modelagem do domínio do problema e para a implementação da solução computacional. Por isso, uma inadequada definição das classes-objetos pode ter um impacto significativo no êxito do projeto. Por exemplo, uma boa escolha dos nomes para as classes-objetos contribuirá para o entendimento do modelo de classes-objetos, embora isto nem sempre seja fácil. Estes nomes devem ser únicos no modelo, descrever o conjunto de objetos que a compõem, pertencer ao vocabulário utilizado pelos especialistas do domínio - *classes de domínio* - e obedecer aos padrões pré-estabelecidos. Cada classe-objetos deve ter uma descrição de seu significado, ter mais de um atributo, pelo menos um serviço que justifica sua existência no contexto da aplicação e estar associada a pelo menos uma outra classe. Além disso, cada classe-objetos deve representar uma abstração significativa para o domínio do problema e cada objeto na classe deve poder ser descrito a partir do conjunto de atributos e serviços especificados.

Portanto, todas as classes-objetos - *domínio e implementação* - devem estar corretamente definidas de acordo com o método de desenvolvimento utilizado e segundo as regras pré-estabelecidas pela organização para o projeto específico.

- **Correção dos Atributos**

Um atributo é uma abstração de uma única característica possuída por uma classe ou seu conjunto de objetos. Cada atributo possui um valor para cada instância de objeto, podendo ser um valor atômico, uma referência ou uma estrutura de dados. Os atributos representam as características da classe-objetos de domínio - *atributos de domínio* - ou resultados de decisões de projeto e implementação - *atributos de implementação.* O objetivo é obter, para cada classe-objetos, um conjunto de atributos que sejam:

a) ***Únicos,*** ou seja, os nomes dos atributos devem ser únicos na classe-objetos - *o que não é verdadeiro quando se considera todas as classes-objetos do modelo.*

b) ***Totalmente fatorados,*** ou seja, cada atributo captura um aspecto separado da abstração da classe-objetos; e

c) ***Completos,*** ou seja, abrangem todas as informações pertinentes à classe-objetos que está sendo definida.

- **Correção dos Serviços**

O serviço é uma atividade realizada por uma classe ou por um objeto. De modo geral, podem ser classificados em dois tipos: *serviços de domínio* - em sua maioria identificados, analisados é especificados durante a fase de análise - e *serviços de implementação* - resultado de decisões de projeto e codificação. Para garantir sua correta implementação, é necessário que cada serviço apresente uma definição precisa e que seja especificado, em pseudo-código ou representação equivalente, obedecendo aos padrões pré-estabelecidos para a organização ou para o projeto específico. Por exemplo, como regra geral, serviços que realizam as mesmas funções devem ter o mesmo nome, possuir o mesmo tipo e retornar o mesmo valor. Isto facilita a geração de protocolos padrão através do polimorfismo e, conseqüentemente, aumenta as possibilidades de reutilização da classe-objetos (Johnson e Foote 1988), (Rumbaugh 1993), (Martin R. 1995). Portanto, uma inadequada definição dos serviços pode trazer problemas no entendimento da implementação do produto.

- **Correção dos Relacionamentos de Associação**

As classes-objetos não existem de maneira isolada. Durante o processo de modelagem, os desenvolvedores estão interessados em identificar as associações entre as classes-objetos e em refletir essas associações no modelo através de relacionamentos precisamente determinados. Estas associações indicam a dependência semântica entre duas classes-objetos, e os indicadores de cardinalidade devem refletir a semântica correta

da associação entre as classes-objetos envolvidas, segundo os requisitos do domínio da aplicação. No relacionamento deve estar indicado claramente o significado semântico, através do uso de um nome simples e preciso. Caso contrário, pode ser uma indicação de um relacionamento incompleto ou que várias associações estão sendo modeladas através de um único relacionamento. Além do nome, qualquer relacionamento de associação precisa indicar as classes-objetos envolvidas, além de uma definição das restrições quando pertinente. Seu correto registro é essencial para o entendimento da solução modelada, sua correta implementação e validação por parte do usuário.

- **Correção dos Relacionamentos de Composição**

A correta modelagem do relacionamento de composição é importante porque nos proporciona um discernimento significativamente maior do domínio do problema e das responsabilidades do sistema. Permite obter um modelo de classe-objetos mais simples de ser entendido, devido à separação de aspectos de interesse do domínio do problema, garantindo a propriedade de visibilidade.

Relacionamentos de composição podem ser avaliados segundo as duas propriedades seguintes (Firesmith 1993), (Rumbaugh et al. 1994):

a) ***Transitividade***, isto é, se a Classe_A faz parte da Classe_B e a Classe_B faz parte da Classe_C, então a Classe_A faz parte da Classe_C.

b) ***Anti-simetria***, isto é, se a Classe_A faz parte da Classe_B, então a Classe_B não faz parte da Classe_A.

- **Correção da Hierarquia de Classes-objetos**

Uma das formas mais comuns de organização do conhecimento é através do uso de representações hierárquicas, que formam uma estrutura de grafo acíclico dirigido. Estas hierarquias facilitam a modelagem pela estruturação de classes-objetos e incorporam, resumidamente, o que é semelhante e o que é diferente em relação a elas, facilitando o entendimento do modelo e evitando a duplicação de código. Cada classe-objetos na hierarquia deve refletir um inter-relacionamento com as demais classes-objetos que compartilham a estrutura. Além disso, o topo de uma hierarquia - *generalização* - deve ser uma classe abstrata e as folhas - *especializações* - devem ser classes-objetos concretas. Uma classe-objetos é considerada uma especialização se herda todos os atributos e serviços da classe de generalização e acrescenta novos a si própria. Portanto, uma especialização deverá sempre representar uma extensão do comportamento de seus pais. Uma correta hierarquia de classes-objetos permite o uso do polimorfismo, que é um dos responsáveis pela facilidade de *extensão* e *manutenção* de um software orientado a objetos (Rumbaugh et al. 1993), (Winblad et al. 1993), (Martin R. 1995), (Wang e Wang 1995).

- **Coesão Semântica das Classes-Objetos**

Coesão é definida como um atributo de qualidade de módulos individuais, descrevendo o grau em que seus componentes são necessários para realizar uma mesma tarefa (Fenton 1991). Para o caso de sistemas orientados a objetos, o termo *módulo,* pode ser substituído pelo termo *classe-objetos* e a definição de coesão passa a ser entendida como a característica da classe-objetos de ter nela especificados um conjunto de atributos e serviços que caracterizam adequadamente a abstração e que permitem que ela possa cumprir com suas responsabilidades. No entanto, uma classe-objetos que apresenta atributos e serviços aparentemente não relacionados indica uma maior complexidade da classe-objetos, aumentando, assim, a possibilidade de erros durante o desenvolvimento da aplicação. Além disso, pode ser uma indicação de que ela precisa ser decomposta em classe-objetos menores. Classes-objetos que representam unidades menores, com atributos e serviços logicamente relacionados, são mais fáceis de entender e modificar e apresentam maiores possibilidades de reutilização do que classes-objetos grandes e complexas.

A ***Coesão Semântica das Classes - Objetos*** pode ser avaliada considerando a seguinte classificação:

a) ***Coesão por Abstração,*** ocorre quando a classe-objetos está composta por um conjunto de atributos e serviços relacionados que caracterizam adequadamente a abstração, visando o cumprimento de suas responsabilidades.

b) ***Coesão por Coincidência,*** ocorre quando a classe-objetos está composta por um conjunto de atributos e serviços que não caracterizam adequadamente a classe-objetos realizando funções não relacionadas com suas responsabilidades.

- **Coesão Semântica dos Serviços**

Classes-objetos apresentam um ou mais serviços, onde cada um deles deve realizar uma única operação. A ***Coesão Semântica do Serviço*** deve ser entendida como a força de associação funcional dos elementos definidos no serviço. Isto é, o serviço deve realizar uma única função e deve ser possível descrever seu propósito de forma precisa, através de uma sentença simples, contendo um único verbo (Yourdon 1994). A baixa coesão de um serviço da classe-objetos (*i.e. o serviço realiza mais de uma função*) indica que, provavelmente, é necessária uma modificação na classe-objetos correspondente para obter-se serviços que possuam o nível de granularidade desejado.

A ***Coesão Semântica de Serviço*** pode ser avaliada considerando a seguinte classificação:

a) ***Coesão Operacional,*** ocorre quando o serviço especificado na classe-objetos realiza uma única operação.

b) ***Coesão por Coincidência,*** ocorre quando o serviço realiza várias operações diferentes, não relacionadas.

- **Coesão Semântica dos *Clusters***

O *cluster* é um mecanismo que permite que um conjunto de classes-objetos seja organizado segundo objetivos específicos *(i.e. entendimento, teste, reutilização, implementação).* Por exemplo, nas fases iniciais do desenvolvimento, as classes-objetos são organizadas com o objetivo de atingir-se a compreensibilidade e ter-se uma visão geral do modelo numa perspectiva abstrata. Por outro lado, este tipo de organização pode mudar a medida que o desenvolvimento evolui. Determinadas funcionalidades do sistema, através de cenários ou da distribuição de tarefas, que deverão ser implementadas entre os membros da equipe de desenvolvimento, são aspectos comumente encontrados no desenvolvimento, e que são compreendidos e gerenciados através de *clusters* adequados. Portanto, um *cluster* é considerado coeso, se todas as suas abstrações forem ao encontro de seu objetivo específico. Um *cluster* deve representar um único objetivo específico e deverá estar composto por um conjunto de classes-objetos relacionadas, onde cada classe-objetos contribui para o atendimento deste objetivo. Um *cluster* coeso permite que um conjunto de classes-objetos possa ser devidamente entendido, testado, reutilizado e implementado.

A ***Coesão Semântica do Cluster*** pode ser avaliada considerando a seguinte classificação:

a) ***Coesão por Classes,*** ocorre quando o *cluster* está composto por um conjunto de classes-objetos relacionadas, visando o objetivo específico definido para o *cluster*.

b) ***Coesão por Coincidência,*** ocorre quando o *cluster* está composto por um conjunto de classes-objetos não relacionadas ao objetivo específico que o *cluster* deve atingir.

- **Acoplamento de Herança**

Herança é um mecanismo capaz de diminuir a complexidade do modelo e melhorar a sua compreensão, pois por meio da fatoração das características comuns, as classes-objetos tornam-se mais simples, com suas diferenças claramente identificadas. Estas características comuns podem ser reutilizadas, não somente pela subclasse em questão, mas também por qualquer outra classe-objetos futura que precise destas características. Através deste mecanismo, também, evita-se a duplicação de esforços na modelagem e na documentação. O *Acoplamento de Herança* deve ser entendido como o grau em que uma determinada classe-objetos utiliza os atributos e serviços (*i.e. estrutura e comportamento*) de sua(s) classe-objetos ancestral (is). Ter-se *Acoplamento de*

Herança elevado é desejável, na medida em que os atributos e serviços herdados são utilizados pela suas especializações, podendo, no caso dos serviços, ser redefinidos.

O ***Acoplamento de Herança*** pode acontecer, de modo geral, de duas formas diferentes:

a) ***Acoplamento por Extensão,*** ou seja, a subclasse utiliza os atributos e serviços herdados, os redefine ou adiciona outros atributos e serviços.

b) ***Acoplamento por Implementação,*** ou seja, serviços e atributos são modificados por conveniência para reutilizar ou compartilhar código. Isto é, a subclasse é tornada uma classe semelhante à desejada. Este tipo de acoplamento é semanticamente errado e conduz a problemas de manutenção e ao serem realizados os testes das classes-objetos, pois não há um relacionamento inerente entre as classes ancestral e descendente. Toda vez que são realizadas extensões e/ou modificações na classe-ancestral, estas não são aplicáveis a suas especializações.

SubFator: Correção Comportamental - *conjunto de atributos de qualidade que avaliam a correção da especificação do ponto de vista dinâmico.*

No desenvolvimento de um modelo de classes-objetos, duas dimensões ortogonais são imprescindíveis para descrever completamente o sistema que está sendo modelado. A dimensão estática - *modelo estático* - centra-se no aspecto passivo do sistema de classes-objetos e relaciona-se com a estrutura estática dos mesmos. A dimensão dinâmica ou comportamental - *modelo dinâmico* -, por sua vez, refere-se ao aspecto ativo do sistema, descrevendo o comportamento do conjunto de classes-objetos.

Nos métodos orientados a objetos, a dimensão comportamental é modelada utilizando diferentes técnicas[5], que descrevem os conceitos chaves da dimensão comportamental - *estado, evento, operação, interação* - como discutido no Capítulo III. A maioria dos métodos utiliza técnicas baseadas em máquinas de estado finito - *diagrama de transição de estado - DTE* (Yeung e Chow 1996) porque através delas é possível descrever como um objeto reage ao receber uma mensagem, identificando seus estados e transições e os serviços que o objeto executa na transição. Desta forma, os

[5] *Diagrama de Comunicação de Objetos* e *Diagrama de Ciclo de Vida dos Objetos* (Yourdon 1994), *Diagrama de Fluxos de Eventos e Diagrama de Transição de Estado* (Rumbaugh et al 1994), *Diagrama de Transição de Estado, Diagrama de Eventos, Diagrama de Operação e Diagrama de Fluxo de Objetos* (Martin e Odell 1992), *Diagrama de Estado* e *Diagrama de Comunicação de Objeto* (Shlaer e Mellor 1992) , *Diagrama de Interação de Objetos* (Embley et al. 1992), *Máquina de Estado, Diagrama de Cenários, Gráfico de Interação* (Coleman et al. 1994), *Diagrama de Transição de Estados* e *Diagrama de Interação(* Booch 1994).

aspectos estáticos e dinâmicos nos métodos de desenvolvimento são modelados com relativa independência.

Para compreender o aspecto comportamental de um modelo de objetos, é necessário conhecer os estados possíveis dos objetos, os eventos que produzem transições e as seqüências de interações - *cenários* - que definem o contexto destes eventos. Portanto, é possível considerar dois níveis de abstração: *um nível de colaboração - interação,* que refere-se à troca de mensagens entre os objetos, e *um nível de comportamento,* que refere-se aos estados, transições e ações internas de cada classe-objeto analisada.

Ao avaliarmos uma especificação orientada a objetos, no que se refere a este subfator, devemos, portanto, considerar os critérios: ***Correção dos Cenários, Correção do Comportamento das Classes-Objetos*** e ***Correção do Comportamento dos Clusters.*** A definição destes critérios está na Tabela V.4. A seguir apresentamos uma discussão sobre os mesmos.

- **Correção dos Cenários**

Um cenário pode ser definido como uma seqüência específica de interações entre classes-objetos com o objetivo de atingir uma funcionalidade desejada para o sistema. Além de ser uma auxílio para compreender o comportamento dinâmico do sistema, sua correta definição é importante, porque permite: *identificar novos objetos; avaliar a necessidade das classes-objetos, atributos, serviços e relacionamentos; refinar e melhorar a distribuição das responsabilidades das classes-objetos; verificar a consistência entre as responsabilidades das classes-objetos; definir casos de testes para as classes-objetos e, conseqüentemente, para todo o sistema, e; rastrear os requisitos da aplicação* (Gossain 1994), (White 1994), (Coad 1995).

Cada função especificada no documento de requisitos está associada a um ou mais cenários. Portanto, para garantir que estamos atendendo os requisitos da aplicação, devemos realizar uma análise detalhada dos cenários. Assim sendo, cada cenário deve ser avaliado, verificando: *os eventos iniciais, os dados associados ao evento; a classe-objetos que recebe o evento inicial; os caminhos de comunicação entre as classe-objetos envolvidas, verificando em cada classe-objetos, suas responsabilidades com relação à funcionalidade desejada para o cenário; os dados acessados no cenário, e; a classe-objetos que fornecerá a resposta (saída)* (Gossman 1994), (Agavanakis e Thrampoulidis 1995).

- **Correção do Comportamento das Classes-Objetos**

Algumas classes-objetos apresentam comportamentos razoavelmente complexo, exigindo que mais tempo seja destinado à compreensão, especificação e posterior avaliação de seu funcionamento. O nível de comportamento de uma classe-objeto diz respeito aos seus estados e transições que, conforme visto, podem ser representados utilizando um diagrama de transição de estados (*DTE*). Valores de atributos e relacionamentos mantidos por um objeto constituem seu estado. As transições representam mudanças nos valores dos atributos, implicando assim em mudanças de estados. Estas transições são produzidas como um reflexo a estímulos - *eventos* - ocorridos dentro ou fora do próprio objeto. Os objetos estimulam uns aos outros, (i.e. trocam mensagens) resultando numa série de alterações de seus estados.

O estado inicial deve ser conhecido, e é aquele em que o objeto se encontra quando é criado. Para muitos objetos o número de possíveis estados seria infinito. Por isso, não é preciso representar todos os estados no diagrama, mas apenas os estados genéricos que determinam algum comportamento específico. Através de um *DTE* é possível especificar de forma clara e precisa o comportamento de uma classe-objetos, permitindo que, posteriormente, o seu funcionamento seja validado. Além disso, uma representação gráfica torna mais fácil a localização de possíveis falhas ou indefinições no projeto, além de sugerir alternativas para a sua simplificação. Muitas classes-objetos possuem um comportamento tão simples que se torna desnecessário representá-lo por um *DTE* . Para outras, no entanto, este é um instrumento valioso, que se for bem utilizado pode auxiliar bastante o projeto.

- **Correção do Comportamento dos *Clusters***

Sendo um *cluster* um conjunto de classes-objetos que trabalham juntas para atingir um objetivo específico, seu comportamento será determinado essencialmente pela colaboração das classes-objetos constituintes que, por sua vez, depende do conjunto de comportamentos individuais exibidos por estas mesmas classes-objetos. Sendo assim, o comportamento de um determinado *cluster* é considerado correto se o conjunto de comportamentos individuais das classes-objetos fornecem o comportamento desejado em relação à funcionalidade definida para o *cluster*.

V.3.2.2. Fator: Suficiência - *conjunto de atributos de qualidade que avaliam se estão presentes na especificação todos os aspectos necessários e somente estes.*

O fator suficiência é avaliado através dos subfatores ***Necessidade, Não Redundância*** e ***Completitude***.

SubFator: Necessidade - *conjunto de atributos de qualidade que avaliam se todos os aspectos considerados na especificação são imprescindíveis.*

Uma especificação deve conter, apenas, os aspectos considerados imprescindíveis evitanto, assim, descrições desnecessárias e irrelevantes. No caso de especificações orientadas a objetos, este subfator torna-se importante pelo fato do desenvolvimento segundo este paradigma ser um processo iterativo, o que obriga a manter uma continua avaliação da necessidade das classes-objetos, atributos, serviços e relacionamentos ao longo do processo de desenvolvimento.

Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, deve ser considerado o critério ***Necessidade dos Requisitos***. A definição deste critério está na Tabela V.4. Uma discussão detalhada do mesmo pode ser encontrada em Clunie (1987).

A avaliação de uma especificação orientada a objetos, no que se refere a este subfator, deve considerar, também, os critérios: ***Necessidade das Classes-objetos, Necessidade dos Atributos, Necessidade dos Serviços*** e ***Necessidade dos Relacionamentos.*** A definição destes critérios está na Tabela V.4. A seguir apresentamos uma discussão sobre os mesmos.

- **Necessidade das Classes-Objetos**

A primeira atividade na construção do modelo de classes-objetos é identificar as classes-objetos que tenham relevância no domínio da aplicação. Uma especificação com classes-objetos irrelevantes, além de aumentar o tamanho do modelo, pode desviar a atenção do problema que está sendo modelado. Para ajudar na avaliação do grau de necessidade das classes-objetos no modelo, é conveniente considerar a seguinte classificação:

a) ***Classes-objetos imprescindíveis,*** são as classes-objetos consideradas relevantes para um implementação adequada do problema. Caso não existam no modelo, implicará num modelo incompleto que compromete a implementação e, conseqüentemente, a aceitação do software.

b) ***Classes-objetos opcionais,*** são as classes-objetos que poderão ou não ser consideradas, pois elas definem responsabilidades relativas a compromissos inicialmente não assumidos.

c) ***Classes-objetos desnecessárias,*** são as classes-objetos que não justificam sua existência no contexto da aplicação.

- **Necessidade dos Atributos**

Cada classe-objetos deve conter, apenas, os atributos considerados imprescindíveis à aplicação em desenvolvimento, evitando assim o tratamento de atributos desnecessários e irrelevantes. Um atributo na classe-objetos é necessário, quando participa nos serviços que a classe-objetos precisa para cumprir com suas responsabilidades ou quando representa um valor (*disponível através de um serviço implícito*) que pode ser acessado por outras classes-objetos, através de mensagens. Isto quer dizer que atributos não referenciados, e/ou que não participam nos serviços especificados na classe-objetos, não são necessários e, portanto, devem ser eliminados. Estes atributos podem ser propriedades válidas da classe-objetos, mas no contexto da aplicação não são necessários. Para avaliar o grau de necessidade dos atributos na classe-objetos, devemos considerar a seguinte classificação:

a) ***Atributos imprescindíveis,*** são os atributos considerados relevantes para que a classe-objetos possa definir seus serviços e possa cumprir com suas responsabilidades. Caso não constem nela, a classe-objetos não terá capacidade de responder a todas as responsabilidades.

b) ***Atributos opcionais,*** são os atributos que ajudam a obter uma classe-objetos mais abrangente. Caso não constem nela, não implicará em uma classe-objetos insuficiente.

c) ***Atributos desnecessários,*** são os atributos que não justificam sua existência na classe-objetos, pois não são utilizados no contexto da aplicação.

- **Necessidade dos Serviços**

Um serviço é necessário, quando contribui à realização das responsabilidades que a classe-objetos deve cumprir no contexto da aplicação. Estes serviços devem ser identificados tanto para as classes como para os objetos, de maneira diferenciada. Uma forma de garantir a identificação dos serviços necessários é listar as responsabilidades da classe-objetos, definindo o conjunto de serviços que são necessários para cumprí-las e assegurar que cada serviço realize uma única função, quanto possível. Da mesma forma que as classes-objetos e atributos, o grau de necessidade dos serviços, também, pode ser avaliado considerando três categorias:

a) ***Serviços imprescindíveis,*** são os serviços considerados relevantes, que garantem a implementação correta do produto. Caso não constem nela, podem comprometer a funcionalidade do produto.

b) ***Serviços opcionais,*** são os serviços que poderão ou não ser considerados. Caso não constem nela, não implicará em uma classe-objetos insuficiente.

c) ***Serviços desnecessários,*** são os serviços que não justificam sua existência na classe-objetos, pois não são utilizados no contexto da aplicação.

- **Necessidade de Relacionamentos**

A falta de serviços que percorram uma associação pode ser um indicativo que o relacionamento é desnecessário, pois nenhum serviço utiliza o caminho, o que talvez seja uma indicação de que as informações que se deseja obter não são necessárias.

Para avaliar o grau de necessidade dos relacionamentos entre classes-objetos, devemos considerar:

a) ***Relacionamentos imprescindíveis,*** são os relacionamentos considerados relevantes. Caso não existam, as classes-objetos afetadas não poderão cumprir com suas responsabilidades e, conseqüentemente, fica comprometida a funcionalidade prevista para o produto.

b) ***Relacionamentos opcionais,*** são os relacionamentos que poderão ou não ser considerados. Caso não constem nela, não comprometerá a funcionalidade das classes-objetos envolvidas.

c) ***Relacionamentos desnecessários,*** são os relacionamentos que não são percorridos por nenhum serviço.

SubFator: Não Redundância - *conjunto de atributos de qualidade que avaliam a presença, na especificação, de aspectos repetitivos.*

A presença de redundâncias pode tornar uma especificação mais compreensível, mas pode, também, ocasionar problemas nos processos de manutenção da especificação. Caso sejam necessárias, devem ser incluídas referências cruzadas explícitas cujo objetivo é permitir que as modificações sejam feitas de forma confiável.

Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, deve ser considerado o critério ***Não Redundância de Informações***. A definição deste critério está na Tabela V.4. Uma discussão detalhada do mesmo pode ser encontrada em Clunie (1987).

Ao se considerar o subfator ***Não Redundância,*** para o caso de especificações orientadas a objetos, deve ser considerado, também, o critério ***Não Redundância de Classes-objetos.*** A definição deste critério está na Tabela V.4. A seguir apresentamos sua discussão.

- **Não Redundância de Classes-objetos**

Ao serem inicialmente definidos os *clusters,* é difícil conhecer com precisão as fronteiras entre eles. É perfeitamente normal que durante o desenvolvimento, dois ou mais grupos, trabalhando em *clusters* adjacentes, gerem modelos que apresentam classes-objetos em comum que são modeladas de formas diferentes. Nesses casos, ao

serem identificadas classes-objetos que expressam a mesma informação, a classe-objetos deve ser nomeada de acordo com os *clusters* aos quais está relacionada, devendo ser representada em um único *cluster*.

Classes-objetos duplicadas podem ser encontradas pela comparação entre suas descrições (*i.e. seus atributos, serviços* e *relacionamentos*), identificando-se aquelas que são conceitualmente iguais. Portanto, redundâncias de classes-objetos devem ser evitadas, pois além de aumentar desnecessariamente o tamanho do modelo, interferem nos processos de testes e manutenção da especificação, podendo introduzir inconsistências.

SubFator: Completitude - *conjunto de atributos de qualidade que avaliam se todos os aspectos que devem ser especificados estão presentes na especificação.*

Segundo Davis (1993) e Alagar e Kourkopoulos (1994), a completitude é o atributo de qualidade da especificação mais difícil de se definir e detectar suas violações. Estas dificuldades são atribuídas a *mudanças nos requisitos, incompletitudes no conhecimento* do domínio do problema e *uso de uma linguagem de especificação inadequada.* De modo geral, duas categorias de incompletitudes podem ser identificadas: *Incompletitude externa,* que ocorre quando a especificação não reflete de forma adequada a aplicação, pela falta de definição de algumas de suas propriedades, e, *incompletitude interna,* que ocorre quando a especificação apresenta entidades indefinidas (termos, classes-objetos, atributos, serviços etc.)

Uma especificação é considerada completa se (IEEE 1984), (Clunie 1987), (Davis 1993):

a) Inclui tudo o que o software supostamente deve fazer (*i.e. requisitos relativos a funcionalidade, desempenho, qualidade, interfaces externas etc.);*

b) Define todas as respostas que o sistema deve fornecer para todas as possíveis classes de entrada constatadas, válidas ou inválidas;

c) Está de acordo com o roteiro estabelecido pela organização desenvolvedora para o projeto;

d) Identifica e faz referência a todas as figuras, tabelas, diagramas, assim como também define todos os termos e unidades de medida utilizados e referencia todo o material utilizado na especificação; e

e) Evita frases indeterminadas como "*A se determinar*". Caso existam, deve ser indicado *quem* é o responsável e *quando* será completada a seção.

Independente do paradigma de desenvolvimento, ao avaliarmos uma especificação segundo este subfator, devem ser considerados os critérios: ***Completitude com Relação ao Roteiro Definido pela Organização, Completitude com Relação ao***

Método de Desenvolvimento e ***Completitude com Relação aos Requisitos.*** A definição destes critérios está na Tabela V.4. Uma discussão detalhada sobre os mesmos pode ser encontrada em Clunie (1987).

V.3.3. Objetivo: Utilizabilidade

Utilizabilidade é um objetivo fundamental, pois não tem sentido se pensar em uma especificação, se esta não pode ser utilizada. Este objetivo refere-se às características de qualidade que tornam possível a utilização da especificação, sob as mais diversas formas e propósitos, durante o processo de desenvolvimento, avaliação, manutenção e implementação. Uma especificação é normalmente utilizada para:

- ser avaliada com relação às suas propriedades inerentes e quanto às necessidades e expectativas dos futuros usuários do produto;
- realização de manutenções;
- ser implementada; e
- ser reutilizada de forma parcial ou total, em outros problemas similares.

O objetivo ***Utilizabilidade*** se realiza através dos seguintes fatores de qualidade: ***Avaliabilidade, Manutenibilidade, Reutilizabilidade*** e ***Implementabilidade.***

V.3.3.1 Fator: Avaliabilidade - *conjunto de atributos de qualidade que avaliam a capacidade da especificação poder ser avaliada com relação à sua forma e conteúdo.*

Uma especificação deve poder ser avaliada para, assim, se garantir o desenvolvimento de um produto final de boa qualidade. Esta avaliação deve poder ser feita com relação à sua forma e ao seu conteúdo. Este fator se realiza através dos subfatores ***Verificabilidade*** e ***Validabilidade.***

Subfator: Verificabilidade - *conjunto de atributos de qualidade que avaliam a facilidade de se avaliar uma especificação segundo à sua forma.*

Deve ser possível avaliar se a especificação foi construída atendendo às normas e padrões pré-estabelecidos pelo método de desenvolvimento utilizado, pela organização desenvolvedora e/ou usuário. Os atributos de qualidade relacionados a este subfator estão indicados nas Tabela V.5 e V.6. Uma justificativa destes relacionamentos pode ser encontrada no Apêndice I.

Subfator: Validabilidade - *conjunto de atributos de qualidade que avaliam a especificação com relação ao seu conteúdo.*

As especificações são a base para todas as demais etapas do desenvolvimento de um produto de software. Portanto, é muito importante poder garantir que ela esteja correta com relação ao seu contéudo. Ou seja, deve ser possível avaliar se a especificação descreve adequadamente o problema (*especificações de requisitos*) ou a solução proposta (*especificação de projeto*).

Os atributos de qualidade relacionados a este subfator estão indicados nas Tabelas V.5 e V.6. Uma justificativa destes relacionamentos pode ser encontrada no Apêndice I.

V.3.3.2 Fator: Manutenibilidade - *conjunto de atributos de qualidade que avaliam a capacidade da especificação poder ser facilmente modificada e detalhada.*

A realização de alterações em produtos de software é uma realidade, pela própria natureza do mundo real, sujeito a alterações, e do próprio processo de desenvolvimento de software.

A construção de software que seja suscetível a manutenção é, portanto, essencial e deve iniciar-se nos primeiros estágios do processo de desenvolvimento e inclui os documentos de análise e projeto. Portanto, devemos construir especificações considerando este fator. É bastante difícil produzir especificações imunes à necessidade de introdução de alterações.

Existem três possíveis categorias de manutenção em software, de acordo com o seu propósito. Estas categorias podem ser aplicadas, também, para especificações:

- ***manutenção corretiva,*** visa corrigir os erros existentes na especificação;
- ***manutenção adaptativa,*** visa modificar a especificação e acomodá-la às alterações ocorridas no ambiente, o que traze como conseqüência mudanças nos requisitos;
- ***manutenção evolutiva,*** visa permitir adições e refinamentos produzidos como conseqüência do processo evolutivo na geração da especificação.

Este fator se realiza através dos subfatores ***Modificabilidade*** e ***Evolutibilidade.***

Subfator: Modificabilidade - *conjunto de atributos de qualidade que avaliam a capacidade da especificação poder ser alterada com facilidade sem com isso perder a sua qualidade.*

A realização de alterações são necessárias pois, freqüentemente, encontramos erros, omissões e inadequações nas especificações - *manutenção corretiva* - ou devido a mudanças nos requisitos por alterações no ambiente - *manutenção adaptativa* . Torna-se necessário que a especificação seja facilmente modificável. Sua organização, estrutura, estilo e modelagem devem, portanto, permitir que tais modificações sejam feitas com facilidade, de forma completa e consistente, sem se perder a qualidade.

Os atributos de qualidade relacionados a este subfator estão indicados nas Tabelas V.5 e V.6. Uma justificativa destes relacionamentos pode ser encontrada no Apêndice I.

Subfator: Evolutibilidade - *conjunto de atributos de qualidade especificação que avaliam a sua facilidade de se poder introduzir novos requisitos ou realizar refinamentos em uma especificação sem que esta perca a qualidade.*

O ambiente real não é estático. Ele sofre mudanças que podem requerer a agregação de novas funcionalidades ao sistema e, conseqüentemente, alterar os requisitos do software. A especificação precisa, portanto, ser *evolutível.* Os atributos relacionados à confiabilidade da representação são fundamentais para que seja possível compreender e manipular a especificação onde se quer realizar modificações e/ou evoluções. Além disso, realizar uma modificação ou uma evolução em especificações não confiáveis sob o ponto de vista conceitual conduz, apenas, a especificações detalhadas não confiáveis, o que não tem qualquer sentido de se produzir. Os atributos de qualidade relacionados a este subfator estão indicados nas Tabelas V.5 e V.6. Uma justificativa destes relacionamentos pode ser encontrada no Apêndice I.

V.3.3.3 Fator: Reutilizabilidade - *conjunto de atributos de qualidade que avaliam se a especificação tem os seus componentes organizados e desenvolvidos de maneira a permitir sua reutilização parcial ou total em outras aplicações similares.*

Atualmente não é, ainda, freqüente a reutilização de componentes de especificações. Cada nova aplicação é especificada e desenvolvida como se nenhuma outra já houvesse sido criada. Devemos, no entanto, considerar a possibilidade de que outros produtos de software para aplicações similares venham a ser desenvolvidos e que se possa aproveitar o trabalho já realizado. Reutilizar especificações orientadas a objetos - *componentes do modelo de classes-objetos e sua documentação* - parcial ou totalmente pode reduzir em muito o volume de esforço requerido no desenvolvimento de novos produtos de software. A preocupação com a reutilização deve dar-se desde as primeiras etapas do processo de desenvolvimento[6]. Para que isto aconteça, devem ser realizadas

[6]A decisão de desenvolver para reuso é uma decisão gerencial, resultado de uma análise custo/benefício. Além disso, deve ser definido o enfoque de *desenvolvimento para reutilização* mais apropriado para a organização.

atividades relacionadas a um *desenvolvimento para reutilização*, visando a preparação de componentes da especificação para sua posterior reutilização em outros contextos[7]. O paradigma da orientação a objetos possui diversas características que facilitam a reutilização: *encapsulamento de classes-objetos, herança, composição de classes-objetos, polimorfismo, classes genéricas* e os *frameworks*. Este fator se realiza através dos subfatores ***Adaptabilidade*** e ***Generalidade***.

SubFator: Adaptabilidade - *conjunto de atributos de qualidade que avaliam a capacidade da especificação poder ser adaptada para corresponder a outra aplicação similar.*

Para que componentes de uma especificação possam ser reutilizados, esses componentes em geral necessitam adaptações para corresponder perfeitamente, à nova aplicação. Isso pode ser facilitado pela existência de padrões de desenvolvimento para seus componentes. Os atributos de qualidade relacionados a este subfator estão indicados nas Tabelas V.5 e V.6. Uma justificativa destes relacionamentos pode ser encontrada no Apêndice I.

SubFator: Generalidade - *conjunto de atributos de qualidade que avaliam se a especificação tem os seus componentes desenvolvidos de forma a poderem ser utilizados em outros contextos.*

Componentes da especificação desenvolvidos para serem reutilizados em outras aplicações devem ser mais flexíveis, do que se fossem desenvolvidos para atender apenas a uma solução específica. O que se pretende não é o desenvolvimento de um componente que satisfaça os requisitos de uma aplicação particular, mas sim de um componente suficientemente genérico e que encapsule funcionalidades úteis em uma diversidade de contextos. Esse ganho em flexibilidade implica inevitavelmente em um aumento da complexidade da componente, o que significa que o projetista deve dedicar uma parcela considerável de tempo ao seu planejamento, se realmente deseja um resultado satisfatório. Os atributos de qualidade relacionados a este subfator estão indicados nas Tabelas V.5 e V.6. Uma justificativa destes relacionamentos pode ser encontrada no Apêndice I.

[7]Estas atividades incluem: análise da variabilidade dos requisitos em relação aos diferentes reutilizadores potenciais, análise de custo/benefício com relação à introdução de novos requisitos e modelagem das componentes com um adequado nível de granularidade, considerando os possíveis reutilizadores do componente (Karlsson ed. 1995), (Frakes e Terry 1996).

		UTILIZABILIDADE					
		VER	VAL	MOD	EVO	ADA	GEN
CONFIABILIDADE DA REPRESENTAÇÃO	**COMUNICABILIDADE**						
	Correção no Uso do Método						
	Correção da Notação	X		X	X	X	
	Correção Sintática	X		X	X	X	
	Correção Semântica	X		X	X	X	
	Correção no Uso do Formato de Documentação	X		X	X	X	
	Uniformidade de Terminologia						
	Uniformidade de Termos	X	X	X	X		X
	Uniformidade de Notação	X	X	X	X		X
	Uniformidade no Nível Abstração						
	Uniformidade de Detalhes da Documentação	X	X	X	X	X	
	Independência de Detalhes de Projeto			X	X		X
	Uniformidade de Abstração das Classes-Objetos	X		X	X	X	
	Uniformidade de Abstração dos Atributos	X		X	X	X	
	Uniformidade de Abstração dos Serviços	X		X	X	X	
	Uniformidade de Abstração dos *Clusters*	X		X	X	X	
	Modularidade da Documentação						
	Coesão das Informações	X		X	X	X	
	Acoplamento entre as Seções	X		X	X	X	
	Estrutura da Documentação	X		X	X	X	
	Correção da Arquitetura						
	Nível de Fatoração da Classe-Objetos	X		X	X	X	
	Nível de Profundidade da Hierarquia de Classes-objetos	X		X	X	X	
	Coesão Estrutural das Classes-Objetos	X		X	X	X	
	Coesão Estrutural dos *Clusters*	X		X	X	X	
	Acoplamento de Relacionamentos das Classes	X		X	X	X	
	Acoplamento de Relacionamento dos *Clusters*	X		X	X	X	
	Acoplamento de Interação das Classes-Objetos	X		X	X	X	
	Acoplamento de Interação dos *Clusters*	X		X	X	X	
	Concisão						
	Complementabilidade	X					
	Tamanho da Classe-Objetos	X		X	X	X	
	Tamanho do Serviço	X		X	X	X	
	Tamanho do *Cluster*	X		X	X	X	
	Tamanho das Interfaces	X		X	X	X	
	Simplicidade						
	Simplicidade dos Serviços	X		X	X	X	
	Simplicidade dos Atributos	X		X	X	X	
	Conformidade						
	Aderência a Normas da Organização Desenvolvedora	X				X	
	Aderência a Normas estabelecidas pelo Contratante	X				X	
	MANIPULABILIDADE						
	Disponibilidade						
	Acessibilidade	X		X	X	X	
	Estar Atualizada	X		X	X	X	
	Rastreabilidade						
	Organização da Documentação	X		X	X	X	
	Localizabilidade Interna	X		X	X	X	
	Localizabilidade Externa	X		X	X	X	

Tabela V.5. - Características de Qualidade do Objetivo Confiabilidade da Representação relacionadas ao Objetivo Utilizabilidade

		UTILIZABILIDADE					
		VER	VAL	MOD	EVO	ADA	GEN
CONFIABILIDADE	**FIDEDIGNIDADE**						
	Consistência						
	Consistência Interna		X	X	X		
	Consistência Externa		X	X	X		
	Não Ambigüidade						
	Ser Explícita		X	X	X		
	Precisão		X	X	X		
	Correção Semântica da Arquitetura						
	Correção das Classes-objetos		X				X
	Correção dos Atributos		X				X
	Correção dos Serviços		X				X
	Correção dos Relacionamentos - Associação		X				X
	Correção dos Relacionamentos - Composição		X				X
	Correção das Estruturas de Hierarquia		X	X	X	X	X
	Coesão Semântica das Classes-objetos		X	X	X	X	
	Coesão Semântica dos Serviços		X	X	X	X	
	Coesão Semântica dos *Clusters*		X	X	X	X	
	Acoplamento de Herança		X	X	X	X	
	Correção Comportamental						
	Correção dos Cenários		X			X	
	Correção Comportamento das Classes-Objetos		X			X	
	Correção Comportamento dos *Clusters*		X			X	
CONCEITUAL	**SUFICIÊNCIA**						
	Necessidade						
	Necessidade dos Requisitos		X				
	Necessidade das Classes-Objetos		X				
	Necessidade dos Atributos		X				
	Necessidade dos Serviços		X				
	Necessidade dos Relacionamentos		X				
	Não Redundância						
	Não Redundância de Informações		X	X	X	X	
	Não Redundância de Classes-objetos		X	X	X	X	
	Completitude						
	Completitude com Relação ao Método de Desenvolvimento		X				
	Completitude com Relação ao Roteiro Definido pela Organização		X				
	Completitude com Relação aos Requisitos		X				

Tabela V. 6. - Características de Qualidade do Objetivo Confiabilidade Conceitual relacionadas ao Objetivo Utilizabilidade

V.3.3.4 Fator: Implementabilidade - *conjunto de atributos de qualidade que avaliam se uma especificação pode ser implementada, tendo em conta aspectos econômicos, financeiros, tecnológicos, de mão de obra, de cronograma e sociais.*

O desenvolvimento de software é uma atividade problemática, devido, principalmente, à escassez de recursos e às dificuldades encontradas durante o processo de desenvolvimento tais como, problemas técnicos, mão de obra inadequada e cronograma insuficiente. Por isso, é necessário realizar uma avaliação da viabilidade de construção do software, após se ter a sua especificação, considerando os vários aspectos

envolvidos, isto é, aspectos econômicos, financeiros, tecnológicos, de mão de obra, de cronograma, de engenharia humana e sociais.

Este fator se realiza através dos subfatores ***Viabilidade Econômica, Viabilidade Financeira, Viabilidade Tecnológica, Viabilidade de Mão de Obra, Viabilidade de Cronograma*** e ***Viabilidade Social.***

Subfator: Viabilidade Econômica - *conjunto de atributos de qualidade que avaliam a compatibilidade entre o custo estimado para o desenvolvimento e operação do software e os benefícios esperados com a sua utilização.*

O problema da viabilidade econômica é um problema de estimativa de todos os custos envolvidos no desenvolvimento e operação do sistema, dos benefícios que advirão do seu uso e da comparação destes custos e benefícios. É, portanto, um problema de análise custo/benefício.

Para o caso específico do paradigma de orientação a objetos, muito pouco tem sido feito em termos de métodos para estimar custos de desenvolvimento. Isto se deve ao fato de existirem poucos relatos de experiências no uso do paradigma em organizações. Conseqüentemente, a inexistência de dados históricos de custos de desenvolvimento de software utilizando este paradigma impede que se tenham padrões de referência, sendo muito difícil estimar de forma confiável o custo de um novo projeto. A literatura apresenta algumas propostas de métodos para estimar custos de desenvolvimento no paradigma de orientação a objetos, destacando-se os trabalhos de Lorenz e Kidd (1994), Bohem et al. (1995) e Philip et al. (1995).

Ao avaliarmos a ***Implementabilidade*** de uma especificação segundo este subfator, devem ser consideramos os critérios: ***Aceitabilidade de Custos, Relevância dos Benefícios*** e ***Compatibilidade Custo/Benefício***. A definição destes critérios está na Tabela V.7. Uma discussão detalhada dos mesmos pode ser encontrada em Clunie (1987).

Subfator: Viabilidade Financeira - *conjunto de atributos de qualidade que avaliam a existência e à disponibilidade de capital necessário para conduzir o desenvolvimento do produto especificado.*

A viabilidade financeira pode ser avaliada através da análise da condição financeira, presente e potencial. Deve-se considerar a capacidade da empresa de assumir, com segurança, o débito adicional que pretende contrair e em suprir, adequada e continuamente, com recursos próprios, os fundos necessários para o desenvolvimento, uso e manutenção do produto. Isto implica na verificação da existência de capital e se a empresa é capaz de torná-lo disponível quando necessário.

Ao avaliarmos a ***Implementabilidade*** de uma especificação segundo este subfator, devemos considerar os critérios: ***Existência de Capital*** e ***Disponibilidade de Capital.*** A definição destes critérios está na Tabela V.7. Uma discussão detalhada destes critérios pode ser encontrada em Clunie (1987).

OBJETIVO: UTILIZABILIDADE			
AVALIABILIDADE			Conjunto de atributos de qualidade que avaliam a capacidade da especificação poder ser avaliada com relação à sua forma e conteúdo.
	Verificabilidade		Conjunto de atributos de qualidade que avaliam a facilidade se se avaliar uma especificação segundo à sua forma.
	Validabilidade		Conjunto de atributos de qualidade que avaliam a especificação com relação ao seu conteúdo.
MANUTENIBILIDADE			Conjunto de atributos de qualidade que avaliam a capacidade da especificação poder ser facilmente modificada e detalhada.
	Modificabilidade		Conjunto de atributos de qualidade que avaliam a capacidade da especificação poder ser alterada com facilidade sem com isso perder a sua qualidade.
	Evolutibilidade		Conjunto de atributos de qualidade que avaliam a facilidade de se poder introduzir novos requisitos ou realizar refinamentos em especificações sem que esta perca a qualidade.
REUTILIZABILIDADE			Conjunto de atributos de qualidade que avaliam se a especificação de tem os seus componentes organizados e desenvolvidos de maneira a permitir sua reutilização parcial ou total em outras aplicações similares.
	Adaptabilidade		Conjunto de atributos de qualidade que avaliam a capacidade da especificação poder ser adaptada para corresponder a outra aplicação similar.
	Generalidade		Conjunto de atributos de qualidade que avaliam se a especificação tem os seus componentes desenvolvidos de forma a poderem ser utilizados em outros contextos.
IMPLEMENTABILIDADE			Conjunto de atributos de qualidade que avaliam se uma especificação poder ser implementada, tendo em conta aspectos econômicos, financeiros, tecnológicos, de mão de obra, de cronograma e sociais.
	Viabilidade Econômica		Conjunto de atributos de qualidade que avaliam a compatibilidade entre o custo estimado para o desenvolvimento e operação do software e os benefícios esperados com sua utilização.
		Aceitabilidade de Custos	Característica que avalia se as estimativas de custos, para o desenvolvimento e/ou produção do software, são aceitas por usuários e desenvolvedores.
		Relevância dos Benefícios	Característica que avalia se as estimativas de benefícios tangíveis e intangíveis são aceitas como relevantes, por usuários e desenvolvedores.
		Compatibilidade Custo/Benefício	Característica que avalia se os custos estimados para o desenvolvimento e operação são compatíveis com os benefícios esperados com sua utilização.

Tabela V. 7 . - Características de Qualidade de Especificações segundo o Objetivo Utilizabilidade

OBJETIVO: UTILIZABILIDADE	
Viabilidade Financeira	Conjunto de atributos de qualidade que avaliam a existência e à disponibilidade de capital necessário para conduzir o desenvolvimento do produto especificado.
Existência de Capital	Característica que avalia se a organização possui capital suficiente para custear o desenvolvimento.
Disponibilidade de Capital	Característica que avalia se a organização é capaz de tornar disponível o capital necessário para o desenvolvimento.
Viabilidade Tecnológica	Conjunto de atributos de qualidade que avaliam a existência e à disponibilidade da tecnologia necessária para conduzir o desenvolvimento do produto especificado.
Existência da Tecnologia	Característica que avalia se existe o nível de tecnologia necessário para conduzir o desenvolvimento.
Disponibilidade da Tecnologia	Característica que avalia se a equipe encarregada do desenvolvimento tem disponível a tecnologia necessária para conduzir o desenvolvimento.
Viabilidade de Mão de Obra	Conjunto de atributos de qualidade que avaliam a existência e à disponibilidade da mão de obra necessária para conduzir o desenvolvimento do produto especificado.
Existência de Mão de Obra	Característica que avalia se existe na instalação a mão de obra necessária para o desenvolvimento.
Disponibilidade de Mão de Obra	Característica que avalia se estão disponíveis os recursos humanos com o conhecimento e experiência necessários para realizar o desenvolvimento e operação do software.
Viabilidade de Cronograma	Conjunto de atributos de qualidade que avaliam se o software pode ser construído dentro do limite de tempo estabelecido.
Adequabilidade de Cronograma	Característica que avalia se o software pode ser construído no tempo previsto pelo cronograma, considerando possíveis ocorrências de imprevistos e sem descuidar da qualidade definida para o produto.
Flexibilidade de Cronograma	Característica que avalia se o cronograma aceito para o desenvolvimento pode atender, na medida do possível, fatores tais como introdução de atividades não projetadas, contingências etc.
Viabilidade Social	Conjunto de atributos de qualidade que avaliam as implicações que o software que vai ser construído terá sobre o grupo social ao qual este deverá servir, assim como sobre qualquer outro grupo social externo.
Aceitabilidade da Engenharia Humana	Característica que avalia se o software que vai ser construído leva em consideração o grau de satisfação e o desenvolvimento do potencial humano previsto para os usuários.
Aceitabilidade dos Impactos Sociais	Característica que avalia se o software que vai ser construído leva em consideração seus impactos sobre o sistema social ao qual deverá servir.

Tabela V. 7 . - Características de Qualidade de Especificações segundo o Objetivo Utilizabilidade (cont.)

Subfator: Viabilidade Tecnológica - *conjunto de atributos de qualidade que avaliam a existência e à disponibilidade da tecnologia necessária para conduzir o desenvolvimento do produto especificado.*

A viabilidade tecnológica refere-se à possibilidade de construir o software considerando se o desenvolvimento atual da tecnologia - *hardware e software* - permite

o atendimento dos requisitos. Conseqüentemente, é necessário verificar se a organização desenvolvedora tem acesso a essa tecnologia e caso isto não aconteça, como esta pode ser adquirida.

Ao avaliarmos a ***Implementabilidade*** de uma especificação segundo este subfator, devemos considerar os critérios: ***Existência da Tecnologia*** e ***Disponibilidade da Tecnologia.*** A definição destes critérios está na Tabela V.7. Uma discussão detalhada dos mesmos pode ser encontrada em Clunie (1987).

Subfator: Viabilidade de Mão de Obra - *conjunto de atributos de qualidade que avaliam a existência e à disponibilidade da mão de obra necessária para conduzir o desenvolvimento do produto especificado.*

A viabilidade de mão de obra está relacionada à possibilidade de se organizar uma equipe de desenvolvimento, formada por gerentes, especialistas do domínio, analistas e programadores, com conhecimento e experiência suficiente para desenvolver o software. Esta equipe precisa dominar a tecnologia a ser utilizada, no que se refere a hardware e a software. Além disso, precisa ter o conhecimento necessário do domínio da aplicação. Se não forem satisfeitas qualquer uma destas condições o êxito do projeto fica comprometido. Assim sendo, ao avaliarmos a ***Implementabilidade*** de uma especificação segundo este subfator, devemos considerar os critérios: ***Existência de Mão de Obra*** e ***Disponibilidade de Mão de Obra.*** A definição destes critérios está na Tabela V.7. Uma discussão detalhada dos mesmos pode ser encontrada em Clunie (1987).

Subfator: Viabilidade de Cronograma - *conjunto de atributos de qualidade que avaliam se o software pode ser construído dentro do limite de tempo estabelecido.*

O processo de desenvolvimento é uma atividade essencialmente dinâmica que envolve muitos problemas, existindo uma clara certeza de que, ao longo do desenvolvimento, os fatos não ocorrerão conforme o planejado. Por isso, deve-se avaliar se o software poderá ser construído dentro dos prazos estipulados no cronograma do projeto, considerando possíveis ocorrências de imprevistos e sem descuidar a qualidade definida para o produto.

Ao avaliarmos a ***Implementabilidade*** segundo este subfator, devemos considerar os critérios: ***Adequabilidade do Cronograma*** e ***Flexibilidade do Cronograma.*** A definição destes critérios está na Tabela V.7. Uma discussão detalhada dos mesmos pode ser encontrada em Clunie (1987).

Subfator: Viabilidade Social - *conjunto de atributos de qualidade que avaliam as implicações que o software, que vai ser construído, terá sobre o grupo social ao qual deverá servir, assim como sobre qualquer outro grupo social externo.*

A viabilidade social refere-se à possibilidade de se poder construir o software considerando as implicações sobre o grupo social ao qual deverá servir, bem como a qualquer outro grupo social externo que poderá ser afetado pelo mesmo. Portanto, um estudo de viabilidade social deve considerar as repercussões sobre a sociedade e estas devem ser cuidadosamente analisadas, assegurando-se a sua aceitabilidade, antes de se iniciar a construção do software.

Ao avaliarmos a ***Implementabilidade*** segundo este subfator, consideramos os critérios: ***Aceitabilidade dos Impactos Sociais*** e ***Aceitabilidade da Engenharia Humana.*** A definição destes critérios está na Tabela V.7. Uma discussão detalhada dos mesmos pode ser encontrada em Clunie (1987).

V.4 Conclusão

Neste capítulo é discutido um conjunto de características de qualidade proposto para avaliar especificações de modo geral e para o caso de especificações orientadas a objetos, no que se refere a modelos de classes-objetos e sua documentação associada. Estes atributos foram definidos a partir de trabalhos sobre qualidade de especificações, da literatura técnica sobre qualidade em orientação a objetos, dos métodos de desenvolvimento orientado a objetos e da experiência de participação em um projeto de desenvolvimento orientado a objetos. A organização da discussão destes atributos foi realizada de acordo com o método de avaliação da qualidade de software proposto por Rocha (1983).

Capítulo VI

Hierarquização dos Atributos de Qualidade de Especificações Orientadas a Objetos

"Conte o que é contábil, meça o que é mensurável e o que não é mensurável torne mensurável".

Galileo - Galilei (1564 -1642)

VI.1. Introdução

No capítulo anterior foram discutidos com detalhes os atributos de qualidade para especificações orientadas a objetos. Neste capítulo, descrevemos os resultados de uma pesquisa de campo realizada com o objetivo de obter um perfil da qualidade desejável para especificações orientadas a objetos. A partir desta pesquisa, verificamos se os atributos de qualidade propostos adequam-se às especificações desenvolvidas utilizando este paradigma e estabelecemos uma hierarquização dos mesmos, de acordo com o seu grau de importância. A pesquisa foi realizada consultando a opinião de profissionais com experiência no uso do paradigma de orientação a objetos no desenvolvimento de projetos reais.

VI.2. Hierarquização dos Atributos de Qualidade de Especificações Orientadas a Objetos

Apresenta-se nesta seção os procedimentos utilizados para determinar o grau de importância dos atributos de qualidade para especificações desenvolvidas segundo o paradigma de orientação a objetos, discutidos com detalhes no Capítulo V. O procedimento utilizado compreende as seguintes etapas: *definição dos objetivos, definição*

dos elementos da população, definição dos instrumentos, processo de aplicação dos instrumentos e *análise estatística dos dados coletados.*

VI.2.1. Objetivo

Esta pesquisa teve como objetivo obter um perfil da qualidade desejável para especificações orientadas a objetos. Para isto, foram realizadas as seguintes atividades:

- Verificação da adequação dos atributos de qualidade propostos às especificações desenvolvidas utilizando este paradigma.
- Análise do conjunto de atributos, estabelecendo uma hierarquização dos mesmos, de acordo com o seu grau de importância.

VI.2.2. Definição dos Elementos da População

Nesta pesquisa foram considerados, na população, três tipos de elementos:

- ***Gerente do projeto***, responsável pela atividade de gerenciamento do projeto ao longo de seu desenvolvimento.
- ***Analista/Projetista***, responsável pelas tarefas relacionadas à elicitação e geração da especificação de requisitos e de projeto.
- ***Programador***, responsável pelas tarefas de implementação segundo as especificações.

VI.2.3. Definição dos Instrumentos

Para a realização da pesquisa, foram desenvolvidos os seguintes documentos de apoio ao processo experimental:

- ***Instrumento para Hierarquizar Critérios de Qualidade para Especificações Orientadas a Objetos***
  Através da aplicação deste instrumento, é obtido o grau de importância dos atributos de qualidade para uma especificação, segundo as visões do analista/projetista e programador (Apêndice III)
- ***Manual para Controle da Qualidade de Especificações Orientadas a Objetos***
  Neste documento são discutidos os atributos de qualidade a serem analisados, apresentando-se sua definição, justificativa, processo de avaliação e sugestões para correção, quando pertinentes (Apêndice I).

VI.2.4. Processo de Aplicação dos Instrumentos

A partir da definição e discussão dos atributos de qualidade foram elaborados os instrumentos definidos no item anterior. Estes instrumentos foram aplicados a três

categorias da população definida na pesquisa (*gerentes*[1], *analistas/projetistas* e *programadores*) que participam em projetos de desenvolvimento orientado a objetos. O conjunto de atributos avaliados completa um total de 76, organizados segundo o método de avaliação da qualidade utilizado. Dos setenta e seis atributos, 53% correspondem a características aplicáveis a especificações em geral e 47% correspondem a características aplicáveis a especificações orientadas a objetos. A aplicação dos instrumentos para a coleta de dados foi realizada da seguinte maneira:

- As entrevistas foram iniciadas fazendo-se uma breve introdução do objetivo da pesquisa, do instrumento e fornecendo orientações gerais sobre o preenchimento do documento.
- O instrumento foi acompanhado de um anexo, contendo as definições de cada atributo de qualidade.
- Devido à extensão do instrumento e por serem atributos de qualidade pertencentes a um novo paradigma de desenvolvimento, optou-se pela aplicação do mesmo sempre em forma de entrevista.
- Foi entregue aos entrevistados o respectivo instrumento. Cada entrevista foi realizada de maneira individual, pelo fato de serem necessárias explanações verbais com relação a cada atributo de qualidade.
- Obteve-se dos entrevistados, para cada atributo avaliado, um valor quantitativo de 0 a 4, segundo a escala descrita na Tabela VI.1. Para o caso de atributos específicos do paradigma de orientação a objetos, foram obtidos dois valores. Um valor correspondente a especificações de requisitos e outro a especificações de projeto.

Grau de Importância	Sigla	Explicação
0	NI	Indica que o atributo que está sendo apresentado não tem nenhuma importância.
1	PI	Indica que o atributo que está sendo apresentado tem pouca importância.
2	ACI	Indica que o atributo que está sendo apresentado tem importância em algumas circunstâncias mas nem sempre.
3	MI	Indica que o atributo que está sendo apresentado é muito importante.
4	IMPRE	Indica de maneira absoluta que não há dúvida que o atributo que está sendo apresentado é imprescindível.

Tabela VI.1 - Escala de Valores

- Salientou-se que não se estava fazendo avaliação de um determinado sistema (desenvolvido ou em desenvolvimento), mas avaliando-se o grau de importância de cada atributo, a partir da experiência do entrevistado no desenvolvimento

[1]Pelo fato dos gerentes representarem uma amostra pequena, foram considerados na categoria analistas/projetistas ou programadores, dependendo de cada caso.

utilizando este paradigma. Isto é, os valores atribuídos a cada atributo retratariam como uma especificação orientada a objetos deveria estar e não o estado em que uma determinada especificação se apresentava.

- Foram realizados pré-testes pilotos com profissionais com o mesmo perfil dos entrevistados, para verificar a aplicabilidade do instrumento.

VI.2.5. Análise Estatística dos Dados

A geração do conjunto de dados foi realizada procurando-se garantir que a amostra, de acordo com o nosso universo pesquisado, fosse representativa. A coleta de dados da pesquisa estendeu-se a 20 entrevistados, todos com experiência em modelagem e programação orientada a objetos, utilizando o método de análise e projeto (OOA - OOP) proposto por Coad e Yourdon (1992;1993) ou o método OMT proposto por Rumbaught et al. (1994). A análise estatística deste dados foi realizada, para cada critério de qualidade, através do pacote estatístico *SPSS/PC for Windows Release 5.01*, da seguinte maneira:

- A estatística descritiva para as amostras foi realizada através das funções *Frequencies* e *Descriptives* para cada critério em separado e em conjunto, para os dados coletados dos analistas/projetistas e dos programadores. Analisaram-se também, em conjunto, todos os elementos população.
- A análise das amostras foi realizada através de medidas de tendência central (*média*[2], $X = \frac{1}{n}\sum_{i=1}^{n} Xi$) e de dispersão (*desvio padrão*[3], $\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(Xi - X)^2}$)

Para um melhor entendimento da análise dos dados coletados, a discussão dos resultados se inicia com os critérios de qualidade relativos a especificações de modo geral e, seguidamente, com os critérios para o caso do paradigma de orientação a objetos, segundo o objetivo confiabilidade da representação e confiabilidade conceitual, respectivamente.

VI.2.5.1 Hierarquização dos Critérios do Objetivo Confiabilidade da Representação

Nas tabelas e nos gráficos que serão apresentados, os critérios de qualidade para especificações orientadas a objetos são hierarquizados de acordo com os intervalos definidos na Tabela VI.1. A discussão e processos de avaliação de cada um destes atributos pode ser encontrado no Apêndice I.

A Tabela VI. 2 e a Figura VI.1 apresentam os resultados classificados em ordem descendente de importância, dos critérios de qualidade relacionados a especificações de modo

[2]Somamos as observações originais ($X_1, X_2, \ldots, X_n$) e dividimos o total por n.

[3]Medida do grau de dispersão das observações, sendo uma boa medida para descrever amostras.

geral para o objetivo confiabilidade da representação. Pode-se perceber, claramente, que a maioria dos critérios (90%) tem graus de importância variando de *muito importante* a *imprescindível*. Dentre o conjunto de atributos listados, os critérios de qualidade relacionados ao subfator *disponibilidade* (acessibilidade e estar atualizada) lideram o conjunto de critérios representados na tabela. Isto torna evidente a preocupação por se ter sempre uma especificação que possa ser facilmente acessada por seus usuários, na sua versão mais atualizada, pois será utilizada por diversas pessoas ao longo do desenvolvimento, devendo, ainda, atender às necessidades da fase de manutenção.

Os critérios pertencentes ao fator *comunicabilidade* também apresentam graus de importância elevados. Isto demonstra a preocupação de desenvolvedores e usuários por se ter disponível uma especificação que possa ser lida e entendida por diferentes pessoas ao longo do desenvolvimento, tornando viável as atividades de implementação, manutenção e reutilização. Apenas os critérios associados ao subfator *uniformidade no nível de abstração* (uniformidade de detalhe e independência de detalhes de projeto) apresentaram graus de importância menores. Acreditamos que isto acontece devido ao fato do desenvolvimento orientado a objetos estar envolvido numa dinâmica iterativa muito grande, que dificulta atingir estes critérios.

CONFIABILIDADE DA REPRESENTAÇÃO	
Critérios	*Grau de Importância*
Acessibilidade (ACE)	MI - IMPRE
Estar Atualizada (ETA)	MI - IMPRE
Uniformidade de Termos (UTT)	MI - IMPRE
Organização da Documentação (ORD)	MI - IMPRE
Correção Semântica (CSM)	MI - IMPRE
Complementabilidade (COM)	MI - IMPRE
Localizabilidade Interna (LOI)	MI - IMPRE
Estrutura da Documentação (EDO)	MI - IMPRE
Uniformidade de Notação (UNT)	MI - IMPRE
Coesão das Informações (COI)	MI - IMPRE
Correção da Notação (CNT)	MI - IMPRE
Aderência às Normas estabelecidas pelo Contratante (ANC)	MI - IMPRE
Correção Sintática (CST)	MI - IMPRE
Localizabilidade Externa (LOE)	MI - IMPRE
Aderência às Normas da Organização Desenvolvedora (ANO)	MI - IMPRE
Correção no uso do Formato de Documentação (CFD)	MI - IMPRE
Acoplamento entre Seções (ACS)	MI - IMPRE
Uniformidade de Detalhe da Documentação (UDD)	ACI - MI
Independência de Detalhes de Projeto (IDP)	ACI - MI

Tabela VI.2 - Hierarquização dos Critérios do Objetivo Confiabilidade da Representação para Especificações em Geral

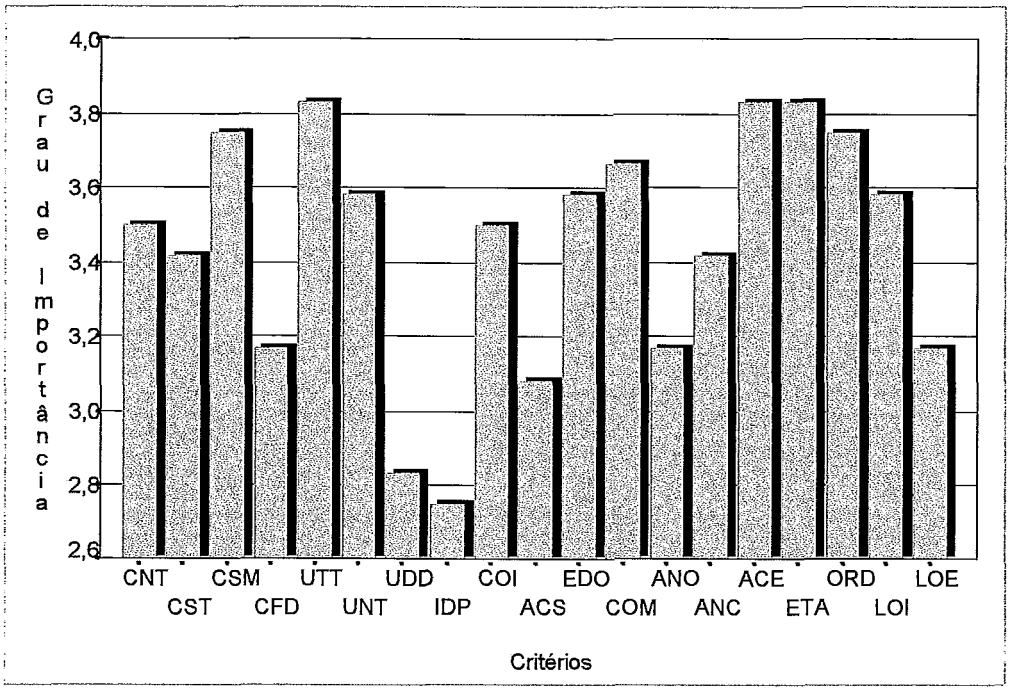


Figura VI.1 - Grau de Importância dos Critérios do Objetivo Confiabilidade da Representação para Especificações em Geral

As Tabelas VI.3 e VI.4 apresentam os resultados obtidos, classificados em ordem de importância, dos critérios referentes ao modelo de requisitos e de projeto orientado a objetos para o objetivo *confiabilidade da representação*. Observa-se que os critérios avaliados com relação ao modelo de requisitos apresentam, de modo geral, graus de importância menores, se comparados com os mesmos critérios avaliados segundo o modelo de projeto (Figura. VI.2). Isto significa que, pela própria natureza iterativa do processo de desenvolvimento orientado a objetos e pela falta de uma clara definição das fronteiras entre as fases, a preocupação pela qualidade se dá com maior atenção ao modelo na fase de projeto, por ser este o modelo a ser implementado. Entretanto, o modelo de requisitos usualmente é considerado incompleto, e terá transformações pelo próprio processo de maturação a que está submetido. Portanto, a equipe de desenvolvimento, nesse momento, está mais preocupada em fornecer soluções de modelagem que capturem os aspectos relevantes do domínio da aplicação, do que em atingir altos níveis de qualidade para um produto que inevitavelmente sofrerá modificações ao longo do desenvolvimento. Pode-se perceber, também, que o critério *tamanho das interfaces,* tem a última posição em grau de importância, quando se refere ao modelo de requisitos. No entanto, torna-se muito importante quando se refere ao modelo de projeto. Na realidade, este fato é natural pois, para o modelo de requisitos, dificilmente são considerados com rigor os argumentos que participam nas mensagens.

CONFIABILIDADE DA REPRESENTAÇÃO	
Critérios	*Grau de Importância*
Coesão Estrutural das Classes-objetos (CEC)	MI - IMPRE
Tamanho dos Serviços (TS)	MI - IMPRE
Acoplamento de Relacionamento das Classes-objetos (ARC)	MI - IMPRE
Simplicidade de Atributos (SIA)	MI
Coesão Estrutural dos *Clusters* (CCL)	MI
Simplicidade dos Serviços (SIS)	ACI - MI
Acoplamento de Interação das Classes-objetos (ACI)	ACI - MI
Nível de Profundidade da Hierarquia de Classes-objetos (NPH)	ACI - MI
Acoplamento de Relacionamento dos *Clusters* (ACRL)	ACI - MI
Nível de Fatoração das Classes-objetos (NFC)	ACI - MI
Uniformidade de Abstração dos *Clusters* (UCL)	ACI - MI
Acoplamento de Interação dos *Clusters* (AICL)	ACI - MI
Tamanho das Classes-objetos (TCO)	ACI - MI
Tamanho dos *Clusters* (TCL)	ACI - MI
Uniformidade de Abstração dos Serviços (UAS)	ACI - MI
Uniformidade de Abstração das Classes-objetos (UAC)	ACI - MI
Uniformidade de Abstração dos Atributos (UAA)	ACI - MI
Tamanho das Interfaces (TI)	PI - ACI

Tabela VI.3 - Hierarquização dos Critérios do Objetivo Confiabilidade da Representação para um Modelo de Requisitos Orientado a Objetos

CONFIABILIDADE DA REPRESENTAÇÃO	
Critérios	*Grau de Importância*
Coesão Estrutural das Classes-Objetos (CEC)	MI - IMPRE
Simplicidade do Serviço (SIS)	MI - IMPRE
Tamanho dos Serviços (TS)	MI - IMPRE
Acoplamento de Interação (ACI)	MI - IMPRE
Acoplamento de Relacionamento das Classes-objetos (ARC)	MI - IMPRE
Simplicidade dos Atributos (SIA)	MI - IMPRE
Coesão Estrutural dos *Clusters* (CCL)	MI - IMPRE
Nível de Profundidade da Hierarquia de Classes-objetos (NPH)	MI - IMPRE
Acoplamento de Interação dos *Clusters* (AICL)	MI - IMPRE
Uniformidade de Abstração dos Serviços (UAS)	MI - IMPRE
Tamanho das Interfaces (TI)	MI
Acoplamento de Relacionamento dos *Clusters* (ARCL)	ACI - MI
Uniformidade de Abstração das Classes-objetos (UAC)	ACI - MI
Nível de Fatoração das Classes-objetos (NFC)	ACI - MI
Tamanho das Classes-objetos (TCO)	ACI - MI
Uniformidade de Abstração dos *Clusters* (UCL)	ACI - MI
Tamanho dos *Clusters* (TCL)	ACI - MI
Uniformidade de Abstração dos Atributos (UAA)	ACI - MI

Tabela VI.4 - Hierarquização dos Critérios do Objetivo Confiabilidade da Representação para um Modelo de Projeto Orientado a Objetos

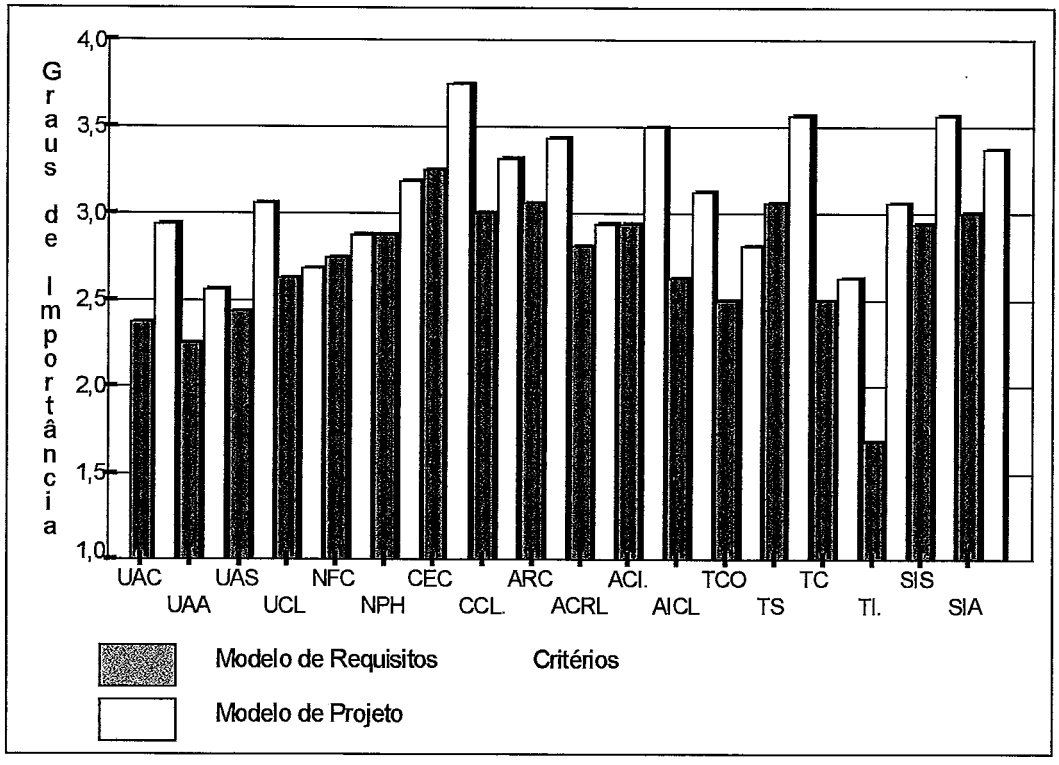


Figura VI.2 - Grau de Importância dos Critérios do Objetivo Confiabilidade da Representação para um Modelo Orientado a Objetos

Cabe salientar que, nos resultados computados referentes aos critérios relacionados a *cluster,* o valor do *desvio padrão* é maior, se comparado com os valores obtidos para os outros critérios. Isto induz a supor que o conceito de *cluster*[4] é entendido e utilizado de várias formas pelos desenvolvedores.

A Tabela VI.5 apresenta os resultados da avaliação do modelo de objetos, segundo a visão do analista/projetista, do programador e de toda a população. Através desta tabela, é possível demonstrar que o grau de exigência dos analistas/projetistas e dos programadores, com relação à qualidade do modelo de objetos, varia de acordo com o critério de qualidade observado.

Computando-se os dados de todos os elementos da população, observa-se que os resultados dos critérios relacionados ao subfator *uniformidade no nível de abstração* têm o mesmo grau de importância. Percebe-se, também, através da Figura.VI.3, que os analistas/projetistas demonstram ter maior preocupação em manter a uniformidade no nível de abstração, considerando o estágio de desenvolvimento. Por outro lado, os programadores são mais exigentes do que os analistas na maioria dos critérios relativos ao subfator *correção da arquitetura, concisão* e *simplicidade*, pois estes atributos estão

[4] A maioria dos entrevistados utilizaram o Método de Coad e Yourdon, portanto, o conceito de *cluster* é associado ao conceito de *assunto.*

relacionados com aspectos de disposição, composição e relacionamentos dos componentes do modelo, fundamentais para garantir uma implementação adequada.

OBJETIVO: CONFIABILIDADE DA REPRESENTAÇÃO			
ATRIBUTOS	(A/P, P)	A	P
Uniformidade No Nível de Abstração			
Uniformidade de Abstração das Classes-Objetos	ACI - MI	ACI - MI	ACI - MI
Uniformidade de Abstração dos Atributos	ACI - MI	ACI - MI	ACI
Uniformidade de Abstração dos Serviços	ACI - MI	ACI - MI	ACI - MI
Uniformidade de Abstração dos *Clusters*	ACI - MI	MI	ACI - MI
Correção da Arquitetura			
Nível de Fatoração da Classe-Objetos	ACI - MI	MI	ACI - MI
Nível de Profundidade da Hierarquia de Classes-objetos	MI	ACI - MI	MI - IMPRE
Coesão Estrutural da Classe-Objetos	MI - IMPRE	MI - IMPRE	MI - IMPRE
Coesão Estrutural dos *Clusters*	MI - IMPRE	MI - IMPRE	MI - IMPRE
Acoplamento de Relacionamentos das Classes-objetos	MI - IMPRE	MI - IMPRE	MI - IMPRE
Acoplamento de Relacionamento dos *Clusters*	ACI - MI	ACI - MI	ACI - MI
Acoplamento de Interação das Classes-Objetos	MI - IMPRE	MI - IMPRE	MI - IMPRE
Acoplamento de Interação dos *Clusters*	ACI - MI	ACI - MI	ACI - MI
Concisão			
Tamanho das Classes-Objetos	ACI - MI	ACI - MI	ACI - MI
Tamanho dos Serviços	MI - IMPRE	MI - IMPRE	MI - IMPRE
Tamanho dos *Clusters*	ACI - MI	ACI - MI	ACI - MI
Tamanho das Interfaces	ACI - MI	ACI - MI	ACI - MI
Simplicidade			
Simplicidade dos Serviços	MI - IMPRE	MI	MI - IMPRE
Simplicidade dos Atributos	MI - IMPRE	MI-IMPRE	MI - IMPRE

Legenda:
A - Analistas P - Programadores

Tabela VI. 5 - Hierarquização dos Critérios (organizados por Subfator) do Objetivo Confiabilidade da Representação para um Modelo Orientado a Objetos

No que diz respeito aos critérios relativos a tamanho, estes apresentam um grau de importância maior quando se refere ao *tamanho do serviço*. Isto demonstra que, apesar de ser um critério bastante controverso, ele se torna muito importante em seus respectivos contextos. Percebe-se, também, que os analistas/projetistas e os programadores estão preocupados em minimizar os impactos causados por alterações na especificação, através da definição e implementação de classes-objetos com serviços e atributos simples, quando possível. Considerando-se todos os dados obtidos dos elementos da população, percebe-se que os resultados ora favorecem as opiniões dos analistas/projetistas, ora favorecem os pareceres dos programadores, estabelecendo um equilíbrio entre ambos.

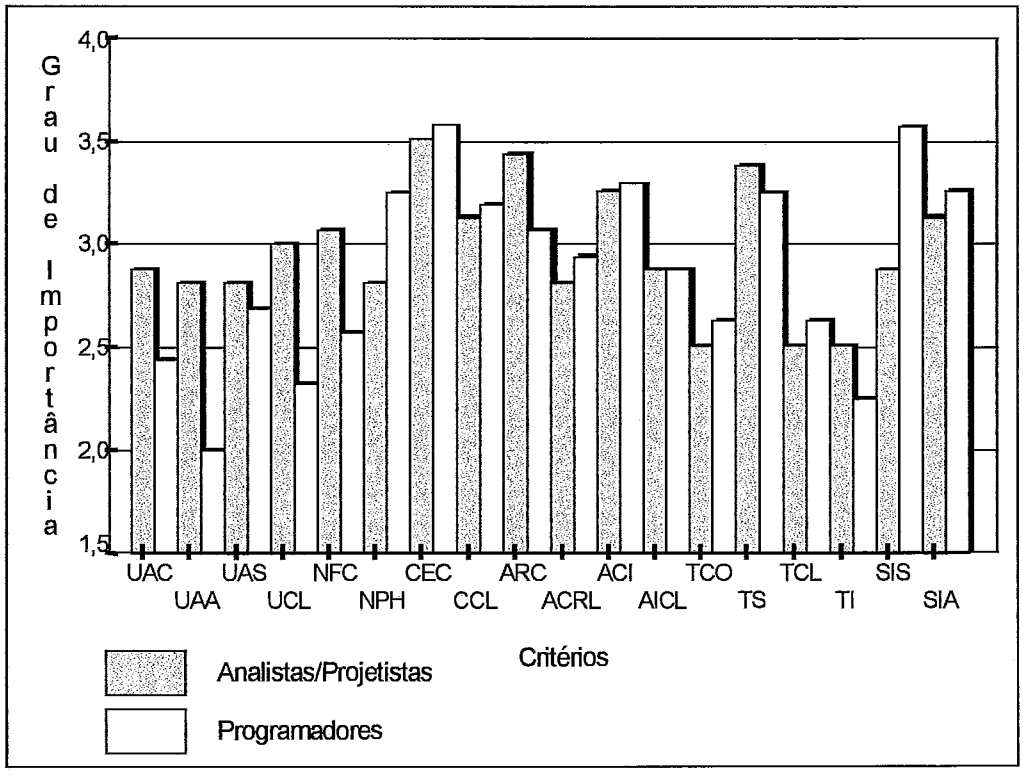


Figura VI.3 - Grau de Importância dos Critérios do Objetivo Confiabilidade da Representação para o Modelo Orientado a Objetos, segundo as visões dos Analistas/Projetistas e Programadores

VI.2.5.2 Hierarquização dos Critérios do Objetivo Confiabilidade Conceitual

A Tabela VI.6 e a Figura VI.4 apresentam os resultados, classificados em ordem de importância, dos critérios de qualidade relacionados a características gerais de especificações para o objetivo confiabilidade conceitual. Todos os critérios de qualidade foram pesquisados tanto para analistas/projetistas, quanto para programadores. Observa-se que a maioria desses critérios tem peso elevado, variando de *muito importante* a *imprescindível*, demonstrando o grande interesse em se ter uma especificação correta com relação ao problema que descreve. Apenas os critérios *não redundância* e *completude com relação ao método de desenvolvimento utilizado* têm um grau de importância menor. A *não redundância* não é considerada como um atributo imprescindível, pois a redundância em si não é considerada um erro, embora, possa levar a erros e trazer problemas no processo de manutenção da especificação. Além disso, em algumas circunstâncias, a redundância é necessária para tornar uma especificação mais compreensível. O critério *completude com relação ao método de desenvolvimento utilizado* tem peso variando de ACI - MI. Isto acontece porque o paradigma da orientação a objetos não atingiu a sua maturidade e os métodos de desenvolvimento orientado a objetos, de modo geral, não apóiam de forma adequada todo o processo de

desenvolvimento. Por isso, em alguns casos, é comum observar que ao longo do desenvolvimento são realizadas modificações e/ou agregações ao método, com o objetivo de adequá-lo ao projeto. Além disso, o método se torna, muitas vezes, mais um guia do que um recurso que deve fornecer subsídios adequados que ajudem a garantir o sucesso do projeto.

CONFIABILIDADE CONCEITUAL	
Critérios	*Grau de Importância*
Consistência Interna (COI)	MI - IMPRE
Consistência Externa (COE)	MI - IMPRE
Ser Explícita (EXP)	MI - IMPRE
Precisão (PRC)	MI - IMPRE
Necessidade dos Requisitos (NRQ)	MI - IMPRE
Não Redundância de Informações (NRI)	ACI - MI
Completude Relação ao Roteiro def. pela Organização (CRO)	MI - IMPRE
Completude Relação ao Método de Desenvolvimento (CMM)	ACI - MI
Completude com Relação aos Requisitos (COR)	MI - IMPRE

Tabela VI.6 - Hierarquização dos Critérios do Objetivo Confiabilidade Conceitual para Especificações em Geral

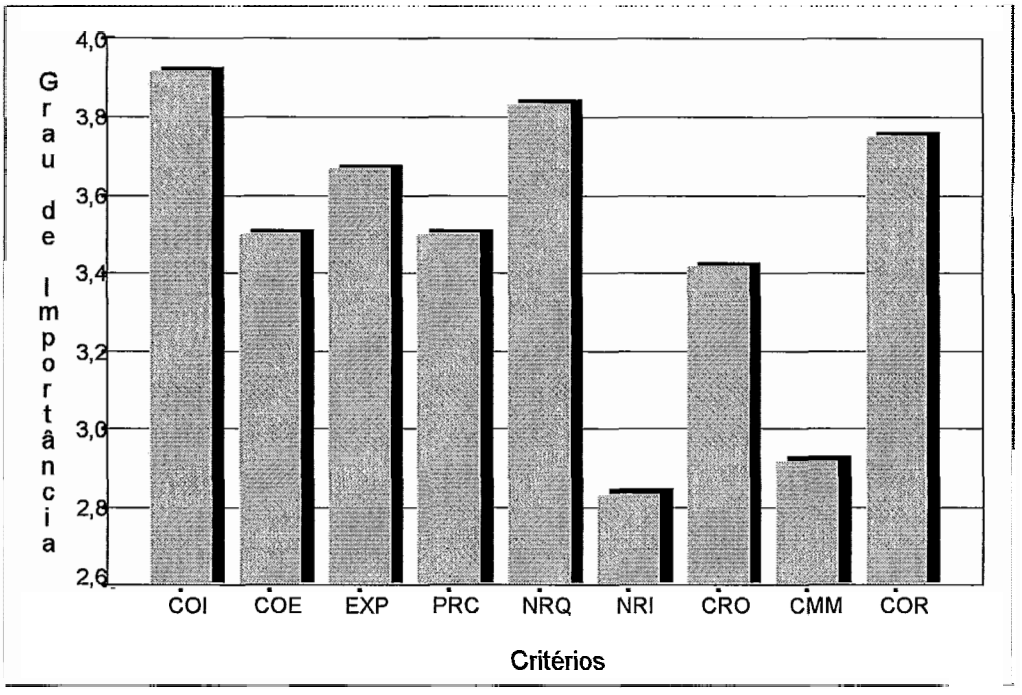


Figura VI.4 - Grau de Importância dos Critérios do Objetivo Confiabilidade Conceitual para Especificações em Geral

As Tabelas VI.7 e VI.8 apresentam os resultados obtidos, classificados em ordem descendente de importância, dos critérios referentes ao modelo de requisitos e de projeto orientado a objetos para o objetivo *confiabilidade conceitual*.

CONFIABILIDADE CONCEITUAL	
Critérios	*Grau de Importância*
Correção das Classes-Objetos (CRC)	MI - IMPRE
Correção da Hierarquia das Classes-Objetos (CHC)	MI - IMPRE
Correção dos Relacionamentos de Composição (CCO)	MI - IMPRE
Correção dos Relacionamentos de Associação (COR)	MI - IMPRE
Acoplamento de Herança (ACH)	MI - IMPRE
Correção dos Atributos (COA)	MI - IMPRE
Correção dos Serviços (COS)	MI - IMPRE
Necessidade dos Relacionamentos (NER)	MI - IMPRE
Correção dos Cenários (CCE)	MI - IMPRE
Correção do Comportamento das Classes-Objetos (CCC)	MI - IMPRE
Não Redundância das Classes-Objetos (NRC)	MI - IMPRE
Necessidade das Classes-Objetos (NEC)	MI - IMPRE
Coesão Semântica das Classes-Objetos (CSC)	MI - IMPRE
Necessidade dos Serviços (NES)	ACI - MI
Coesão Semântica dos Serviços (CSS)	ACI - MI
Necessidade dos Atributos (NEA)	ACI - MI
Coesão Semântica dos *Clusters* (CSCL)	ACI - MI
Correção do Comportamento dos *Clusters* (CCS)	ACI - MI

Tabela VI.7. - Hierarquização dos Critérios do Objetivo Confiabilidade Conceitual para um Modelo de Requisitos Orientado a Objetos

CONFIABILIDADE CONCEITUAL	
Critérios	*Grau de Importância*
Correção do Comportamento das Classes-Objetos (CCC)	MI - IMPRE
Necessidade das Classes-Objetos (NEC)	MI - IMPRE
Coesão dos Serviços (COS)	MI - IMPRE
Acoplamento de Herança (ACH)	MI - IMPRE
Correção dos Cenários (CCE)	MI - IMPRE
Não Redundância das Classes-Objetos (NRC)	MI - IMPRE
Correção da Hierarquia das Classes-Objetos (CHC)	MI - IMPRE
Necessidade dos Atributos (NEA)	MI - IMPRE
Necessidade de Relacionamentos (NER)	MI - IMPRE
Correção dos Relacionamentos de Composição (CRC)	MI - IMPRE
Correção dos Relacionamentos de Associação (COR)	MI - IMPRE
Correção das Classes-Objetos (CCO)	MI - IMPRE
Necessidade dos Serviços (NES)	MI - IMPRE
Correção dos Atributos (COA)	MI - IMPRE
Correção do Comportamento dos *Clusters* (CCS)	MI - IMPRE
Coesão Semântica dos Serviços (CSS)	MI - IMPRE
Coesão Semântica das Classes-Objetos (CSC)	MI - IMPRE
Coesão Semântica dos *Clusters* (CSCL)	MI

Tabela VI. 8 - Hierarquização dos Critérios do Objetivo Confiabilidade Conceitual para um Modelo de Projeto Orientado a Objetos

Observa-se que os critérios avaliados com relação ao modelo de requisitos apresentam, de modo geral, graus de importância menores, se comparados com os mesmos critérios avaliados segundo o modelo de projeto (Figura. VI.5). Um

comportamento similar foi observado com relação aos critérios do objetivo *confiabilidade da representação*. Isto supõe, também, que os analistas/projetistas e os programadores são mais exigentes com a qualidade do modelo de projeto. Nota-se que, para o modelo de requisitos, critérios relacionados com o agrupamento de classes-objetos (*clusters*) apresentam graus de importância menores. Isto acontece porque o modelo de requisitos não apresenta, num primeiro momento, uma completude conceitual que viabilize realizar tarefas a nível de *clusters*. Cabe salientar, também, que, da mesma forma que na confiabilidade da representação, os resultados obtidos com relação aos critérios relacionados a *cluster* apresentam valores superiores do *desvio padrão*, se comparados com os valores obtidos para os outros critérios, confirmando, uma vez mais, que o conceito de *cluster* é utilizado de diversas formas pelos desenvolvedores de software orientado a objetos.

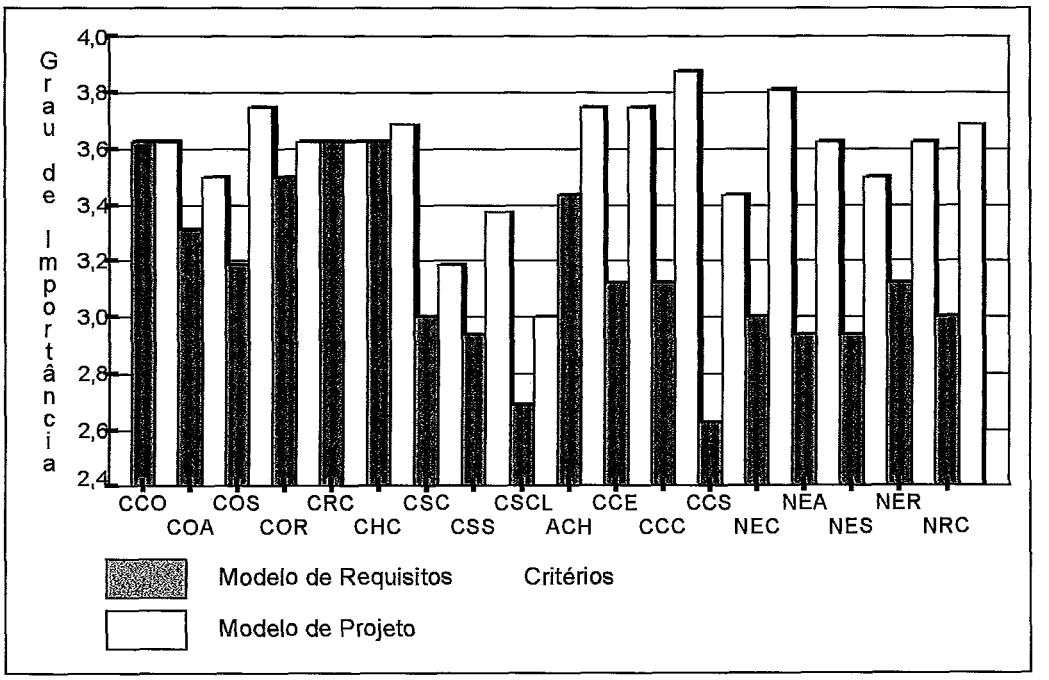


Figura VI.5 - Grau de Importância dos Critérios do Objetivo Confiabilidade Conceitual para um Modelo Orientado a Objetos

A Tabela VI.9 nos dá uma visão macroscópica dos critérios de qualidade relacionados ao objetivo confiabilidade conceitual. Esta tabela mostra-nos os pontos de vista de analistas/projetistas e de programadores, formando uma composição dos valores obtidos de ambos, no que diz respeito à qualidade de modelos orientados a objetos. Pode-se observar, além de uma uniformidade nas apreciações dos analistas/projetistas e dos programadores com relação aos critérios relacionados a este objetivo, que os graus de importância associados a cada critério indicam um alto grau de exigência em relação a estes critérios. Isto se justifica pela importância que têm na qualidade do produto final (Figura VI.6).

CONFIABILIDADE CONCEITUAL			
ATRIBUTOS	(AP, P)	A/P	P
Correção Semântica da Arquitetura			
Correção das Classes-Objetos (CCO)	MI - I	MI - I	MI - I
Correção dos Atributos (COA)	MI - I	MI - I	MI - I
Correção dos Serviços (COS)	MI - I	MI - I	MI - I
Correção dos Relacionamentos de Associação (COR)	MI - I	MI - I	MI - I
Correção dos Relacionamentos de Composição (CRC)	MI - I	MI - I	MI - I
Correção da Hierarquia de Classes-Objetos (CHC)	MI - I	MI - I	MI - I
Coesão Semântica das Classes - Objetos (CSC)	MI - I	MI - I	MI - I
Coesão Semântica dos Serviços (CSS)	MI - I	MI - I	MI - I
Coesão Semântica dos *Clusters* (CSCL)	AI - MI	AI - MI	AI - MI
Acoplamento de Herança (ACH)	MI - I	MI - I	MI - I
Correção Comportamental			
Correção dos Cenários (CCE)	MI - I	MI - I	MI - I
Correção do Comportamento das Classes-Objetos (CCC)	MI - I	MI - I	MI - I
Correção do Comportamento dos *Clusters* (CCS)	MI	AI - MI	MI - I
Necessidade			
Necessidade das Classes-Objetos (NEC)	MI - I	MI - I	MI - I
Necessidade dos Atributos (NEA)	MI - I	MI - I	MI - I
Necessidade dos Serviços (NES)	MI - I	MI - I	MI - I
Necessidade de Relacionamentos (NER)	MI - I	MI - I	MI - I
Não Redundância			
Não Redundância de Classes-Objetos (NRC)	MI - I	MI - I	MI - I

Tabela VI.9. - Hierarquização dos Critérios (organizados por Subfator) do Objetivo Confiabilidade Conceitual para o Modelo Orientado a Objetos

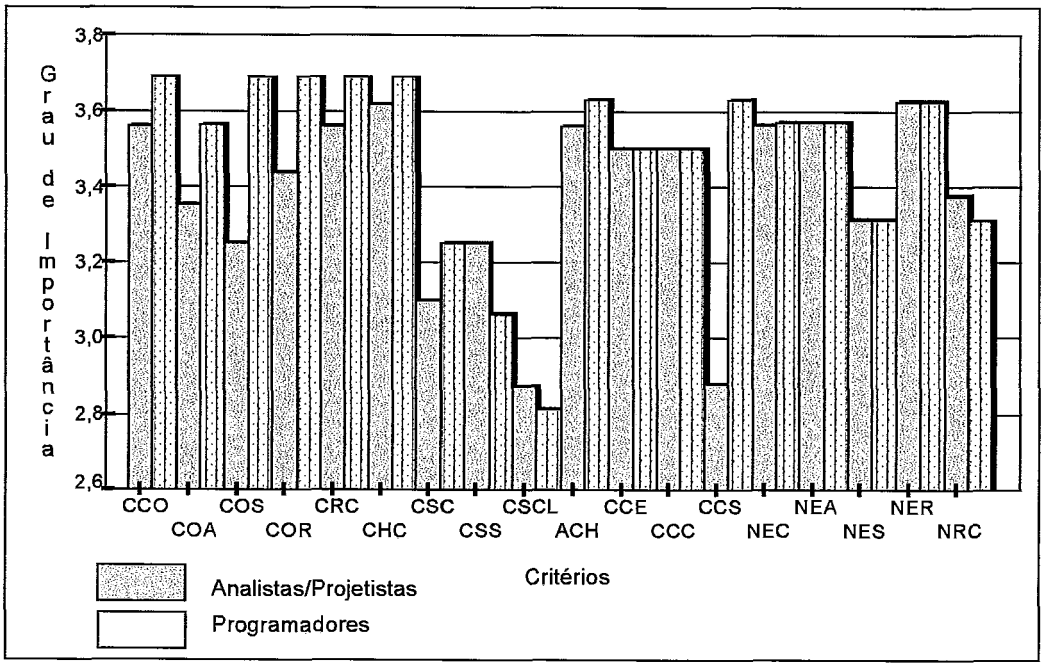


Figura VI. 6 - Grau de Importância dos Critérios do Objetivo Confiabilidade Conceitual para o Modelo Orientado a Objetos segundo as visões dos Analistas/Projetistas e Programadores

VI.2.5.3 Hierarquização dos Critérios do Fator Implementabilidade do Objetivo Utilizabilidade

A Tabela VI.10 e a Figura VI.7 mostram os resultados classificados dos critérios relacionados a especificações de requisitos para o fator implementabilidade, do objetivo Utilizabilidade. Observa-se, claramente, que a maioria dos critérios apresenta pesos elevados. Dentre o conjunto de atributos listados, o critério *aceitabilidade dos impactos sociais,* relacionado ao subfator *viabilidade social,* apresentou um nível de importância menor. Isto se deve ao fato de que, para a maioria dos profissionais entrevistados, aspectos relacionados a repercusões do software sobre a sociedade não são relevantes.

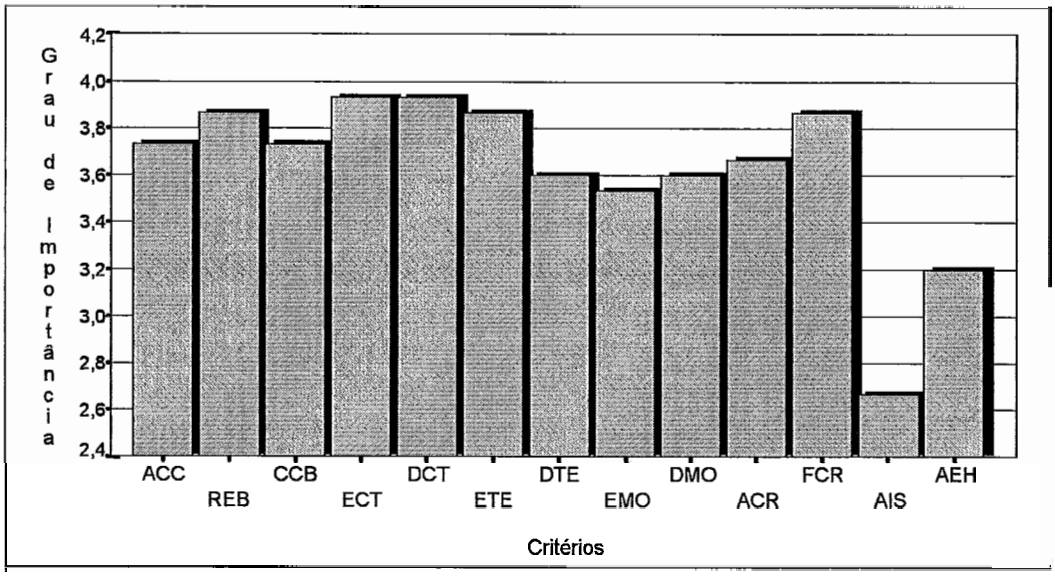


Figura VI. 7 - Grau de Importância dos Critérios do Fator Implementabilidade do Objetivo Utilizabilidade

UTILIZABILIDADE	
Critérios	*Grau de Importância*
Disponibilidade de Capital (DCT)	MI - IMPRE
Existência de Capital (ECT)	MI - IMPRE
Existência de Tecnologia (ETE)	MI - IMPRE
Flexibilidade de Cronograma (FCR)	MI - IMPRE
Relevância de Benefícios (REB)	MI - IMPRE
Aceitabilidade de Custos (ACC)	MI - IMPRE
Compatibilidade Custo/Benefício (CCB)	MI - IMPRE
Adequabilidade de Cronograma (ACR)	MI - IMPRE
Disponibilidade de Mão de Obra (DMO)	MI - IMPRE
Disponibilidade de Tecnologia (DTE)	MI - IMPRE
Existência de Mão de Obra (EMO)	MI - IMPRE
Aceitabilidade de Engenharia Humana (AEH)	MI - IMPRE
Aceitabilidade dos Impactos Sociais (AIS)	ACI - MI

Tabela VI.10 - Hierarquização dos Critérios do SubFator Implementabilidade do Objetivo Utilizabilidade

VI.3 Conclusão

Os resultados apresentados, neste capítulo, através da hierarquização de critérios, poderão auxiliar equipes geradoras de especificações orientadas a objetos a definir como devem ser despendidos os esforços na construção das especificações. As tabelas e gráficos gerados, de modo geral, confirmam-nos que a maioria dos critérios de qualidade propostos, segundo os pontos de vista de analistas/projetistas e programadores, deverão ter graus de importância elevados ao serem avaliados. Assim sendo, todos os atributos de qualidade são importantes, portanto, estes atributos devem ser considerados ao avaliar especificações desenvolvidas segundo o paradigma de orientação a objetos.

Cabe salientar que os resultados obtidos nesta pesquisa foram tratados utilizando estatística descritiva, onde as variações das respostas obtidas nos questionários foram analisadas através do desvio padrão. Uma análise estatística mais detalhada, considerando outras variáveis, como por exemplo: o método de desenvolvimento orientado a objetos utilizado, a experiência da equipe e o domínio da aplicação, poderia ter um impacto nos resultados obtidos. Para que isto possa ser realizado é necessário que a tecnologia evolua, e se tenham disponíveis dados sobre desenvolvimento de projetos, em diferentes domínios de aplicação utilizando diversos métodos de desenvolvimento. Desta forma, será possível atingir resultados mais precisos com relação à adequação dos atributos de qualidade propostos às especificações desenvolvidas utilizando este paradigma.

Capítulo VII

Experiência de Avaliação de Especificações Orientadas a Objetos

"Aumentando significativamente as atividades de prevenção e tornando mais eficientes as atividades de avaliação, os custos totais de qualidade podem ser cortados pela metade"

Zultner 1988

VII.1. Introdução

No Capítulo V foi apresentado um conjunto de atributos de qualidade para especificações orientadas a objetos. Foi possível observar, através da discussão de cada atributo, as conseqüências de se gerar uma especificação sem qualidade. Neste capítulo, descrevemos uma experiência de avaliação de especificações orientadas a objetos. O capítulo se inicia com a descrição, em linhas gerais, do projeto de desenvolvimento orientado a objetos que serviu de base para a realização desta experiência de avaliação. Em seguida, tendo como referência este projeto, apresentamos a experiência de avaliação, segundo os atributos propostos e os processos de avaliação definidos, visando, também, demonstrar sua usabilidade em outros projetos desenvolvidos segundo o paradigma de orientação a objetos.

VII.2. Caracterização do Projeto

A adoção do paradigma de orientação a objetos no processo de desenvolvimento de software tem crescido bastante, segundo as fortes tendências de utilização desta tecnologia como abordagem capaz de oferecer maior flexibilidade no desenvolvimento, facilidade de manutenção, possibilidade de reutilização e compreensão dos sistemas desenvolvidos. Por este motivo, este paradigma de desenvolvimento tem sido utilizado em projetos de variada extensão, com objetivos diferentes e aplicados em diversos domínios de aplicação. Contudo,

apesar de todo o interesse nesta tecnologia, são poucas as experiências de utilização deste paradigma em projetos reais e de grande porte em empresas brasileiras.

Como uma experiência de transferência de tecnologia entre *Universidade - Empresa*, iniciou-se, em 1993 um projeto de desenvolvimento com o uso de técnicas de orientação a objetos numa empresa brasileira de grande porte. Participaram do projeto engenheiros e analistas da empresa e pesquisadores e alunos de pós-graduação de engenharia de software da *COPPE/UFRJ*. Os pesquisadores da *COPPE* participaram do projeto no que se refere à definição do processo de desenvolvimento, treinamento da equipe, e acompanhamento do desenvolvimento do projeto, numa atividade típica de transferência de tecnologia.

A motivação básica para usar o enfoque orientado a objetos residia no interesse da empresa em iniciar a adoção de métodos utilizando este paradigma. Este projeto de cooperação teve como objetivo principal a *transferência de tecnologia* e o *treinamento* no uso deste novo enfoque de desenvolvimento. Alcançado este objetivo, esperava-se desencadear um processo de compromisso e adoção gradual desta tecnologia como enfoque de desenvolvimento na empresa. Além disso, as principais características desejáveis para o desenvolvimento do sistema escolhido - *flexibilidade na interação, rapidez* e sobretudo *facilidade de manutenção,*- constituem as vantagens esperadas na utilização da orientação a objetos como paradigma de desenvolvimento de sistemas.

Para um melhor entendimento das características do projeto, descrevemos, a seguir, alguns aspectos relativos ao processo de desenvolvimento adotado, ao controle da qualidade, à equipe de desenvolvimento, ao método e ferramentas de construção utilizadas, ao treinamento, à gerência do processo de desenvolvimento e à dimensão do projeto.

VII.2.1. O Processo de Desenvolvimento

Para a realização do projeto, foi gerado um documento, com base nas normas *ISO 9000-3* (ISO 9000-3 1990) e *ISO 9126* (ISO 9126 1991), que definiu o processo de desenvolvimento de software. O documento descreve o processo de desenvolvimento onde é definido o *modelo de ciclo de vida, os métodos e as ferramentas de construção, os roteiros da documentação, os procedimentos e os métodos para o controle da qualidade e gerência do processo de desenvolvimento* (Rocha et al. 1994).

Como ponto de partida para a definição do processo de desenvolvimento, considerou-se, além da norma ISO 9000-3:

- o desejo da empresa de iniciar o desenvolvimento de produtos segundo o paradigma de orientação a objetos;
- as características do produto que tornavam adequada a utilização deste paradigma;

- a experiência da equipe de Engenharia de Software da *COPPE*, no desenvolvimento de software com orientação a objetos, que tornava viável e seguro iniciar um projeto segundo este paradigma.

VII.2.2. Controle da Qualidade do Produto

A avaliação da qualidade do produto se deu ao longo de todo o processo de desenvolvimento. Durante as diversas etapas do projeto foram realizados dois tipos de avaliação: *avaliações intermediárias* e *avaliação final do produto de uma etapa*. No caso do projeto, estas avaliações foram realizadas através de reuniões de *walkthrough* ou de inspeção.

Uma avaliação intermediária foi realizada sempre que se deu por concluída uma tarefa cujo resultado iria impactar na próxima tarefa. Esta avaliação tinha por objetivo ter-se uma aprovação do usuário, conforme pertinente em cada situação, ou o consenso da equipe técnica sobre a adequação e completeza do produto da tarefa. As avaliações intermediárias previstas foram identificadas no modelo de ciclo de vida adotado.

As avaliações finais de produtos foram realizadas ao se dar por concluídas as tarefas de uma etapa e antes de se passar à próxima etapa do desenvolvimento. Foram, sempre, realizadas através de reuniões de inspeção das quais participaram, obrigatoriamente, os coordenadores do projeto por parte da *COPPE* e da empresa, representantes dos usuários e desenvolvedores e dois avaliadores que não tinham participado no desenvolvimento do produto.

VII.2.3. Equipe de Desenvolvimento

Uma equipe composta por onze profissionais responsáveis pelo desenvolvimento do sistema, entre funcionários da empresa e pesquisadores da universidade, deu início ao projeto. A equipe estava supervisionada pela gerência do projeto, formada por um coordenador por parte da empresa e uma equipe por parte da universidade, que incluía um coordenador e dois professores pesquisadores que acompanharam o projeto. Nas fases de projeto e construção, a equipe foi expandida, tendo sido incorporados cinco programadores, dos quais dois pertenciam à universidade e os demais eram funcionários da empresa. Totalizando, o projeto reuniu cerca de dezesseis pessoas, doze profissionais diretamente responsáveis pelo desenvolvimento, quatro profissionais a nível de gerência.

VII.2.4. O Método

O método de desenvolvimento orientado a objetos utilizado no projeto foi Análise e Projeto Orientado a Objetos de Coad e Yourdon (Coad e Yourdon 1992; 1993). As razões para a escolha do método se justificam pelo fato do objetivo principal do projeto ter sido a transferência de tecnologia. Este método foi escolhido por se apresentar adequado para o desenvolvimento de sistemas de informação e por ser considerado um método simples, portanto, fácil de ser rapidamente absorvido por uma equipe de desenvolvimento sem experiência em orientação a objetos. Além disso, as razões para esta escolha envolveram aspectos relativos à disponibilidade de ferramentas de apoio e à experiência anterior da equipe da universidade no uso do método.

VII.2.5. Ferramenta CASE

A definição do processo de desenvolvimento do projeto previu a utilização da ferramenta *ObjectTool*[1] (Nicola et al. 1993) para prover suporte ao Método de Coad e Yourdon durante as fases de análise e projeto. A escolha baseou-se, principalmente, no fato desta ferramenta ter sido desenvolvida pela própria equipe que propõe o método e pelo fato de não existir, naquele momento, um leque variado de opções para a escolha de ferramentas.

VII.2.6. Treinamento

Por se tratar de um projeto que visava à transferência de tecnologia, o treinamento dos membros da equipe de desenvolvimento aconteceu durante todo o processo de desenvolvimento, tendo como principal canal de efetividade a experiência dos profissionais envolvidos. No entanto, no planejamento do projeto, foi prevista a realização de cursos em cada fase do processo, para a capacitação inicial e básica da equipe de desenvolvimento com os instrumentos e ferramentas para construção do sistema (*ObjectTool*, C++, *Unix*[2], *OSF/Motif*[3] e *UIM*[4]) que seriam utilizados nestas fases.

VII.2.7. Gerência do Processo de Desenvolvimento

A gerência do projeto estava sob a responsabilidade de um coordenador, funcionário da empresa e um coordenador, professor da *COPPE*. Os dois coordenadores

[1] *ObjectTool* - Marca Registrada de Object International, Inc.
[2] *Unix* - Marca Registrada de UNIX Systems Laboratories
[3] *Motif* - Marca Registrada de Open Systems Foundations
[4] *UIM* - Marca Registrada de IBM Corp.

trabalharam em estreita colaboração, mantendo-se informados de todos os acontecimentos de interesse do projeto. Para facilitar a gerência e acompanhamento do projeto, foi definido o documento *Relatório Histórico do Projeto* (Rocha et al. 1994). Este documento contém o registro de todas as atividades realizadas no âmbito do projeto, bem como dos responsáveis e participantes na realização de cada atividade envolvida.

VII.2.8. Dimensão do Projeto

O projeto de desenvolvimento foi organizado e desenvolvido em dois grandes subsistemas. Na fase de análise, o modelo de domínio do sistema estava composto por 90 classes-objetos, distribuídas da seguinte maneira: O *Subsistema A* formado por 65 classes-objetos e o *Subsistema B* por 25 classes-objetos. Na fase de projeto o *Subsistema A* teve um incremento aproximado de 10% de classes-objetos de domínio, passando a ter o total de 71 classes-objetos. No entanto, o *Subsistema B* teve um aumento significativo de 40% de classes-objetos, passando a ter 42 classes-objetos. O modelo de domínio do *Sistema*, na fase de projeto, manteve-se estabilizado com aproximadamente 113 classes-objetos de domínio. As classes-objetos de suporte (*classes-objetos internas, janelas, gerenciadores* e *relatórios*) ficaram em torno de 400 classes-objetos.

VII.3. Experiência de Avaliação de Especificações segundo os Atributos e Processos de Avaliação

Na seção anterior foram discutidas, em linhas gerais, as características do projeto de desenvolvimento orientado a objetos que viabilizou a realização da experiência de avaliação das especificações segundo os atributos e processos de avaliação definidos. Através dele, foi possível adquirir conhecimento e experiência prática no desenvolvimento e avaliação de especificações de requisitos e especificações de projeto utilizando este paradigma (Clunie et al. 1995b).

Visando reformular e/ou aperfeiçoar os processos de avaliação definidos na segunda versão do *Manual para Controle de Qualidade de Especificações Orientadas a Objetos* (Clunie et al. 1997 e Apêndice I), apresentamos nesta seção uma experiência de avaliação de uma especificação. O resultado desta experiência também, visa demonstrar a viabilidade de utilização deste instrumento de avaliação, em projetos de desenvolvimento de software que utilizem o paradigma de orientação a objetos.

A fim de realizar a avaliação das especificações segundo os atributos e processos de avaliação propostos, foram utilizados dados do projeto descrito na seção anterior, a partir da *Especificação de Requisitos* e da *Especificação de Projeto*. Os comentários,

resultado da aplicação dos processos de avaliação, serão apresentados, inicialmente, com os critérios relacionados ao objetivo confiabilidade da representação e, em seguida, com os critérios relacionados à confiabilidade conceitual.

VII.3.1 Avaliação da Confiabilidade da Representação

Subfator: Correção no Uso do Método *(Critérios:Correção da Notação, Correção Semântica, Correção Sintática, Correção no Uso do Formato de Documentação)*

O avaliação mostrou que as especificações foram geradas utilizando corretamente o método de desenvolvimento no que se refere à sua notação e sintaxe. Verificou-se, também, como resultado da avaliação, que o formato de documentação que o método sugere não foi utilizado, sendo necessária a definição de um formato de documentação que fosse mais adequado ao projeto.

Como comentário a este resultado, pode-se afirmar que esta tendência é bastante natural no desenvolvimento orientado a objetos, sendo este um critério (*correção no uso do formato de documentação*) que apresenta, de modo geral, um grau de importância menor se comparado com os outros critérios do subfator, como pode ser visto na Figura VI.1.

Subfator: Uniformidade de Terminologia *(Critérios: Uniformidade de Termos, Uniformidade da Notação)*

A avaliação mostrou que na especificação, nos itens relacionados à descrição geral do processo de planejamento, descrição geral do sistema, definição dos requisitos do sistema e documentação dos modelos de classes-objetos, a terminologia foi utilizada de forma uniforme. Isto é, os termos e notação obedeceram a definições pré-estabelecidas da área considerada.

Subfator: Uniformidade No Nível de Abstração *(Critérios: Uniformidade de Detalhe da Documentação, Independência de Restrições de Projeto, Uniformidade de Abstração das Classes-objetos, Uniformidade de Abstração dos Atributos, Uniformidade de Abstração dos Serviços, Uniformidade de Abstração dos Clusters)*

A avaliação mostrou que as especificações apresentaram um nível suficientemente detalhado para permitir seu entendimento, embora o grau de detalhamento tenha variado, uma vez que diferentes profissionais têm idéias diferentes acerca do nível de detalhe necessário. A descrição dos aspectos na especificação de requisitos e a modelagem do domínio da aplicação mantiveram a desejável independência

de detalhes de projeto, evitando, quanto possível, impor restrições próprias de outras fases. As classes-objetos, atributos, serviços e *clusters* definidos no modelo de domínio e de projeto apresentaram nível de abstração compatível com o estágio do desenvolvimento.

Como comentário a este resultado, pode-se observar que a avaliação do critério *uniformidade de detalhe da documentação* se torna subjetiva e difícil de se realizar, embora necessária, pela grande quantidade de informações que devem ser consideradas e pela iteratividade do processo de desenvolvimento. Além disso, a uniformidade semântica que fornece o paradigma, faz com que o critério *independência de detalhes de projeto,* algumas vezes seja difícil de ser mantido. Pode-se observar, também, a partir da Figura VI.1, a confirmação deste fato, pois, pela opinião dos analistas e programadores, estes critérios apresentam um menor grau de importância.

Subfator: Modularidade da Documentação *(Critérios: Coesão das Informações, Acoplamento entre Seções, Estrutura da Documentação)*

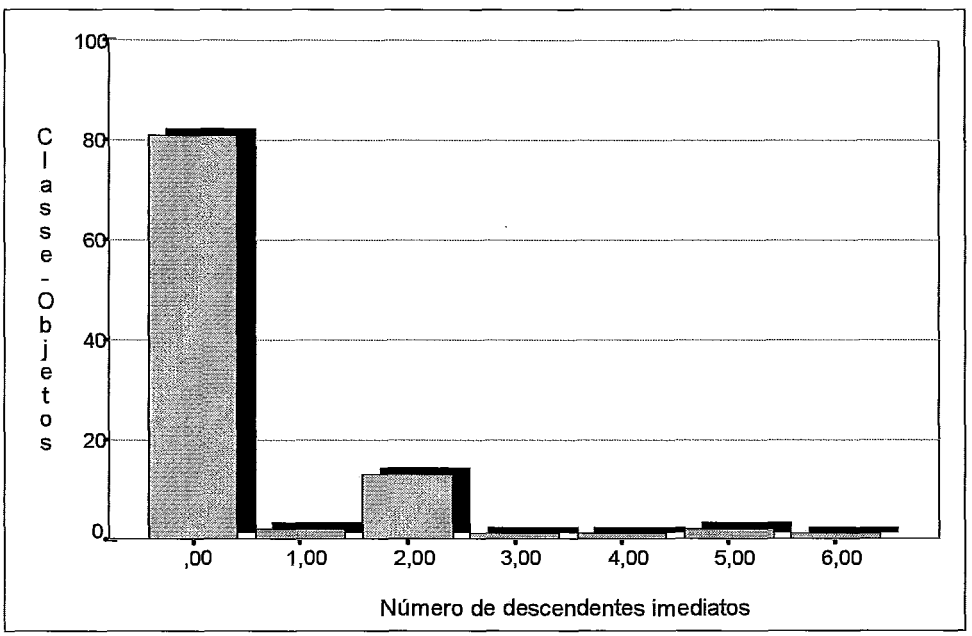
A avaliação mostrou que as especificações foram escritas em partes e sub-divididas em capítulos e seções. Sua estrutura, definida através de um sumário baseado no roteiro estabelecido para o desenvolvimento, forneceu uma clara visão do conteúdo das especificações. As informações descritas nas seções e parágrafos, além de serem detalhadas, descreveram aspectos relacionados, existindo um alto grau de independência entre as diferentes seções da especificação.

Como comentários a este resultado, pode-se dizer que a existência de um bom roteiro previamente estabelecido na definição do processo é de grande ajuda para se atingir estes critérios

Subfator: Correção da Arquitetura *(Critérios: Nível de Fatoração da Classe-Objetos, Nível de Profundidade da Hierarquia de Classes-objetos, Coesão Estrutural das Classes-objetos, Coesão Estrutural dos Clusters, Acoplamento de Relacionamento das Classes-objetos, Acoplamento de Interação das Classes-objetos, Acoplamento de Relacionamento dos Clusters, Acoplamento de Interação dos Clusters).*

O resultado da aplicação do *processo de avaliação* (NFC = Número de classes-objetos imediatas) para o critério *nível de fatoração da classe-objetos,* mostrou que, computando-se os dados de todas as classes-objetos do domínio, a maioria das classes-objetos (80%) não apresenta descendentes imediatos. Apenas 4% das classes-objetos apresentaram de 4 a 6 descendentes imediatos (Figura VII.1).

Pode-se comentar, a partir destes resultados, que algumas destas classes-objetos se tornaram mais flexíveis, pois forneceram serviços numa maior variedade de contextos, embora a reutilização através da herança não tenha sido um objetivo perseguido. Por outro lado, aspectos relacionados à compreensão e testes das classes-objetos do modelo não foram afetados por causa de classes-objetos com muitos descendentes.



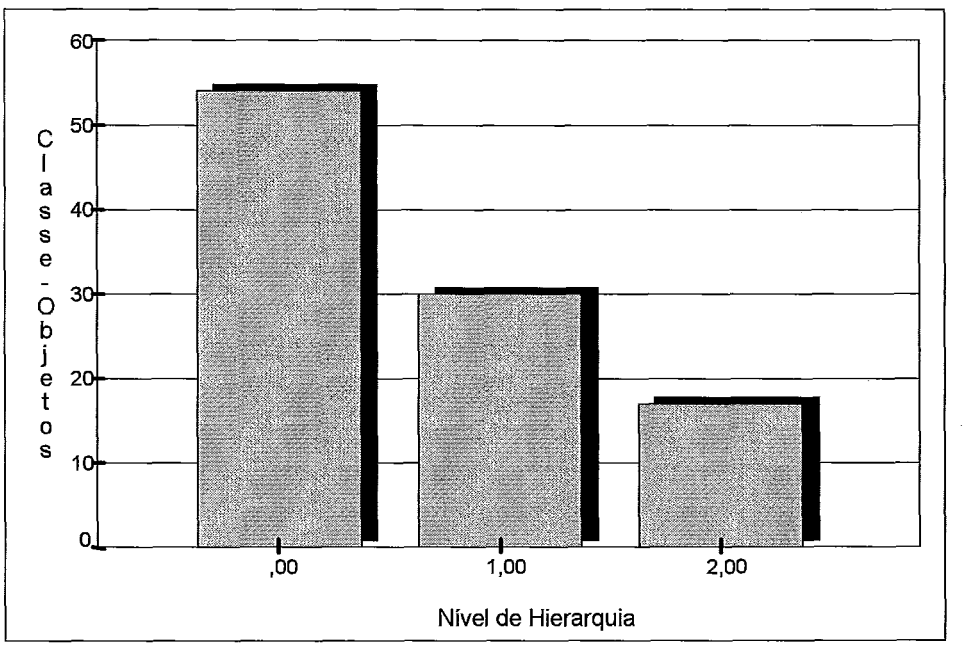
Critério	Média	Mediana	Máximo	Mínimo
NFC	0,505	0,241	6,000	0,000

Figura VII.1 - Nível de Fatoração das Classes-Objetos

Como resultado da aplicação do processo de avaliação para *NPC = nível de profundidade da classe-objetos,* percebe-se que 53% das classes-objetos que compõem o modelo de objetos são classes-objetos raiz (nível 0) ou não formam parte de estruturas de hierarquias de classes-objetos. Apenas 30% das classes-objetos tem um único nível de hierarquia e 17% estão alocadas num segundo nível. Ou seja, as classes-objetos tendem a estar perto da classe-objetos raiz (Figura VII.2). Observa-se, também, que os modelos têm hierarquias de classes-objetos pouco profundas, com um máximo de 2 níveis de abstração. Isto facilitou o entendimento da arquitetura total do sistema, pois as classes-objetos localizadas no segundo nível da hierarquia não apresentam o problema de herdar muitos serviços definidos por várias classes-objetos ancestrais. Portanto, foram obtidos valores menores que 2, para o nível de profundidade da hierarquia de classes-objetos.

O resultado da avaliação do critério *coesão estrutural das classes-objetos* $CEC(C_i) = (1 - M/N)$ onde M = número de conjuntos disjuntos, N = número de argumentos e C_i = identificação da classe-objetos (Figura VII.3), mostrou que, para

valores de coesão para uma amostra de classes-objetos de domínio e gerenciadores, as classes-objetos gerenciadores, apresentaram valores baixos para este critério. Isto pressupõe que estas classes-objetos realizam várias funções e atingem diferentes objetivos; conseqüentemente torna-se menos previsível determinar o seu comportamento. Os serviços nele especificados trabalham com uma variedade maior de argumentos, gerando uma maior quantidade de conjuntos disjuntos. Percebe-se também que, para algumas classes-objetos, o tamanho foi um indicativo do grau de coesão. Estes valores servem para identificar as classes-objetos que, possivelmente, podem ser subdivididas em classes-objetos menores. Cabe salientar que, para classes-objetos que apresentam poucos serviços, estes devem ser avaliados através do critério coesão semântica da classe-objetos, pois os valores obtidos para estes serviços resultaram pouco adequados.

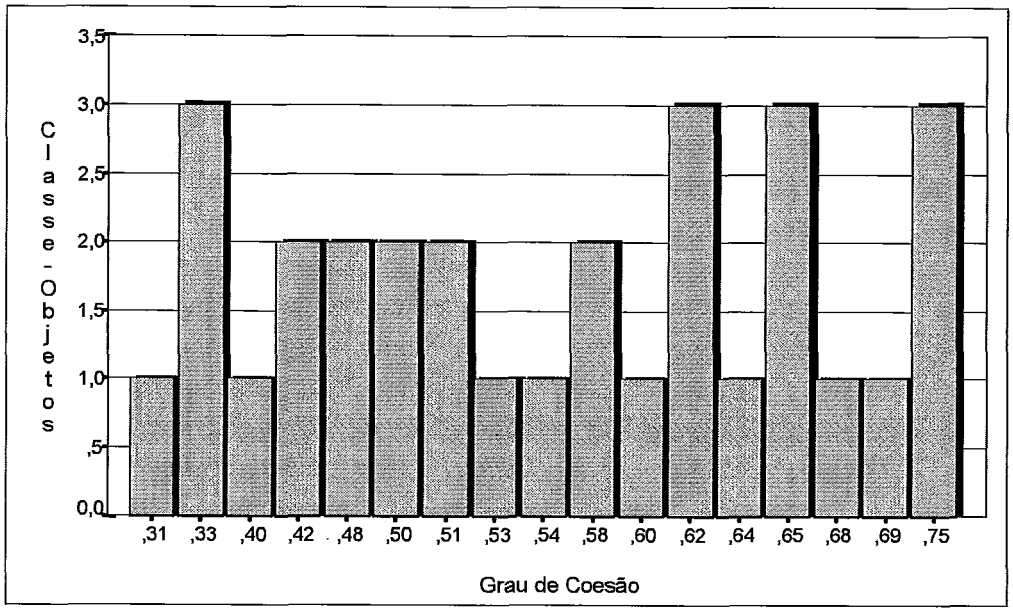


Variável	Média	Mediana	Mínimo	Máximo
NPC	0,634	0,000	0,000	2,000

Figura VII. 2 - Nível de Profundidade das Classes-Objetos

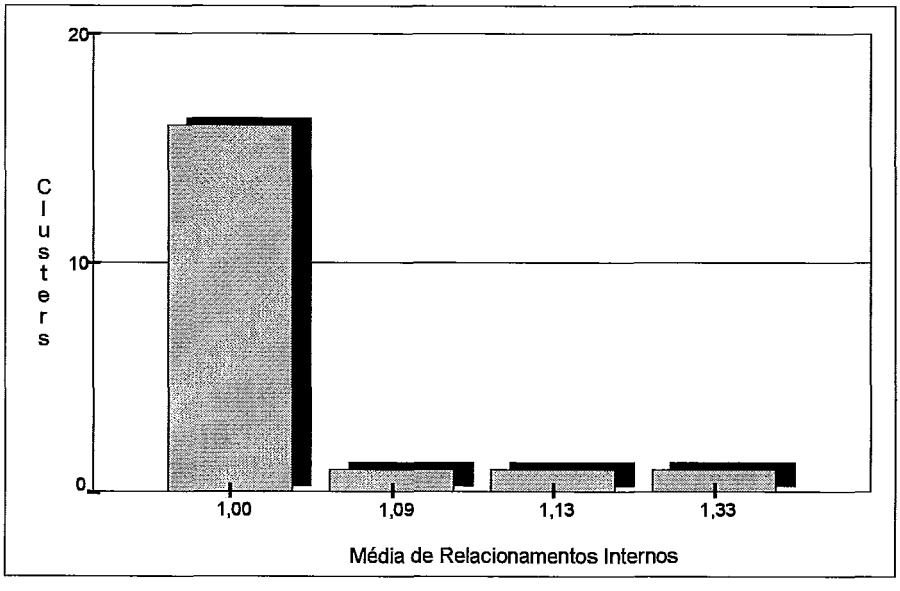
A avaliação do critério *coesão estrutural dos clusters* $CCL(CL_i) = (r + 1)/n$ onde CL_i = identificação do *cluster*, r = número de relacionamentos internos ao *cluster* e n = número de classes-objetos no *cluster* (Figura VII.4) mostrou que a grande maioria dos *clusters* apresenta valores mínimos para a coesão estrutural, isto é, possuem n-1 relacionamentos entre as classes-objetos que os compõem. Valores menores que 1 indicam pouco relacionamento entre as classes-objetos no *cluster*, sendo necessária uma revisão do *cluster*. Observa-se que 16% dos *clusters* estão compostos por classes-objetos que apresentam uma maior quantidade de relacionamentos internos. Isto indica,

de modo geral, que os *clusters* estão compostos por classes-objetos que fornecem serviços relacionados ao objetivo definido para o *cluster*.



Critério	Média	Mediana	Mínimo	Máximo
CEC	0,547	0,560	0,310	0,750

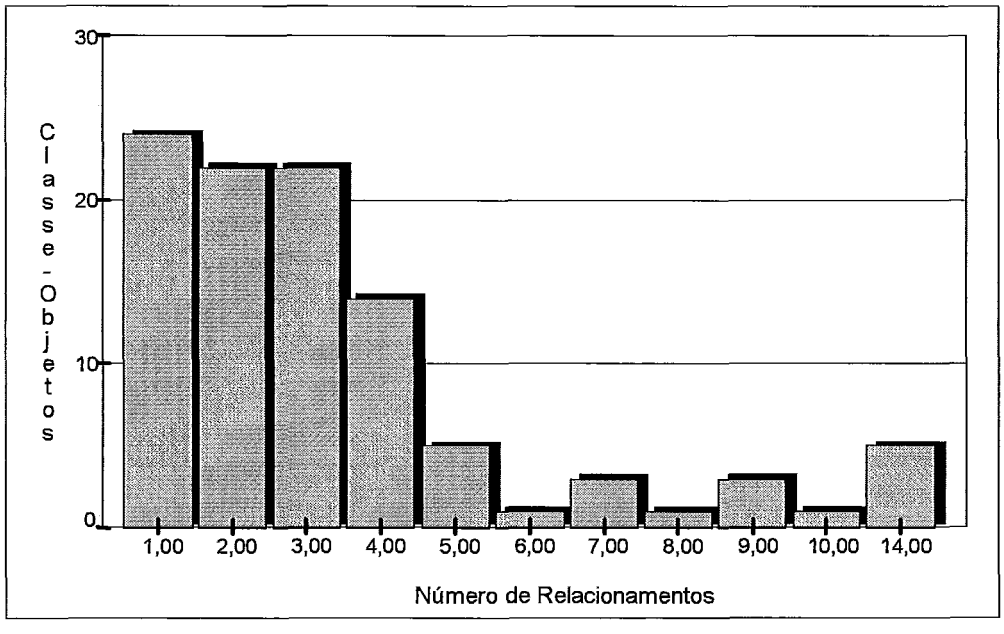
Figura VII.3 - Coesão Estrutural das Classes-Objetos



Critério	Média	Mediana	Mínimo	Máximo
CCL	1,029	1,000	1,000	1,330

Figura VII.4 - Coesão Estrutural dos *Clusters*

O resultado da avaliação segundo o critério *acoplamento de relacionamento das classes-objetos* $ACR(C_i) = \sum_{i=1}^{n} r_i$ onde, C_i = identificação da classe-objetos e r = número de relacionamentos com outras classes-objetos (Figura VII.5) mostrou que as classes-objetos apresentam em média 3,5 relacionamentos, o que representa 81% das classes-objetos do modelo. Isto indica que a maioria das classes-objetos apresentam relativamente pouca dependência estática. No entanto observa-se, também, que as classes-objetos com maior quantidade de relacionamentos desempenham responsabilidades mais complexas e possuem mais atributos e serviços, se comparadas com as classes-objetos que apresentam menos relacionamentos. Desta forma, foi posssível observar quais classes-objetos e que partes do modelo devem ser observadas com maior atenção, avaliando o grau de necessidade destes relacionamentos.

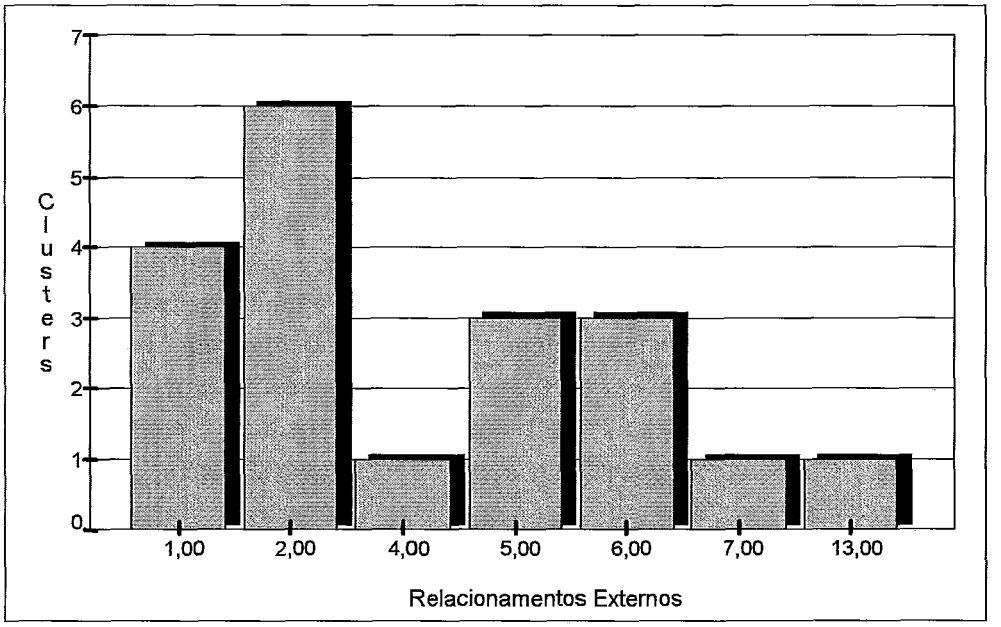


Critério	Média	Mediana	Mínimo	Máximo
ACR	3,535	3,000	1,000	14,000

Figura VII.5 - Acoplamento de Relacionamento das Classes-Objetos

O resultado da avaliação segundo o critério *acoplamento de relacionamento dos clusters* $ACRL(CL_i) = \sum_{i=1}^{n} ri$ onde CL_i = identificação dos *cluster* e r = número de relacionamentos de classes-objetos com outros *clusters* (Figura VII.6), mostrou que o número de relacionamentos externos aos *cluster* é bastante variado. A média obtida foi de quatro relacionamentos externos por cada *cluster.* Cabe salientar que os *clusters* que apresentam uma maior quantidade de relacionamentos (6, 7 e 13) possuem uma dependência estática com uma maior quantidade de outros *clusters* (4, 6). Além disso, o

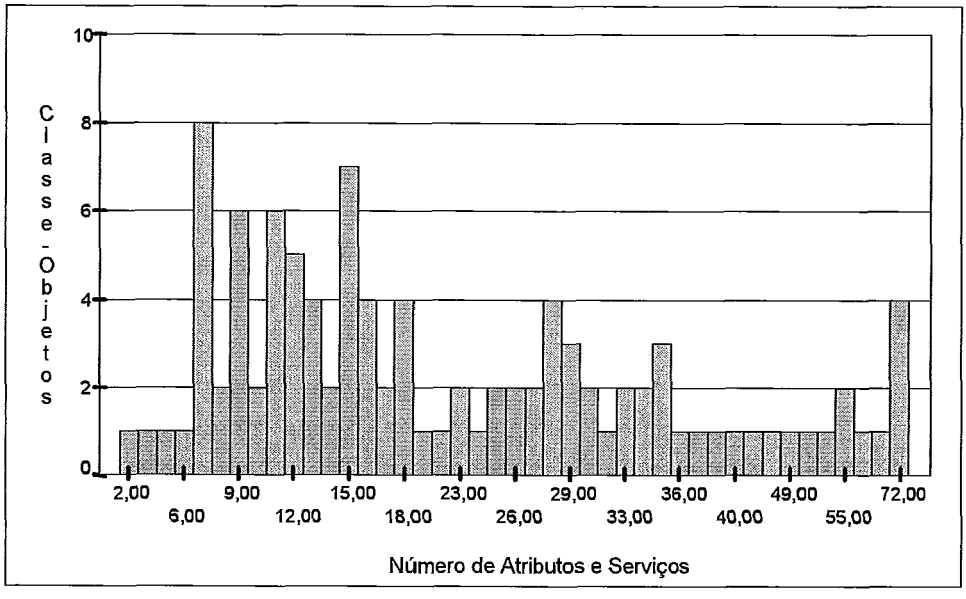
desenvolvimento e testes das classes-objetos pertencentes a estes *clusters* foi mais difícil, devido ao fato de apresentarem uma maior complexidade. A comunicação entre os grupos de desenvolvimento também foi afetada.



Critério	Média	Mediana	Mínimo	Máximo
ACRL	3,842	2,000	1,000	13,000

Figura VII.6 - Acoplamento de Relacionamento dos *Clusters*

O resultado da avaliação do critério *tamanho das classes-objetos*, utilizando o processo de avaliação $TCL(C_i) = NA + NS$; onde NA = número de atributos e NS = número de serviços, mostrou a existência de classes-objetos com uma quantidade variada de atributos e serviços (Figura VII.7). De modo geral, as classes-objetos do domínio da aplicação apresentaram um tamanho menor, se comparadas com as classes-objetos tipo gerenciadores e interfaces com usuários. O tamanho dos gerenciadores obedece ao fato de serem classes-objetos que realizam várias funções, pela própria natureza de suas responsabilidades, e das interfaces, pelo fato de que precisam ser especificados muitos atributos para se definirem de maneira adequada os recursos de interface necessários à aplicação. Pode-se comentar, também, que algumas classes-objetos que apresentam um valor de TC muito grande são classes-objetos que possuem maior número de relacionamentos, serviços mais complexos e uma coesão mais baixa. Conseqüentemente, foram classes-objetos candidatas a serem subdivididas em classes-objetos menores.

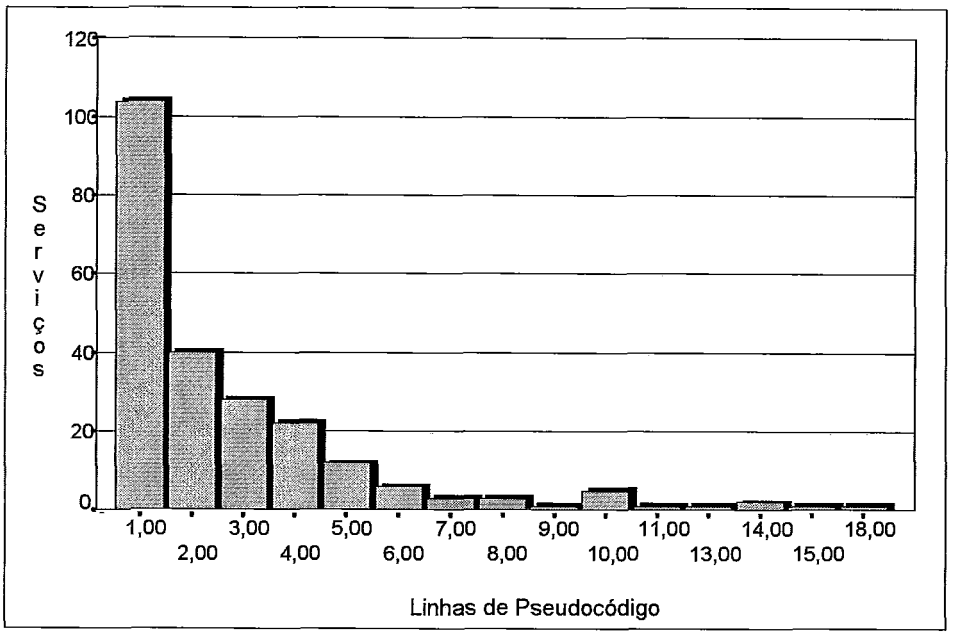


Critério	Média	Mediana	Mínimo	Máximo
TC	23,406	17,000	2,000	72,000

Figura VII.7 - Tamanho das Classes-objetos

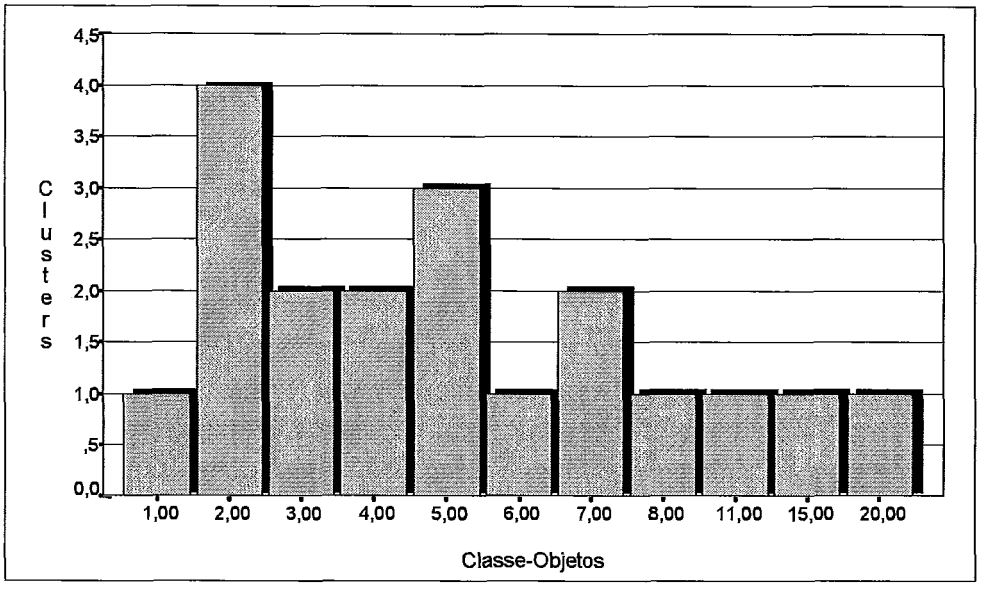
O resultado da avaliação segundo o critério *tamanho do serviço*, utilizando o processo de avaliação TS (S_i) = número de linhas de pseucodódigo, onde S_i = serviço definido na classe-objetos, mostrou que a grande maioria dos serviços (90%) contém de 1 a 5 linhas de pseudocódigo, com uma média de 3 linhas (Figura VII.8). Cabe salientar que estes serviços, em sua grande maioria, são serviços implícitos e serviços que implementam os relacionamentos entre classes-objetos (serviços de associação e desassociação). Os serviços referentes ao domínio da aplicação e os pertencentes às classes-objetos de gerenciamento apresentaram em média 11 linhas de pseudocódigo, pois estes serviços respondem a funções mais complexas. Por ser a linguagem de pseudocódigo definida para o projeto próxima à linguagem de implementação C++, alguns serviços apresentaram tamanhos muito grandes, sendo necessária, sua revisão, pois em certas ocasiões o pseudocódigo gerado orientou-se mais a funções do que a objetos.

O resultado da avaliação segundo o critério *tamanho do cluster*, utilizando o processo de avaliação TCL (CL_i) = número de classes-objetos, onde CL_i = identificação do *cluster*, mostrou que os *clusters* definidos no modelo apresentam uma quantidade variada de classes-objetos. Não se observou uma relação entre tamanho e complexidade nos *clusters*. No entanto, este critério teve relação com a distribuição de tarefas à equipe de desenvolvimento e conseqüentemente ao cumprimento dos cronogramas previstos para o projeto. Portanto, este critério torna-se útil no controle gerencial do desenvolvimento.



Critério	Média	Mediana	Mínimo	Máximo
TS	2,787	2,000	1,000	18,000

Figura VII.8 - Tamanho dos Serviços



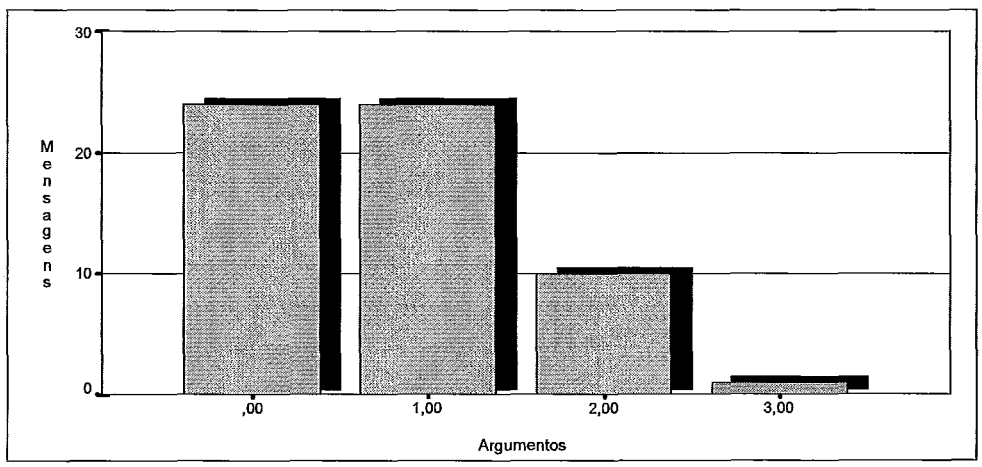
Critério	Média	Mediana	Mínimo	Máximo
TCL	5,895	5,000	1,000	20,000

Figura VII.9 - Tamanho dos *Clusters*

O resultado da avaliação do critério *tamanho das interfaces*, utilizando o processo de avaliação TI (M_i) = número de argumentos, onde M_i = identificação da mensagem, mostrou que, a partir das amostras coletadas dos serviços pertencentes a classes-objetos

de domínio e classes-objetos gerenciadores, a maioria das mensagens analisadas apresenta poucos argumentos (Figura VII.10). Este comportamento pode ser estendido às demais classes-objetos do modelo.

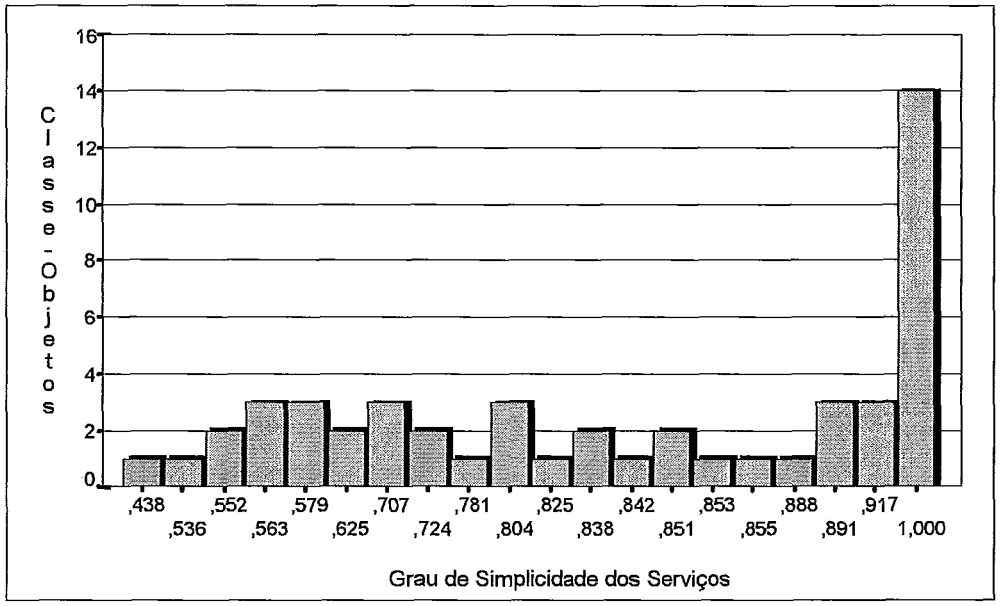
Além disso, podemos comentar, também, que se considerarmos aspectos de reutilização, a redução do número de argumentos para a geração de protocolos padrões não se faz necessária. Os resultados obtidos, ao avaliar este critério, estão dentro dos intervalos citados na literatura.



Critério	Média	Mediana	Mínimo	Máximo
TI	0,797	1,000	0,000	3,000

Figura VII.10 - Tamanho das Interfaces

O resultado da avaliação do critério *simplicidade dos serviços,* utilizando o *processo de avaliação* $SIS(Ci) = \sum_{i=1}^{n} si \ / \ n$, onde C_i = identificação da classe-objetos, s_i = valor associado ao grau de simplicidade de cada serviço e n = número de serviços na classe-objetos, mostrou que as classes-objetos têm em média um grau de simplicidade de 0,812 (Figura VII.11). Isto indica, de modo geral, que a maioria dos serviços especificados nas classes-objetos apresentam uma baixa complexidade, se consideramos as estruturas de controle utilizadas. As classes-objetos tipo gerenciadores apresentaram, de modo geral, uma complexidade maior se comparadas com as outras classes-objetos do modelo. Isto se deve ao fato de serem classes-objetos com serviços que interagem constantemente com classes-objetos de domínio e com as classes-objetos de interface. Por outro lado, os valores máximos de simplicidade são atribuídos, em sua maioria, aos serviços implícitos especificados nas classes-objetos.



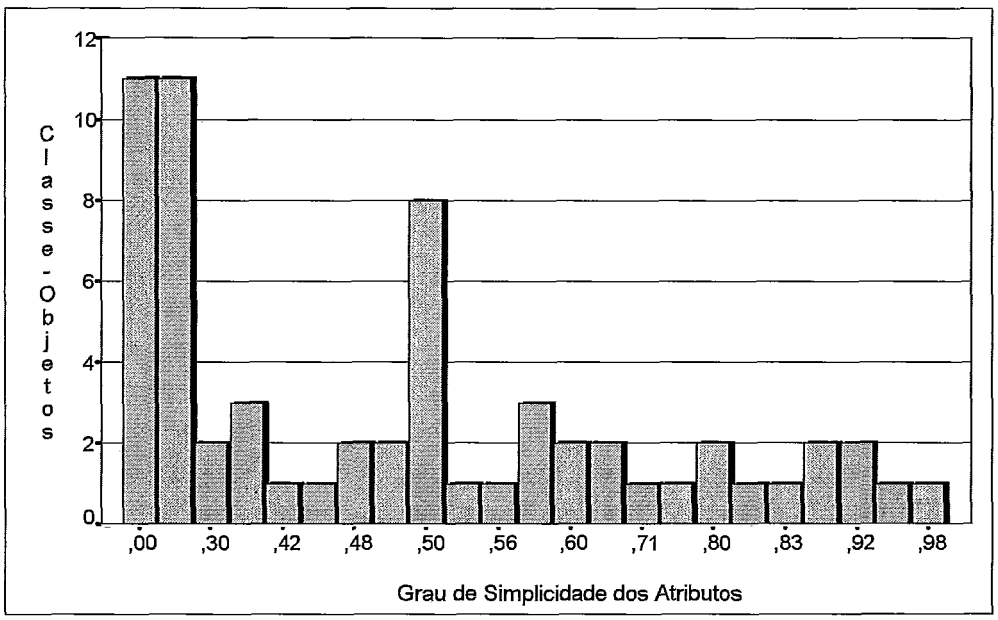
Critério	Média	Mediana	Mínimo	Máximo
SIS	0,812	0,847	0,438	1,000

Figura VII.11 - Simplicidade dos Serviços

O resultado da avaliação do critério *simplicidade dos atributos,* utilizando o processo de avaliação $SIA(Ci) = \sum_{i=1}^{n} ai \ / \ n$, onde C_i = identificação da classe-objetos, a_i = valor associado ao grau de simplicidade de cada atributo e n = número de atributos na classe-objetos, mostrou que as classes-objetos apresentam em média um grau de simplicidade com relação a seus atributos especificados de 0,440 (Figura VII.12). Observou-se, nos dados coletados, que a maioria dos atributos de domínio apresentam um alto grau de simplicidade, se consideramos sua estrutura. A complexidade dos atributos, numa determinada classe-objetos, aumentava pela existência de atributos de associação que foram implementados utilizando estruturas mais complexas (exemplo: listas, vetores). Na realidade, as classes-objetos que realizam o trabalho de gerenciamento das informações de domínio e interface com usuário apresentaram um grau de simplicidade menor, pois elas, de modo geral, possuem muitos atributos de associação.

Com relação aos critérios *acoplamento de interação das classes-objetos* e *acoplamento dos clusters*, não foi possível obter os dados necessários para calcular indicadores do grau de interação das classes-objetos e dos *cluster* no modelo. Isto se deve a que a seção da documentação da classe-objetos, onde se registra a interação com outras classes-objetos, através do envio e recebimento de mensagens, e seus serviços associados, foi difícil de ser mantida, pela própria dinâmica do processo de desenvolvimento. No entanto, com base na experiência, pode-se observar que as classes-objetos gerenciadores apresentam uma dinâmica de interação muito grande, pelo fato de

terem que interagir com classes-objetos de domínio e de interface. Por outro lado, as classes-objetos que possuem uma maior quantidade de relacionamentos estáticos interagem com uma maior quantidade de classes-objetos. Isto é estendido, também, quando são analisadas as interações entre *clusters*.



Critério	Média	Mediana	Mínimo	Máximo
SIA	0,440	0,486	0,000	0,980

Figura VII.12 - Simplicidade dos Atributos

Subfator: Conformidade *(Critérios: Aderência a Normas da Organização Desenvolvedora, Aderência a Normas estabelecidas pelo Contratante)*

O resultado da avaliação evidenciou que as especificações foram desenvolvidas obedecendo a normas definidas pela organização desenvolvedora e pela equipe de desenvolvimento. Isto ajudou a gerar uma melhor documentação e facilitou a comunicação entre os membros da equipe. Por outro lado, por ser este o primeiro projeto de desenvolvimento orientado a objetos realizado na organização contratante, não existiam normas estabelecidas pela mesma a serem consideradas.

Subfator: Disponibilidade *(Critérios: Acessibilidade, Estar Atualizada)*

O processo de avaliação mostrou que a preservação da disponibilidade das especificações, no que se refere à facilidade de acesso por usuários autorizados, e à sua atualização, exige providências adequadas durante cada modificação, resultado de

revisões e/ou atualizações. Em tais situações, as cópias existentes foram substituídas por versões mais atualizadas. Pode-se afirmar que a obediência a este critério se torna fundamental, como mostra a Figura VI.1, pela dinâmica do processo de desenvolvimento orientado a objetos.

Subfator: Rastreabilidade *(Critérios: Estrutura da Documentação, Localizabilidade Interna, Localizabilidade Externa)*

O resultado da avaliação indicou que a organização dos documentos gerados nas especificações permitiu uma hierarquização do mesmo, desde sua visão mais geral - *especificações de requisitos* - até uma visão mais detalhada - *especificações de projeto*. Para o caso das especificações de projeto, foi necessário, devido ao grande volume de classes-objetos, organizar os documentos de acordo com as tarefas definidas para cada subsistema. Portanto, cada documento apresentava seu próprio sumário. Pelo esforço gasto em manter atualizados estes documentos, não foram geradas facilidades para se localizarem todas as especificações e demais documentos relacionados a um determinado aspecto ou assunto.

VII.3.2 Avaliação da Confiabilidade Conceitual

Subfator: Consistência *(Critérios: Consistência Interna, Consistência Externa)*

O resultado da avaliação mostrou que, à medida em que o desenvolvimento avançava, os aspectos evoluíam através das diversas especificações. Conseqüentemente, o tratamento de conflitos de termos, características específicas do produto, conflitos lógicos, temporais ou relacionados a comportamento foram eliminados paulatinamente, como resultado de reuniões de inspeção ou através de reuniões internas entre os membros da equipe de desenvolvimento, principalmente quando os aspectos conflitantes estavam relacionados a decisões de projeto. Os quesitos definidos para avaliar estes critérios foram elaborados considerando os aspectos citados acima.

Subfator: Não Ambigüidade *(Critérios: Ser Explícita, Precisão)*

O resultado da avaliação mostrou que a especificação de requisitos exige que o usuário seja muito cuidadoso ao estabelecer requisitos, explicando com detalhes seus objetivos, sem que fossem admitidas suposições ou definições explícitas. Portanto, o documento de especificação apresentava, de forma explícita e precisa, os compromissos assumidos, considerando as restrições gerais e dependências que afetavam o produto.

Subfator: Correção Semântica da Arquitetura *(Critérios: Correção de Classes-objetos, Correção de Atributos, Correção de Serviços, Correção dos Relacionamentos de Associação, Correção dos Relacionamentos de Composição, Correção da Hierarquia de Classes-Objetos, Coesão Semântica das Classes-Objetos, Coesão Semântica dos Serviços, Coesão Semântica dos Clusters, Acoplamento de Herança)*

Os processos de avaliação definidos para classes-objetos, atributos, serviços, relacionamentos e estruturas de hierarquia de classes-objetos visam avaliar aspectos semânticos envolvidos na modelagem. Na realidade, para realizar a avaliação na fase de especificação de requisitos, faz-se necessária uma seleção de quesitos, para o caso específico dos critérios relacionados à correção de classes-objetos, atributos, serviços, relacionamentos e estruturas de hierarquia. Isto se deve ao fato de que o modelo de domínio nesta etapa, ainda não está completo o suficiente para responder a toda a lista de verificação. No entanto, na fase de projeto, todos estes quesitos podem ser avaliados. Por outro lado, os processos de avaliação definidos para os critérios coesão semântica, coesão dos serviços, coesão dos *clusters* e acoplamento de herança foram mais apropriados ao avaliar o modelo de projeto. A Figura VI.5 mostra que estes critérios apresentaram um grau de importância maior quanto de trata do modelo de projeto, confirmando este fato.

Subfator: Correção Comportamental *(Critérios: Correção dos Cénarios, Correção do Comportamento das Classes-objetos, Correção do Comportamento dos Clusters)*

Avaliar aspectos comportamentais num modelo de objetos torna-se muitas vezes uma tarefa dificil de ser realizada. Portanto, é preciso, ao longo do desenvolvimento, fazer uma seleção criteriosa das classes-objetos que apresentam comportamentos complexos e que, através de um diagrama de transição de estado, possam ser modeladas. Os processos de avaliação definidos para estes critérios consideram estes aspectos, definindo quesitos que avaliam a correção de cenários, comportamento das classes-objetos e de *clusters*. Observa-se que nas classes-objetos de interface estes diagramas se tornam imprescindíveis, pois é necessário realizar testes para verificar se os métodos definidos fornecem o comportamento esperado no momento da interação com o usuário.

Subfator: Necessidade *(Critérios: Necessidade dos Requisitos, Necessidade das Classes-objetos, Necessidade dos Atributos, Necessidade dos Serviços, Necessidade de Relacionamentos)*

Ao longo do desenvolvimento das especificações, foi de fundamental importância manter uma contínua avaliação da necessidade dos aspectos especificados, sejam estes requisitos, classes-objetos, atributos, serviços e relacionamentos. Conseqüentemente, os

processos de avaliação definidos para estes critérios se tornam necessários e adequados. Na fase de especificação de requisitos, para avaliar o grau de necessidade dos compromissos assumidos e na fase de projeto, para avaliar o grau de necessidade das abstrações que serão implementadas e/ou reutilizadas.

Subfator: Não Redundância *(Critérios: Não Redundância de Informações, Não Redundância de Classes-objetos)*

Os processos de avaliação definidos para estes critérios visam eliminar informações desnecessárias e redundância de classes-objetos. Como parte do documento de especificação foi escrito em linguagem natural, onde não existem restrições formais de vocabulário, sintaxe e semântica e, como na produção da especificação, profissionais de diversas áreas, torna-se necessário avaliar este critério, através de uma reunião de inspeção. Por outro lado, para avaliar redundância de classes-objetos, o processo de avaliação definido foi mais adequado para o caso do modelo de projeto.

Subfator: Completude *(Critérios: Completude com Relação ao Roteiro Definido pela Organização, Completude com Relação ao Método de Desenvolvimento, Completude com Relação aos Requisitos)*

O resultado da avaliação mostrou a necessidade de se ter um roteiro bem definido para construir as especificações. Embora este roteiro, ao longo do desenvolvimento, possa sofrer transformações visando adequar-se melhor às características específicas do projeto. Com relação ao método de desenvolvimento utilizado, os resultados da avaliação mostraram que este não forneceu um guia adequado para o desenvolvimento do projeto.

Como comentário a este resultado, pode-se afirmar que esta situação é bastante comum nos métodos orientados a objetos, sendo considerada uma característica pouco relevante (Figura VI.4). Na avaliação do critério completude com relação aos requisitos, os quesitos definidos no processo de avaliação mostraram que, no documento de especificação, estavam definidas todas as categorias de requisitos necessárias para o produto e que o modelo de classes-objetos respondia aos compromissos descritos na especificação.

VII.4 Conclusão

Neste capítulo foi descrita uma experiência de avaliação de uma especificação segundo os atributos de qualidade e os processos de avaliação definidos no Capítulo V e no Apêndice I. Os resultados obtidos nesta experiência demonstram a viabilidade de se

utilizarem estes atributos e seus respectivos processos de avaliação em projetos de desenvolvimento orientado a objetos. Além disso, os resultados e comentários fornecem subsídios aos desenvolvedores para a construção de especificações mais confiáveis, permitindo controlar sua qualidade ao longo do desenvolvimento.

Esta experiência de avaliação identificou, também, algumas dificuldades. Para o caso específico dos processos relativos ao modelo de classes-objetos, a ausência de uma ferramenta automatizada tornou a coleta de dados e o cálculo de valores um processo demorado e difícil, sendo necessário, na maioria dos casos, considerar amostras de dados. Pela quantidade e diversidade de processos de avaliação que deverão ser aplicados, torna-se necessário se ter definido, ao longo do desenvolvimento das especificações, os marcos e pontos de controle, bem como as técnicas de avaliação da qualidade que deverão ser utilizadas.

Um assunto relevante que merece ser mencionado é o relacionado à avaliação dos aspectos dinâmicos na orientação a objetos. As propostas para modelar os aspectos dinâmicos têm sido adaptadas de propostas existentes em outros paradigmas. Isto tem trazido dificuldades no processo de integração dos aspectos estáticos e dinâmicos e, conseqüentemente, na consistência entre os modelos gerados. Os atributos e processos de avaliação definidos neste trabalho, para o caso da avaliação dos aspectos dinâmicos, precisam de um maior refinamento para serem mais específicos, considerando a técnica utilizada pelo método.

Capítulo VIII

Conclusões e Perspectivas Futuras

"Lide com a dificuldade, enquanto ainda está fácil. Resolva os problemas maiores quando eles ainda são pequenos. Evite os problemas maiores. Dar pequenos passos é mais fácil do que resolvê-los. Por pequenas ações grandes coisas são atingidas".

Lao Tzu

Esta tese teve como principal objetivo definir atributos de qualidade e procedimentos para avaliação de software desenvolvido segundo o paradigma de orientação a objetos, considerando as fases de especificação de requisitos e especificação de projeto.

O trabalho realizado insere-se no contexto das pesquisas realizadas em qualidade de software pela linha de pesquisa de Engenharia de Software da *COPPE/UFRJ*, tendo sido, também, um dos projetos do Programa Brasileiro da Qualidade e Produtividade, no Subprograma Setorial da Qualidade e Produtividade em Software - *SSPQ/SW* (Weber 1996).

A realização deste trabalho iniciou-se com a definição de uma metodologia de pesquisa, como visto no Capítulo I, onde foram apresentados os objetivos gerais da tese e especificadas as atividades a serem desenvolvidas ao longo do trabalho (Clunie et al. 1995).

A partir da definição de um processo de geração de especificações, foi identificado, no Capítulo V, um conjunto de atributos de qualidade e foram definidos seus respectivos processos de avaliação que devem ser considerados para avaliar especificações orientadas a objetos (Clunie et al. 1994a; 1994b; 1994c; 1995a; 1996; 1996a; 1996b; 1997). Estes atributos de qualidade surgem a partir de trabalhos sobre qualidade de especificações descritos em (Clunie e Rocha 1987; 1987a; 1987b), da literatura técnica sobre qualidade em orientação a objetos, apresentada no Capítulo IV, dos métodos de desenvolvimento

orientados a objetos, discutidos no Capítulo III e da experiência de participação em um grande projeto de desenvolvimento orientado a objetos, onde está envolvida a *Fundação COPPE/UFRJ* (Clunie et al. 1995b; 1995c). A organização e discussão destes atributos foi realizada de acordo com o método de avaliação da qualidade, proposto por Rocha (1983).

Após a finalização destas atividades, foi realizada uma pesquisa de campo, descrita no Capítulo VI, entre profissionais envolvidos em projetos de desenvolvimento orientados a objetos com o intuito de estabelecer uma hierarquização dos atributos propostos com relação ao seu grau de importância, considerando a opinião de analistas e programadores.

Por fim, tendo como referência o projeto anteriormente mencionado, foi apresentada no Capítulo VII, uma experiência de avaliação de uma especificação orientada a objetos, segundo os atributos de qualidade definidos e seus respectivos processos de avaliação.

São contribuições deste trabalho:

- a particularização do Método Rocha de Avaliação da Qualidade de Software para o caso específico de software desenvolvido segundo o paradigma de orientação a objetos, para as fases de especificação de requisitos e de projeto, através da definição de atributos de qualidade e processos de avaliação específicos;

- a hierarquização, pelo grau de importância, dos atributos de qualidade identificados através da realização de uma pesquisa de campo, a partir da qual é possível delinear um perfil de qualidade para especificações orientadas a objetos;

- a experiência de avaliação de uma especificação orientada a objetos gerada num projeto real, de grande porte e complexo, segundo os atributos e processos definidos, que culminou numa série de observações sobre a qualidade de especificações orientadas a objetos; e

- um Manual para Avaliação da Qualidade de Especificações Orientadas a Objetos, contendo a definição dos atributos e processos para a avaliação da qualidade durante o processo de desenvolvimento do software.

Várias perspectivas para pesquisas podem ser sugeridas para dar continuidade a este trabalho:

- a utilização dos atributos de qualidade na avaliação de outros projetos de desenvolvimento orientado a objetos, com o intuito de aprimorar os processos de avaliação e obter indicadores de qualidade para especificações orientadas a objetos desenvolvidas em uma determinada organização;

- a realização de experimentos de validação, com o intuito de determinar o grau de relacionamento dos atributos de qualidade propostos e as atividades de testes e manutenção.

No contexto do Ambiente Memphis - *Ambiente de Desenvolvimento de Software Baseado em Reutilização,* em desenvolvimento na *COPPE/UFRJ* (Rocha et al. 1996), (Werner et al. 1996):

- o desenvolvimento de uma ferramenta baseada em conhecimento, que assista ao engenheiro de software na avaliação de especificações de requisitos e projeto produzidas utilizando métodos de desenvolvimento orientado a objetos;

- o desenvolvimento de uma ferramenta que auxilie na coleta de métricas e sua visualização, a partir dos modelos de classes-objetos que estão sendo gerados nas fases de especificação de requisitos e projeto.

Sabemos, entretanto, que o assunto não está ainda esgotado. Medidas em orientação a objetos ainda não estão sendo comumente utilizadas pela indústria, devido, principalmente, a este paradigma não ter, ainda, atingido a sua maturidade. Por outro lado, o uso da tecnologia de orientação a objetos está num processo de expansão, sendo já utilizada em aplicações complexas e críticas. Conseqüentemente, a urgência pela qualidade torna-se um aspecto fundamental.

Embora a literatura disponível sobre orientação a objetos ofereça formas de como utilizar os métodos orientados a objetos, estes métodos não fornecem um tratamento direto dos aspectos relacionados à qualidade. Assim sendo, formas de prevenção e correção de erros ainda não estão definidas. À medida em que a tecnologia evoluir, através do desenvolvimento de projetos em diferentes domínios de aplicação e do relato e discussão destas experiências, será possível identificar a origem e causa dos defeitos, formas para localizar e prevenir erros e

mecanismos eficientes para sua correção. Acreditamos que, para o caso da orientação a objetos, a uniformidade semântica desde a análise até a implementação deve ser melhor aproveitada, no sentido de que as atividades de validação - *testes* - possam ser realizadas mais cedo, tornando possível a identificação e correção de erros nas fases iniciais do processo de desenvolvimento.

Por fim, consideramos oportuno citar as palavras de Rocha em Weber et al. (1997), *"uma questão determinante do progresso da Engenharia de Software no Brasil e, como conseqüência, determinante da qualidade dos produtos de software é a necessidade de maior relação entre universidade e empresas. Através desta interação, será possível evidenciar questões relevantes à pesquisa e produzir uma verdadeira transferência de tecnologia. Como conseqüência, deverá levar a processos de desenvolvimento e de avaliação mais eficazes para a produção de software com a qualidade e produtividade desejadas"*.

Essa foi a nossa experiência. A participação em um projeto real, complexo e de grande porte foi determinante para a realização deste trabalho, pois nos permitiu vivenciar e refletir sobre os problemas que acontecem no desenvolvimento de software orientado a objetos.

Referências Bibliográficas

(Abreu 1992). F. B. Abreu , "As Métricas na Gestão de Projetos de Desenvolvimento de Sistemas de Informação", Em ***6as Jornadas para a Qualidade de Software,*** dezembro, Lisboa.

(Abreu e Carapuça 1994a). F. B. Abreu, R. Carapuça , "Candidate Metrics for Object-Oriented Software within a Taxonomy Framework", Em ***Journal of Systems and Software***, volume 26, no.1, pp. 87- 96.

(Abreu e Carapuça 1994b). F. B. Abreu, R. Carapuça, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", Em ***Proceedings of 4th International Conference on Software Quality***, McLean VA, USA, outubro.

(Agavanakis e Thrampoulidis 1994). K.N. Agavanakis, K. X. Thrampoulidis, "Object Interaction Diagram: A New Technique in Object-Oriented Analysis and Design", Em ***Journal of Object-Programming,*** A SIG Publications, vol 8, no.3, pp. 25-32, junho.

(Ambler 1994). S. W. Ambler, "In Search of a Generic SDLC for Object Systems", Em ***Object Magazine,*** vol 4, no.6, pp. 76 -78.

(Andrade 1991). C. J. Andrade, ***ANAFOR: Um Analisador de Programas FORTRAN;*** Tese de Mestrado, COPPE/UFRJ.

(Alagar e Kourkpoulos 1994). V. S. Alagar, D. Kourkopoulos, "(In)Completeness in Specifications", Em ***Information and Software Technology***, no. 36 (6), pp. 331-342.

(Archer e Stinson 1995). C. Archer, M. Stinson, ***Object-Oriented Software Measures,*** Technical Report SEI-95-TR-002, Pittsburgh, PA.

(Argila 1994). C. A. Argila, "Finding and Keeping Good Objects", In ***American Programmer***, pp. 36-43, outubro.

(Arnold et al. 1991). P. Arnold, Bodoff S., D. Coleman, H. Gilchrist, H. Hayes "Evaluation of five Object-Oriented Development Methods", Em ***Journal of Object-Programming: Focus on Analysis and Design***, *SIG Publication*, pp. 101 -121.

(Arthur 1994). L.J.Arthur , ***Melhorando a Qualidade do Software - Um Guia Completo para o TQM,*** IBPI Press, Rio de Janeiro.

(Ashely 1993a). N. Ashely, "The Introduction to METKIT", Em ***Information and Software Technology,*** volume 35, no.2, fevereiro, pp. 108 -110.

(Ashely 1993b). N. Ashley, "The METKIT Industrial Package", Em ***Information and Software Technology,*** volume 35, no.2, fevereiro, pp. 111-114.

(Bahia 1992). H. D. Bahia, ***Avaliação da Complexidade de Software Científico,*** Tese de Mestrado, COPPE/UFRJ.

(Bailin 1989). S. Bailin, "An Object-Oriented Requirements Specification Methods" Em ***Communications of the ACM,*** (32)(5), maio, pp. 609-623.

(Basili 1981). V. Basili, ***Data Collection, Validation and Analysis Software Metrics: An Analysis and Evaluation,*** Cambridge, MIT Press.

(Basili e Weiss 1984). V. Basili, D. Weiss, "A Methodology for Collecting Valid Software Engineering Data", Em ***IEEE Transactions on Software Engineering,*** volume 10, no 11, pp. 758 - 773.

(Basili e Rombach 1988). V. Basili, H. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environment", Em ***IEEE Transactions on Software Engineering,*** volume 14, no 6.

(Basili e Musa 1990). V. Basili, J.Musa, "The Future Engineering of Software: a Management Perspective", Em ***IEEE Computer,*** setembro.

(Basili et al. 1995). V. Basili, L. Briand, W. Melo, ***A Validation of Object-Oriented Design Metrics,*** Technical Report CS-TR-3443, University of Maryland, Dep. of Computer Science, abril.

(Basili et al. 1996). V. Basili , L. Briand, W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", Em ***IEEE Transactions on Software Engineering,*** vol 22, no.10, outubro, pp. 1-29.

(Basili et al. 1994). V. Basili, G. Caldeira, D. Rombach, "The Goal Question Metric Approach", Em ***Encycolpedia of Software Engineering***, University of Maryland, em ftp.cs.umd.edu/pub/sel/papers/gqm.ps.

(Bazzana et al. 1993). G. Bazzana, O. Anderson, Y. Jokela, "ISO 9126 and ISO 9000: Friends of Foes?", Em ***Proceedings of Software Engineering Standards Sympossium,*** Brigthon, United Kingdom.

(Beck e Cunninghan 1989). K. Beck, W. Cunninghan, "A Laboratory for Teaching Object-Oriented Thinking", Em ***OOPSLA '89 - Conference Proceedings,*** Special Issue of SIGPLAN Notices, 24(10), pp. 1-6.

(Beizer 1983). B. Beizer, ***Software Testing Techniques.*** Van Nostrand Reinhold.

(Belchior 1992). A. D. Belchior, ***Avaliação da Qualidade de Software Financeiro,*** Tese de Mestrado, COPPE/UFRJ.

(Bieman 1992). J. Bieman, "Deriving Measures of Software Reuse in Object Oriented Systems", Em ***Proceeding of BCS-FACS Workshop on Formal Aspects of Measurement,*** Springer-Verlag.

(Binder 1994). R. Binder, "Design for Testability in Object-Oriented Systems", Em ***Communications of ACM,*** vol 37, no. 9, pp. 101, setembro.

(Blaschek 1995). J. R. Blaschek, ***Planejamento de Sistemas de Informação,*** Tese de Doutorado, COPPE/UFRJ, Brasil.

(Boeg et al. 1993). J. Boeg, Hausen H. L, Welzel D, "A Practitioners Guide to Evaluations of Software", Em ***Proceedings of Software Engineering Standards Sympossium,*** Brigthon, United Kingdom.

(Boehm et al. 1978). B. W. Boehm, J. R. Brown, H Kaspar, M. Lipow, G. J. Macleod, M.J. Merritt, ***Characteristics of Software Quality,*** Amsterdam, North-Holland.

(Boehm 1986). B. W. Boehm, "A Spiral Model of Software Development and Enhancement", Em ***ACM Software Engineering Notes,*** 11(4), pp. 14 -24.

(Boehm 1984). B. W. Boehm, "Verifying and Validating Software Requirements and Design Specification", Em ***IEEE Software,*** volume, 1, no.1, janeiro.

(Boehm 1987). B. W. Boehm, "Improving Software Productivity", Em ***Computer,*** setembro.

(Boehm et al. 1995). B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, R. Selby, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", Em ***Annals of Software Engineering - Software Process and Product Measurement,*** Baltzer Science Publishers - vol 1.

(Booch 1991). G. Booch, ***Object-Oriented Design with Applications,*** Benjamin/Cummings Publishing Company, Inc.

(Booch 1994). G. Booch, ***Object - Oriented Analysis and Design with Applications***, Benjamin/Cummings Publishing Company, Inc.

(Bordoloi e Lee 1994). B. Bordoloi, M. Lee, "An Object-Oriented View: Produtivity Comparison with Structured Development" , Em ***Information Systems Management***, winter, pp. 22-30.

(Braga et al. 1995). L.O.Braga, A.R. Rocha, G. Campos, "Avaliação de Produto de Software Musical para fins Educacionais", Em ***Workshop de Qualidade de Software***, Recife - Brasil, outubro.

(Briand et al. 1994). L. Briand, S. Morasca, V. Basili, ***Defining and Validating High-Level Design Metrics***. Politecnico de Milano, Dipartamento di Elettronica e Informazione, Rapporto Interno 94.076, University of Maryland, Department of Computer Science, Technical Report CS-TR 3301.

(Brooks 1993). I. Brooks, "Object-Oriented Metrics Collection and Evaluation with a Software Process", Em ***OOPSLA'93, Workshop Program: Process and Metrics for Object-Oriented Software Development.***

(Brown 1991). P. G. Brown, "QED: Echoing the Voice of the Customer", Em ***AT&T Technical Journal***, março - abril, pp. 18 - 32.

(Bunge 1977). M. Bunge, ***Treatise on Basic Philosophy: Ontology I: The Furniture of the World***; Boston Riedel.

(Campos 1994). F. C. Campos, ***Hipermídia na Educação: Paradigma e Avaliação da Qualidade***, Tese de Mestrado, COPPE/UFRJ.

(Campos 1994). G. H. B. Campos, ***Metodologia para Avaliação da Qualidade de Software Educacional: Diretrizes para Desenvolvedores e Usuários***, Tese de Doutorado, COPPE/UFRJ.

(Capretz e Lee 1992). L. F. Capretz, P. A. Lee, "A Classification of the Object-Oriented Development Methodologies", Em ***Anais de VII Simpósio Brasileiro de Engenharia de Software***, Gramado, UFRGS.

(Champeaux e Faure 1992). D. Champeaux, P. Faure, "A Comparative Study of Object-Oriented Analysis Methods", Em ***Journal of Object-Oriented Programming***, 5(1): pp. 21-33, março/abril.

(Chen e Lu 1993). J-Y Chen, J-F Lu , "A New Metric for Object-Oriented Design", Em ***Information and Software Technology***, volume 35, no.4, abril, pp. 232 - 239.

(Chen 1976). P.P.S.Chen, "The Entity Relationship Model - Toward a Unified View of Data", Em *ACM Transaction on Data Base Systems*, (1), (1), pp. 9-36.

(Chidamber e Kemerer 1991). S. Chidamber, C. Kemerer, "Towards a Metrics Suite for Object - Oriented Design", Em *OOPSLA'91*, pp. 197-211.

(Chidamber e Kemerer 1993). S. Chidamber, C. Kemerer, *A Metrics Suite for Object-Oriented Design*. Working Papers, CISR WP No. 249, Sloan WP No. 3524-93, Massachusetts Institute of Technology.

(Chidamber e Kemerer 1994). S. Chidamber , C. Kemerer, "A Metrics Suite for Object-Oriented Design", Em *IEEE Transactions on Software Engineering*, 7(5), pp. 510-518.

(Churcher e Shepperd 1995a). N. I. Churcher, M. J. Shepperd, "Toward a Conceptual Framework for Object-Oriented Software Metrics", Em *Software Engineering Notes 20, 4*, abril, pp. 69-75.

(Churcher e Shepperd 1995b). N. I. Churcher, M. J. Shepperd, "Comment on A Metrics Suite for Object Oriented Design" Em *IEEE Transactions on Software Engineering*, volume 21, no. 3, pp. 263-265.

(Chonoles et al. 1995). M. J.Chonoles, J. Schardt, P. Magrogan, "Object-Oriented Systems from a Quality Perspective", Em *Report on Object Analysis Design*, vol 2, no.4, novembro-dezembro, pp. 46-52,

(Clunie 1987). C. E. Clunie, *Verificação e Validação de Software na Fase de Especificação de Requisitos*, Tese de Mestrado, COPPE/UFRJ.

(Clunie e Rocha 1987a), C. E. Clunie, A. R. Rocha, "Verificação e Validação de Software na Fase de Especificação de Requisitos" Em *XIII Conferencia Latinoamericana de Informática*, Bogotá - Colombia.

(Clunie e Rocha 1987b). C. E. Clunie, A. R. Rocha, *Manual para Controle da Qualidade de Especificações*, Relatório Técnico ES-10-87, COPPE/UFRJ.

(Clunie e Clunie 1988). C. E. Clunie, G. Clunie, *Metodología para el Desarrollo Estructurado de Sistemas*, Editorial Barriles, Panamá.

(Clunie e Werner 1994). C. E. Clunie, C. Werner, *Métodos de Desenvolvimento de Software Orientado a Objetos (M.D.O.O.) - Uma Abordagem Comparativa*, Relatório Técnico ES-329-94, COPPE/UFRJ.

(Clunie et al. 1994a). C. E. Clunie, C. Werner, A.R. Rocha, ***Avaliação da Qualidade de um Modelo de Análise e Projeto Orientado a Objetos,*** Relatório Técnico ES-328-94, COPPE/UFRJ.

(Clunie et al. 1994b). C. E. Clunie, C. Werner, A.R. Rocha, "Avaliação da Qualidade de um Modelo Orientado a Objetos", Em ***XX Conferencia Latinoamericana de Informática,*** México D.F.

(Clunie et al. 1994c). C. E. Clunie, C. Werner, A. R. Rocha, "Critérios de Qualidade para um Modelo Orientado a Objetos", Em ***Workshop de Qualidade de Software,*** Curitiba, Brasil.

(Clunie e Werner 1995). C . E. Clunie, C. Werner, A.R. Rocha, "Qualidade de Especificações de Requisitos e de Projeto Orientadas a Objetos: Definição e Avaliação", Em ***Workshop de Qualidade de Software,*** Recife, Brasil.

(Clunie et al. 1995a). C. E. Clunie, C. Werner, A. R. Rocha, "Metrics of Quality and their Relation with the Object-Oriented Modeling", Em ***Proceedings of the 5th.International Symposium on Systems Research, Informatics and Cybernetics*** - *ISAS,* Baden-Baden, Alemanha.

(Clunie et al. 1995b). C. E. Clunie, R. M. Araujo, C. M. Werner, A. R. Rocha, G. H. Travassos, "Una Experiencia en el Desarrollo de un Sistema Orientado a Objetos", Em ***Proceedings of the II International Congress on Information Engineering - ICIE' 95,*** Buenos Aires, Argentina.

(Clunie et al. 1995c). C. E. Clunie, R. M. Araujo, C. M. Werner, A. R. Rocha; G. H. Travassos, "Uma Experiência no Desenvolvimento de um Sistema utilizando o Método de Coad e Yourdon", Em ***I Workshop de Orientação a Objetos,*** COPPE/UFRJ, dezembro.

(Clunie e Xexeo 1996). C. E. Clunie, G. Xexeo, ***Avaliação da Qualidade de um Modelo Orientado a Objetos: Uma Abordagem com Lógica Nebulosa,*** Relatório Técnico ES 371/96, COPPE/UFRJ.

(Clunie et al. 1996). C. E. Clunie, C. M. Werner, A. R. Rocha, "How to Evaluate the Quality of Object-Oriented Specification", Em ***Proceedings of the 6th. International Conference of Software Quality,*** Canada, outubro, pp. 283-293.

(Clunie et al. 1996a). C. E. Clunie, C. M. Werner, A. R. Rocha, "Avaliação da Qualidade de Especificações Orientadas a Objetos", Em ***Workshop de Teses em Engenharia de Software, X Simposio Brasileiro de Engenharia de Software,*** San Carlos, São Paulo, outubro, pp. 43-44.

(Clunie et al. 1996b). C. E. Clunie, C. M. Werner, A. R. Rocha, "Qualidade de Especificações Orientadas a Objetos" Em ***II Workshop de Orientação a Objetos***, COPPE/UFRJ, dezembro.

(Coad e Yourdon 1992). P. Coad, E. Yourdon, ***Análise Baseada em Objetos***, Editora Campus, Rio de Janeiro.

(Coad e Yourdon 1993). P. Coad, E. Yourdon, ***Projeto Baseado em Objetos***, Editora Campus, Rio de Janeiro.

(Coad et al. 1995). P. Coad, D. North, M. Mayfield, ***Strategies, Patterns, and Applications***, Yourdon Press, Englewood Cliff, New Jersey.

(Collins et al. 1994). W. R. Collins, K. W. Miller, B. J. Spielman, P. Wherry, "How Good is Good Enough? An Ethical Analysis of Software Construction and Use", Em ***Communications of the ACM***, janeiro.

(Comerlato 1994). C. Comerlato, ***Avaliação da Reutilizabilidade de Componentes de Software***, Tese de Mestrado, COPPE/UFRJ.

(Conte et al. 19986). S. D. Conte, H. E. Dunsmore, V. Y. Shen, ***Software Engineering Metrics and Models***, Benjamin / Cummings, Menlo Park, C.A.

(Coppick e Cheatham 1992). Ch. Coppick, T. Cheatham, "Software Metrics for Object-Oriented Systems", Em ***Proceedings: ACM CSC " 92 Conference***, Kansas City, Misouri, março 3-5, New York, pp. 317-322.

(Cox 1991). B. Cox, ***Programação Orientada a Objetos***, Editora McGraw-Hill.

(Curtis 1981). B. Curtis, ***The Measurement of Software Quality and Complexity - Software Metrics: An Analysis and Evaluation***, Cambridge, MIT Press.

(Davis 1990). A. Davis, ***Software Requirements - Analysis & Specifications***, Englewood Cliff, New Jersey, Prentice-Hall.

(Davis 1993). A. Davis, ***Software Requirements: Object, Functions, and States***, Englewood Cliff, New Jersey, Prentice-Hall.

(Davis 1995). A. Davis, "Object-Oriented Requirements to Object-Oriented Design: An Easy Transition?", Em ***Journal Systems and Software***, volume 30, julho-agosto, pp. 151-159.

(DeMarco 1978). T. DeMarco, ***Structured Analysis and Systems Specification***, Prentice - Hall.

(DeMarco 1989). T. DeMarco, ***Controle de Projetos de Software***, Editora Campus, Rio de Janeiro.

(Denicoff e Grafton 1981). M. Denicoff, M. R. Grafton, ***Software Metrics a Research Iniciative - Software Metrics: An Analysis and Evaluation***, Cambridge, MIT.

(Dromey 1995). G. Dromey, "A Model for Software Quality Product Quality", Em ***IEEE Transaction on Software Engineering***, volume 21, no.2, fevereiro, pp 146-162.

(Dromey 1996). G. Dromey, "Cornering the Chimera", Em ***IEEE Software***, janeiro, pp. 33- 43.

(Duke et al. 1991). R. Duke, P. King, F. Rose, F. Smith, ***The Object-Z Specification Language***, Technical Report 91-1 (Version 1), University of Queensland, Department of Computer Sciences, Software Verification Research Centre, maio.

(Dorling 1993). A. Dorling, "SPICE: Software Process Improvement and Capability Determination", Em ***Information and Software Technology***, vol. 35, no. 6.

(Dyer 1987). M. Dyer, "A Formal Approach to Software Error Removal", Em ***Journal Systems and Software***, volume 7.

(Ejiogu 1991). L. Ejiogu, ***Software Engineering with Formal Metrics***, QED Technical Publishing Group.

(Ejiogu 1993). L. Ejiogu, "Five Principles for the Formal Validation Models of Software Metrics", Em ***ACM SIGPLAN Notices***, 28(8), pp. 67 -76.

(Fenton 1991). N. E. Fenton, ***Software Metrics: A Rigorous Approach.*** Chapman & Hall.

(Fenton 1994). N. E. Fenton, "Software Measurement: A Necessary Scientific Basis", Em ***IEEE Transaction on Software Engineering***, volume 20, no.3, pp. 199-206.

(Fichman e Kemerer 1992). R. G. Fichman, CH. Kemerer, "Object-Oriented and Conventional Analysis and Design Methodologies", Em ***IEEE Computer***, outubro.

(Firesmith 1993). D. Firesmith, ***Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach***, Editora John Wiley & Son.

(Firesmith 1994). D. Firesmith, "Clusters of Classes: A Bigger Building Block", Em ***ROAD***, volume 1, no.4, novembro-dezembro.

(Fowler 1993). M. Fowler, "Position Paper on Process and Metrics for OO A&D", Em ***OOPLA'93 Workshop Program: Process and Metrics for Object-Oriented Software Development.***

(Frakes e Terry 1996). W. Frakes, C. Terry, "Software Reuse: Metrics and Models", Em ***ACM Computing Surveys***, vol. 28, no.2, junho, pp. 415 - 435.

(Gane e Sarson 1979). Ch. Gane, T. Sarson, ***Structured Systems Analysis: Tools and Techniques***, Prentice-Hall.

(Ghezzi et al. 1991). C. Ghezzi, M. Jazayeri, D. Mandriole, ***Fundamentals of Software Engineering***. Prentice-Hall.

(Gilb 1988). T. Gilb, ***Principals Software Engineering Management***, Addison- Wesley, 1988.

(Garvin 1984). D. Garvin, "What Does Produce Quality Really Mean?" ***Sloan Management Review***, 1984, pp. 25- 45.

(Gillies 1992). A. Gillies, ***Software Quality - Theory and Management***, Chapman & Hall.

(Gilpin 1993). M. Gilpin, "A Comparison of Object Oriented Analysis and Design Method ", Em ***Case World***, março.

(Grady e Caswell 1987). R. Grady, D. Caswell, ***Software Metrics: Establishing a Company Wide Program***, Prentice-Hall.

(Gustafson et al. 1993). D.Gustafson, T. Tan, P. Weaver, "Software Measure Specification", Em ***Proceedings of the First ACM Sigsoft Symposium on the Foundations of Software Engineering***, Los Angeles.

(Halstead 1977). M. H. Halstead, ***Elements of Software Sciences***. Elsevier North - Holland, New York.

(Hasia et al. 1994). P. Hasia, S. Jararajan, S. Gao, D. Kung, "Formal Approach to Scenario Analysis", Em ***IEEE Software***, março, pp. 33-40.

(Hausen e Welzel 1993). H. Hausen, D. Welzel, ***Guides to Software Evaluation, Arbeitspapiere der GMD 746***, Projeto SCOPE, abril.

(Henderson-Sellers e Edward 1993). B. Henderson-Sellers, J. M. Edward, "The Fountain Model for Object Oriented Systems Development", Em ***Object Magazine***, vol 3, no. 2, pp. 71- 79.

(Henderson-Sellers 1994). B. Henderson-Sellers, "Methodologies-Framework for OO Success", Em ***American Programmer***, outubro, pp. 1-9.

(Henderson-Sellers 1995). B. Henderson-Sellers, "OO Metrics Programme", Em ***Object Magazine***, outubro, pp. 73-79.

(Henderson-Sellers 1996). B. Henderson-Sellers, ***Object-Oriented Metrics - Measures of Complexity***, Prentice-Hall, Inc.

(Hetzel 1995). W. Hetzel, "The Sorry State of Software Pratice Measurement and Evaluation", Em ***Journal Systems and Software***, vol 31, pp. 171 - 179.

(Henry e Kafura 1984). S. Henry, D. Kafura, "The Evaluation of Software System's Structure Using Quantitative Software Metrics", Em ***Software - Practice and Experience***, volume 14, no. 16, pp. 561 - 573.

(Henry e Kafura 1981). S. Henry, D. Kafura, "Software Structure Metrics Based on Information Flow ". Em ***IEEE Transactions on Software Engineering***, 7(5), pp. 510 -518.

(Hitz e Montazeri 1996). M. Hitz, B. Montazeri, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective". **Em** ***IEEE Transactions on Software Engineering***, vol 22, no.4, abril, pp. 267- 270.

(Holdsworth 1992). J. Holdsworth, "The AMI Project: Validating Quantitative Approaches to Software Management", Em ***Proceedings Eurometrics '92 European Conference on Quantitative Evaluation of Software and Systems: Practical and Theorical Issues***, Brussels, Belgium (13-15), pp. 125-134.

(Hutt 1995a). A. Hutt, ***Object Analysis and Desing - Description of Methods***, OMG - Object Management Group, Jonh Wiley & Sons, Inc.

(Hutt 1995b). A. Hutt, ***Object Analysis and Desing - Comparison of Methods***, OMG - Object Management Group, Jonh Wiley & Sons, Inc.

(IEEE 1984). IEEE, ***IEEE Guide to Software Requirements Specifications***, Computer Society New York.

(Ince 1990). D. Ince, "Software Metrics: Introduction", Em ***Information and Software Technology***, volume 32, no. 4, pp. 297-303.

(Ingalls 1990). D. Ingalls, "Design Principles Behind Smalltalk", Em ***Tutorial Object-Oriented Computing - Concepts***, volume 1, Ed. Gerald Peterson.

(ISO 8402 1990). ISO, "Generic Terms and Definition", Em ***ISO/CD 8402 - Quality Concepts and Terminology - Part 1***, dezembro.

(ISO 9000 1987). ISO, "Quality Management and Quality Assurance Standards - Guidelines for the Application of ISO 9000" , Em ***ISO 9000***, setembro.

(ISO 9000-3 1990). ISO, "Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software". Em ***ISO 9000-3***, setembro.

(ISO/IEC 9126 1991). ISO/IEC, "Information Technology - Software Evaluation - Quality Characteristics and Guidelines for the Use", Em ***ISO/IEC 9126***, dezembro.

(Jacobson 1992). I. Jacobson, ***Object-Oriented Software Engineering***, Addison Wesley.

(Jones 1991). T. C. Jones, ***Applied Software Measurement: Assuring Productivity and Quality***, McGraw -Hill.

(Johnson 1994). A. Johnson, "Software Process Improvement Experience in the DP/MIS Function" , Em ***Proceedings of the 16th International Conference on Software Engineering***, Sorrento, Italia, maio.

(Jonhson e Foote 1988). R. Jonhson, B. Foote, "Designing Reusable Classes", Em ***Journal of Object Oriented Programming***, vol. 1, no.2.

(Juran e Gryna 1991). J. M. Juran, F.M. Gryna, ***Controle da Qualidade - Handbook***. Makron Books, Rio de Janeiro.

(Kafura e Henry 1981). D. Kafura, S. Henry, "Software Quality Metrics Based on Interconnectivity ", Em ***Journal Systems and Software***, volume 2, no.2, pp. 121-131.

(Kumar 1995). K. Kumar, "A Practical Strategy for OO Design", Em ***Dr. Dobb's Journal***, vol 231, junho, pp. 34 - 39.

(Kaposi e Myers 1990). A. Kaposi, M. Myers, "Quality Assuring Specifications and Design ", Em ***Software Engineering Journal***, London.

(Karam e Casselaman 1993). G. M. Karam, R.S. Casselaman, "Cataloging Framework for Software Development Methods", Em ***IEEE Computer***, 26(2): pp. 35 - 46, março.

(Karlsson 1995) . E. Karlsson (ed), ***Software Reuse***, John Wiley & Sons.

(Karunanithi e Bieman 1993). S. Karunanithi, J. Bieman, "Candidate Reuse Metrics for Object Oriented and Ada Software", Em ***Proceedings of IEEE International Software Metrics Symposium***, pp. 120 - 128.

(Kitchenaham 1987). B. A. Kitchenham, "Towards a Construtive Quality Model. Part I: Software Quality Modeling, Measurement and Prediction", Em ***Software Engineering Journal***, volume 2, no.4, pp. 105-113.

(Kitchenham e Plfeeger 1995). B. Kitchenham, S. Pfleeger, "Towards a Framework for Software Measurement Validation", Em ***IEEE Transaction on Software Engineering***, vol 21, no. 12, pp. 929-943.

(Kitchenham e Pfleeger 1996). B. Kitchenham, S. Pfleeger, "Software Quality: The Elusive Target ", Em ***IEEE Software***, janeiro, pp. 12 - 21.

(Kitson e Masters 1993). D. H. Kitson, S. M. Masters, "An Analysis of SEI Process Assesment Results: 1987 - 1991", Em ***Proceedings of the 15th International Conference on Software Engineering***, Baltimore, Maryland, maio.

(Kival et al. 1997). Ch. W. Kival, J. C. De Luca, A. R. Rocha, ***Qualidade e Produtividade em Software*** - *Termo de Referência do SubPrograma Setorial da Qualidade e Produtividade em Software, do Programa Brasileiro da Qualidade e Produtividade* - ***PBQP***, Makron Books, Brasil.

(Knight e Myers 1993). J. C. Knight, E. A. Myers, "An Improved Inspection Technique", Em ***Communication of the ACM***, volume 36, no. 11, pp. 51-61.

(Knight e Myers 1993). J. Knight, A. Myers, "An Improved Inspection Technique", Em ***Communications of the ACM***, novembro.

(Kogure e Akao 1983). M. Kogure, Y. Akao, "Quality Function Deployment and CWQC", Em ***Japan Quality Progress***, pp. 25 -29.

(Kolewe 1993). R. Kolewe, "Metrics in Object-Oriented Design and Programming", Em ***Software Development***, outubro, pp. 53-62.

(Korson e McGregor 1992). T. Korson, J. D. McGregor, "Technical Criteria for the Specification and Evaluation of Object-Oriented Libraries", Em ***Software Engineering Journal***, março 1992, pp. 85 - 94.

(Lano e Haughton 1994). K. Lano, L. Haughton, ***Object-Oriented Specification CASE Studies***, Prentice-Hall.

(Lattanzi e Sallie 1994). M. Lattanzi, H. Sallie, ***Object Oriented Metrics: Generation and Application***, Technical Report No. 94-20, Computer Science Department, Virginia Tech.

(Leite 1990). J. C. S. Leite, "Validação de Requisitos: O Uso de Pontos de Vista" , Em ***Revista Brasileira de Computação***, vol no.6, no. 2, outubro-dezembro, pp 39-52.

(Li e Henry 1993a). W. Li, S. Henry, "Object-Oriented Metrics that Predict Maintainability", Em ***Journal Systems and Software***, volume 23, pp. 111-122.

(Li e Henry 1993b). W. Li, S. Henry, "Maintenance Metrics for the Object Oriented Paradigm", Em ***Proceedings of the First International Software Metrics Symposium***, Los Alamitos, CA, pp. 52 - 60.

(Li et al. 1995). W. Li, S. Henry, D. Kafura, R. Schulman, "Measuring Object-Oriented Design", Em ***Journal Object Oriented Programming***, julho - agosto, pp. 48-55.

(Lindland et al. 1994). O. Lindland, G. Sindre, A. Solverg, "Understanding Quality in Conceptual Modelling", Em ***IEEE Software***, março.

(Linger 1994). R. Linger, "Cleanroom Process Model", Em ***IEEE Software***, março, pp.50- 58.

(Linkman e Walker 1991). S. Linkman, J.G. Walker, "Controlling Programmes Through Measurement", Em ***Information and Software Technology***, volume 33, no.1, pp. 93-102.

(Lorenz 1993). M. Lorenz, ***Object-Oriented Software Development: A Practical Guide***, Prentice-Hall, Englewood Cliff, New Jersey.

(Lorenz e Kidd 1994). M. Lorenz, J. Kidd, ***Object-Oriented Software Metrics***, PTR Prentice Hall, Englewood Cliff, New Jersey.

(Love 1991). T. Love , "Timeless Design of Information Systems", Em ***Object Magazine***, nov/dez.

(Love 1993). T. Love, ***Object Lessons - Lessons Learned in Object-Oriented Development Projects***, SIGS Books, New York.

(MacLennan 1990). B. J. MacLennan, "Values and Objects in Programming Languages", Em ***Tutorial Object-Oriented Computing - Concepts***, volume 1, Ed. Gerald Peterson.

(Maffeo 1992). B. Maffeo, ***Engenharia de Software e Especificação de Sistemas***, Editora Campus, Rio de Janeiro.

(Mann e Coleman 1988). T. Mann, M. Coleman, ***Software Quality Assurance***, McMillan Education.

(Martin 1995). C. Martin, ***Designing Object-Oriented C++. Application Using The Booch Method***, Prentice - Hall, Inc.

(Martin W. 1996). W. Martin, "Taking the Measure of Metrics", Em ***Object Expert***, volume 1(2), janeiro-fevereiro, pp. 43-45.

(Martin e Odell 1994). J. Martin, J. Odell, ***Princípios de Análise e Projeto Baseados em Objetos***, Editora Campus, Brasil.

(McCabe 1976). T. J. McCabe, - "A Complexity Measure", Em ***IEEE Transaction on Software Engineering***, volume 2, no.4, pp. 308 -320.

(McCall 1979). J. A. McCall, ***An Introduction to Software Quality Metrics***, Software Quality Management, Petrocelli.

(McCall e Cavano 1978). J. A. McCall, J. P. Cavano, "A Framework for the Measurement to Software Quality", Em ***Proceedings of the ACM Software Quality Assurance Workshop***, novembro.

(McCall et al. 1979). J. McCall, J. Cooper, M. J. Fisher, "An Introduction to Software Quality Metrics", Em ***Software Quality Management***, Petrocelli.

(McGregor 1995). J. McGregor, "Managing Metrics in a Iterative Environment". Em ***Object Magazine***, outubro, pp. 65-71.

(McMenamin e Palmer 1984). S. M. McMenamin, F. Palmer, ***Essencial Systems Analysis***, Yourdon Press.

(Melo et al. 1995). W. Melo, L. Briand, V. Basili, ***Measuring the Impact Reuse on Quality and Produtivity in Object-Oriented Systems*** - Technical Report CS-TR-3395, University of Maryland, Dept of Computer Sciences.

(Meyer 1988). B. Meyer, ***Object-Oriented Software Construction***, Prentice-Hall, Englewood Cliffs, New Jersey

(Miller 1956). G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information", Em ***Psychological Review*** 63 (2), pp. 81-97.

(Mills 1971). H. Mills, "Chief Programmer Teams, Principles, and Procedure", Em ***IBM Federal Systems Division Report***, FSC 71-5108, Gaithersburg.

(Miyoshi e Azuma 1993). T. Miyoshi, M. Azuma, "An Empirical Study of Evaluating Software Development Environment Quality, Em ***IEEE Transactions on Software Engineering***, volume 19, no.5, pp. 425- 435.

(Möller e Paulish). K. Möller, D. Paulish, ***Software Metrics: A Practitioner's Guide to Improved Product Development***, Chapman & Hall.

(Monarchi e Putchen 1992). D. E. Monarchi, G.I. Putchen, "A Research Typology for Object-Oriented Analysis and Design", Em ***Communications of the ACM***, 35 (9): pp. 35-47.

(Montazeri e Hitz 1996). B. Montazeri, M. Hitz, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective", Em ***IEEE Transaction on Software Engineering***, volume 22, no.4, abril, pp. 267 - 270.

(Mosley 1993). D. Mosley, ***The Handbook of MIS Application Software Technology***, Yourdon Press - Prentice-Hall.

(Müllerburg e Meyerhoff 1993). M. Müllerburg, D. Meyerhoff, "The METKIT CAI System", Em ***Information and Software Technology***, volume 35, No.2, fevereiro, pp. 120 - 126.

(Munson 1995). J. Munson, "Software Measurement: Problems and Practice", Em ***Annals of Software Engineering - Software Process and Product Measurement***, volume (1), pp. 255 -285.

(Murine e Murine 1995). G. Murine, B. Murine, "Implementing a Software Quality Metric Program based on the Rome Laboratory Initiatives", Em ***Annals of Software Engineering - Software Process and Product Measurement***, volume (1), pp. 155-177.

(Oivo e Basili 1992). M. Oivo, V. R. Basili, "Representating Software Engineering Models: The TAME Goal Oriented Approach", Em ***IEEE Transactions on Software Engineering***, vol 18, no. 10, outubro.

(Oliveira 1995). K. M. Oliveira, ***Avaliação da Qualidade de Sistemas Especialistas***, Tese de Mestrado, COPPE/UFRJ, Brasil.

(Oliveira et al. 1995). K. M. Oliveira, C. Asanome, A. Rabelo, A. R. Rocha, Avaliação da Confiabilidade de um Sistema Especialista para Diagnóstico Médico de Infarto Agudo do Miocárdio, Em ***Workshop de Qualidade de Software***, Recife, Brasil, pp. 127 - 131.

(O_2 1993). O_2 Technology, ***The O_2 User Manual - Version 4.3,*** França, julho.

(Pages-Jones 1988). M. Page-Jones, ***Projeto Estruturado de Sistemas,*** Editora McGraw-Hill.

(Palermo e Rocha 1993). S. Palermo, A. R. Rocha, "An Experience on Evaluating Software Quality for High Energy Physics", Em ***Computer Physics Communications.***

(Passos 1991). M. C. Passos, ***Uma Ferramenta para Construção e Avaliação de Gráficos de Estrutura,*** Tese de Mestrado, COPPE/UFRJ.

(Paulk et al. 1993). M. Paulk, C. Curtis, B. Chrissis, C. V. Weber, ***Capability Maturity Model*** - V. 1.1, Software Engineering Institute (SEI), Pittsburg, PA - USA.

(Perlis 1981). A. J. Perlis, ***Controlling Software Development Through the LifeCicle Model Software Metrics: An Analysis and Evaluation,*** Cambridge, MIT Press.

(Peter et al. 1995). K. Peter, V. Zumer, J. Brest, M. Mernik, "PROMIS: A Software Metrics Tool Generator", Em ***ACM SIGPLAN Notices,*** volume 30, no. 5, maio, pp. 37 - 42.

(Pfleeger 1995a). Sh.L. Pfleeger, "Experimental Design and Analysis in Software Engineering", Em ***Annals of Software Engineering-Software Process and Product Measurement,*** volume (1), pp. 219 - 253.

(Pfleeger Sh. L. 1995b). Sh. L. Pfleeger, "Maturity, Models, and Goals: How to Build a Metrics Plan", Em ***Journal Systems and Software,*** vol 31, pp. 143- 155.

(Philip et al. 1995). H. Philip, T. Menzies, G. Phipps, ***Using the Size of Classes and Methods as the Basis for Early Effort Prediction; Empirical Observations, Initial Application; A Practitioners Experience Report,*** *Technical Report,* Monash University, Melbourne, Australia.

(Pressman 1995). R. Pressman, ***Engenharia de Software: Enfoque Prático,*** McGraw-Hill.

(Queiroz e Ribeiro 1995). O. A. Queiroz, R. P. Ribeiro, ***Testes de Programas OO,*** Relatório Técnico ES-338/95, COPPE-UFRJ.

(Robson 1990). D. Robson, "Object-Oriented Software Systems", Em ***Tutorial Object-Oriented Computing - Concepts,*** volume 1, Ed. Gerald Peterson.

(Rocha 1983). A. R. Rocha, ***Um Modelo para Avaliação da Qualidade de Especificações,*** Tese de Doutorado, PUC-RJ.

(Rocha 1987). A. R. Rocha, ***Análise e Projeto Estruturado de Sistemas***, Editora Campus, Rio de Janeiro.

(Rocha e Palermo 1989). A. R. Rocha, S. Palermo, ***A Proposal to Evaluate Program Quality for the Delphi Off-Line Environment***, Relatório Técnico ES, COPPE/UFRJ.

(Rocha et al. 1994). A. R. Rocha, C. Werner, V. Werneck, G. Travassos, G. Xexeo, "Uma Experiência na Definição do Processo de Desenvolvimento e Avaliação de Software segundo as Normas ISO", Em ***Anais do Workshop Qualidade de Software***, Curitiba.

(Rocha et al. 1996). A. R. Rocha, C. Werner, V. Werneck, G. Travassos, ***Processo de Desenvolvimento de Software Baseado em Reutilização,*** Relatório Técnico do Projeto MEMPHIS, 1/96, COPPE/UFRJ.

(Roche e Jackson 1994). J. Roche, M. Jackson, "Software Measurement Methods: Recipes for Sucess?" Em ***Information and Software Technology***, 36(4), pp. 173-179.

(Rombach 1987). H. D. Rombach, "A Controlled Experiment on the Impact of Software Structure on Maintainability", Em ***IEEE Transaction on Software Engineering***, volume 13, no. 3, pp. 344 - 354.

(Rumbaugh et al. 1994). J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, ***Modelagem e Projeto Baseados em Objetos***, Editora Campus, Rio de Janeiro, Brasil.

(Scalet 1995). D. Scalet, "Avaliação da Qualidade de Produto de Software" ,Em ***Workshop de Qualidade de Software***, Recife, Brasil, outubro.

(Schneidewind 1992). N. F. Schneidewind, "Methodology for Validating Software Metrics", Em ***IEEE Transactions on Software Engineering***, volume 18, no.5, pp. 410- 422.

(SEPIN/MCT 1995). SEPIN/MCT, ***Qualidade no Setor de Software Brasileiro,*** Ministério da Ciência e Tecnologia, Secretaria de Política de Informática e Automação.

(Sharble e Cohen 1993). R. C. Sharble, S. S. Cohen, The Object-Oriented Bewery: A Comparison of two Object-Oriented development methods, Em ***ACM SIGSOFT Software Engineering Notes***, 18(2), pp. 60-73.

(Shaw 1981). M. Shaw, ***Annotated Bibliography on Software Metrics - Software Metrics: An Analysis and Evaluation***, Cambridge, MIT Press.

(Shepperd 1992). M. Shepperd, "Products, Process and Metrics", Em ***Information and Software Technology***, volume 34, no.10.

(Shlaer e Mellor 1990) S. Shlaer, S. Mellor, ***Análise de Sistemas Orientada a Objetos***, Editora McGraw-Hill.

(Shlaer e Mellor 1992). S. Shlaer, S. Mellor, ***Object LifeCycles: Modeling the World in States***, Prentice-Hall.

(SPICE 1995). SPICE - ***SPICE- Project Overview***, http://www.cit.gu.au/research/center/sqi/spice.

(Spriet e Vansteenksite 1982). J. A. Spriet, G. C. Vansteenksite, ***Computer-Aided Modelling and Simulation***, Academic Press.

(Stiliglic et al. 1995). B. Stiliglic, M. Hericko, R. Ivan, "How to Evaluate Object-Oriented Software Development?", Em ***ACM SIGPLAN Notices***, volume 30, no. 5, maio, pp. 3-10.

(Sullo 1994). G. Sullo, ***Object Engineering: Designing Large-Scale Object - Oriented Systems***, John Wiley & Sons, Inc, 1994.

(Sutcliffe 1991). A. G. Sutcliffe, "Object-Oriented Systems Development: Survey of Structured Methods", Em ***Information and Software Technology***, 33, pp. 433 - 442, julho-agosto.

(Tegarden et al. 1992). D. P. Tegarden, S. D. Sheetz, D. E. Monarchi, "The Effectiveness of Traditional Metrics for Object Oriented Systems", Em ***Proceedings of the Twenty - Fifth Hawaii International Conference on System Sciences***, volume IV, IEEE Computer Society Press, Washington D.C., pp. 359-368.

(Tegarden et al. 1995). D. P. Tegarden, S. D. Sheetz, D. E. Monarchi, "A Software Complexity Model of Object-Oriented Systems, Em ***Decision Support Systems***, no. 13, pp. 241-262.

(Travassos et al. 1995). G. H. Travassos, C. M. Werner, F. M. Vasconcelos, "Testing Complex Object Oriented Models: a Practical Approach", Em ***Proceedings of the Second International Congress on Information Engineering - ICIE' 95***. - Buenos Aires, Argentina.

(Tsukumo et al. 1995). A.Tsukumo, A. L. Andrade, C.M. Rego, G.F. Azevedo, M. Jino, R. Tulami, S. Maintinguer, "Avaliação do Produto de Software: Algumas Questões Relevantes e a ISO/IEC 9126", Em ***Workshop de Qualidade de Software***, Recife, Brasil, outubro.

(Walker 1992). J. I. Walker, "Requirements on an Object - Oriented Design Method", Em ***Software Engineering Journal***, pp. 102 - 113, março.

(Wang e Wang 1994). B. L.Wang, J. Wang, "Is A Deep Class Hierarchy Considered Harmulf?", Em ***Object Magazine***, pp. 35-36, novembro-dezembro.

(Weber et al. 1997) K. Weber, J. C. M. Luca, A.R. Rocha, *Qualidade e Produtividade em Software*, 2a. Edição - Revisão Ampliada, Makron Books.

(Werner 1995) C. Werner, "Orientação a Objetos e Reutilização de Software", Em ***I Workshop de Orientação a Objetos***, COPPE/UFRJ, dezembro.

(Werner et al. 1996). C. Werner, A. R. Rocha, V. Werneck, G. Travassos, ***MEMPHIS: Um Ambiente para Desenvolvimento de Software Baseado em Reutilização***. - Definição da Arquitetura, Publicações Técnicas 3/96, COPPE-UFRJ.

(West 1996). M. West, "Taking the Measure of Metrics" , Em ***Object Expert***, A SIG Publication, volume (12), janeiro/fevereiro, pp. 35 - 40.

(Weyuker 1988) E. J. Weyuker, "Evaluating Software Complexity Measures", Em ***IEEE Transactions on Software Engineering***, volume 14, no.9, pp. 1357-1365.

(White 1994). I. White, ***Using the Booch Method***, Redwood City: Benjamin/Cummings.

(Whitty 1996), R. Whitty, "Object-Oriented Metrics: A Status Report", Em ***Object Expert***, A SIG Publication, volume (12), janeiro/fevereiro, pp. 35 - 40.

(Wild e Huitt 1992). N. Wild, R. Huitt, "Maintenance Support for Object-Oriented Programs", Em ***IEEE Transaction Software Engeneering***, volume 18, pp. 1038 -1044.

(William 1993). J. Williams, "Metrics for Object Oriented Projects", Em ***OOPLSA'93 Workshop Program: Process and Metrics for Object-Oriented Software Development.***

(Winblad et al. 1993). A. Winblad, S. Edwards, D. King, ***Software Orientado a Objetos***, Makron Books do Brasil.

(Winston 1993). A. B. Winston, "Impact of the ISO 9000 Standard Series", Em ***Conference on Software Maintenance***, Montreal, Canada, setembro.

(Wirfs-Brock 1989). R. E. Wirfs-Brock, "Object-Oriented Design: A Responsability-Driven Approach", Em ***OOPSLA'89***.

(Wirfs-Brock 1990a). R. E. Wirfs-Brock, ***Design Object-Oriented Software***, Prentice-Hall, Englewood Cliff, New Jersey.

(Wirf-Brock 1990b). R. E. Wirf-Brock, "Surveying Current Research in Object Oriented Design", Em ***Communications of the ACM***, volume 33, no.9.

(Yourdon 1989). E. Yourdon, ***Modern Structured Analysis***. Prentice-Hall.

(Yourdon e Constantine 1978). E. Yourdon, L. Constantine, ***Structured Design***, Englewood Cliffs, N. J., Yourdon Press/Prentice-Hall.

(Zage e Zage 1995), W. Zage, D. Zage, "Avoiding Metric Monsters: A Design Metrics Approach", Em ***Annals of Software Engineering***, volume 1, pp. 43-55.

(Zuse 1993). H. Zuse, "Support of Experimentation by Measurement Theory", Em ***Lecture Notes in Computer Science***, vol 706, New York: Spring Verlag, 1993, pp. 137-140.

Apêndice I

Manual para Controle da Qualidade de Especificações Orientadas a Objetos

Sumário

I. Introdução

Partindo da premissa de que os projetos de software apresentam sinais evidentes de aumento de sua complexidade, e, que o paradigma de orientação a objetos ainda não atingiu sua maturidade, seria pouco realista esperar que os métodos de desenvolvimento orientado a objetos, pudessem resolver os problemas de qualidade no desenvolvimento de software. Por outro lado, como a qualidade de um sistema de software, como a de qualquer produto de alta tecnologia, depende da qualidade de suas especificações, torna-se muito importante identificar quais são os atributos que determinam sua qualidade e, definir processos que permitam avaliá-las de acordo com estes atributos. Portanto, com o intuito de auxiliar na avaliação das especificações de requisitos e de projeto desenvolvidas segundo o paradigma de orientação a objetos, e, permitindo que esta seja feita numa forma organizada e metódica, é proposto um *Manual para Controle da Qualidade de Especificações Orientadas a Objetos.* O objetivo deste manual é fornecer subsídios que permitam avaliar a qualidade das especificações que estão sendo geradas durante o desenvolvimento utilizando o paradigma de orientação a objetos.

II. Método de Avaliação da Qualidade de Software

O método para avaliação da qualidade utilizado neste manual foi proposto por Rocha (1983) e baseia-se nos seguintes conceitos (Figura A1.1):

- ***Objetivos de Qualidade***: são propriedades gerais que o produto deve possuir;
- ***Fatores de Qualidade do Produto:*** determinam a qualidade do produto sob o ponto de vista dos seus diferentes usuários do produto (*usuário final, mantenedores, etc.)*;
- ***Critérios:*** são atributos primitivos possíveis de serem avaliados;
- ***Processos de Avaliação***: determinam o processo e os instrumentos a serem utilizados para se medir o grau de presença, no produto, de um determinado critério;
- ***Medidas:*** indicam o grau de presença, no produto, de um determinado critério;
- ***Medidas Agregadas***: são o resultado da agregação das medidas obtidas na avaliação segundo os critérios que quantificam os fatores.

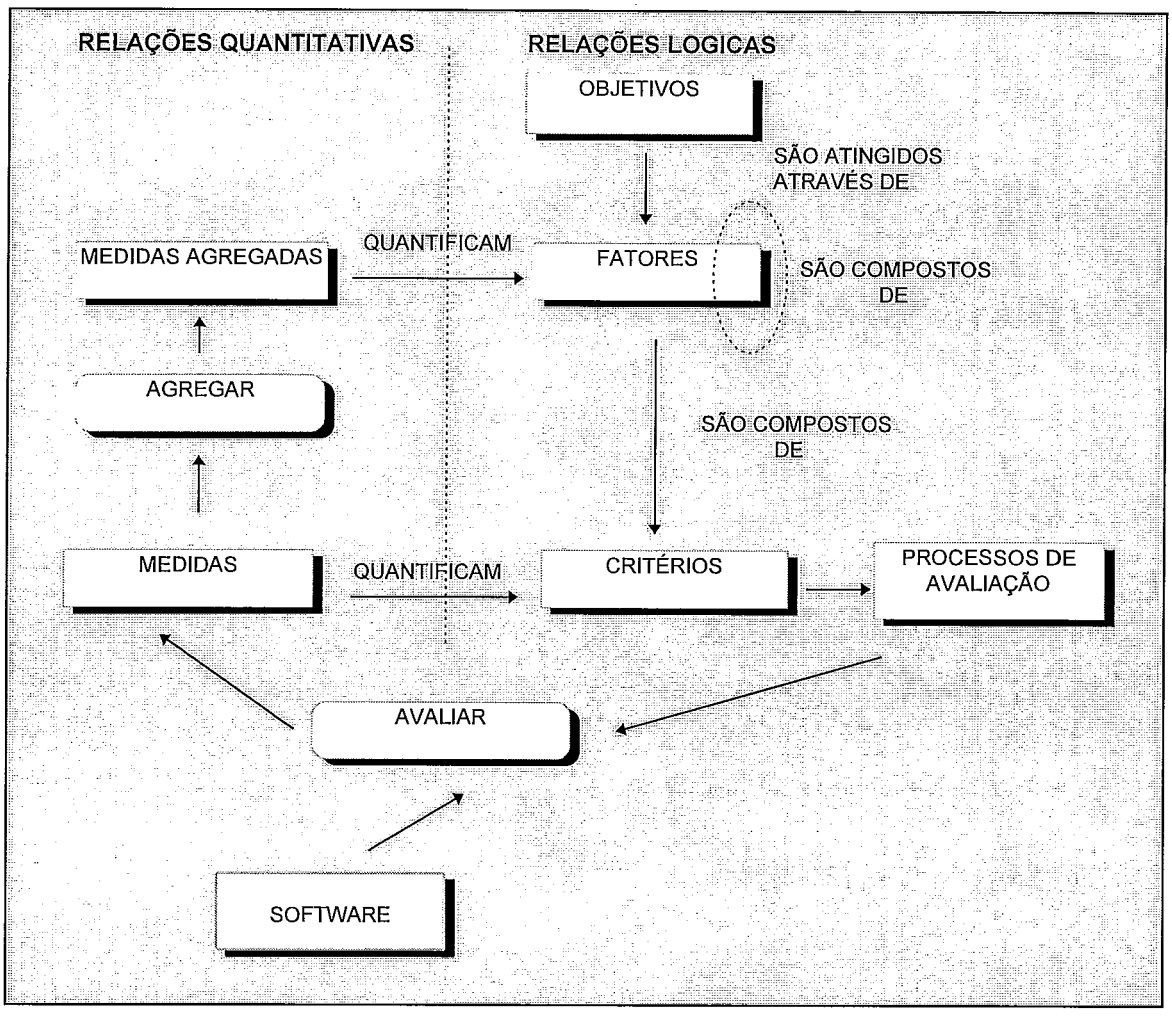


Figura A1. 1 - Método de Avaliação da Qualidade de Software

Os objetivos de qualidade são atingidos através dos fatores de qualidade, que podem ser compostos por outros fatores e são avaliados através de critérios. Os critérios definem atributos de qualidade para os fatores. Medidas são valores resultantes da avaliação de um produto segundo um critério específico. Objetivos e fatores não são diretamente mensuráveis e só podem ser avaliados através de critérios. Um critério é um atributo primitivo, isto é, um atributo independente de todos os outros atributos. Nenhum critério isolado é uma descrição completa de um determinado fator ou sub-fator. Da mesma forma, nenhum fator define completamente um objetivo.

A partir do método descrito e considerando-se que a especificação é desenvolvida para atender às necessidades de seus usuários, determinados objetivos devem ser atingidos:

- ***Confiabilidade Conceitual,*** dado que a especificação precisa satisfazer as necessidades e requisitos que motivaram sua construção.

- ***Confiabilidade da Representação***, que refere-se às características de representação do produto que afetam sua comprensão e manipulação, e,
- ***Utilizabilidade,*** que determina a conveniência e a viabilidade de utilização da especificação ao longo do processo de desenvolvimento. Para que uma especificação seja utilizável são necessárias a confiabilidade conceitual e a confiabilidade da representação.

III. Estrutura do Manual

O conteúdo do *Manual de Controle da Qualidade para Especificações Orientadas a Objetos* define os atributos de qualidade de uma especificação orientada a objetos, como mostra a Figura A1.2 e A1.3, organizados de acordo com o método de avaliação da qualidade definido por Rocha (1983). Indica, também, os procedimentos necessários para realizar avaliações de especificações segundo o método. Para cada objetivo de qualidade, são definidos os fatores e subfatores de qualidade a ele relacionados. Para cada critério, é descrito:

- ***Definição***: é apresentada uma definição precisa e explícita do critério;
- ***Fase:*** momento durante o desenvolvimento onde o critério pode ser avaliado;
- ***Técnica de Avaliação:*** método de controle da qualidade sugerido para realizar a avaliação;
- ***Processo de Avaliação***: o processo de avaliação adotado mede o grau de presença de cada critério em um intervalo de 0 a 1, utilizando uma escala ordinal como indicado na Tabela A1.1. A variável V_x utilizada em alguns processos de avaliação representam valores de aceitação estabelecidos ao longo do desenvolvimento do projeto e/ou considerando a experiência da organização no desenvolvimento de sistemas orientados a objetos.

ESCALA	EQUIVALÊNCIA	INTERPRETAÇÃO
1	*Total Presença*	Indica que não há dúvida que o critério está totalmente presente.
0,75	*Alta Presença*	Indica um alto grau de presença do critério, mas não total.
0,50	*Moderada Presença*	Indica um grau de presença aceitável do critério.
0,25	*Baixa Presença*	Indica um baixo grau de presença do critério, sendo necessário o uso de medidas corretivas.
0	*Total Ausência*	Indica de maneira absoluta que o critério está ausente.

Tabela A1.1 - Graus de Presença do Critério - Escala Ordinal

- ***Sugestões para Correção***: são indicadas medidas corretivas quando pertinente.
- ***Atributos de Qualidade Relacionados:*** são identificados os atributos de qualidade relacionados ao critério que está sendo avaliado, explicando brevemente o motivo deste relacionamento.

IV. Características de Qualidade de Especificações Orientadas a Objetos

A qualidade de uma especificação é avaliada através de três objetivos: *Confiabilidade da Representação, Confiabilidade Conceitual* e *Utilizabilidade.*

IV.1. Avaliação do Objetivo Confiabilidade da Representação

O objetivo Confiabilidade da Representação refere-se às características que tornam a especificação confiável para seus usuários, considerando-se aspectos relativos à sua forma e que tornam possível a sua compreensão e manipulação, tendo em conta que a especificação evolui gradativamente ao longo do desenvolvimento e que deve ser modificada durante a vida útil do produto, ao serem realizadas manutenções.

Este objetivo se realiza através dos fatores de qualidade ***Comunicabilidade*** e ***Manipulabilidade*** (Figura A1.2).

IV.1.1. Fator: Comunicabilidade

Conjunto de atributos de qualidade que avaliam a capacidade da especificação de comunicar o seu conteúdo.

É atingido através dos seguintes sub-fatores de qualidade:

- ***Correção no Uso do Método:*** Conjunto de atributos de qualidade que avaliam a correção da especificação do ponto de vista do uso do método de desenvolvimento.
- ***Uniformidade de Terminologia:*** Conjunto de atributos de qualidade que avaliam se a especificação foi escrita com uniformidade de termos e notação.
- ***Uniformidade no Nível de Abstração:*** Conjunto de atributos de qualidade que avaliam se a especificação possui um nível de detalhe uniforme, considerando-se um determinado estágio de desenvolvimento.
- ***Modularidade da Documentação:*** Conjunto de atributos de qualidade que avaliam se as partes da especificação podem ser entendidas e modificadas de forma independente.
- ***Correção da Arquitetura:*** Conjunto de atributos de qualidade que avaliam a correção da modelagem, considerando a disposição, composição, e relacionamento de seus componentes.
- ***Concisão:*** Conjunto de atributos de qualidade que avaliam se a especificação contém um volume mínimo de texto por ter-se maximizado o volume de informação por unidade de texto.
- ***Simplicidade:*** Conjunto de atributos de qualidade que avaliam em cada classe-objetos o grau de simplicidade dos seus serviços e atributos.
- ***Conformidade:*** Conjunto de atributos de qualidade que avaliam se a especificação foi gerada considerando as normas estabelecidas para o desenvolvimento do produto.

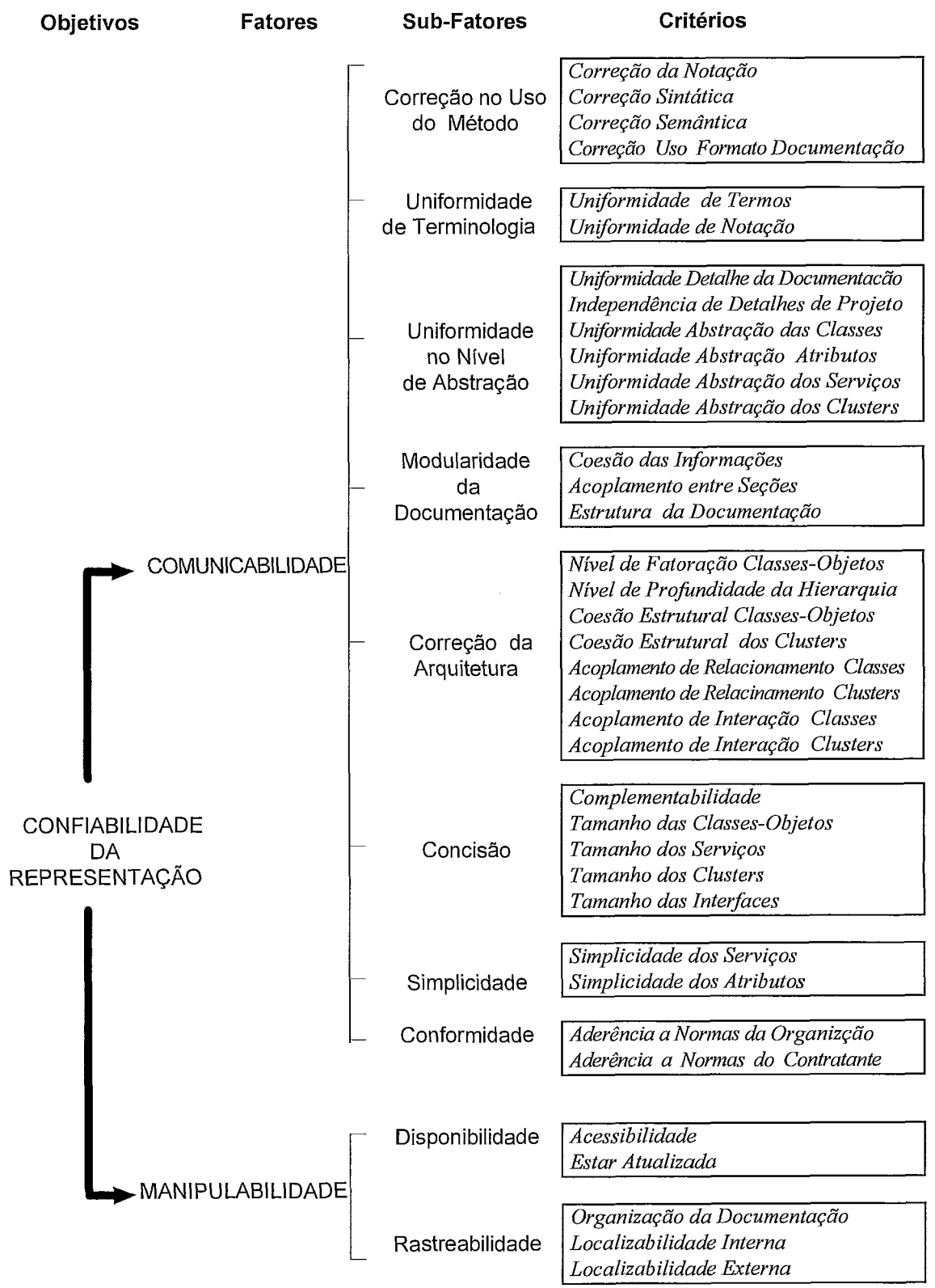


Figura A1.2. - Características de Qualidade de Especificações segundo o Objetivo Confiabilidade da Representação

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Correção no Uso do Método **Critério:** ***Correção da Notação (CNT)***	**Sub-Produto:** Modelo de Classes-objetos **Técnica de Avaliação:** Inspeção Individual
Definição do Critério: Característica que avalia se a notação do método foi usada de forma correta.	
Discussão: Um método de especificação para ser útil tem que ter uma notação definida com precisão. Além disso, para que se possa entender um documento escrito utilizando um método, é necessário que esta notação seja utilizada de forma correta, observando sua definição.	
Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta: • A especificação está escrita utilizando de forma correta a notação pré-definida no método de especificação?	**Escala** Sim → 1 Não → 0
Atributos de Qualidade Relacionados: • ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Especificações escritas corretamente, sob o ponto de vista da notação pré-definida no método, facilitam os processos de manutenção, avaliação e aumentam as possibilidades de reutilização da especificação, pois ajudam a torná-la entendível.	

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Correção no Uso do Método **Critério:** ***Correção Sintática (CST)***	**Sub-Produto:** Modelo de Classes-objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se o conjunto de regras sintáticas, pré-definidas no método de especificação, foi usado de forma correta.

Discussão:

A sintaxe da linguagem definida para o método é descrita a partir de um conjunto de regras que permitem utilizar os simbolos e combiná-los. É necessário entender a sintaxe antes de tentar compreender o sentido de uma determinada sentença da linguagem. O conhecimento da sintaxe da linguagem fornce um guia para a compreensão do sentido de uma sentença. Dada uma sentença, sua sintaxe resulta em uma especie de mapa que pode ser usado para investigar seu significado.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• A especificação está escrita utilizando de forma correta o conjunto de regras sintáticas pré-definidas no método de especificação?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Especificações escritas obedecendo ao conjunto de regras sintáticas pré-definidas na linguagem facilitam o processo de manutenção, avaliação e aumenta as possibilidades de reutilização da especificação, pois ajudam a torná-la entendível.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Correção no Uso do Método **Critério:** ***Correção Semântica (CSM)***	**Sub-Produto:** Modelo de Classes-objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se o conjunto de regras semânticas pré-definidas no método foi usado de forma correta.

Discussão:

A sintaxe trata da maneira de reunir os símbolos para formar sentenças corretas em uma determinada linguagem. A semântica trata do significado contido nestas sentenças sintaticamente corretas. Embora seja possível gerar sentenças em uma linguagem obedecendo à regras sintáticas, não é garantido que estas sentenças tenham algum significado. Regras sintáticas não são suficientes para caracterizar uma linguagem. É necessário introduzir-se um outro grupo de regras chamado de regras semânticas. A razão é obvia: intérpretes podem partilhar a mesma estrutura lingüística com regras semânticas diferentes sendo, então, incapazes de se comunicar. É neste contexto que se introduz a dimensão semântica, a qual pode ser definida como o estudo da relação entre objetos e sua representação.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• A especificação está escrita utilizando de forma correta a conjunto de regras semânticas pré-definidas no método de especificação?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Especificações escritas obedecendo ao conjunto de regras semânticas pré-definidas na linguagem facilitam o processo de manutenção, avaliação e aumenta as possibilidades de reutilização da especificação, pois ajudam a torná-la entendível.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Correção no Uso do Método **Critério:** ***Correção no Uso do Formato de Documentação (CFD)***	**Sub-Produto:** Modelo de Classes-objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se o formato de documentação definido no método foi usado de forma correta.

Discussão:

Um método de especificação deve ter definido um formato de documentação, fornecendo assim uma padronização. A ausência de um formato padronizado para documentação pode trazer problemas de entendimento da especificação. A existência de um formato de documentação bem organizado e sua correta utilização facilitam a leitura e compreensão da especificação. Assim sendo, o formato de documentação ajuda na produção de uma documentação organizada, que possa servir de guia para as atividades subseqüentes.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• A especificação está escrita utilizando de forma correta o formato de documentação definido no método?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Especificações escritas utilizando um método de especificação com formato de documentação bem organizado e utilizado corretamente, facilitam a manutenção, avaliação e possibilidades de reutilização da especificação, pois ajudam a torná-la entendível.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Uniformidade de Terminologia **Critério:** ***Uniformidade de Termos (UTT)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se os termos técnicos foram utilizados de forma uniforme ao longo do texto e de acordo com as definições pré-estabelecidas.

Discussão:

O entendimento da especificação sofre os efeitos da inexistência de uma terminologia consistente, que descreva de maneira uniforme todos os elementos da especificação. O uso uniforme de termos técnicos ao longo dos documentos produzidos, permite que a pessoa que está trabalhando com a especificação possa entender o especificado e os desenvolvedores possam desempenhar a sua função na construção do software.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Os termos técnicos são utilizados de forma uniforme ao longo da especificação e obedecem a definições pré-estabelecidas ?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** O uso uniforme de terminologia, no que se refere a termos técnicos, garantem o entendimento dos aspectos especificados e, conseqüentemente, pode-se realizar avaliações e modificações confiáveis.
- ***Reutilizabilidade:*** A existência de um padronização de termos torna possível sua aplicação em outros problemas similares, sempre que os termos técnicos definidos sejam entendidos com o mesmo sentido.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Uniformidade de Terminologia **Critério:** ***Uniformidade de Notação (UNT)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se a notação foi utilizada de forma uniforme ao longo do texto.

Discussão:

Mesmo quando se utilizam linguagens gráficas nos métodos orientados a objetos, existe uma documentação associada que é descrita em linguagem natural. Conseqüentemente, o uso de notações pode ser necessário para completar as informações dos diagramas. Por este motivo é necessário que parte da especificação escrita em linguagem natural utilize, de forma uniforme, uma notação precisa. É ainda desejável que, no caso necessário, o uso de simbologia, esta tenha significado óbvio.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• A notação foi utilizada de forma uniforme ao longo da especificação e obedece a definições pré-estabelecidas?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Especificações escritas com emprego de uma notação uniforme garantem o seu entendimento e, conseqüentemente, as atividades de avaliação e manutenção podem ser realizadas com maior confiabilidade.
- ***Reutilizabilidade:*** Para facilitar a reutilização futura de componentes da especificação, é essencial que este seja facilmente compreendido pelo reutilizador. Torna-se necessário o uso de uma notação que seja fácil de entender.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Uniformidade No Nível de Abstração **Critério:** ***Uniformidade de Detalhes da Documentação (UDD)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se todos os aspectos estão descritos na especificação com o mesmo nível de detalhamento, considerando-se um determinado estágio de desenvolvimento.

Discussão:

A especificação deve definir detalhes o mais tarde possível, assegurando desta forma a disponibilidade de toda a informação requerida para avaliar a necessidade e o escopo destes detalhes. Caso perceba-se que determinado aspecto está detalhado demais, deve-se considerar a hipótese de mudar as porções com detalhe excessivo para especificações a serem produzidas em passos de refinamentos posteriores. Deste modo, para um determinado nível de abstração, teremos todos os aspectos especificados com o mesmo grau de detalhamento.

Processo de Avaliação:

Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Os aspectos descritos na especificação apresentam o mesmo nível de detalhamento, considerando-se o estágio de desenvolvimento	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Especificações não uniformes são difíceis de entender e por conseguinte difíceis de serem avaliadas e alteradas com relação ao seu conteúdo, pois torna-se inadequado avaliar, num determinado nível de abstração, aspectos que apresentam diferentes graus de detalhamento.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Uniformidade No Nível de Abstração **Critério:** *Independência de Detalhes de Projeto (IDP)*	**Sub-Produto:** Documentação, Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia na especificação de requisitos a existência de restrições com relação à escolha de alternativas de solução próprias da fase de projeto.

Discussão:

Especificações de requisitos não devem conter restrições estabelecidas com relação à escolha de alternativas próprias de passos posteriores de refinamentos, pois estas tendem a retringir de forma desnecessária outras possíveis alternativas de solução, tornando pouco flexíveis as soluções encontradas. A inexistência de restrições estabelecidas com relação à escolha de alternativas próprias da fase de projeto permitem que as especificações seja possíveis de serem reutilizadas em uma outra aplicação similar e evoluam com mais facilidade.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• A especificação de requisitos não apresenta restrições com relação à escolha de alternativas próprias da fase de projeto e/ou implementação.	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** A inexistência de restrições, estabelecidas com relação à escolha de alternativas próprias da fases de projeto e implementação, permitem definir uma especificação mais uniforme, possibilitando realizar evoluções e modificações confiáveis, aumentando também as possibilidades de reutilização em outras aplicações.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Uniformidade No Nível de Abstração **Critério:** ***Uniformidade de Abstração das Classes-objetos (UAC)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia o nível de abstração de cada classe-objetos, considerando-se um determinado estágio de desenvolvimento.

Discussão:

Um bom modelo de classes-objetos incorpora, na fase de análise, os aspectos fundamentais de um problema e omite os demais porque, se este contém detalhes em excesso, limita desnecessariamente a escolha de decisões próprias de outras fases, desviando a atenção dos problemas reais.

Classes-objetos mais estáveis são aquelas que descrevem, de maneira precisa e uniforme, o domínio do problema e as responsabilidades do sistema dentro desse domínio. Por maior que seja o sistema, seu aspecto mais estável (classes-objetos no domínio do problema, no escopo das responsabilidades do sistema) permanecerá o mesmo através de quaisquer alterações potencialmente importantes. Desta forma, o nível adequado de abstração das classes-objetos possibilita que os componentes do modelo de análise ou de projeto possam ser reutilizados em outras aplicações.

Processo de Avaliação:

Assinale de acordo com a escala o valor que melhor representa o grau em que o critério foi atingido:

- As classes-objetos apresentam o nível de abstração que lhe corresponde, segundo o estágio de desenvolvimento.

Escala	Interpretação
1 →	Total Presença
0,75 →	Alta Presença
0,50 →	Moderada Presença
0,25 →	Baixa Presença
0 →	Total Ausência

Sugestões para Correção:

- Este critério se torna importante ao início do desenvolvimento, quando o objetivo é a definição de aspectos conceituais do domínio da aplicação, através da modelagem de classes-objetos adequadas. Nesse momento, é recomendável , não considerar classes-objetos que pertencem a outras fases do desenvolvimento.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** A não uniformidade de abstração das classes-objetos, em um determinado estágio de desenvolvimento, pode trazer problemas de entendimento do modelo como um todo, que incidirão no processo de avaliação, manutenção e reutilização. Por exemplo, torna-se mais difícil, nas etapas iniciais do desenvolvimento, avaliar, manter ou reutilizar classes-objetos de domínio que apresentam detalhes de implementação.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Uniformidade No Nível de Abstração **Critério:** ***Uniformidade de Abstração dos Atributos (UAA)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia o nível de abstração dos atributos em cada classe-objetos, considerando-se um determinado estágio de desenvolvimento.

Discussão:

A uniformidade de abstração dos atributos de uma determinada classe-objetos permite caracterizá-la dentro do contexto do estágio do processo de desenvolvimento em questão. A não uniformidade de abstração dos atributos pode trazer problemas de entendimento da classe-objetos. Por exemplo, na fase de análise, o objetivo é a compreensão da classe-objetos e seus relacionamentos entre classes-objetos no nível conceitual. Portanto, não devem ser modelados atributos de implementação, pois este tipo de tratamento será considerado nas fases posteriores. Isto é, classes-objetos com atributos que representam diferentes níveis de abstração são difíceis de serem reutilizadas.

Processo de Avaliação:

Assinale de acordo com a escala o valor que melhor representa o grau em que o critério foi atingido:

- Os atributos da classe-objetos apresentam o nível de detalhamento adequado que lhe corresponde, segundo o estágio de desenvolvimento.

Escala	Interpretação
1 →	Total Presença
0,75 →	Alta Presença
0,50 →	Moderada Presença
0,25 →	Baixa Presença
0 →	Total Ausência

Sugestões para Correção:

- Verificar se os atributos das classes-objetos apresentam um grau de detalhamento que afeta o entendimento da classe-objetos. Caso seja necessário, uniformizar os atributos considerando os atributos com detalhamento satisfatório.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Classes-objetos compostos por atributos que apresentam diferentes níveis de detalhamento são mais difíceis de entender e, conseqüentemente, mais difíceis de serem alteradas, reutilizadas e avaliadas com relação ao seu conteúdo, pois apresentam num mesmo instante diferentes níveis de abstração.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Uniformidade No Nível de Abstração **Critério:** ***Uniformidade de Abstração dos Serviços (UAS)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia o nível de abstração dos serviços em cada classe-objetos, considerando-se um determinado estágio de desenvolvimento.

Discussão:

Um serviço é uma função ou transformação que pode ser aplicada a objetos, ou por estes, a uma classe. Os serviços descritos na fase de análise apresentam, de modo geral, um nível de detalhe menor do que os descritos na fase de projeto. Na fase de projeto, são especificados ou detalhados os serviços relacionados à própria implementação da classe-objetos, situação esta que não acontece na fase de análise. O importante, na fase de análise, é atingir o entendimento do modelo, sem detalhes desnecessários de projeto e implementação. Portanto, os serviços especificados na classe-objetos devem ter o nível de abstração correspondente ao estágio em questão.

Processo de Avaliação:

Assinale de acordo com a escala o valor que melhor representa o grau em que o critério foi atingido:

- Os serviços da classe-objetos apresentam o nível de detalhamento adequado que lhe corresponde, segundo o estágio de desenvolvimento.

Escala	Interpretação
1 →	Total Presença
0,75 →	Alta Presença
0,50 →	Moderada Presença
0,25 →	Baixa Presença
0 →	Total Ausência

Sugestões para Correção:

- Verificar os serviços que apresentam muito ou pouco detalhamento, afetando o entendimento da classe-objetos. Caso seja necessário, uniformizar de acordo com os serviços considerados com detalhamento satisfatório.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Classes-objetos com detalhamento não uniforme a nível de serviços são mais difíceis de entender e, conseqüentemente, mais difíceis de serem alteradas, reutilizadas e avaliadas com relação a seu conteúdo, pois as classes-objetos apresentam num mesmo instante diferentes níveis de abstração.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Uniformidade No Nível de Abstração **Critério:** ***Uniformidade de Abstração dos Clusters (UCL)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia o nível de abstração das componentes modeladas no *cluster,* considerando-se um determinado estágio de desenvolvimento.

Discussão:

No caso de sistemas muito grandes, compostos por centenas de classes-objetos, o desenvolvimento do software pode ser realizado por uma equipe de desenvolvimento igualmente grande. Esta equipe é subdividida em equipes menores responsáveis por determinada parte do sistema. Estas partes, compostas por um conjunto de classes-objetos - *cluster* -, devem manter o mesmo nível de abstração, que corresponda à atividade do desenvolvimento que se está realizando. Caso contrário, tornam-se mais difíceis de serem entendidas e de serem feitas modificações e/ou evoluções, pois o grupo encarregado pelo seu desenvolvimento necessita lidar com diferentes graus de abstração. Além disso, torna-se difícil a comunicação entre os diferentes membros da equipe que trabalham com classes-objetos que estão relacionadas, mas que pertencem a *clusters* diferentes.

Processo de Avaliação:

Assinale de acordo com a escala o valor que melhor representa o grau em que o critério foi atingido:

	Escala	**Interpretação**
• O *cluster* apresenta todas suas abstrações com o nível de detalhamento adequado que corresponde ao estágio de desenvolvimento.	1 →	Total Presença
	0,75 →	Alta Presença
	0,50 →	Moderada Presença
	0,25 →	Baixa Presença
	0 →	Total Ausência

Sugestões para Correção:

- Uniformizar de acordo com os *clusters* considerados com abstrações modeladas com detalhamento satisfatório.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** *Clusters* compostos por abstrações que apresentam diferentes níveis de abstração são mais difíceis de serem entendidos, avaliados, alterados e reutilizados.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Modularidade da Documentação **Critério:** ***Coesão de Informações (COI)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia o grau de associação das informações de um capítulo e, dentro deste, seções.

Discussão:

A coesão modular, no caso de um documento de especificação, pode ser avaliada observando o grau de associação das informações descritas num capítulo e, dentro destes, das seções. Numa especificação é conveniente que um mesmo capítulo e, dentro deste, as diferentes seções, tenham uma elevada funcionalidade dos diversos aspectos da especificação.

Processo de Avaliação:

Assinale de acordo com a escala o valor que melhor representa o grau em que o critério foi atingido:

- A informação que apresenta o módulo (capítulo ou seção) descreve aspectos relacionados ao capítulo ou seção que está sendo descrito.

Escala	Interpretação
1 →	Total Presença
0,75 →	Alta Presença
0,50 →	Moderada Presença
0,25 →	Baixa Presença
0 →	Total Ausência

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Especificações escritas em módulos, descrevendo aspectos relacionados, permitem que modificações e avaliações sejam feitas com maior facilidade. Além disso, possibilitam sua reutilização em outras aplicações na mesma área de conhecimento, pois é possível ter um conhecimento de toda a funcionalidade fornecida pela componente que será reutilizada.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Modularidade da Documentação **Critério:** ***Acoplamento entre Seções (ACS)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia o grau de interdependência entre os módulos (capítulos ou seções) da especificação.

Discussão:

Acoplamento é o grau de interdependência entre capítulos ou seções. Este relacionamento entre capítulos deve ser mínimo. Quanto mais baixo for o relacionamento entre capítulos ou seções, maior será o grau de modularidade da especificação. Com este procedimento, pretende-se reduzir a probabilidade que a modificação de um capítulo ou seção provoque alterações em outros capítulos ou seções. Um elevado grau de interdependência entre os capítulos ou seções, permite que o leitor da especificação, interessado em um determinado aspecto, possa concentrar-se neste aspecto, de forma independente, e obter um perfeito entendimento de tudo o relacionado a esse aspecto.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• O conteúdo de um módulo (capítulo ou seção) faz referência ao conteúdo de outro módulo, apenas como complemento, sem modificá-lo?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Especificações escritas em módulos (capítulo ou seções), com dependência mínima, permitem que, ao realizarem-se avaliações com relação ao conteúdo, não será necessário ler toda a documentação, mesmo quando só deseja-se avaliar um determinado aspecto da especificação. Além disso, ao realizar uma modificação na especificação, não repercutirá, de forma significativa, em outras partes do documento. As possibilidades de reutilização aumentam, pois a componente a ser reutilizada têm uma dependência mínima com as outras partes da especificação.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Modularidade da Documentação **Critério:** ***Estrutura da Documentação (EDO)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se a estrutura da especificação, composta de capítulos e seções, está organizada numa seqüência lógica.

Discussão:

Para atingir o atributo modularidade, na especificação, é ainda necessário que a organização dada aos capítulos e seções tenha uma sequência lógica. Esta organização se reflete no índice da especificacão. Uma organização adequada fornece uma visão mais clara da estrutura da especificação e torna possível o entendimento da mesma, pois as informações fluem de forma lógica.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• A especificação tem seus capítulos e, dentre destes, as seções organizadas numa seqüência lógica?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade, Rastreabilidade:*** Especificações com uma estrutura adequada permitem sua avaliação, manutenção e reutilização, pois fornecem uma visão clara da sua estrutura, permitindo, também, localizar com facilidade determinado aspecto ou assunto.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Correção da Arquitetura **Critério:** ***Nível de Fatoração das Classes-Objetos (NFC)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia a quantidade de descendentes imediatos de uma classe-objetos.

Discussão:

A estrutura hierárquica de classes-objetos indica em que medida as especializações herdam os atributos e serviços de sua classe ancestral. Desta forma, o número de especializações associadas a uma classe-objetos é um indicador do potencial de influência que determinada classe-objetos tem sobre seus descendentes. Assim sendo, o nível de fatoração da classe-objetos (*NFC*) é entendido como a quantidade de subclasses imediatas de uma classe ancestral. Se a classe-objetos apresenta uma grande quantidade de descendentes imediatos, é uma indicação de quantas classes de especializações vão herdar os atributos e serviços da classe ancestral. Além disso, isto pode indicar uma modelagem inadequada das especializações ou uma classe ancestral inapropriada.

Processo de Avaliação:

Este critério pode ser avaliado segundo a seguinte relação matemática:

NFC (Ci) = Número de classes-objetos imediatas da classe-objetos Ci

Escala		Interpretação
1	→	$1 \leq NFC \leq V_{nfc}$
0	→	$NFC > V_{nfc}$

Sugestões para Correção:

- Eliminar as especializações que não sejam relevantes para o domínio da aplicação, através da análise do grau de necessidade dos atributos e serviços adicionados pelas especializações, assim como os nomes utilizados relacionados com essas características.
- Tentar criar classes abstratas e reorganizar as especializações. Desta forma, é reduzida a quantidade de especializações da classe-objetos em questão. Caso a redução das especializações implique em um aumento dos níveis da estrutura de generalização-especialização, devem ser consideradas as diretrizes estabelecidas para o critério *Nível de Profundidade da Hierarquia de Classes-Objetos.*
- Verificar se existe uma percentagem alta de métodos herdados. Caso contrário, é um indicador de que a classe-objetos especialização é fraca e deve ser revista.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** Uma classe-objetos com muitos descendentes poderá requerer um maior esforço durante o processo de validação dos serviços das classes-objetos, pois os serviços das classes-objetos ancestral possivelmente precisam ser novamente testados nas classes-objetos descendentes.
- ***Reutilizabilidade:*** Se uma classe-objetos apresenta uma grande quantidade de descendentes imediatos, é uma indicação de quantas classes de especialização vão herdar os atributos e serviços da classe-ancestral, sendo a herança considerada como uma forma de reutilização.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Correção da Arquitetura **Critério:** *Nível de Profundidade da Estrutura Hierárquica (NPH)*	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia a quantidade média de níveis em uma hierarquia de classes-objetos.

Discussão:

Uma estrutura generalização-especialização representa uma hierarquia de classes-objetos formada por vários níveis. Através de uma análise desta hierarquia, é possível identificar em que medida as classes ancestrais afetam as demais classes, visualizando-se a profundidade das classes na hierarquia e o número de serviços herdados. Estas informações são indicadores do grau de complexidade da estrutura.

O aumento dos níveis de profundidade da hierarquia implica em um maior esforço para o entendimento e manutenção das classes-objetos. Isto se deve ao fato de que a herança afeta a localizabilidade e o princípio de encapsulamento. Ao invés de localizar todas as características da classe-objetos na própria classe-objetos e encapsular todos os seus dados, estas declarações aparecem dispersas ao longo de toda a hierarquia. No caso de *herança múltipla,* estes problemas tendem a se agravar ainda mais.

Processo de Avaliação:

Este critério pode ser avaliado da seguinte maneira:

Sejam $C_1, \ldots, C_n$ as classes-objetos e $NPC(Ci)$ o nível da classe-objetos na hierarquia, o nível de profundidade NPH é dada por:

$$NPH = \frac{\sum_{i=1}^{n} NPC(Ci)}{n}$$

Escala		Interpretação
1	→	$0 \leq NPH \leq V_{nph}$
0	→	$NPH > V_{nph}$

Sugestões para Correção:

- Verificar se as superclasses e subclasses podem fundir-se e formar um único nível.
- Verificar a possibilidade de agrupar serviços e atributos, gerando estruturas de hierarquia menores.
- Verificar a possibilidade de se incorporar as características da(s) subclasse(s) na superclasse.
- Procurar formas semânticas de se substituir o relacionamento de herança por outros tipos de relacionamentos (exemplo: composição)

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** O aumento dos níveis de profundidade da hierarquia de classes-objetos implica em um maior esforço para o entendimento das classes-objetos, tornando mais complexo predizer seus comportamentos. Por conseguinte, a avaliação das classes-objetos são afetadas, pois ao se realizar os testes dos serviços nas classes-objetos ancestrais há a necessidade de testá-los nas classes-objetos descendentes ao longo de toda a hierarquia.
- ***Manutenibilidade:*** O aumento nos níveis de hierarquia de classes-objetos torna mais difícil a realização de modificações e evoluções nas classes-objetos. Isto se deve ao fato de que a herança afeta a localizabilidade e o princípio de encapsulamento.
- ***Reutilizabilidade:*** Uma estrutura de hierarquia de classes-objetos muito profunda aumenta o potencial de reutilização dos atributos e serviços herdados.

Objetivo: Confiabilidade da Representação | **Sub-Produto:** Modelo de Classes-Objetos
Fator: Comunicabilidade | **Técnica de Avaliação:** Inspeção Individual
Sub-Fator: Correção da Arquitetura
Critério: ***Coesão Estrutural da Classe - Objetos (CEC)***

Definição do Critério:

Característica que avalia o grau de relacionamento dos serviços especificados nas classes-objetos.

Discussão:

Se uma classe-objetos tem diferentes serviços, realizando funções diferentes sobre um mesmo conjunto de argumentos, então a classe é *coesa*. Para realizar esta verificação, é possível determinar o grau de similaridade entre os serviços especificados na classe-objetos, observando o conjunto de argumentos que participam nos serviços. Serviços que trabalham com os mesmos argumentos pode ser uma indicação de que os serviços estão relacionados.

Um valor alto de *CEC* é indicativo de que existe uma alto relacionamento entre os serviços especificados na classe-objetos. Por outro lado, um valor pequeno pode indicar que a classe-objetos deva ser dividida em duas ou mais classes-objetos, pois ela provavelmente realiza serviços não relacionados, tornando a classe-objetos mais complexa e aumentando as possibilidades de erros durante o desenvolvimento.

Processo de Avaliação:

Este critério pode ser avaliado da seguinte maneira:

Seja a classe-objetos *C*, com *n* serviços *S*1, ... , Sn, e {A} o conjunto de argumentos declarado no serviço *S*i , onde $1 \leq i \leq n$. Então, existem *N* conjuntos {A1}, ... , {An}, tal que *M* (número de conjuntos disjuntos) é gerado a partir da interseção entre os conjuntos de argumentos Ai $\cap$ Aj .

$$CEC(Ci) = (1 - M/N)$$

Escala		Interpretação
1	→	$CEC \geq V_{cec}$
0	→	$CEC < V_{cec}$

Sugestões para Correção:

- Verificar a possibilidade de dividir os atributos e serviços da classe-objetos em duas ou mais subclasses.
- Verificar a possibilidade de realocar atributos e serviços em outras classes-objetos.
- Verificar a possibilidade de criar novas classes-objetos com uma funcionalidade definida.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** Classes-objetos com serviços não relacionados tornam-se mais difíceis de serem avaliadas com relação às funcionalidades que descrevem. A classe-objetos se torna mais complexa e, provavelmente, será necessária a definição de testes mais minuciosos.
- ***Manutenibilidade:*** Classes-objetos com serviços, realizando funções sobre conjuntos diferentes de argumentos, provavelmente, apresentam uma baixa coesão. A classe-objetos tende a ser mais complexa e o processo de modificação e/ou evolução se torna mais difícil de se realizar.
- ***Reutilizabilidade:*** Classes-objetos com baixa coesão apresentam menos chances de serem reutilizadas, pois não apresentam uma clara visão de suas funcionalidades.
- ***Coesão Semântica da Classe-Objetos:*** Para ter-se um indicador mais preciso da *coesão da classe-objetos*, deve-se considerar também o aspecto semântico envolvido.

Objetivo: Confiabilidade da Representação
Fator: Comunicabilidade
Sub-Fator: Correção da Arquitetura
Critério: ***Coesão Estrutural dos Clusters (CCL)***

Sub-Produto: Modelo de Classes-Objetos
Técnica de Avaliação: Inspeção Individual

Definição do Critério:

Característica que avalia o grau de relacionamento entre as classes-objetos de um determinado *cluster*..

Discussão:

O aspecto da coesão de um conjunto de classes-objetos pode ser avaliado através da média do número de relacionamentos internos entre as classes-objetos. Um valor alto de ***CCL*** é uma indicação de que existe um alto relacionamento entre as classes-objetos no *cluster*. Caso contrário, o objetivo definido para o *cluster* deve ser revisto.

Processo de Avaliação:

Este critério pode ser avaliado da seguinte maneira:

Seja *CL* um *cluster* específico e ***r*** o número de relacionamentos que são internos ao *cluster* (*i.e. que não estão conectados com classes-objetos fora do cluster*). Seja *n* o número de classes-objetos no *cluster*. Então, o indicador da *Coesão Estrutural do Cluster (CCL)* pode ser calculado da seguinte maneira:

$$CCL(CLi) = \frac{(r + 1)}{n}$$

(O valor de 1 na equação evita que *CCL* = 0, quando n=1)

Escala		Interpretação
1	→	$CCL \geq V_{ccl}$
0	→	$CCL < V_{ccl}$

Sugestões para Correção:

- Verificar a possibilidade de realocar classes-objetos em *cluster* adjacentes.
- Verificar a possibilidade de criar novos *clusters* com objetivos perfeitamente definidos.

Atributos de Qualidade Relacionados:

- ***Reutilizabilidade:*** *Clusters* com classes-objetos não relacionadas, apresentam menos chances de serem reutilizadas em outra aplicação, pois não apresenta uma visão clara de seu objetivo.
- ***Coesão Semântica dos Clusters:*** Para se ter um indicador mais preciso da coesão do *cluster,* deve-se considerar também, o aspecto semântico envolvido.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Correção da Arquitetura **Critério:** ***Acoplamento de Relacionamento das Classes-Objetos (ACR)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia o grau de dependência estática das classes-objetos, excluindo os relacionamentos por herança.

Discussão:

Um modelo com classes-objetos com um alto *Acoplamento de Relacionamento* pode tornar o modelo difícil de entender e levar a erros durante o desenvolvimento. Uma mudança - *ou uma falha* - em uma classe-objetos pode ter um efeito de propagação em outras classes-objetos no modelo. Isto dificulta a atividade de manutenção e a realização de testes. Os serviços que implementam os relacionamentos - *construção, destruição, associação* e *desassociação* - entre classes-objetos podem tornar-se complexos. Além disso, o tempo de desenvolvimento é reduzido, pois o projetista precisa de menos tempo para compreender detalhes de outras classes-objetos. Desta forma, as atividades de extensão e/ou customização são realizadas com maior facilidade.

Processo de Avaliação:

Este critério pode ser avaliado da seguinte maneira:

Seja C_i uma classe-objetos do modelo e r a indicação do número de relacionamentos com outras classes-objetos. Então, *ACR* é dada por:

$$ACR(Ci) = \sum_{i=1}^{n} r_i$$

Escala	**Interpretação**
1 →	$0 < ACR \leq V_{acr}$
0 →	$ACR > V_{acr}$

Sugestões para Correção:

- Eliminar os relacionamentos desnecessários, avaliando o seu grau de necessidade através do critério *necessidade de relacionamentos*.
- Minimizar a quantidade de relacionamentos considerados necessários, mantendo os considerados imprescindíveis.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Classes-objetos com alto acoplamento deste tipo dificultam a atividade de manutenção e a realização de testes, pois os projetista precisa de menos tempo para entender detalhes relacionados a outras classes-objetos.
- ***Reutilizabilidade:*** Um baixo acoplamento deste tipo maximiza o encapsulamento e aumenta as possibilidades de reutilização das classes-objetos, pois os objetos com poucas dependências são mais facilmente reutilizados em outras aplicações.
- ***Necessidade de Relacionamentos:*** A remoção dos relacionamentos extras deve ser um dos objetivos do desenvolvedor, uma vez que acoplamentos desnecessários não são percorridos por nenhum serviço.

Objetivo: Confiabilidade da Representação
Fator: Comunicabilidade
Sub-Fator: Correção da Arquitetura
Critério: *Acoplamento de Relacionamento dos Clusters (ARCL)*

Sub-Produto: Modelo de Classes-Objetos
Técnica de Avaliação: Inspeção Individual

Definição do Critério:

Característica que avalia o grau de dependência estática dos *clusters*.

Discussão:

Um *cluster* específico, além de estar composto por um conjunto de classes-objetos relacionadas, deve ter uma dependência mínima em relação a outros *clusters*. Assim sendo, o acoplamento entre clusters é definido pelo número de relacionamentos entre classes-objetos do *cluster* e classes-objetos que pertencem a *clusters* diferentes. Um valor baixo para *ARCL* é desejável, na medida em que os relacionamentos entre as classes-objetos ficam confinadas aos *clusters,* sem atravessar suas fronteiras. Isto permite que o conjunto de classes-objetos possa ser implementado, testado e reutilizado de modo quase independente, sem afetar os demais *clusters*.

Processo de Avaliação:

Este critério pode ser avaliado da seguinte maneira:

Seja *CL1 ,..., CLn* um conjunto de *clusters* e *r* a indicação do número de relacionamentos entre classes-objetos que pertencem a *clusters* diferentes. Então, *ARCL* é dada por:

$$ARCL(CLi) = \sum_{i=1}^{n} r_i$$

Escala		Interpretação
1	→	$0 < ACRL \leq V_{acrl}$
0	→	$ACRL > V_{acrl}$

Sugestões para Correção:

- Eliminar os relacionamentos desnecessários avaliando seu grau de necessidade através do critério *necessidade de relacionamentos*.
- Minimizar a quantidade de relacionamentos considerados necessários, mantendo os considerados imprescindíveis.

Atributos de Qualidade Relacionados:

Avaliabilidade, Manutenibilidade, Reutilizabilidade: *Clusters* com um baixo acoplamento são mais fáceis de serem avaliados, alterados e reutilizados, pois é possível que os objetivos definidos possam ser bem entendidos, sem necessidade de entender os objetivos de outros *clusters*.

Objetivo: Confiabilidade da Representação
Fator: Comunicabilidade
Sub-Fator: Correção da Arquitetura
Critério: *Acoplamento de Interação das Classes-objetos (ACI)*

Sub-Produto: Modelo de Classes-Objetos
Técnica de Avaliação: Inspeção Individual

Definição do Critério:

Característica que avalia o grau de comunicação via mensagens das classes-objetos.

Discussão:

Uma classe-objetos tem *Acoplamento de Interação* com outra classe-objetos se uma delas atua sobre a outra via mensagens, ou seja, se os seus serviços utilizam os serviços ou atributos da outra classe-objetos. A partir dos serviços especificados na classe-objetos, é possível determinar o grau de interação que uma classe-objetos tem com as outras classes-objetos.Um valor alto de *ACI* é uma indicação de que a classe-objetos se comunica via mensagem com uma grande quantidade de classes-objetos, o que pode significar uma alta complexidade da classe-objetos.

Processo de Avaliação:

Este critério pode ser avaliado da seguinte maneira:

Seja *Ci* uma classe-objetos com *S1, . . . , Sn,* os serviços declarados, e *Ri* o conjunto de serviços chamados por $Si = \{Ri\ j\}$, então o indicador que mede o grau de comunicação da classe-objetos é:

$ACI(Ci) = Si \cup \{Ri\ j\}$

Escala		Interpretação
1	→	$0 < ACI \leq V_{aci}$
0	→	$ACI > V_{aci}$

Atributos de Qualidade Relacionados:

Avaliabilidade, Manutenibilidade, Reutilizabilidade: Através deste critério é possível determinar a complexidade dos testes nas várias partes do modelo e possíveis problemas de desempenho. Idealmente, para maximizar o encapsulamento e facilitar a manutenção, espera-se que os valores de *ACI* sejam pequenos.

Objetivo: Confiabilidade da Representação
Fator: Comunicabilidade
Sub-Fator: Correção da Arquitetura
Critério: *Acoplamento de Interação dos Clusters (AICL)*

Sub-Produto: Modelo de Classes-Objetos
Técnica de Avaliação: Inspeção Individual

Definição do Critério:

Característica que avalia o grau de comunicação via mensagens dos *clusters*.

Discussão:

Um *cluster* tem *Acoplamento de Interação (AICL)* se existe comunicação via mensagens entre classes-objetos do *cluster* e classes-objetos que pertencem a *clusters* diferentes.

Processo de Avaliação:

Este critério pode ser avaliado da seguinte maneira:

Seja CL_i um cluster formado pelas classes-objetos $C_1, \ldots, C_n$ e $ACI(C_i)$ o acoplamento de interação da classe-objetos C_i. O indicador do grau de *Acoplamento de Interação do Cluster* é calculado a partir de:

$$AICL(CL_i) = \frac{\sum_{i=1}^{n} ACI(Ci)}{n}$$

Escala		Interpretação
1	→	$0 < AICL \leq V_{aicl}$
0	→	$AICL > V_{aicl}$

Atributos de Qualidade Relacionados:

Avaliabilidade, Manutenibilidade, Reutilizabilidade: Um valor baixo para *AICL* é desejável, pois indica que a maioria das interações entre as classes-objetos se realizam dentro dos limites do *cluster*. Isto permite que o conjunto de classes-objetos possa ser implementado, reutilizado e testado com maior facilidade. Para o caso específico dos testes do *cluster*, estes tornam-se mais complexos, pois devem ser planejados casos de testes considerando as classes-objetos que estão dentro e fora do *cluster*.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Concisão **Critério:** ***Complementabilidade (COM)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se a especificação faz uso de documentos auxiliares - *referências, glossários, dicionários de dados* - que facilitam seu entendimento.

Discussão:

Em muitas aplicações, os usuários empregam um vocabulário próprio, que pode ser desconhecido para os desenvolvedores, a menos que estes tenham uma experiência substancial na organização. Em tais situações, o grupo que produz a especificação muitas vezes se vê obrigado a incluir explicações adicionais, com o objetivo de tornar mais compreensível a especificação. Na maioria dos casos, esse aumento de compreensão ocasiona redundâncias e aumento do volume de texto. Estes problemas podem ser contornados com a existência de referências a documentos prévios e/ou complementares, bem como à existência de um glossário para termos, simbologia e notações, que sirva como complemento para o entendimento da especificação.

Processo de Avaliação: Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. Existe um glossário com definições de termos técnicos, simbologia e notação utilizados na especificação e que não são de uso comum?	Sim → 1 Não→ 0
2. Existe um dicionário de dados centralizando as informações?	Sim → 1 Não→ 0
3. Existe referência a documentos prévios e/ou complementares necessários ao entendimento da especificação?	Sim → 1 Não→ 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** A existência de documentos auxiliares ajudam ao entendimento dos aspectos especificados e, por conseguinte, possibilitam sua avaliação com relação ao problema que descreve.
- ***Manutenibilidade***: A existência de documentos auxiliares tornam mais compreensível a especificação. Desta forma, os processos de manutenção são realizados com maior facilidade.
- ***Não Ambigüidade***: Especificações escritas utilizando dicionário de dados e/ou glossário como complemento reduzem a possibilidade de descrever e/ou entender aspectos iguais em forma diferente, evitando assim más interpretações.
- ***Consistência:*** Especificações escritas utilizando dicionário de dados e/ou glossário como complemento reduzem a possibilidade que se gerem conflitos entre aspectos na especificação.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Concisão **Critério:** ***Tamanho das Classes-Objetos (TC)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia a quantidade de atributos e serviços definidos na classe-objetos.

Discussão:

O tamanho adequado para uma classe-objetos varia dependendo da complexidade da aplicação e de sua responsabilidade no contexto do problema. O ***Tamanho da Classe-Objetos (TC)*** é definido como a quantidade de atributos e serviços declarados na classe-objetos.

Uma classe-objetos, com um valor de *TC* muito grande - *se comparado com valores para outras classes-objetos* -, pode ser uma indicação da necessidade de divisão em classes-objetos menores. Caso isso seja necessário, deve-se observar também o grau de coesão dos serviços definidos na classe-objetos. O número exato de atributos e serviços desejável para uma classe-objetos é difícil de se definir, pois este depende das particularidades da classe-objetos no contexto do domínio da aplicação.

Processo de Avaliação:

Seja a classe-objetos C_i , com *NA* atributos e *NS* serviços. O tamanho da classe-objetos pode ser calculado da seguinte forma:

$$TC(C_i) = NA + NS$$

Escala	**Interpretação**
$1 \rightarrow$	$0 \leq TC \leq V_{tc}$
$0 \rightarrow$	$TC > V_{tc}$

Sugestões para Correção:

- Verificar se a classe-objetos está realizado mais responsabilidades do que as que deve fazer. Neste caso, é possível realocar atributos e serviços em outras classes-objetos ou criar novas classes-objetos.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** Classes-objetos muito grandes tornam mais complexas as atividades de testes, pois uma maior quantidade de atributos e serviços precisam ser verificados e validados.
- ***Manutenibilidade:*** Uma classe-objetos com uma grande quantidade de atributos e serviços tende a ser mais complexa, limitando as capacidades do projetista de realizar modificações e desenvolver novas extensões para a aplicação.
- ***Reutilizabilidade:*** Do ponto de vista da reutilização, o tamanho da classe-objetos é um dilema. Por um lado, considerando-se o custo/benefício, uma classe-objetos com uma grande quantidade de atributos e serviços é desejável, na medida em que suas especializações possam herdar e utilizar uma grande quantidade de atributos e serviços. Entretanto, o tamanho de uma classe-objetos também possa ser uma indicação, de que a classe-objetos foi projetada para uma aplicação específica, reduzindo suas possibilidades de reutilização em outras aplicações.
- ***Coesão Estrutural da Classe-Objetos, Coesão Semântica da Classe-Objetos:*** Quantidade de atributos e serviços numa classe-objetos não deve ser considerado de maneira isolada. Ao analisar este critério, deve-se observar também o grau de relacionamento dos serviços especificados, caso a classe-objetos possa ser dividida em classes-objetos menores por causa de seu tamanho.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Concisão **Critério:** ***Tamanho do Serviço (TS)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia a quantidade de linhas de pseudocódigo definidas no serviço.

Discussão:

Embora não existam, ainda, pesquisas que determinem valores desejáveis para o tamanho dos serviços de uma classe-objetos, principalmente pelo fato deste ser altamente dependente da linguagem utilizada e do estilo de codificação do desenvolvedor, é evidente que esta característica afeta o entendimento da solução modelada. Serviços muito grandes devem ser subdivididos em serviços menores, preservando sua coesão. Quando um serviço cresce muito, aumenta a chance de incluir bagagem extra.

Podemos indicar algumas consequencias, não desejáveis, da existência de serviços muito longos:

- São difíceis de ler e entender;
- Podem ser um indicador de que o código que vai ser gerado é mais orientado a funções do que orientado a objetos;
- São, provavelmente, mais complexos e mais voltados para aplicações específicas, e,
- Apresentam, provavelmente, uma baixa coesão.

Processo de Avaliação:

Seja a S_1 ... S_n os serviços definidos numa classe-objetos C_i . O tamanho do serviço pode ser obtido a partir de:

$TS(S_i)$ = *número de linhas de pseudocódigo do serviço Si*

Escala	Interpretação
1 →	$0 \leq TS \leq V_{ts}$
0 →	$TS > V_{ts}$

Sugestões para Correção:

- Serviços muito grandes devem ser verificados quanto a possibilidade de serem divididos, sempre que isto não comprometa a coesão do serviço e da própria classe-objetos.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Serviços muito extensos são mais difíceis de serem lidos, entendidos e provavelmente mais complexos, dificultando a realização de modificações e o desenvolvimento de futuras extensões nas especificações.
- ***Reutilizabilidade:*** Serviços muito longos são provavelmente voltados para aplicações específicas, minimizando as possibilidades de reutilização.
- ***Coesão Semântica das Classes-Objetos, Coesão Semântica dos Serviços:*** Por ser este critério dependente da linguagem utilizada e do estilo de codificação do desenvolvedor, não deve ser considerado de maneira isolada. Caso seja necessário dividir um serviço em serviços menores, deve-se preservar a coesão do serviço e, conseqüentemente, a coesão da própria classe-objetos.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Concisão **Critério:** ***Tamanho do Cluster (TCL)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia a quantidade de classes-objetos definidas num *cluster*.

Discussão:

O *Tamanho do Cluster (TCL)* é entendido como a quantidade de classes-objetos definidas no *cluster*. Através do emprego de um mecanismo de identificação incremental das classes-objetos (*i.e. classes-objetos que vão surgindo ao longo do desenvolvimento e obedecendo a critérios pré-estabelecidos, com relação ao número de classes-objetos em cada cluster),* é possível ter-se um melhor controle entre os vários grupos que participam no desenvolvimento, melhorar a documentação, entender melhor os sistemas complexos, minimizar erros e aumentar a produtividade. Valores exatos da quantidade de classes-objetos para um determinado *cluster* não são, ainda, estabelecidos. No entanto, sugere-se que um *cluster* adequado contém em média cinco a vinte classes-objetos. Ao estabelecer restrições de tamanho aos *clusters,* pretende-se garantir que as classes-objetos que o compõem possam ser analisadas, projetadas, codificadas, reutilizadas e testadas com facilidade por uma equipe de desenvolvimento composta por duas a cinco pessoas. As restrições de tamanho poderão ser estabelecidas em função do tamanho da equipe de desenvolvimento e da complexidade da aplicação.

Processo de Avaliação:

Seja CL_i um cluster composto pelas classes-objetos C_1 ... C_n . Seu tamanho pode ser calculador através de:

TCL (CL_i) = *número de classes-objetos do cluster* CL_i

Escala		Interpretação
1	→	$0 \leq TCL \leq V_{tcl}$
0	→	$TCL > V_{tcl}$

Sugestões para Correção:

- Verificar se o conjunto de classes-objetos atende a mais de um objetivo que possibilite sua divisão, sem prejudicar a *coesão do cluster*.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Restrições de tamanho aos *clusters* pretendem garantir que as classes-objetos que o compõem possam ser avaliadas, mantidas e reutilizadas com maior facilidade.
- ***Viabilidade de Cronograma***: O tamanho dos *clusters* e, conseqüentemente, a quantidade de *cluster* envolvidos no desenvolvimento, tem um grande impacto no cumprimento dos cronogramas previstos para o projeto, pois o desenvolvimento da aplicação pode ser gerenciado através do uso destas abstrações.
- ***Coesão Estrutural, Coesão Semântica dos Clusters:*** Ao serem realizadas divisões nos *cluster* por causa de seu tamanho, deve-se sempre manter a coesão entre as abstrações que a compõem.

Objetivo: Confiabilidade da **Representação**	**Sub-Produto:** Modelo de Classes-Objetos
Fator: Comunicabilidade	**Técnica de Avaliação:** Inspeção Individual
Sub-Fator: Concisão	
Critério: ***Tamanho das Interfaces (TI)***	

Definição do Critério:

Característica que avalia a quantidade de argumentos definidos nas mensagens.

Discussão:

Segundo Jonhson e Foote (1988), uma forma de gerar projetos de alta qualidade, com classes reutilizáveis, é através do aperfeiçoamento da qualidade dos protocolos padrões (*i.e. novos comportamentos das mensagens e métodos)*. Isto pode ser realizado através da redução do número de argumentos que participam nas mensagens. Assim sendo, o *Tamanho das Interfaces (TI)* é entendido como a quantidade de argumentos definidos em uma mensagem. Um bom número para *TI* é seis, ou perto disso (Jonhson e Foote 1988). Por outro lado, outros autores recomendam que o protocolo de mensagem deve ser tão simples quanto possível e conter poucos parâmetros (em média não mais do que três parâmetros).

Processo de Avaliação:

Seja M_i uma mensagem composta pelos argumentos A_1 .. A_n . Seu tamanho pode ser calculado através de:

$TI\ (M_i)$ = *número de argumentos da mensagem* M_i

Escala		**Interpretação**
1	→	$0 \leq TI \leq V_{ti}$
0	→	$TI > V_{ti}$

Sugestões para Correção:

Reduzir o número de argumentos, dividindo uma mensagem em várias. Isto aumenta o número de mensagens com argumentos similares, que poderiam ter o mesmo nome. Além disso, é possível considerar a criação de uma nova classe-objetos que representa um pequeno grupo de argumentos.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Mensagens com muitos argumentos tornam mais complexa a comunicação entre as classes-objetos, conseqüentemente as atividades de testes e manutenção tendem a ser mais difíceis de serem realizadas.
- ***Reutilizabilidade:*** Uma das regras apontadas por Jonhson e Foote (1988) para gerar classes-objetos reutilizáveis é através do aperfeiçoamento da qualidade dos protocolos padrões, reduzindo o número de argumentos que participam nas mensagens.

Objetivo: Confiabilidade da Representação
Fator: Comunicabilidade
Sub-Fator: Simplicidade
Critério: ***Simplicidade dos Serviços (SIS)***

Sub-Produto: Modelo de Classes-Objetos
Técnica de Avaliação: Inspeção Individual

Definição do Critério:

Característica que avalia o grau de simplicidade dos serviços definidos na classe-objetos.

Discussão:

O critério simplicidade dos serviços (*SIS*) avalia o grau de simplicidade de cada classe-objetos através da análise de cada um de seus serviços. Desta forma, é conhecido o esforço necessário para desenvolver, testar e manter a classe-objetos. De fato, quanto maior o valor *SIS*, provavelmente, mais fácil será a reutilização, manutenção e teste da classe-objetos, pois se torna menos complexa. Espera-se, portanto, um valor de *SIS* o maior possível.

Processo de Avaliação:

Seja C_i uma classe, com n serviços $S_1, S_2, \ldots, S_n$, e sejam $s_1, s_2, \ldots, s_n$ os valores associados ao grau de simplicidade de cada serviço, considerando as instruções CASE, IF's e Ciclos utilizados e pela sua dificuldade de ser implementada, de acordo com a seguinte escala:

1 → Serviços sem instruções IF's, Case, Ciclos
0,75 → Serviços com instruções Ciclos
0,50 → Serviços com instruções IF's
0,25 → Serviços com instruções IF's, Ciclos
0 → Serviços com instruções IF's, Case, Ciclos

O valor da *Simplicidade dos Serviços* da classe-objetos é dada por:

$$SIS(Ci) = \sum_{i=1}^{n} si / n$$

Escala		Interpretação
1	→	$SIS \geq V_{SIS}$
0	→	$SIS < V_{SIS}$

Sugestões para Correção:

Alguns serviços nas classes-objetos são pela sua natureza complexos. Por outro lado, existirão situações nas quais será necessário manter em um mesmo serviço várias funcionalidades, tornando o serviço mais complexo. Dependendo do tipo da classe-objetos - *domínio ou suporte* - e de sua responsabilidade, cada serviço deverá ser avaliado caso a caso.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Classes-objetos com serviços muitos complexos são mais difíceis de serem testados e mantidos.
- ***Reutilizabilidade:*** Classes-objetos com serviços complexos, pode significar um grande impacto em suas subclasses, devido à herança de serviços.

Objetivo: Confiabilidade da Representação
Fator: Comunicabilidade
Sub-Fator: Simplicidade
Critério: ***Simplicidade dos Atributos (SIA)***

Sub-Produto: Modelo de Classes-Objetos
Técnica de Avaliação: Inspeção Individual

Definição do Critério:

Característica que avalia o grau de simplicidade dos atributos definidos na classe-objetos.

Discussão:

O critério simplicidade dos atributos (*SIA*) avalia o grau de simplicidade dos atributos definidos em cada classe-objetos. Desta forma, é conhecido o esforço necessário para desenvolver, testar e manter a classe-objetos. Um valor alto para *SIA* é um indicador da existência de uma classe-objetos menos complexa, provavelmente mais fácil de ser entendida, modificada, estendida e reutilizada.

Processo de Avaliação:

Seja C_i uma classe, com n atributos $A_1, A_2, ..., A_n$, e sejam $a_1, a_2, ..., a_n$ os valores associados ao grau de simplicidade de cada atributo, considerando sua estrutura:

- 1 → Inteiro, Real, Booleano
- 0,75 → Caracter
- 0,50 → Registro/Estrutura
- 0,25 → Referências
- 0 → Lista/Vetores

O valor da simplicidade dos atributos na classe-objetos é dada por:

$$SIA(Ci) = \sum_{i=1}^{n} ai / n$$

Escala		Interpretação
1	→	$SIA \geq V_{SIA}$
0	→	$SIA < V_{SIA}$

Atributos de Qualidade Relacionados

- ***Avaliabilidade, Manutenibilidade:*** Classes-objetos com atributos muitos complexos são mais difíceis de serem testados e mantidos.
- ***Reutilizabilidade:*** Classes-objetos com atributos complexos, pode significar um grande impacto em suas subclasses, devido à herança de serviços.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Conformidade **Critério:** ***Aderência a Normas da Organização Desenvolvedora (ANO)***	**Sub-Produto:** Documentação, Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se a especificação foi gerada considerando as normas estabelecidas pela organização desenvolvedora.

Discussão:

Uma especificação deve estar elaborada obedecendo a uma padronização pré-estabelecida, como conseqüência de normas da organização desenvolvedora da aplicação. Uma característica importante desta padronização é a homogeneização dos conhecimentos que permitem a geração de uma documentação com qualidade e garantem o desenvolvimento organizado do produto. Além disso, a comunicabilidade entre os membros da equipe é facilitada ao longo de toda a construção do produto.

Processo de Avaliação:

Este critério pode ser avaliado através da respostas à seguinte pergunta:	**Escala**
• A especificação foi gerada considerando as normas estabelecidas pela organização desenvolvedora?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Consistência:*** O uso de padrões pré-estabelecidos durante desenvolvimento permite que sejam geradas especificações consistentes.
- ***Não Ambigüidade:*** A obediência a uma padronização pré-estabelecida minimiza o surgimento de problemas de interpretação ao longo do desenvolvimento das especificações.

Objetivo: Confiabilidade da Representação **Fator:** Comunicabilidade **Sub-Fator:** Conformidade **Critério:** *Aderência as Normas estabelecidas pelo Contratante (ANC)*	**Sub-Produto:** Documentação, Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se a especificação foi gerada considerando as normas estabelecidas pelo contratante.

Discussão:

Toda especificação deve ser desenvolvida considerando padrões e convenções estabelecidas pelo contratante. Isto garante que o produto que está sendo construído apresente características da organização contratante, pois posteriormente formará parte do ambiente operacional da organização e será utilizada por diferentes conjuntos de pessoas e para diferentes propósitos. Além disso, as especificações deverão ser modificados ao longo do ciclo de vida por pessoal da organização, que possivelmente não participou na construção do produto.

Processo de Avaliação:	**Escala**
Este critério pode ser avaliado através da respostas à seguinte pergunta: • A especificação foi gerada considerando as normas estabelecidas pelo contratante ?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados

- ***Consistência:*** O uso de normas definidas pelo contratante, ajuda na geração de especificações consistentes.
- ***Não Ambigüidade:*** A obediência a normas minimiza os possível problemas relacionados a más interpretações.

IV.1.2. Fator: Manipulabilidade

Conjunto de atributos de qualidade que avaliam a facilidade de manipulação da especificação para diversas formas de uso.

É atingido através dos seguintes subfatores de qualidade:

- ***Disponibilidade:*** Conjunto de atributos de qualidade que avaliam a facilidade de acesso de uma especificação para seus usuários autorizados, na sua versão mais atualizada.
- ***Rastreabilidade:*** Conjunto de atributos de qualidade que avaliam a facilidade de se percorrer as especificações de requisitos e de projeto, identificando a agregação de detalhes a um determinado aspecto, desde sua visão mais global até a mais detalhada e vice-versa.

Objetivo: Confiabilidade da Representação **Fator:** Manipulabilidade **Sub-Fator:** Disponibilidade **Critério:** *Acessibilidade (ACE)*	**Sub-Produto:** Documentação, Modelo de Classes-objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se qualquer usuário autorizado pode facilmente consultar a especificação e/ou obter uma cópia da mesma.

Discussão:

A especificação deverá ser utilizada por diversas pessoas ao longo do desenvolvimento, devendo, também, atender às necessidades das atividades subseqüente (*avaliação, testes e manutenção)*. Portanto, deve ser facilmente acessível por seus usuários autorizados. Além disso, ela deve ser facilmente reproduzida após cada modificação, de modo que se possa dispor dela e utilizá-la sempre que necessário.

Processo de Avaliação: Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. A especificação - *modelo e sua documentação* - é possível de ser acessada por todos seus usuários autorizados?	Sim → 1 Não → 0
2. A especificação pode ser facilmente reproduzida?	Sim → 1 Não → 0
3. Existe uma cópia de segurança da especificação fora do local de trabalho?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Para poder conduzir uma avaliação, realizar uma manutenção e possibilitar a reutilização de uma especificação em outras aplicações similares, ela deve poder ser facilmente acessada e reproduzida por todos os seus usuários autorizados.

Objetivo: Confiabilidade da Representação	**Sub-Produto:** Documentação, Modelo de Classes-Objetos
Fator: Manipulabilidade	
Sub-Fator: Disponibilidade	**Técnica de Avaliação:** Inspeção Individual
Critério: *Estar Atualizada (ETA)*	

Definição do Critério:

Característica que avalia se a especificação reflete a informação mais recente.

Discussão:

Especificações poderão estar incorretas e/ou incompletas devido à evolução do problema a ser resolvido, ou pelo próprio desconhecimento de todas as suas implicações. Assim sendo, é perfeitamente normal para o usuário da especificação encontrar-se em passos posteriores do desenvolvimento, e/ou durante o seu uso, com a necessidade de efetuar mudanças ou acertos em especificações criadas em passos posteriores. Como a especificação serve de insumo para a geração de diversos produtos no desenvolvimento, torna-se evidente a necessidade de que ela esteja sempre atualizada, contendo as informações mais recentes.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Escala
1. O conteúdo da especificação - *modelo e documentação* - corresponde à informação mais recente, fruto da última manutenção?	Sim → 1 Não → 0
2. As atualizações realizadas estão todas datadas?	Sim → 1 Não → 0
3. As atualizações realizadas estão claramente indicadas?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade, Viabilidade de Implementação:*** Especificações não atualizadas são de grande risco ao proceder-se a avaliação, manutenção e implementação, pois a confiabilidade conceitual deixa de existir se a especificação não reflete a visão mais atualizada do problema.

Objetivo: Confiabilidade da Representação **Fator:** Manipulabilidade **Sub-Fator:** Rastreabilidade **Critério:** ***Organização da Documentação (ORD)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se o conjunto de especificações está organizada, de forma a facilitar sua manipulação.

Discussão:

Durante o processo de desenvolvimento, é gerado um conjunto de especificações com um grau variável de detalhe. É virtualmente impossível conseguir-se manipular e entender este conjunto de especificações sem que se disponha de uma documentação que esteja organizada. Com uma documentação organizada será possível ao leitor da especificação percorrer as especificações de requisitos e de projeto, identificando a agregação de detalhes a um determinado aspecto, desde sua visão mais global até a mais detalhada e vice-versa.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Escala
1. A organização do conjunto de documentos - *especificações de requisitos* - permite uma hierarquização do mesmo, desde a especificação mais detalhada até a mais macroscópica e vice-versa?	Sim → 1 Não → 0
2. A organização do conjunto de documentos - *especificações de projeto* - permite uma hierarquização do mesmo, desde a especificação mais detalhada até a mais macroscópica e vice-versa?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados

- ***Avaliabilidade, Manutenibilidade, Reutilizabilidade:*** Um conjunto de especificações com uma organização adequada permite identificar com facilidade os aspectos que serão avaliados, mantidos ou reutilizados.
- ***Modularidade da Documentação:*** A existência de especificações modulares facilitam a rastreabilidade da especificação.

Objetivo: Confiabilidade da Representação **Fator:** Manipulabilidade **Sub-Fator:** Rastreabilidade **Critério:** ***Localizabilidade Interna (LOI)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se existem facilidades para se localizar todos os elementos, dentro de uma especificação, relacionados com determinado aspecto ou assunto.

Discussão:

Os requisitos devem ser indexados, segmentados e referenciados de forma cruzada para facilitar o uso e modificação da especificação. Portanto, a especificação é um documento que deve fornecer caminhos que podem ser facilmente percorridos, permitindo, a partir de certos requisitos, alcançar outros requisitos. Essa facilidade de percorrer itens dentro da especificação está relacionada à existência de mecanismos que facilitem o acesso a todos os elementos da especificação que estão relacionados. Para facilitar a localização dos itens relacionados a um determinado aspecto ou assunto, a especificação deve conter sumários, índices remissivos e tabelas de referências cruzadas explícitas.

A especificação não deve ter referências externas a ela. Caso seja necessário, a informação que se deseja referenciar deve ser reproduzida na especificação.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. A especificação apresenta um sumário?	Sim → 1 Não → 0
2. Existem índices remissivos?	Sim → 1 Não → 0
3. Existem referências cruzadas explícitas?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** Especificações devem fornecer mecanismos que viabilizem a execução de avaliações dos requisitos (testes nas fases posteriores), permitindo seu rastreabilidade desde sua primeira formulação até a mais detalhada.
- ***Manutenibilidade:*** Especificações que fornecem mecanismos de acesso a seus elementos de informação facilitam a localização de aspectos que foram alterados como resultado de modificações e/ou evoluções na especificação.

Objetivo: Confiabilidade da Representação **Fator:** Manipulabilidade **Sub-Fator:** Rastreabilidade **Critério:** ***Localizabilidade Externa (LOE)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se existem facilidades para se localizar todas as especificações e demais documentos relacionados a um determinado aspecto ou assunto.

Discussão:

Os itens da especificação têm sua antecedência definida claramente nas etapas iniciais da especificação, ou nas declarações dos objetivos do sistema. Para grandes especificações, cada item deve indicar o item ou os itens das especificações iniciais das quais foram derivadas.

Assim como desejamos localizar os itens relacionados a um determinado assunto numa mesma especificação, também desejamos poder localizar todas as especificações e demais documentos que tratem de um determinado assunto e, dentro destes, localizar onde é tratado o assunto procurado.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Escala
1. Existem sumários que organizam os documentos?	Sim → 1 Não → 0
2. Existem índices remissivos?	Sim → 1 Não → 0
3. Existem referências cruzadas explícitas?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Manutenibilidade:*** É evidente que, para realizar mudanças em especificações, é necessário que ela forneça facilidades para localizar o item, ou itens, que vão a ser alterados e, além disso, localizar todas as especificações e documentos externos afetados pela mudança.

IV.2. Avaliação do Objetivo Confiabilidade Conceitual

Uma especificação deve registar todas as características que o software deve conter, com o principal objetivo de satisfazer as necessidades dos seus usuários. Portanto, a especificação deve estar correta com relação ao seu conteúdo, isto é, ao problema que descreve. Assim sendo, o objetivo ***Confiabilidade Conceitual*** refere-se às características de qualidade que avaliam se uma especificação é confiável para seus usuários, do ponto de vista de seu conteúdo, satisfazendo os requisitos que motivara a sua construção.

Este objetivo se realiza através dos fatores de qualidade ***Fidedignidade*** e ***Suficiência*** (Figura A1.3).

IV.2.1. Fator: Fidedignidade

Conjunto de atributos de qualidade que avaliam se a especificação representa o que é entendido como sendo as necessidades e expectativas dos usuários do produto.

É atingido através dos seguintes subfatores de qualidade:

- ***Consistência:*** Conjunto de atributos de qualidade que avaliam se a especificação está isenta de contradições entre os aspectos especificados.
- ***Não Ambigüidade:*** Conjunto de atributos de qualidade que avaliam se o conteúdo da especificação está expresso de forma a evitar a possibilidade de diferentes interpretações para qualquer aspecto ou assunto.
- ***Correção Semântica da Arquitetura:*** Conjunto de atributos de qualidade que avaliam a correção das abstrações modeladas de acordo com o produto que especifica.
- ***Correção Comportamental:*** Conjunto de atributos que avaliam a correção da especificação do ponto de vista dinâmico.

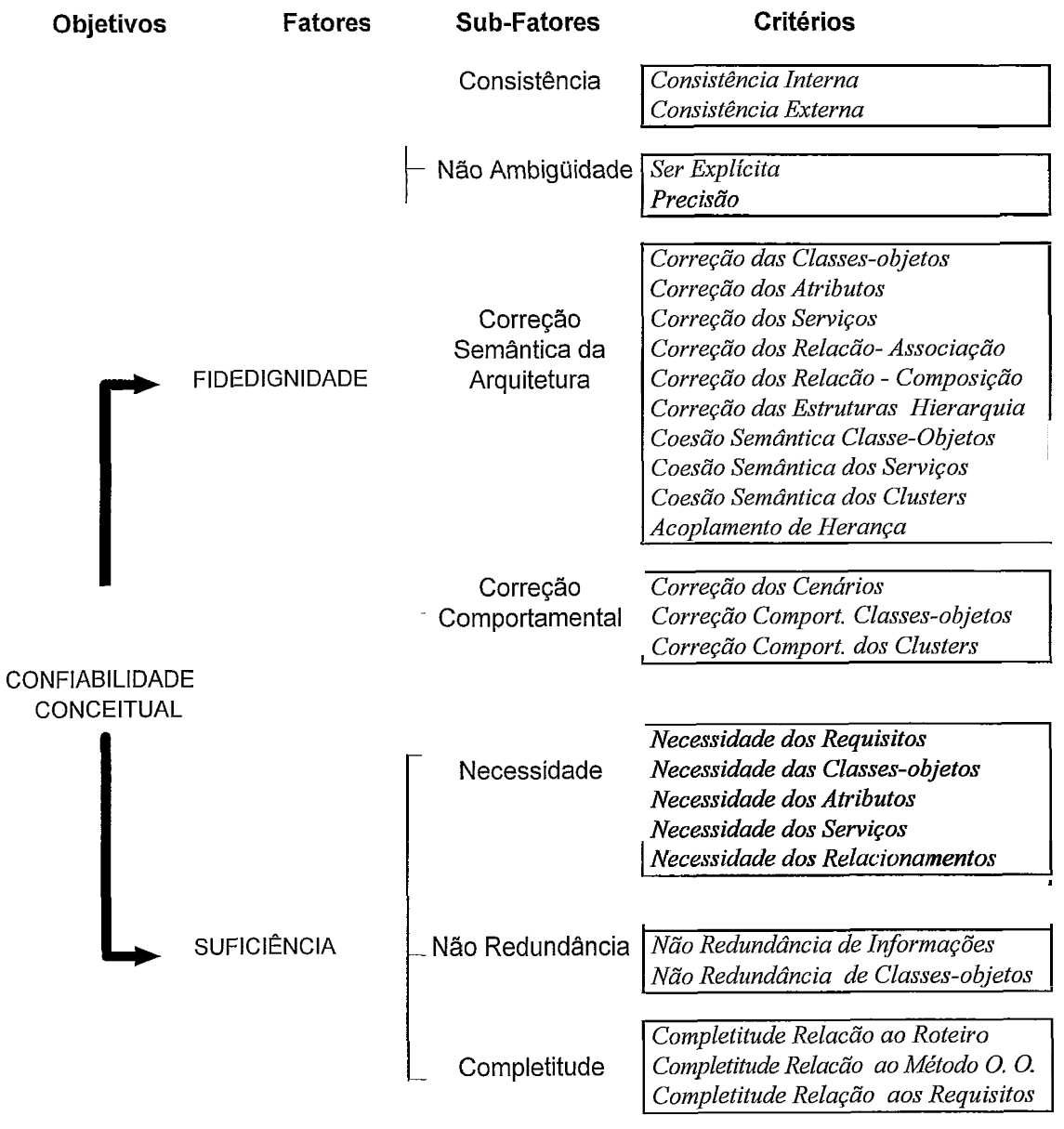


Figura A1-3. - Características de Qualidade de Especificações segundo o Objetivo Confiabilidade Conceitual

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Consistência **Critério:** ***Consistência Interna (COI)***	**Sub- Produto:** Documentação, Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia a existência de conflitos entre aspectos especificados na mesma especificação.

Discussão:

Como as especificações evoluem em detalhe a medida que avança o processo de desenvolvimento, é perfeitamente possível que passos de refinamentos subseqüentes tornem não válidos aspectos aceitos em passos anteriores. É provável que, como conseqüência da atividade de evolução, sejam geradas contradições que originem conflitos com aspectos anteriormente especificados, ou seja, um mesmo aspecto descrito é considerado de forma diferente em outro lugar da mesma especificação. Existe a possibilidade de que, ao realizar manutenções na especificação, sejam introduzidas contradições entre aspectos que inicialmente eram consistentes.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Escala
1. A especificação não apresenta termos diferentes que possuem o mesmo significado e que são utilizados para descrever um mesmo objeto, que é tratado em contextos e lugares diferentes?	Sim → 1 Não → 0
2. A especificação não apresenta contradições entre características específicas do produto?	Sim → 1 Não → 0
3. A especificação não apresenta conflitos lógicos ou temporais entre requisitos do produto que dependem do tempo?	Sim → 1 Não → 0
4. A especificação não apresenta conflitos na descrição de aspectos de comportamento do produto?	Sim → 1 Não → 0
5. A especificação não apresenta conflitos entre o modelo de classes-objetos e o modelo dinâmico.	Sim → 1 Não → 0
6. A especificação não apresenta conflitos em relação a decisões de modelagem?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Manutenibilidade, Avaliabilidade:*** Especificações com aspectos conflitantes conduzem a avaliações não confiáveis. Além disso, conduzem a evoluções e modificações que podem gerar novos conflitos e inclusive agravar os já existentes de maneira a ocasionar repercusões graves, nas fases posteriores.
- ***Não Ambigüidade:*** Especificações com aspectos conflitantes são de grande risco, pois podem causar más interpretações e, portanto, uma inadequada implementação do produto.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Consistência **Critério:** ***Consistência Externa (COE)***	**Sub-Produto:** Documentação, Modelo de Classe - Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia a existência de conflitos entre aspectos especificados em outras especificações ou entidades externas.

Discussão:

Assim como podem surgir contradições entre aspectos especificados na mesma especificação, pode acontecer que informações introduzidas contenham contradições, quando comparadas com outras especificações anteriores ou posteriores. As inconsistências podem, ainda, estar relacionadas a entidades externas, tais como especificações de sistemas existentes ou em desenvolvimento na organização, especificações de software para o tratamento das interfaces com o usuário, pacotes de software etc.

Processo de Avaliação:

Este critério pode ser avaliado através da respostas à seguinte pergunta:	**Escala**
• A especificação não apresenta aspectos conflitantes com relação a outras especificações ou entidades externas ?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Não Ambigüidade:*** Especificações livres de aspectos conflitantes com outras especificações ou entidades externas reducem o problema de más interpretações.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Não Ambigüidade **Critério:** ***Ser Explícita (EXP)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se na especificação existem condições, hipóteses e/ou restrições definidas por contexto.

Discussão:

Especificações com aspectos descritos de forma implícita podem causar soluções diferentes da esperada. Além disso, podem gerar problemas na aceitação do produto pelo usuário, por não terem ficado bem definidas nas condições contratuais. Esta situação pode exigir o cumprimento de requisitos implícitos diferentes dos que originalmente se assumiu. Por isso, a especificação não deve conter condições, hipóteses e /ou restrições definidas por contexto, que possam conduzir a suposições que gerem alternativas e/ou soluções não apropriadas.

Processo de Avaliação:	
Este critério pode ser avaliado através da respostas à seguinte pergunta: • A especificação não apresenta aspectos conflitantes com relação a outras especificações ou entidades externas ?	**Escala** Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Não se pode pretender realizar avaliações e alterações de forma confiável, se na especificação existem aspectos que não ficaram completamente compreendidos pelos usuários e desenvolvedores.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Não Ambigüidade **Critério:** ***Precisão (PRC)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se os aspectos especificados estão descritos de forma precisa e, sempre que possível, quantificada.

Discussão:

É muito importante que se possa determinar se o software a ser desenvolvido vai satisfazer os requisitos estabelecidos. Para que isto aconteça, a especificação deverá poder ser validada com relação às necessidades e expectativas dos futuros usuários do produto. Portanto, os aspectos deverão, na medida do possível, ser especificados numa forma exata e quantificável.

Processo de Avaliação: Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. Os requisitos do produto estão descritos de forma possível de serem validados?	Sim → 1 Não → 0
2. Os termos de significado múltiplo, apresentam uma definição precisa?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** Aspectos especificados, numa forma exata e quantificável, permitem que sejam realizadas avaliações confiáveis.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Correção das Classes-Objetos(CRC)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia a correta definição das classes-objetos.

Discussão:

Definir uma classe-objetos como uma abstração do mundo real é essencial para o entendimento da modelagem do domínio do problema e para a implementação da solução computacional. Por isso, uma inadequada definição das classes-objetos pode ter um impacto significativo no êxito do projeto.

Processo de Avaliação: Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. A classe-objeto representa uma abstração significativa para o domínio do problema?	Sim → 1 Não → 0
2. O nome da classe-objetos obedece a padrões pré-estabelecidos e correspondem ao vocabulário utilizado pelos especialistas de domínio, caso sejam classes-objetos de domínio?	Sim → 1 Não → 0
3. O nome da classe-objetos é único na especificação?	Sim → 1 Não → 0
4. Cada objeto na classe é descrito a partir do conjunto de atributos e serviços?	Sim → 1 Não → 0
5. O conjunto de atributos e serviços identificados caracterizam adequadamente a classe-objetos no contexto do domínio do problema?	Sim → 1 Não → 0
6. Cada classe-objetos tem uma descrição precisa de seu significado no contexto do domínio do problema?	Sim → 1 Não → 0

Sugestões para Correção:

- Nas primeiras semanas de modelagem do domínio, o apoio de pessoal especialista em orientação a objetos, se torna necessário, para garantir a seleção e definição de classes-objetos adequadas.

Atributos de Qualidade Relacionados:

- ***Viabilidade de Implementação:*** Especificações com classes-objetos incorretas são de grande risco ao proceder-se a implementação, isto porque a confiabilidade conceitual do modelo deixa de existir.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Correção dos Atributos (COA)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia a correta definição dos atributos especificados na classe-objetos.

Discussão:

Um atributo é uma abstração de uma única característica possuída por uma classe ou seu conjunto de objetos. Cada atributo possui um valor para cada instância de objeto, podendo ser um valor atômico, uma referência ou uma estrutura de dados. Os atributos representam as características da classe-objetos de domínio - *atributos de domínio* - ou resultados de decisões de projeto e implementação - *atributos de implementação.* O objetivo é obter, para cada classe-objetos, um conjunto de atributos que sejam:

- *únicos,* ou seja, os nomes dos atributos devem ser únicos na classe-objetos - *o que não é verdadeiro quando se considera todas as classes-objetos do modelo.*
- *totalmente fatorados,* ou seja, cada atributo captura um aspecto separado da abstração da classe-objetos.
- *completos,* ou seja, abrangem todas as informações pertinentes à classe-objetos que está sendo definida.

Processo de Avaliação: Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. Os nomes dos atributos obedecem a padrões pré-estabelecidos e correspondem ao vocabulário utilizado pelos especialistas do domínio, caso sejam atributos de domínio?	Sim → 1 Não → 0
2. Os nomes dos atributos são únicos na classe-objetos?	Sim → 1 Não → 0
3. Cada atributo captura um aspecto separado da abstração da classe-objetos e possui um valor para cada instância de objeto, no contexto do domínio do problema?	Sim → 1 Não → 0
4. Os atributos abrangem todas as informações pertinentes à classe-objetos que está sendo descrita?	Sim → 1 Não → 0
5. Cada atributo apresenta uma descrição precisa de seu significado, no contexto do domínio do problema?	Sim → 1 Não → 0
6. Cada atributo apresenta, quando pertinente: • seu tipo; • enumeração de todos seus valores possíveis; • referência do documento onde estão especificados e/ou listados estes valores; • identificação de valor(es) default; • regras que determinam quais valores são permitidos; • identificação do intervalo de valores possíveis; • visibilidade?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Viabilidade de Implementação:*** Classes-objetos com atributos incorretos são de grande risco ao proceder-se a implementação, isto porque a confiabilidade conceitual da classe-objetos deixa de existir.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Correção dos Serviços (COS)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia a correta definição dos serviços especificados na classe-objetos.

Discussão:

O serviço é uma atividade realizada por uma classe ou por um objeto. De modo geral, podem ser classificados em dois tipos: *serviços de domínio* - em sua maioria identificados, analisados e especificados durante a fase de análise - e *serviços de implementação* - resultado de decisões de projeto e codificação. Para garantir sua correta implementação, é necessário que cada serviço apresente uma definição precisa e que seja especificado, em pseudo-código ou representação equivalente, obedecendo aos padrões pré-estabelecidos para a organização ou para o projeto específico.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Escala
1. Os nomes dos serviços obedecem aos padrões pré-estabelecidos?	Sim → 1 Não → 0
2. Os nomes dos serviços são únicos na classe-objetos?	Sim → 1 Não → 0
3. Cada serviço apresenta uma descrição precisa de seu signifcado, no contexto do domínio da aplicação?	Sim → 1 Não → 0
4. Cada serviço apresenta sua pseudocodificação, obedecendo as regras sintáticas e semânticas da linguagem de projeto definida para o projeto?	Sim → 1 Não → 0
5. Serviços com nomes diferentes realizam funções diferentes?	Sim → 1 Não → 0
6. Os serviços especificados na classe são aplicáveis a cada um dos objetos da classe?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Viabilidade de Implementação:*** Classes-objetos com serviços incorretos são de grande risco ao proceder-se a implementação, isto porque a confiabilidade conceitual da classe-objetos deixa de existir.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Correção dos Relacionamentos de Associação (COR)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia a correta definição dos relacionamentos de associação.

Discussão:

As classes-objetos não existem de maneira isolada. Durante o processo de modelagem, os desenvolvedores estão interessados em identificar as associações entre as classes-objetos e em refletir essas associações no modelo através de relacionamentos precisamente determinados. Estas associações indicam a dependência semântica entre duas classes-objetos, e os indicadores de cardinalidade devem refletir a semântica correta da associação entre as classes-objetos envolvidas, segundo os requisitos domínio da aplicação. No relacionamento deve estar indicado claramente o significado semântico, através do uso de um nome simples e preciso. Caso contrário, pode ser uma indicação de um relacionamento incompleto ou que várias associações estão sendo modeladas através de um único relacionamento. Além do nome, qualquer relacionamento de associação precisa indicar as classes-objetos envolvidas, além de uma defimição das restrições quando pertinente. Seu correto registro, é essencial para o entendimento da solução modelada e sua correta implementação.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Escala
1. Os relacionamentos apresentam um descritor indicando claramente a semântica do relacionamento, indicando o objeto cliente e o servidor?	Sim → 1 Não → 0
2. O sentido de dependência é consistente com o intercâmbio de mensagens?	Sim → 1 Não → 0
3. Para cada relacionamento entre pares de objetos, a quantidade (m) ou intervalo (m,n), a partir da perspectiva de cada objeto, está de acordo com às restrições de cardinalidade impostas, segundo os requisitos do domínio do problema?	Sim → 1 Não → 0
4. Para cada par de relacionamentos existe um mapeamento no domínio do problema?	Sim → 1 Não → 0
5. Para cada par de relacionamentos estão associadas as responsabilidades do sistema?	Sim → 1 Não → 0
6. Se o relacionamento requerido não pode ser representado, existe uma forma textual de indicar esta situação?	Sim → 1 Não → 0
7. Para cada par de relacionamentos estão definidos os serviços de criação, destruição, associação, desassociação e tratamento de referências.	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Uma correta definição dos relacionamentos de associação permite que avaliações e manutenções possam ser realizadas com confiabilidade. Além disso, é possível que sejam realizadas validações por parte do usuário.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Correção dos Relacionamentos de Composição (CRC)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia a correta definição dos relacionamentos de composição especificados.

Discussão:

A correta modelagem do relacionamento de composição é importante porque nos proporciona um discernimento significativamente maior do domínio do problema e das responsabilidades do sistema. Permite obter um modelo de classes-objetos mais simples de ser entendido, devido à separação de aspectos de interesse do domínio do problema, garantindo a propriedade de visibilidade.

Relacionamentos de composição podem ser avaliados segundo as duas propriedades seguintes (Firesmith 1993, Rumbaugh et al. 1994):

- *Transitividade,* isto é, se a Classe_A faz parte da Classe_B e a Classe_B faz parte da Classe_C, então a Classe_A faz parte da Classe_C.
- *Anti-simetria,* isto é, se a Classe_A faz parte da Classe_B, então a Classe_B não faz parte da Classe_A.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Escala
1. Para cada relacionamento de composição, a quantidade (m) ou intervalo (m,n), a partir da perspectiva de cada objeto, está de acordo com as restrições de cardinalidade impostas segundo os requisitos do domínio do problema?	Sim → 1 Não → 0
2. O relacionamento satisfaz a propriedade de transitividade?	Sim → 1 Não → 0
3. O relacionamento satisfaz a propriedade de anti-simetria?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** A correta modelagem dos relacionamentos de composição torna mais comprensível o modelo, permitindo que avaliações e manutenções sejam feitas com maior confiabilidade.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Correção da Hierarquia de Classes-Objetos (CHC)***	**Produto:** Modelo Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia a correta definição das estruturas de hierarquia de classes-objetos.

Discussão:

Uma das formas mais comuns de organização do conhecimento é através do uso de representações hierárquicas, formando uma estrutura de grafo acíclico dirigido. Estas hierarquias facilitam a modelagem pela estruturação de classes-objetos e incorporam, resumidamente, o que é semelhante e o que é diferente em relação a elas, facilitando o entendimento do modelo e evitando a duplicação de código. Cada classe-objetos na hierarquia deve refletir um inter-relacionamento com as demais classes-objetos que compartilham a estrutura. Além disso, o topo de uma hierarquia - *generalização* - deve ser uma classe abstrata e as folhas - *especializações* - devem ser classes-objetos concretas. Uma classe-objetos é considerada uma especialização se herda todos os atributos e serviços da classe de generalização podendo redefini-las e/ou acrescentar novos a si própria. Portanto, uma especialização poderá representar uma extensão do comportamento de seus pais.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Valores
1. Cada generalização e especialização é nomeada de acordo com os padrões pré-estabelecidos?	Sim → 1 Não → 0
2. A classe de generalização é sempre uma classe abstrata e as classes de especialização são sempre classes concretas?	Sim → 1 Não → 0
3. Uma especialização adiciona pelo menos um atributo ou serviço (pode estar redefinido) com relação a sua classe de generalização?	Sim → 1 Não → 0 Sim → 1 Não → 0
4. Existem pelo menos um atributo ou um serviço (pode estar redefinido) que distingue as especializações de uma mesma classe de generalização?	Sim → 1 Não → 0
5. Cada classe-objetos na hierarquia reflete um inter-relacionamento com as demais classes-objetos que compartilham a estrutura?	Sim → 1 Não → 0
6. A estrutura de hierarquia tem nomes de atributos e serviços únicos em relação aos ancestrais e descendentes (quando pertinente)?	Sim → 1 Não → 0
7. A estrutura de hierarquia tem nomes de atributos e serviços diferentes nas classes-objetos de generalização para cada parte de um entrelaçamento (herança múltipla) ?	Sim → 1 Não → 0
8. São utilizados discriminadores para indicar o critério de divisão da estrutura de hierarquia?	Sim → 1 Não → 0
9. As características de uma classe de generalização são aplicáveis a todas as classes-objetos especializações?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Manutenibilidade:*** Uma correta hierarquia de classes-objetos permitem o uso de polimorfismo, responsável pela facilidade de extensão e modificação de um software orientado a objetos.
- ***Concisão:*** Uma estrutura de hierarquia de classes-objetos facilita o entendimento e evita a duplicação de código e documentação.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Coesão Semântica das Classes-Objetos (CSC)***	**Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia o grau de relacionamento semântico entre os conceitos especificados na classe-objetos.

Discussão:

Coesão é definida como um atributo de qualidade de módulos individuais, descrevendo o grau em que seus componentes são necessários para realizar uma mesma tarefa. Para o caso de sistemas orientados a objetos, o termo *módulo* pode ser substituído pelo termo *classe-objetos* e a definição de coesão passa a ser entendida como a característica da classe-objetos de ter nela especificados um conjunto de serviços relacionados que contribuem para que ela possa cumprir com suas responsabilidades.

A *Coesão Semântica das Classes-objetos* pode ser avaliada considerando a seguinte classificação:

- *coesão por abstração,* ocorre quando a classe-objetos está composta por um conjunto de atributos e serviços relacionados que caracterizam adequadamente a abstração, visando o cumprimento de suas responsabilidades.
- *coesão por coincidência,* ocorre quando a classe-objetos está composta por um conjunto de atributos e serviços que não caracteriza adequadamente a classe-objetos realizando funções não relacionadas com suas responsabilidades.

Processo de Avaliação:

Assinale de acordo com a escala o valor que melhor representa o grau em que o critério foi atingido:

- A classe-objetos está composta por atributos e serviços relacionados que caracterizam adequadamente a abstração, visando o cumprimento de suas responsabilidades ?

Escala	Interpretação
1 →	Total Presença
0,75 →	Alta Presença
0,50 →	Moderada Presença
0,25 →	Baixa Presença
0 →	Total Ausência

Sugestões para Correção:

- Verificar a possibilidade de dividir os atributos e serviços da classe-objetos em duas ou mais subclasses.
- Verificar a possibilidade de realocar atributos e serviços em outras classes-objetos.
- Verificar a possibilidade de criar novas classes-objetos com responsabilidades definidas.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Uma classe-objetos que apresenta atributos e serviços aparentemente não relacionados indica uma maior complexidade da classe-objetos, aumentando, assim, a possibilidade de erros durante o desenvolvimento da aplicação. Conseqüentemente, as atividades de avaliação (testes) e manutenção são mais difíceis de serem realizadas.
- ***Reutilizabilidade:*** Classes-objetos que representam unidades menores, com atributos e serviços relacionados, são mais fáceis de entender e modificar e apresentam maiores possibilidades de reutilização do que as grandes e complexas.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Coesão Semântica dos Serviços(CSS)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia o grau de relacionamento semântico entre os elementos de informação definidos no serviço.

Discussão:

Classes-objetos apresentam um ou mais serviços, onde cada um deles deve realizar uma única operação. A *Coesão Semântica do Serviço* deve ser entendida como a força de associação funcional dos elementos definidos no serviço. Isto é, o serviço deve realizar uma única função e deve ser possível descrever seu propósito de forma precisa, através de uma sentença simples, contendo um único verbo (Yourdon 1994). A baixa coesão de um serviço da classe-objetos *(i.e. o serviço realiza mais de uma função)* indica que, provavelmente, é necessária uma modificação na classe-objetos correspondente para obter-se serviços que possuam o nível de granularidade desejado.

A *Coesão Semântica de Serviço* pode ser avaliada considerando a seguinte classificação:

- *coesão operacional,* ocorre quando o serviço especificado na classe-objetos realiza uma única operação.
- *coesão por coincidência,* ocorre quando o serviço realiza várias operações diferentes, não relacionadas.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta: • Cada serviço especificado na classe-objeto realiza uma única operação ?	**Escala** Sim → 1 Não → 0

Sugestões para Correção:

- Tentar dividir em serviços menores, um para cada propósito que a classe-objetos deve servir.
- Tentar criar e /ou modificar a estruturas de hierarquia de classe-objetos, se necessário.
- Adicione novas classes-objetos sempre que os serviços não possam ser realocados em classes-objetos existentes.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Classes-objetos com serviços que realizam uma única operação são provavelmente menos complexos, portanto, os processos de avaliação (testes) e manutenção da classe-objetos podem ser realizados com maior facilidade.
- ***Reutilizabilidade:*** Classes-objetos com serviços que realizam uma única operação tem maior possibilidade de serem reutilizadas, pois são mais fáceis de entender e provavelmente menos complexos.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Coesão Semântica dos Clusters(CSCL)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia o grau de relacionamento semântico entre os serviços definidos nas classes-objetos que compõem o *cluster.*

Discussão:

O *cluster* é um mecanismo que permite que um conjunto de classes-objetos seja organizado segundo objetivos específicos *(i.e. entendimento, teste, reutilização, implementação).* Um *cluster* é considerado coeso, se todas as suas abstrações forem ao encontro de seu objetivo específico. Um *cluster* deve representar um único objetivo específico e deverá estar composto por um conjunto de classes-objetos relacionadas, onde cada classe-objetos contribui para o atendimento deste objetivo. Assim sendo, a *Coesão de Cluster* pode ser avaliada considerando os seguintes tipo de coesão:

- *coesão das classes,* ocorre quando o *cluster* está composto por um conjunto de classes-objetos relacionadas, visando o objetivo específico definido para o *cluster.*
- *coesão por coincidência,* ocorre, quando, o *cluster* está composto por um conjunto de classes-objetos não relacionadas ao objetivo definido que o *cluster* deve atingir.

Processo de Avaliação:

Assinale de acordo com a escala o valor que lhe parece melhor representar o grau em que o critério foi atingido:

- O *cluster* está composto por um conjunto de classes-objetos que realizam funções relacionadas de acordo com o objetivo definido?

Escala	Interpretação
1 →	Total Presença
0,75 →	Alta Presença
0,50 →	Moderada Presença
0,25 →	Baixa Presença
0 →	Total Ausência

Sugestões para Correção:

- Verificar a possibilidade de realocar classes-objetos em *cluster* adjacentes.
- Verificar a possibilidade de criar novos *clusters* com objetivos perfeitamente definidos.

Atributos de Qualidade Relacionados

- ***Avaliabilidade, Manutenibilidade:*** *Clusters* que realizam múltiplos objetivos, ou que gerenciam apenas parte de um objetivo, tornam-se mais difíceis de serem avaliados com relação ao objetivo que deve cumprir. Além disso, o processo de modificação e/ou evolução são mais difíceis de se realizar.
- ***Reutilizabilidade:*** *Clusters* com baixa coesão apresentam menos chances de serem reutilizados em outras aplicações, pois não apresentam uma visão clara de seu objetivo.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Semântica da Arquitetura **Critério:** ***Acoplamento de Herança (ACH)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia o grau em que são utilizados os atributos e serviços herdados.

Discussão:

O *Acoplamento de Herança* deve ser entendido como o grau em que uma determinada classe-objetos utiliza os atributos e serviços (*i.e. estrutura e comportamento*) de sua(s) classe-objetos ancestral (is). Ter-se *Acoplamento de Herança* elevado é desejável, na medida em que os atributos e serviços herdados são utilizados pelas suas especializações, podendo, no caso dos serviços, ser redefinidos. Assim sendo, o *Acoplamento de Herança* pode acontecer, de modo geral, de duas formas diferentes:

- *acoplamento por extensão,* ou seja, a sub-classe utiliza os atributos e serviços herdados, os redefine ou adiciona outros atributos e serviços.
- *acoplamento por implementação,* ou seja, serviços e atributos são modificados por conveniência para reutilizar ou compartilhar código. Isto é, a subclasse é torna-se uma classe semelhante à desejada.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Numa hierarquia de classes-objetos, as sub-classes utilizam os atributos e serviços herdados, os redefinem ou adicionam outros atributos ou serviços?	Sim → 1 Não → 0

Sugestões para Correção:

- Buscar estruturas de hierarquia alternativas, nas quais cada subclasse herda e utiliza os atributos e serviços.

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** A presença de acoplamento por implementação é semanticamente errado e conduz a problemas de manutenção, e ao serem realizados os testes das classes-objetos, pois não há um relacionamento coerente entre as classes ancestral e descendente. Isto é, toda vez que são realizados extensões e/ou modificações na classe-ancestral, estas não são aplicáveis às suas especializações.
- ***Reutilizabilidade:*** Através do acoplamento de herança, características comuns podem ser reutilizadas, não somente pela subclasse em questão, mas também por qualquer outra classe-objetos futura que precise destas características.
- ***Concisão:*** Através do mecanismo de herança, evita-se a duplicação do esforço da modelagem e na documentação.
- ***Nível de Fatoração das Classes-objetos:*** Uma classe-objetos, com uma grande quantidade de descendentes imediatos, é uma indicação de quantas subclasses vão herdar os atributos e serviços da classe ancestral.
- ***Nível de Profundidade da Hierarquia de Classes-objetos:*** Através de uma análise de uma hierarquia, é possível identificar em que medida as classes ancestrais afetam as demais classes, visualizando-se a profundidade das classes na hierarquia e o número de serviços herdados.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Comportamental **Critério:** ***Correção dos Cenários (CCE)***	**Sub - Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se na especificação está definido de forma correta a seqüência de interações entre classes-objetos.

Discussão:

Cada função especificada no documento de requisitos está associada a um ou mais cenários. Portanto, para garantir que estamos atendendo os requisitos da aplicação, devemos realizar uma análise detalhada dos cenários. Um cenário pode ser definido como uma seqüência específica de interações entre classes-objetos, com o objetivo de atingir uma funcionalidade desejada para o sistema. Além de ser uma auxílio para compreender o comportamento dinâmico do sistema, sua correta definição é importante, porque permite: *identificar novos objetos; avaliar a necessidade das classes-objetos, atributos, serviços e relacionamentos; refinar e melhorar a distribuição das responsabilidades das classes-objetos; verificar a consistência entre as responsabilidades das classes-objetos; definir casos de testes para as classes-objetos e, conseqüentemente, para todo o sistema, e; rastrear os requisitos da aplicação.*

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. Cada cenário está associado a uma funcionalidade requerida pelo produto?	Sim → 1 Não → 0
2. Cada cenário tem identificado seus eventos iniciais?	Sim → 1 Não → 0
3. Cada cenário tem identificado seus dados iniciais?	Sim → 1 Não → 0
4. Cada cenário tem identificada a classe-objetos que recebe o evento inicial (entrada) e a resposta (saída)?	Sim → 1 Não → 0
5. Cada cenário tem identificado os caminhos de comunicação entre as classes-objetos envolvidas?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** Através da identificação de cenários corretos é possível definir diversos casos de testes para classes-objetos e para todo o sistema.
- ***Necessidade:*** Através de cenários é possível avaliar a necessidade de classes-objetos, atributos, serviços e relacionamentos especificados.
- ***Rastreabilidade:*** Através dos caminhos de comunicação entre as classes-objetos é possível rastrear os requisitos do problema.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Comportamental **Critério:** ***Correção do Comportamento das Classes (CCC)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se na especificação está definido de forma correta o comportamento de cada classe-objetos.

Discussão:

Algumas classes-objetos apresentam comportamentos razoavelmente complexo, exigindo que mais tempo seja destinado à compreensão, especificação e posterior avaliação de seu funcionamento. O nível de comportamento de uma classe-objeto diz respeito aos seus estados e transições que, conforme visto, podem ser representados utilizando um diagrama de transição de estados (*DTE*). Valores de atributos e relacionamentos mantidos por um objeto constituem seu estado. As transições representam mudanças nos valores dos atributos, implicando assim em mudanças de estados. Estas transições são produzidas como um reflexo a estímulos - *eventos* - ocorridos dentro ou fora do próprio objeto. Os objetos estimulam uns aos outros (i.e. trocam mensagens), resultando numa série de alterações de seus estados.

O estado inicial deve ser conhecido, e é aquele em que o objeto se encontra quando é criado. Para muitos objetos o número de possíveis estados seria infinito. Por isso, não é preciso representar todos os estados no diagrama, mas apenas os estados genéricos que determinam algum comportamento específico. Através de um *DTE,* é possível especificar de forma clara e precisa o comportamento de uma classe-objetos, permitindo que, posteriormente, o seu funcionamento seja validado. Além disso, uma representação gráfica torna mais fácil a localização de possíveis falhas ou indefinições no projeto, além de sugerir alternativas para a sua simplificação.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Para cada classe-objetos considerada com comportamento complexo, é gerado um diagrama de transição de estado ?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** Através de um diagrama de transição de estado é possível avaliar o funcionamento de uma classe-objeto.

Objetivo: Confiabilidade Conceitual **Fator:** Fidedignidade **Sub-Fator:** Correção Comportamental **Critério:** ***Correção do Comportamento dos Clusters (CCS)***	**Sub- Produto:** Modelo Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se na especificação está definido de forma correta o comportamento de cada *cluster.*

Discussão:

Sendo um *cluster* um conjunto de classes-objetos que trabalham juntas para atingir um objetivo específico, seu comportamento será determinado essencialmente pela colaboração das classes-objetos constituintes que, por sua vez, depende do conjunto de comportamentos individuais exibidos por estas mesmas classes-objetos. Sendo assim, o comportamento de um determinado *cluster* é considerado correto se o conjunto de comportamentos individuais das classes-objetos fornecem o comportamento desejado em relação à funcionalidade definida para o *cluster.*

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
1. O conjunto de comportamento individuais das classes-objetos fornecem o comportamento desejado em relação à funcionalidade definida para o *cluster?*	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade:*** Através dos comportamentos a nível de *cluster* é possível validar a funcionalidade fornecida pelo sistema.

IV.2.2. Fator: Suficiência

Conjunto de atributos de qualidade que avaliam se estão presentes na especificação todos os aspectos necessários e somente estes.

É atingido através dos seguintes subfatores de qualidade:

- ***Necessidade:*** Conjunto de atributos de qualidade que avaliam se todos os aspectos considerados na especificação são imprescindíveis.
- ***Não Redundância:*** Conjunto de atributos de qualidade que avaliam a presença, na especificação, de aspectos repetitivos.
- ***Completitude:*** Conjunto de atributos de qualidade que avaliam se todos os aspectos que devem ser especificados estão presentes na especificação.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Necessidade **Critério:** ***Necessidade dos Requisitos (NRQ)***	**Sub-Produto:** Documentação **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se na especificação estão descritos os requisitos considerados imprescindíveis.

Discussão:

Uma especificação deve conter apenas os aspectos considerados imprescindíveis, evitando assim descrições desnecessárias e irrelevantes.

Para verificar o grau de necessidade dos requisitos especificados, é conveniente classificá-los em três grupos:

- *requisitos imprescindíveis:* são aqueles requisitos considerados como obrigatórios. Caso não constem na especificação, implicaria na não aceitação do software.
- *requisitos desejáveis:* são aqueles requisitos que ajudam a obter um software mais abrangente. Caso não constem na especificação não implicará na não aceitação do software.
- *requisitos opcionais:* são aqueles requisitos que poderão ou não ser considerados.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Os aspectos especificados são considerados imprescindíveis?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Concisão:*** A existência de aspectos irrelevantes na especificação aumenta desnecessariamente o volume de documentação da especificação.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Necessidade **Critério:** ***Necessidade das Classes-Objetos(NEC)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se na especificação estão definidas as classes-objetos consideradas imprescindíveis.

Discussão:

A primeira atividade na construção do modelo de classes-objetos é identificar as classes-objetos que tenham relevância no domínio da aplicação. Uma especificação com classes-objetos irrelevantes, além de aumentar o tamanho do modelo, pode desviar a atenção do problema que está sendo modelado. Para ajudar na avaliação do grau de necessidade das classes-objetos no modelo, é conveniente considerar a seguinte classificação:

- *classes-objetos imprescindíveis*, são as classes-objetos consideradas relevantes para um implementação adequada do problema. Caso não existam no modelo, implicará num modelo incompleto que compromete a implementação e, conseqüentemente, a aceitação do software.
- *classes-objetos opcionais*, são as classes-objetos que poderão ou não ser consideradas, pois elas definem responsabilidades relativas a compromissos não assumidos inicialmente, e que poderiam estar associados a aspectos de reutilização ou evolução futura do software.
- *classes-objetos desnecessárias,* são as classes-objetos que não justificam sua existência no contexto da aplicação.

Processo de Avaliação:	**Escala**
Este critério pode ser avaliado através da resposta à seguinte pergunta: • A classe-objetos é considerada imprescindível no contexto do domínio do problema ?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Avaliabilidade, Manutenibilidade:*** Um modelo com classes-objetos irrelevantes aumenta desnecessáriamente o esforço dos processos de avaliação e manutenção da especificação.
- ***Concisão:*** A existência de classes-objetos irrelevantes aumenta desnecessariamente o volume de documentação da especificação.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Necessidade **Critério:** ***Necessidade dos Atributos (NEA)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se em cada classe-objetos estão definidos os atributos considerados imprescindíveis.

Discussão:

Cada classe-objetos deve conter, apenas, os atributos considerados imprescindíveis, evitando assim o tratamento de atributos desnecessários e irrelevantes. Um atributo na classe-objetos é necessário, quando participa nos serviços que a classe-objetos precisa para cumprir com suas responsabilidades, ou quando representa um valor - *disponível através de um serviço implícito* - que pode ser acessado por outras classes-objetos, através de mensagens. Isto quer dizer que atributos não referenciados, e/ou que não participam nos serviços especificados na classe-objetos, não são necessários e, portanto, devem ser eliminados. Estes atributos podem representar propriedades válidas da classe-objetos, mas no contexto da aplicação não são necessários. Para avaliar o grau de necessidade dos atributos na classe-objetos, devemos considerar a seguinte classificação:

- *atributos imprescindíveis*, são os atributos considerados relevantes para que a classe-objetos possa cumprir com suas responsabilidades. Caso não constem nela, a classe-objetos não terá capacidade de responder a todas as responsabilidades.
- *atributos opcionais*, são os atributos que ajudam a obter uma classe-objetos mais abrangente. Caso não constem nela, não implicará em uma classe-objetos insuficiente.
- *atributos desnecessários*, são os atributos que não justificam sua existência na classe-objetos, pois não são utilizados no contexto da aplicação.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Os atributos definidos na classe-objetos são considerados imprescindíveis no contexto do domínio da aplicação ?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados

- ***Concisão:*** Uma classe-objetos com atributos irrelevantes aumenta desnecessariamente o tamanho da classe-objetos e possivelmente a torna mais complexa. Além disso, aumenta desnecessariamente o volumen da documentação.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Necessidade **Critério:** *Necessidade dos Serviços (NES)*	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se em cada classe-objetos estão definidos os serviços considerados imprescindíveis.

Discussão:

Um serviço é necessário, quando contribui à realização das responsabilidades que a classe-objetos deve cumprir no contexto da aplicação. Estes serviços devem ser identificados tanto para as classes como para os objetos, de maneira diferenciada. Uma forma de garantir a identificação dos serviços necessários é listar as responsabilidades da classe-objetos, definindo o conjunto de serviços que são necessários para cumprí-las e assegurar que cada serviço realize uma única função, quanto possível. Da mesma forma que as classes-objetos e atributos, o grau de necessidade dos serviços, também, pode ser avaliado considerando três categorias:

- *serviços imprescindíveis*, são os serviços considerados relevantes, que garantem a implementação correta do produto. Caso não constem nela, podem comprometer a funcioanlidade do produto.
- *serviços opcionais*, são os serviços que poderão ou não ser considerados. Caso não constem nela, não implicarão em uma classe-objetos insuficiente.
- *serviços desnecessários*, são os serviços que não justificam sua existência na classe-objetos, pois não são utilizados no contexto da aplicação.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Os serviços definidos na classe-objetos são considerados imprescindíveis no contexto do domínio da aplicação ?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados:

- ***Concisão:*** Uma classe-objetos com serviços irrelevantes aumenta desnecessariamente o tamanho da classe-objetos e possivelmente a torna mais complexa. Além disso, aumenta desnecessariamente o volume da documentação.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Necessidade **Critério:** ***Necessidade dos Relacionamentos (NER)***	**Sub- Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se na especificação estão definidos os relacionamentos considerados imprescindíveis.

Discussão:

A falta de serviços que percorram uma associação pode ser um indicativo que o relacionamento é desnecessário, pois nenhum serviço utiliza o caminho, o que talvez seja uma indicação de que as informações que se deseja obter não são necessárias.

Para avaliar o grau de necessidade dos relacionamentos entre classes-objetos, devemos considerar:

- *relacionamentos imprescindíveis*, são os relacionamentos considerados relevantes. Caso não estejam representados, as classes-objetos não poderão cumprir suas responsabilidades e conseqüentemente fica comprometida a funcionalidade do software produto.
- *relacionamentos opcionais*, são os relacionamentos que podem ou não ser considerados. Caso não constem nela, não comprometerão a funcionalidade das classes-objetos envolvidas.
- *relacionamentos desnecessários*, são os relacionamentos que não são percorridos por nenhum serviço.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Os relacionamentos definidos entre pares de classes-objetos são considerados imprescindíveis no contexto do domínio da aplicação?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados

- ***Avaliabilidade, Manutenibilidade:*** A existência de relacionamentos desnecessários torna o modelo de classes-objetos mais complexos e, conseqüentemente, mais difíceis de serem avaliados e alterados.
- ***Reutilizabilidade:*** A remoção de relacionamentos desnecessários deve ser um dos objetivos do desenvolvedor, pois aumenta a capacidade da classe-objetos de ser reutilizada.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Não Redundância **Critério:** ***Não Redundância de Informações (NRI)***	**Sub- Produto:** Documentação **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se um mesmo aspecto é descrito em mais de um lugar da especificação.

Discussão:

O atributo não redundância refere-se ao fato de que um mesmo requisito não deve aparecer em mais do que um lugar na especificação. A presença de redundâncias pode tornar uma especificação mais legível, mas pode também ocasionar problemas nos processos de manutenção da especificação.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• A especificação não apresenta aspectos redundantes?	Sim → 1 Não → 0

Sugestões para Correção:

- Caso sejam necessárias redundâncias, devem ser incluídas referências cruzadas explícitas, cujo objetivo é permitir que as modificações sejam feitas de forma confiável.

Atributos de Qualidade Relacionados:

- ***Manutenibilidade:*** Especificações não redundantes evitam que um mesmo aspecto tenha que evoluir ou ser modificado em mais de um lugar, na especificação.
- ***Consistência:*** Especificações livres de aspectos repetitivos tornam-se mais consistentes.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Não Redundância **Critério:** ***Não Redundância de Classes-Objetos (NRC)***	**Sub-Produto:** Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia a duplicidade de classes-objetos na especificação.

Discussão:

Ao serem inicialmente definidos os *clusters,* é difícil conhecer com precisão as fronteiras entre eles. É perfeitamente normal que durante o desenvolvimento, dois ou mais grupos, trabalhando em *clusters* adjacentes, gerem modelos que apresentam classes-objetos em comum que são modeladas de formas diferentes. Nesses casos, ao serem identificadas classes-objetos que expressam a mesma informação, a classes-objetos deve ser nomeada de acordo com os *clusters* aos quais está relacionada, devendo ser representada em um único *cluster.*

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• *Clusters* adjacentes não apresentam redundância de classes-objetos?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados

- ***Avaliabilidade, Manutenibilidade:*** Redundância de classes-objetos interferem nos processos de testes e manutenção da especificação, podendo introduzir inconsistências.
- ***Concisão:*** Redundância de classes-objetos devem ser evitadas, pois aumenta desnecessariamente o tamanho do modelo.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Completitude **Critério:** ***Completitude com Relação ao Roteiro Definido pela Organização Desenvolvedora (CRO)***	**Sub-Produto:** Documentação, Modelo de Classes-Objetos **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se o roteiro definido pela organização desenvolvedora foi totalmente coberto pela especificação.

Discussão:

A especificação deve apresentar aos usuários e desenvolvedores uma descrição completa e objetiva das condições a que deve obedecer o software que vai ser construído. Para que isto aconteça, cada organização desenvolvedora deve definir um roteiro para elaboração da especificação. As especificações produzidas deverão obedecer a este roteiro.

Existe a possibilidade de que uma determinada seção do roteiro não seja aplicável ao problema que está sendo especificado. Nesse caso, deve ser incluída, contendo uma breve explicação do motivo pela qual não é aplicável.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• O roteiro definido pela organização desenvolvedora foi totalmente coberto pela especificação?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados

- ***Avaliabilidade:*** Especificações incompletas com relação ao roteiro definido pela organização desenvolvedora são de grande risco. Isto porque o problema que está sendo especificado não poderá ser avaliado de forma completa. Além disso, o produto gerado pode ser implementado de forma inadequada.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Completitude **Critério:** ***Completitude com Relação ao Método de Desenvolvimento Utilizado (CMM)***	**Sub- Produto:** Documentação **Técnica de Avaliação:** Inspeção Individual

Definição do Critério:

Característica que avalia se foram utilizados todos os recursos previstos no método de desenvolvimento.

Discussão:

Especificações são produzidas utilizando métodos e linguagens de especificação. Portanto, ao avaliar uma especificação deve-se considerar que ela esteja completa com relação à documentação prevista pelo método de desenvolvimento utilizado.

Processo de Avaliação:

Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
• Todos os recursos que auxiliam na produção da documentação, que o método que está sendo aplicado fornece, são utilizados?	Sim → 1 Não → 0

Atributos de Qualidade Relacionados

- ***Avaliabilidade:*** Especificações incompletas, com relação à documentação prevista pelo método, são de grande risco. Isto porque o problema que está sendo especificado não poderá ser avaliado completamente. Além disso, o produto gerado pode ser implementado de forma inadequada.

Objetivo: Confiabilidade Conceitual **Fator:** Suficiência **Sub-Fator:** Completitude **Critério:** ***Completitude com Relação aos Requisitos (COR)***	**Sub-Produto:** Documentação, Modelo de Classes-Objetos **Técnica de Avaliação:** Reunião de Inspeção

Definição do Critério:

Característica que avalia se na especificação estão definidos todos os requisitos estabelecidos para o produto.

Discussão:

Especificar envolve o entendimento do problema e seu registro. Este processo é fundamental para a qualidade do produto a ser gerado. Portanto, a principal função de uma especificação é registrar todos os aspectos a partir dos quais o software deve ser projetado e construído. Posteriormente, deve-se poder avaliar se estes aspectos foram completamente desenvolvidos. De forma geral, uma especificação é completa se contém todos os requisitos funcionais, requisitos de interfaces, requisitos de qualidade, requisitos de desempenho e requisitos de informação necessários para especificar o problema.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:

	Escala
1. Todas as funções a serem desempenhadas pelo software estão definidas?	Sim → 1 Não→ 0
2. Todas as necessidades de informação estão identificadas?	Sim → 1 Não→ 0
3. Todas as interfaces com o usuário estão identificadas?	Sim → 1 Não→ 0
4. Todas as interfaces com o ambiente de hardware estão identificadas?	Sim → 1 Não→ 0
5. Todas as interfaces com outros software estão identificadas?	Sim → 1 Não→ 0
6. Todas as possíveis respostas que o sistema deve fornecer para todas as possíveis classes de entrada constatadas, válidas ou inválidas, estão identificadas?	Sim → 1 Não→ 0
7. Todas as características relativas ao desempenho do software estão identificadas?	Sim → 1 Não→ 0
8. Todas as características de qualidade exigidas pelo usuário estão identificadas?	Sim → 1 Não→ 0
9. Todas as características de qualidade inerentes ao software são identificadas?	Sim → 1 Não→ 0
10. Todas as categorias possíveis de tratamento de erros e exceções, por falhas de hardware e software estão identificadas?	Sim → 1 Não→ 0

Atributos de Qualidade Relacionados:

- ***Viabilidade de Implementação:*** Só é possível implementar adequadamente o produto que se especifica, se o conteúdo da especificação atende as necessidades dos usuários, pois tem definido todo o que supostamente deve fazer.
- ***Reutilizabilidade:*** Para que uma componente possa ser reutilizada, devem ser completamente entendidos seus requisitos funcionais e não funcionais.

IV.3. Fator: Implementabilidade

Conjunto de atributos de qualidade que avaliam se uma especificação de requisitos pode ser implementada, tendo em conta aspectos econômicos, financeiros, tecnológicos, de mão de obra, de cronograma e sociais.

É atingido através dos seguintes subfatores de qualidade:

- ***Viabilidade Econômica:*** Conjunto de atributos de qualidade que avaliam se a compatibilidade entre o custo estimado para o desenvolvimento e operação do software e os benefícios esperados com sua utilização.
- ***Viabilidade Financeira:*** Conjunto de atributos de qualidade que avaliam a existência e à disponibilidade de capital necessário para conduzir o desenvolvimento do produto especificado.
- ***Viabilidade Tecnológica:*** Conjunto de atributos de qualidade que avaliam a existência e à disponibilidade da tecnologia necessária para conduzir o desenvolvimento do produto especificado.
- ***Viabilidade de Mão de Obra:*** Conjunto de atributos de qualidade que avaliam a existência e à disponibilidade da mão de obra necessária para conduzir o desenvolvimento do produto especificado.
- ***Viabilidade de Cronograma:*** Conjunto de atributos de qualidade que avaliam se o software pode ser construído dentro do limite de tempo estabelecido.
- ***Viabilidade Social:*** Conjunto de atributos de qualidade que avaliam as implicações que o software que vai ser construído terá sobre o grupo social ao qual este deverá servir, assim como sobre qualquer outro grupo social externo.

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Econômica
Critério: ***Aceitabilidade de Custos (ACC)***

Definição do Critério:

Característica que avalia se as estimativas de custos, para o desenvolvimento e/ou produção do software, são aceitas por usuários e desenvolvedores.

Discussão:

A aceitabilidade de custos para o desenvolvimento do software está ligada ao problema da estimativa de custos. Para o caso específico do paradigma de orientação a objetos, muito pouco tem sido feito em termos de métodos para estimar custos de desenvolvimento. De modo geral, as estimativas de custos de desenvolvimento de software envolve dois problemas abrangentes:

- o alto nível de incerteza e risco na estimativa, e,
- a inexistência de medidas precisas dos custos de projetos anteriores.

Cabe salientar, a inexistência de dados históricos de custos de desenvolvimento de software. Sem padrões de referência é muito difícil estimar, de forma confiável, o custo de um novo projeto.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. O custo estimado para o desenvolvimento do software é aceitável?	Sim → 1 Não→ 0
2. O custo estimado para operação do software é aceitável?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Econômica
Critério: ***Relevância de Benefícios (REB)***

Definição do Critério:

Característica que avalia se as estimativas de benefícios tangíveis e intangíveis são aceitas como relevantes, por usuários e desenvolvedores.

Discussão:

A dificuldade para quantificação dos benefícios é, também, um obstáculo para a realização da análise de custos/benefícios necessária para avaliar a implementabilidade do software especificado. Ao se estimarem os benefícios das alternativas propostas, estes devem ser comparados, sempre que possível, com os do sistema existente.

Processo de Avaliação: Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. As estimativas de benefícios tangíveis são aceitáveis?	Sim → 1 Não→ 0
2. As estimativas de benefícios intangíveis (atribuindo valores) são aceitáveis?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Econômica
Critério: ***Compatibilidade Custo/Benefíco (CCB)***

Definição do Critério:

Característica que avalia se os custos estimados para o desenvolvimento e operação são compatíveis com os benefícios esperados com sua utilização.

Discussão:

Através da análise de custo/benefício podemos verificar se o custo estimado para o desenvolvimento e operação do software é compatível com os benefícios esperados.

A análise de custo/benefício vai, portanto, comparar as medidas referentes aos custos e benefícios do software que vai ser construído. O resultado desde análise permitirá a decisão sobre a viabilidade do desenvolvimento do produto sob o aspecto econômico.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
1. A relação custo/benefício é aceitável?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Financeira
Critério: ***Existência de Capital (ECT)***

Definição do Critério:

Característica que avalia se a organização possui capital suficiente para custear o desenvolvimento.

Discussão:

A especificação não é uma declaração definitiva do que o usuário quer e pode dispor. Ela pode registrar o que o usuário idealmente quer, mas os custos envolvidos são muitas vezes razão para negociação. Por este motivo, faz-se necessária uma avaliação sob o ponto de vista financeiro de forma a se poder analisar as estimativas de custos e os fluxos de caixa necessários durante o desenvolvimento do software. Precisa-se saber, então, se a empresa possui os recursos necessários para o desenvolvimento. A viabilidade financeira implica, em parte, na verificação da existência de capital suficiente para custear o desenvolvimento e operação.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
1. A empresa possui capital suficiente para custear o desenvolvimento?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Financeira
Critério: ***Disponibilidade de Capital (DCT)***

Definição do Critério:

Característica que avalia se a organização é capaz de tornar disponível o capital necessário para o desenvolvimento.

Discussão:

De modo geral, as empresas tem uma capacidade limitada de investimentos no desenvolvimento de software. É comúm, também, que as empresa tenham, simultaneamente, diversos sistemas em desenvolvimento e manutenção. Isso faz com que tenham que compatilhar, entre os sistemas, recursos financeiros existentes. Em tais circunstâncias a empresa se vê obrigada a distribuir o capital de acordo com suas prioridades, criando, muitas vezes um problema de disponibilidade de capital.

Processo de Avaliação: Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
1. Existem recursos financeiros disponíveis?	Sim → 1 Não→ 0
2. Caso não estejam disponíveis os recursos, existe a possibilidade de serem obtidos?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Tecnológica
Critério: ***Existência da Tecnologia (ETE)***

Definição do Critério:
Característica que avalia se existe o nível de tecnologia necessário para conduzir o desenvolvimento.

Discussão:
É necessário avaliar este aspecto com bastante cuidado, pois a tecnologia necessária poderá não existir ou poderá existir e não ser do conhecimento da equipe desenvolvedora. É de grande importância, neste momento, a consulta à literatura técnica e às empresas fabricantes e distribuidores de software e hardware.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
1. Existe a tecnologia necessária para desenvolver o software especificado?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Tecnológica
Critério: ***Disponibilidade da Tecnologia (DTE)***

Definição do Critério:

Característica que avalia se a equipe encarregada do desenvolvimento tem disponível a tecnologia necessária para conduzir o desenvolvimento.

Discussão:

Embora a tecnologia que se precisa para a construção do software exista, ela pode não estar disponível no lugar em que está situada a empresa. Ainda mais, pode ser inviável adquirí-la em outro lugar. Pode também ser muito costoso adquirir o hardware e o software que se necessita. Essa, entretanto, é uma questão econômica que deve ser considerada ao se avaliar a viabilidade econômica.

Processo de Avaliação: Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. A equipe encarregada do desenvolvimento dispõe da tecnologia que precisa em termos de hardware e software?	Sim → 1 Não→ 0
2. Caso a tecnologia de hardware não esteja disponível, ela pode ser adquirida?	Sim → 1 Não→ 0
3. A equipe encarregada do desenvolvimento dispõe da tecnologia que precisa em termos de software?	Sim → 1 Não→ 0
4. Caso a tecnologia de software não esteja disponível, ela pode ser adquirida?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade de Mão de Obra
Critério: ***Existência de Mão de Obra (EMO)***

Definição do Critério:

Característica que avalia se existe na instalação a mão de obra necessária para o desenvolvimento.

Discussão:

Para poder construir o software, é necessário que exista na empresa pessoal com capacitação técnica, tanto a nível de hardware quanto de software, e que possua, conhecimentos sobre a área de aplicação. Esta capacitação técnica depende não somente da experiência, mas, também da formação e experiência no uso das metodologias e ferramentas a serem utilizadas para desenvolver o software.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
1. Existe na instalação a mão de obra necessária para construir o software?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade de Mão de Obra
Critério: ***Disponibilidade de Mão de Obra (DMO)***

Definição do Critério:

Característica que avalia se estão disponíveis os recursos humanos com o conhecimento e experiência necessários para realizar o desenvolvimento e operação do software.

Discussão:

A falta de experiência e conhecimento, inclui qualquer aspecto do desenvolvimento em que os membros da equipe não possuem experiência direta. Estes aspectos podem ser: *uso de um novo método de desenvolviemento, uso de um novo hardware, uso de um novo sistema operacional, uso de novas linguagens de programação.* Caso na instalação não exista o recurso humano com conhecimento e experiência suficiente para desenvolver e operar o software, deve existir a possibilidade de torná-lo disponível através de treinamento do pessoal próprio, determinando o prazo para sua capacitação.

Poderá ser necessária a contratação de pessoal em virtude da aquisição de nova tecnologia (hardware e software), devido a que a empresa considerou inviável o treinamento de pessoal próprio.

Processo de Avaliação: Este critério pode ser avaliado através das respostas à seguinte lista de verificação:	**Escala**
1. A organização dispõe da mão de obra para o desenvolvimento e operação do software em termos de conhecimento e domínio da tecnologia?	Sim → 1 Não→ 0
2. Caso no disponha da mão de obra, para o desenvolvimento e operação é possível adquirí-la por meio de treinamento?	Sim → 1 Não→ 0
3. Caso não disponha da mão de obra, para o desenvolvimento e operação, é possível adquirí-la por meio de contratação de pessoal?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade de Cronograma
Critério: ***Adequabilidade de Cronograma (ACR)***

Definição do Critério:

Característica que avalia se o software pode ser construído no tempo previsto pelo cronograma, considerando possíveis ocorrências de imprevistos e sem descuidar da qualidade definida para o produto.

Discussão:

É impossível desenvolver um software de qualidade elevada, sem especificações adequadas. Chegar o mais rápido possível ao código não é uma indicação satisfatória de que o produto está sendo bem desenvolvido. Nesses casos, o resultado será um produto inadequado que provavelmente terá um alto custo de manutenção. Portanto, é necessário ter-se uma noção clara dos prazos necessários para o desenvolvimento. Por isso, a viabilidade de implementação deve, portanto, também ser avaliada, verificando se o software poderá ser construído dentro dos prazos estipulados pelo cronograma, considerando possíveis ocorrências de imprevistos e sem descuidar a qualidade definida para o produto.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
1. O software é possível de ser construído no tempo previsto pelo cronograma?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade de Cronograma
Critério: ***Flexibilidade de Cronograma (FCR)***

Definição do Critério:

Característica que avalia se o cronograma aceito para o desenvolvimento pode atender, na medida do possível, fatores tais como introdução de atividades não projetadas, contingências etc.

Discussão:

Ao se estabelecer um cronograma devemos considerar as estimativas de recursos requeridos para cada atividade, juntamente com a disponibilidade de recursos existentes, permitindo um relacionamento lógico entre as atividades, estabelecendo datas de início e fim para cada atividade.

Embora seja possível estabelecer um cronograma considerando todas as condições, sempre ocorrem imprevistos. Portanto, o cronograma deve ser flexível para atender:

- a possível introdução de atividades não projetadas;
- as contingências, e,
- a gerência.

Processo de Avaliação:	
Este critério pode ser avaliado através da resposta à seguinte pergunta:	**Escala**
1. Existe flexibilidade de cronograma?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Social
Critério: ***Aceitabilidade da Engenharia Humana (AEH)***

Definição do Critério:

Característica que avalia se o software que vai ser construído leva em consideração o grau de satisfação e o desenvolvimento do potencial humano previsto para os usuários.

Discussão:

A viabilidade desde o ponto de vista da engenharia humana dirige-se aos seres humanos que interagem com o software, procurando simplificar essa interação, tornando-a também agradável. Está relacionada aos fatores psicológicos e físicos que determinam conforto, tolerância e amenidade ao uso contribuindo para a utilizabilidade do software.

Processo de Avaliação: Este critério pode ser avaliado através das respostas às seguintes perguntas:	**Escala**
1. Poderá o software especificado proveer uma forma satisfatória para que o usuário possa desempenhar a sua função operacional?	Sim → 1 Não→ 0
2. Poderá o software especificado satisfazer as necessidades humanas em seus diversos níveis?	Sim → 1 Não→ 0
3. Poderá o software especificado ajudar ao desenvolvimento das capacidades humanas?	Sim → 1 Não→ 0

Objetivo: Utilizablidade
Fator: Implementabilidade
Sub-Fator: Viabilidade Social
Critério: ***Aceitabilidade dos Impactos Sociais (AIS)***

Definição do Critério:

Característica que avalia se o software que vai ser construído leva em consideração seus impactos sobre o sistema social ao qual deverá servir.

Discussão:

As repercusões de um software sobre o ambiente podem significar impactos sociais de diferentes tipos: desemprego, insastisfação no trabalho, segurança etc. Estes impactos devem ser avaliados antes de se decidir pela construção do produto. Sua gravidade pode, inclusive, decidir a inviabilidade de operação do produto.

Processo de Avaliação: Este critério pode ser avaliado através das respostas às seguintes perguntas:	**Escala**
1. O software especificado leva em consideração seus impactos sobre o sistema social ao qual deverá servir?	Sim → 1 Não→ 0

Formulário de Avaliação de Especificações Orientadas a Objetos

Critérios de Avaliação de Especificações Orientadas a Objetos

I. Instruções

Assinale de acordo com a escala ordinal de 0 - 1 (Tabela A1.1) o valor que lhe parece melhor representar o grau em que cada critério foi atingido.

ESCALA	EQUIVALÊNCIA	INTERPRETAÇÃO
1	*Total Presença*	Indica que não há dúvida que o critério está totalmente presente.
0,75	*Alta Presença*	Indica um alto grau de presença do critério, mas não total.
0,50	*Moderada Presença*	Indica um grau de presença aceitável do critério.
0,25	*Baixa Presença*	Indica um baixo grau de presença do critério, sendo necessário o uso de medidas corretivas.
0	*Total Ausência*	Indica de maneira absoluta que o critério está ausente.

Tabela A1.1 - Graus de Presença do Critério - Escala Ordinal

Nome do Avaliador: _____

Especificação de Requisitos ☐ **Especificação de Projeto** ☐

Documentação

1. Uniformidade de Termos (UTT)	Técnica de Avaliação: Inspeção Individual
Definição: Característica que avalia se os termos técnicos foram utilizados de forma uniforme ao longo do texto e de acordo com as definições pré-estabelecidas.	
Processo de Avaliação: • Os termos técnicos são utilizados de forma uniforme ao longo da especificação e obedecem a definições pré-estabelecidas?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

2. Uniformidade de Notação (UNT)	Técnica de Avaliação: Inspeção Individual
Definição: Característica que avalia se a notação foi utilizada de forma uniforme ao longo do texto.	
Processo de Avaliação: • A notação foi utilizada de forma uniforme ao longo da especificação e obedece a definições pré-estabelecidas?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

3. Uniformidade de Detalhes da Documentação (UDD)	Técnica de Avaliação: Reunião de Inspeção
Definição: Característica que avalia se todos os aspectos estão descritos na especificação com o mesmo nível de detalhamento, considerando-se um determinado estágio de desenvolvimento.	
Processo de Avaliação: • Os aspectos descritos na especificação apresentam o mesmo nível de detalhamento, considerando-se o estágio de desenvolvimento?	**Valores** 0 0,25 0,50 0,75 1 ☐ ☐ ☐ ☐ ☐
Comentários:	**Valor Obtido:**

4. Independência de Detalhes de Projeto (UDP)	Técnica de Avaliação: Reunião de Inspeção
Definição: Característica que avalia na especificação de requisitos a existência de restrições com relação à escolha de alternativas de solução próprias da fase de projeto.	
Processo de Avaliação: • A especificação de requisitos não apresenta descritas, restrições com relação à escolha de alternativas próprias da fase de projeto e/ou implementação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

5. Coesão de Informações (COI)	Técnica de Avaliação: Reunião de Inspeção

Definição: Característica que avalia o grau de associação das informações de um capítulo e, dentro deste, seções.

Processo de Avaliação:	Valores
• A informação que apresenta o módulo - *capítulo ou seção* - descreve aspectos relacionados ao capítulo ou seção que está sendo descrito?	0 0,25 0,50 0,75 1 ☐ ☐ ☐ ☐ ☐
Comentários:	**Valor Obtido:**

6. Acoplamento entre Seções (ACS)	Técnica de Avaliação: Inspeção Individual

Definição: Característica que avalia o grau de interdependência entre os módulos - *capítulos ou seções* - da especificação.

Processo de Avaliação:	Valores
• O conteúdo de um módulo - *capítulo ou seção* - faz referência ao conteúdo de outro módulo, apenas como complemento, sem modificá-lo?	1 0 ☐ ☐
Comentários:	**Valor Obtido:**

7. Estrutura da Documentação (EDO)	Técnica de Avaliação: Inspeção Individual

Definição: Característica que avalia se a estrutura, composta de capítulos e, dentre seções, está organizada numa seqüência lógica.

Processo de Avaliação:	Valores
• A especificação tem seus capítulos e seções organizadas numa seqüência lógica?	1 0 ☐ ☐
Comentários:	**Valor Obtido:**

8. Complementabilidade (COM)	Técnica de Avaliação: Inspeção Individual

Definição: Característica que avalia se a especificação faz uso de documentos auxiliares - *referências, glossários, dicionários de dados* - que facilitam seu entendimento.

Processo de Avaliação:	Valores
1. Existe um glossário com definições de termos técnicos, simbologia e notação utilizados na especificação e que não são de uso comum?	1 0 ☐ ☐
2. Existe um dicionário de dados centralizando as informações?	1 0 ☐ ☐
3. Existe referência a documentos prévios e/ou complementares necessários ao entendimento da especificação?	1 0 ☐ ☐
Comentários:	**Valor Obtido: (Média)[1]**

[1]Média = $\frac{1}{n}\sum_{i=1}^{n} Xi$). Somamos as respostas com valor 1 ($X_1, X_2, \ldots, X_n$) e dividimos o total por n perguntas.

9. Aderência a Normas da Organização Desenvolvedora (EDO)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se a especificação foi gerada considerando as normas estabelecidas pela organização desenvolvedora.	
Processo de Avaliação: • A especificação foi gerada considerando as normas estabelecidas pela organização desenvolvedora?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

10. Aderência a Normas estabelecidas pelo Contratante (ANC)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se a especificação foi gerada considerando as normas estabelecidas pelo contratante.	
Processo de Avaliação: • A especificação foi gerada considerando as normas estabelecidas pelo contratante?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

11. Acessibilidade (ACE)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se qualquer usuário autorizado pode facilmente consultar a especificação e/ou obter uma cópia da mesma.	
Processo de Avaliação:	**Valores**
1. A especificação é possível de ser acessada por todos os seus usuários autorizados?	1 ☐ 0 ☐
2. A especificação pode ser facilmente reproduzida?	1 ☐ 0 ☐
3. Existe uma cópia de segurança da especificação fora do local de trabalho?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

12. Estar Atualizada (ETA)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se a especificação reflete a informação mais recente.	
Processo de Avaliação:	**Valores**
1. O conteúdo da especificação corresponde à informação mais recente, fruto da última manutenção?	1 ☐ 0 ☐
2. As atualizações realizadas estão todas datadas?	1 ☐ 0 ☐
3. As atualizações realizadas estão claramente indicadas?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

13. Localizabilidade Interna (LOI)	Técnica de Avaliação: Inspeção Individual
Definição: Característica que avalia se existem facilidades para se localizar todos os elementos, dentro de uma especificação, relacionados com determinado aspecto ou assunto.	
Processo de Avaliação:	**Valores**
1. A especificação apresenta um sumário?	1 ☐ 0 ☐
2. Existem índices remissivos?	1 ☐ 0 ☐
3. Existem referência cruzadas explícitas?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

14. Localizabilidade Externa (LOE)	Técnica de Avaliação: Inspeção Individual
Definição: Característica que avalia se existem facilidades para se localizar todas as especificações e demais documentos relacionados a um determinado aspecto assunto.	
Processo de Avaliação:	**Valores**
1. Existem sumários que organizam os documentos?	1 ☐ 0 ☐
2. Existem índices remissivos?	1 ☐ 0 ☐
3. Existem referência cruzadas explícitas?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

15. Consistência Interna (COI)	Técnica de Avaliação: Reunião de Inspeção
Definição: Característica que avalia a existência de conflitos entre aspectos especificados na mesma especificação.	
Processo de Avaliação:	**Valores**
1. A especificação não apresenta termos diferentes que possuem o mesmo significado e que são utilizados para descrever um mesmo objeto, que é tratado em contextos e lugares diferentes?	1 ☐ 0 ☐
2. A especificação não apresenta contradições entre características específicas do produto?	1 ☐ 0 ☐
3. A especificação não apresenta conflitos lógicos ou temporais entre requisitos do produto que dependem do tempo?	1 ☐ 0 ☐
4. A especificação não apresenta conflitos na descrição de aspectos de comportamento do produto?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

16. Consistência Externa (COE)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia a existência de conflitos entre aspectos especificados em outras especificações ou entidades externas.	
Processo de Avaliação: • A especificação não apresenta aspectos conflitantes com relação a outras especificações ou entidades externas?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

17. Ser Explícita (EXP)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação existem condições, hipótesis e/ou restrições definidas por contexto.	
Processo de Avaliação: • A especificação não apresenta aspectos definidos por contexto?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

18. Precisão (PRC)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se os aspectos especificados estão descritos de forma precisa e, sempre que possível, quantificada.	
Processo de Avaliação: 1. Os termos de significado múltiplo, apresentam uma definição precisa?	**Valores** 1 ☐ 0 ☐
2. Os requisitos do produto estão descritos de forma possível de serem validados?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

19. Necessidade dos Requisitos (NQR)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação estão descritos os requisitos considerados imprescindíveis.	
Processo de Avaliação: • Os aspectos descritos na especificação são considerados imprescindíveis?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

20. Não Redundância de Informações (NRI)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se um mesmo aspecto é descrito em mais de um lugar da especificação.	
Processo de Avaliação: • A especificação não apresenta aspectos redundantes?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

21. Completitude com Relação ao Roteiro Definido pela Organização Desenvolvedora (CRO)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se o roteiro definido pela organização desenvolvedora foi totalmente coberto pela especificação.	
Processo de Avaliação: • O roteiro definido pela organização desenvolvedora foi totalmente coberto pela especificação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

22. Completitude com Relação ao Método de Desenvolvimento (CMM)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se foram utilizados todos os recursos previstos no método de desenvolvimento.	
Processo de Avaliação: • Todos os recursos que auxiliam na produção da documentação, que o método que está sendo aplicado fornece, são utilizados?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

23. Completitude com Relação aos Requisitos (CMM)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação estão definidos todos os requisitos do produto.	
Processo de Avaliação:	**Valores**
1. Todas as funções a serem desempenhadas pelo software estão definidas e modeladas?	1 0 ☐ ☐
2. Todas as necessidades de informação estão identificadas?	1 0 ☐ ☐
3. Todas as interfaces com o usuário estão identificadas?	1 0 ☐ ☐

4. Todas as interfaces com o ambiente de hardware estão identificadas?	1 ☐	0 ☐
5. Todas as interfaces com outros software estão identificadas?	1 ☐	0 ☐
6. Todas as possíveis respostas que o sistema deve fornecer para todas as possíveis classes de entrada constatadas, válidas ou inválidas, estão identificadas?	1 ☐	0 ☐
7. Todas as características relativas ao desempenho do software estão identificadas?	1 ☐	0 ☐
8. Todas as características de qualidade exigidas pelo usuário estão identificadas?	1 ☐	0 ☐
9. Todas as características de qualidade inerentes ao software são identificadas?	1 ☐	0 ☐
10. Todas as categorias possíveis de tratamento de erros e exceções, por falhas de hardware e software estão identificadas?	1 ☐	0 ☐
Comentários:	**Valor Obtido: (Média)**	

Modelo de Classes-objetos

1. Correção da Notação (CNT)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se a notação do método foi usada de forma correta.	
Processo de Avaliação: • A especificação está escrita utilizando de forma correta a notação pré-definida no método de especificação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

2. Correção Sintática (CST)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se o conjunto de regras sintáticas, pré-definidas no método de especificação, foi usado de forma correta.	
Processo de Avaliação: • A especificação está escrita utilizando de forma correta o conjunto de regras sintáticas pré-definidas no método de especificação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

3. Correção Semântica (CSM)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se o conjunto de regras semânticas pré-definidas no método foi usado de forma correta.	
Processo de Avaliação: • A especificação está escrita utilizando de forma correta o conjunto de regras semânticas pré-definidas no método de especificação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

4. Correção no Uso do Formato de Documentação (CFD)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia se o formato de documentação definido no método foi usado de forma correta.	
Processo de Avaliação: • A especificação está escrita utilizando de forma correta o formato de documentação definido no método?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

5. Independência de Detalhes de Projeto (UDP)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia na especificação de requisitos a existência de restrições com relação à escolha de alternativas de solução próprias da fase de projeto.	
Processo de Avaliação: • A especificação de requisitos não apresenta restrições com relação à escolha de alternativas próprias da fase de projeto e/ou implementação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

6. Uniformidade de Abstração das Classes-Objetos (UAC)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia o nível de abstração de cada classe-objetos, considerando-se um determinado estágio do desenvolvimento.	
Processo de Avaliação: • As classes-objetos apresentam o nível de abstração que lhe corresponde segundo o estágio de desenvolvimento.	**Valores** 0 0,25 0,50 0,75 1 ☐ ☐ ☐ ☐ ☐
Comentários:	**Valor Obtido:**

7. Uniformidade de Abstração dos Atributos (UAA)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia o nível de abstração dos atributos em cada classe-objetos considerando-se um determinado estágio do desenvolvimento.	
Processo de Avaliação: • Os atributos da classe-objetos apresentam o nível de detalhamento adequado que lhe corresponde, segundo o estágio de desenvolvimento.	**Valores** 0 0,25 0,50 0,75 1 ☐ ☐ 1
Comentários:	**Valor Obtido:**

8. Uniformidade de Abstração dos Serviços (UAS)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia o nível de abstração dos serviços em cada classe-objetos, considerando-se um determinado estágio de desenvolvimento.	
Processo de Avaliação: • Os serviços da classe-objetos apresentam o nível de detalhamento adequado que lhe corresponde, ao estágio de desenvolvimento.	**Valores** 0 0,25 0,50 0,75 1 ☐ ☐ ■ ☐ ☐
Comentários:	**Valor Obtido:**

9. **Uniformidade de Abstração dos Clusters** (UCL)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia o nível de abstração das componentes modeladas no *cluster*, considerando-se um determinado estágio do desenvolvimento.	
Processo de Avaliação: O *cluster* apresenta todas suas abstrações com o nível de detalhamento adequado que corresponde ao estágio de desenvolvimento.	**Valores** 0 0,25 0,50 0,75 1 ☐ ☐ ☐ ☐ ☐
Comentários:	**Valor Obtido:**

11. **Nível de Fatoração das Classes-Objetos** (NFC)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a quantidade de descendentes imediatos de uma classe-objetos.	
Processo de Avaliação: *NFC(Ci)* = Número de classes-objetos imediatas	**Valores** 1 ☐ → $1 \leq NFC \leq V_{nfc}$ 0 ☐ → $NFC > V_{nfc}$
Comentários:	**Valor Obtido:**

12. **Nível de Profundidade da Hierarquia de Classes-Objetos** (NPH)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a quantidade média de níveis em uma hierarquia de classes-objetos.	
Processo de Avaliação: *NPH* = $\sum$ Nível de cada Classe / Número de Classes-Objetos	**Valores** 1 → $0 \leq NPH \leq V_{nph}$ 0 ☐ → $NPH > V_{nph}$
Comentários:	**Valor Obtido:**

13. **Coesão Estrutural da Classe-Objetos** (CEC)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia o grau de relacionamento dos serviços especificados na classe-objetos.	
Processo de Avaliação: M = número de conjuntos disjuntos N = número de argumentos *CEC(Ci) = (1 - M / N)*	**Valores** $CEC \geq V_{cec}$ $CEC < V_{cec}$
Comentários:	**Valor Obtido:**

14. Coesão Estrutural dos Clusters (CCL)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia o grau de relacionamento entre as classes-objetos de um determinado *cluster.*	
Processo de Avaliação: CL_i = identificação do *cluster* r = número de relacionamentos internos ao *cluster* n = número de classes-objetos no *cluster* $CCL(CLi) = \frac{(r+1)}{n}$	**Valores** 1 □ → $CCL \geq V_{ccl}$ 0 □ → $CCL < V_{ccl}$
Comentários:	**Valor Obtido:**

15. Acoplamento de Relacionamento das Classes-Objetos (ACR)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia o grau de dependência estática das classes-objetos excluindo os relacionamentos por herança.	
Processo de Avaliação: C_i = identificação da classe-objetos r = número de relacionamentos com outras classes-objetos $ACR(C_i) = \sum_{i=1}^{n} r_i$	**Valores** 1 □ → $0 < ACR \leq V_{acr}$ 0 □ → $ACR > V_{acr}$
Comentários:	**Valor Obtido:**

16. Acoplamento de Relacionamento dos Clusters (ARCL)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia o grau de dependência estática dos *clusters.*	
Processo de Avaliação: CL_i = identificação do *cluster* r = número de relacionamentos de classes-objetos com outros *clusters* $ARCL(CL_i) = \sum_{i=1}^{n} r_i$	**Valores** 1 □ → $0 < ACRL \leq V_{acrl}$ 0 □ → $ACRL > V_{acrl}$
Comentários:	**Valor Obtido:**

17. Acoplamento de Interação das Classes-Objetos (ACI)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia o grau de comunicação via mensagens das classes-objetos.	
Processo de Avaliação: C_i = identificação da classe-objetos S_i = identificação do serviço R_i = conjunto de serviços chamados por S_i $ACI(C\,i) = Si \cup \{Ri\,j\}$	**Valores** 1 ☐ → $0 < ACI \leq V_{aci}$ 0 ☐ → $ACI > V_{aci}$
Comentários:	**Valor Obtido:**

18. Acoplamento de Interação dos Clusters (AICL)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia o grau de comunicação via mensagens dos *clusters*.	
Processo de Avaliação: CL_i = identificação do *cluster* C_i = identificação da classe-objetos $AICL\,(CL_i) = \frac{\sum_{i=1}^{n} ACI(C_i)}{n}$	**Valores** 1 ☐ → $0 < AICL \leq V_{aicl}$ 0 ☐ → $AICL > V_{aicl}$
Comentários:	**Valor Obtido:**

19. Tamanho das Classes-objetos (TCO)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a quantidade de atributos e serviços definidos na classe-objetos.	
Processo de Avaliação: NA = número de atributos NS = número de serviços C_i = identificação da classe-objetos $TC(C_i) = NA + NS$	**Valores** 1 ☐ → $0 \leq TC \leq V_{tc}$ 0 ☐ → $TC > V_{tc}$
Comentários:	**Valor Obtido:**

20. Tamanho do Serviço (TS)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a quantidade de linhas de pseudocódigo definidas no serviço.	
Processo de Avaliação: S_i = serviço definido na classe-objetos C_i $TS\,(S_i)$ = número de linhas de pseudocódigo do serviço S_i	**Valores** 1 ☐ → $0 \leq TS \leq V_{ts}$ 0 ☐ → $TS > V_{ts}$
Comentários:	**Valor Obtido:**

21. Tamanho do Cluster (TCL)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a quantidade de classes-objetos definidas num *cluster*.	
Processo de Avaliação: CL_i = identificação do *cluster* *TCL* (CL_i) = *número de classes-objetos do cluster* CL_i	**Valores** 1 → $0 \leq TCL \leq V_{tcl}$ 0 → $TCL > V_{tcl}$
Comentários:	**Valor Obtido:**

22. Simplicidade dos Serviços (SIS)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia o grau de simplicidade dos serviços definidos na classe-objetos.	
Processo de Avaliação: C_i = identificação da classe-objetos s_i = valor associado ao grau de simplicidade do serviço n = número de serviços na classe-objetos 1 → Serviços sem instruções IF's, Case, Ciclos 0,75 → Serviços com instruções Ciclos 0,50 → Serviços com instruções IF's 0,25 → Serviços com instruções IF's, Ciclos 0 → Serviços com instruções IF's, Case, Ciclos $SIS(C_i) = \sum_{i=1}^{n} s_i / n$	**Valores** 1 → $1 \geq SIS \leq V_{sis}$ 0 → $SIS < V_{sis}$
Comentários:	**Valor Obtido:**

23. Simplicidade dos Atributos (SIA)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia o grau de simplicidade dos atributos definidos na classe-objetos.	
Processo de Avaliação: C_i = identificação da classe-objetos a_i = valor associado ao grau de simplicidade do atributo n = número de atributos na classe-objetos 1 → Inteiro, Real, Booleano 0,75 → Caracter 0,50 → Registro/Estrutura 0,25 → Referências 0 → Lista/Vetores $SIA(C_i) = \sum_{i=1}^{n} a_i / n$	**Valores** 1 → $1 \geq SIA \leq V_{sia}$ 0 → $SIA < V_{sia}$
Comentários:	**Valor Obtido:**

24. Tamanho das Interfaces (TI)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a quantidade de argumentos definidas nas mensagens.	
Processo de Avaliação: M_i = identificação da mensagem *TI (M_i) = número de argumentos da mensagem M_i*	**Valores** 1 ☐ → $0 \leq TI \leq V_{ti}$ 0 ☐ → $TI > V_{ti}$
Comentários:	**Valor Obtido:**

25. Acessibilidade (ACE)	**Técnica de Avaliação:** Inspeção Individual	
Definição: Característica que avalia se qualquer usuário autorizado pode facilmente consultar a especificação e/ou obter uma cópia da mesma.		
Processo de Avaliação:	**Valores**	
1. Os modelos de classes-objetos são possíveis de serem acessados por todos os seus usuários autorizados?	1 ☐	0 ☐
2. Existe uma cópia de segurança dos modelos de classes-objetos fora do local de trabalho?	1 ☐	0 ☐
Comentários:	**Valor Obtido: (Média)**	

26. Estar Atualizada (ETA)	**Técnica de Avaliação:** Inspeção Individual	
Definição: Característica que avalia se a especificação reflete a informação mais recente.		
Processo de Avaliação:	**Valores**	
1. O conteúdo dos modelos de classes-objetos correspondem à informação mais recente, fruto da última manutenção?	1 ☐	0 ☐
2. As atualizações realizadas estão todas datadas?	1 ☐	0 ☐
3. As atualizações realizadas estão claramente indicadas?	1 ☐	0 ☐
Comentários:	**Valor Obtido: (Média)**	

27. Consistência Externa (COE)	**Técnica de Avaliação:** Reunião de Inspeção	
Definição: Característica que avalia a existência de conflitos entre aspectos especificados em outras especificações ou entidades externas.		
Processo de Avaliação:	**Valores**	
O modelo não apresenta aspectos conflitantes com relação a outras especificações ou entidades externas?	1 ☐	0 ☐
Comentários:	**Valor Obtido:**	

28. Consistência Interna (COI)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia a existência de conflitos entre aspectos especificados na mesma especificação.	
Processo de Avaliação:	**Valores**
1. Os modelos de classes-objetos não apresentam contradições entre características específicas do produto?	1 ☐ 0 ☐
2. Os modelos de classes-objetos não apresentam conflitos lógicos ou temporais entre requisitos do produto que dependem do tempo?	1 ☐ 0 ☐
3. Não existem conflitos entre o modelo de classes-objetos e o modelo dinâmico?	1 ☐ 0 ☐
4. O modelo de classes-objetos não apresenta conflitos em relação a decisões de modelagem?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

29. Correção das Classes-objetos (CRC)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia a correta definição das classes-objetos.	
Processo de Avaliação:	**Valores**
1. A classe-objetos representa uma abstração significativa para o domínio do problema?	1 ☐ 0 ☐
2. O nome da classe-objetos obedece a padrões pré-estabelecidos e correspondem ao vocabulário utilizado pelos especialistas do domínio, caso sejam classes-objetos de domínio?	1 ☐ 0 ☐
3. O nome da classe-objetos é único na especificação?	1 ☐ 0 ☐
4. Cada objeto na classe é descrito a partir do conjunto de atributos e serviços?	1 ☐ 0 ☐
5. O conjunto de atributos e serviços identificados caracterizam adequadamente a classe-objetos no contexto do domínio do problema?	1 ☐ 0 ☐
6. Cada classe-objetos tem uma descrição precisa de seu significado no contexto do domínio do problema?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

30. Correção dos Atributos (COA)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a correta definição dos atributos.	
1. Processo de Avaliação:	**Valores**
2. Os nomes dos atributos obedecem a padrões pré-estabelecidos e correspondem ao vocabulário utilizado pelos especialistas do domínio, caso sejam atributos de domínio?	1 ☐ 0 ☐
3. Os nomes dos atributos são únicos na classe-objetos?	1 ☐ 0 ☐
4. Cada atributo captura um aspecto separado da abstração da classe-objetos e possui um valor para cada instância de objeto, no contexto do domínio do problema?	1 ☐ 0 ☐
5. Os atributos abrangem todas as informações pertinentes à classe-objetos que está sendo descrita?	1 ☐ 0 ☐
6. Cada atributo apresenta uma descrição precisa de seu significado, no contexto do domínio do problema?	1 ☐ 0 ☐
7. Cada atributo apresenta, quando pertinente: • seu tipo; • enumeração de todos seus valores possíveis; • referência do documento onde estão especificados e/ou listados estes valores; • identificação de valor(es) default; • regras que determinam quais valores são permitidos; • identificação do intervalo de valores possíveis; • visibilidade?	1 ☐ 0 ☐
Comentários:	**Valor Obtido:** **(Média)**

31. Correção dos Serviços (COS)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a correta definição dos serviços.	
Processo de Avaliação:	**Valores**
1. Os nomes dos serviços obedecem aos padrões pré-estabelecidos?	**1** ☐ 0 ☐
2. Os nomes dos serviços são únicos na classe-objetos?	1 ☐ 0 ☐
3. Cada serviço apresenta uma descrição precisa de seu signifcado, no contexto do domínio da aplicação?	1 ☐ 0 ☐
4. Cada serviço apresenta sua pseudocodificação, obedecendo as regras sintáticas e semânticas da linguagem de projeto definida para o projeto ?	1 ☐ 0 ☐
5. Serviços com nomes diferentes realizam funções diferentes?	1 ☐ 0 ☐
6. Os serviços especificados na classe são aplicáveis a cada um dos objetos da classe?	1 ☐ 0 ☐
Comentários:	**Valor Obtido:** **(Média)**

32. Correção dos Relacionamentos de Associação (COR)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a correta definição dos relacionamentos de associação especificados.	
Processo de Avaliação:	**Valores**
1. Os relacionamentos apresentam um descritor indicando claramente a semântica do relacionamento, o objeto cliente e o servidor?	1 ☐ 0 ☐
2. O sentido de dependência é consistente com o intercâmbio de mensagens?	1 ☐ 0 ☐
3. Para cada relacionamento entre pares de objetos, a quantidade (m) ou intervalo (m,n), a partir da perspectiva de cada objeto, está de acordo com às restrições de cardinalidade impostas, segundo os requisitos do problema?	1 ☐ 0 ☐
4. Para cada par de relacionamentos existe um mapeamento no domínio do problema?	1 ☐ 0 ☐
5. Se o relacionamento requerido não pode ser representado, existe uma forma textual de indicar esta situação?	1 ☐ 0 ☐
6. Para cada par de relacionamentos estão definidos os serviços de criação, destruição, associação, desassociação e tratamento de referências?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

33. Correção dos Relacionamentos de Composição (CRC)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a correta definição dos relacionamentos de composição especificados.	
Processo de Avaliação:	**Valores**
1. O relacionamento satisfaz a propriedade de transitividade?	1 ☐ 0 ☐
2. O relacionamento satisfaz a propriedade de anti-simetria?	1 ☐ 0 ☐
3. Para cada relacionamento de composição, a quantidade (m) ou intervalo (m,n), a partir da perspectiva de cada objeto, está de acordo com às restrições de cardinalidade impostas segundo os requisitos do domínio do problema?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

34. Correção da Hierarquia de Classes-Objetos (HC)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia a correta definição das estruturas de hierarquia de classes-objetos.	
Processo de Avaliação:	**Valores**
1. Cada generalização e especialização é nomeada de acordo com os padrões pré-estabelecidos?	1 ☐ 0 ☐
2. A classe de generalização é sempre uma classe abstrata e as classes de especialização são sempre classes concretas?	1 ☐ 0 ☐
3. Uma especialização adiciona pelo menos um atributo ou serviço (pode estar redefinido) com relação a sua classe de generalização?	1 ☐ 0 ☐
4. Existem pelo menos um atributo ou um serviço (pode estar redefinido) que distingue as especializações de uma mesma classe de generalização?	1 ☐ 0 ☐
5. Cada classe-objetos na hierarquia reflete um inter-relacionamento com as demais classes-objetos que compartilham a estrutura?	1 ☐ 0 ☐
6. A estrutura de hierarquia tem nomes de atributos e serviços únicos em relação aos ancestrais e descendentes (quando pertinente)?	1 ☐ 0 ☐
7. A estrutura de hierarquia tem nomes de atributos e serviços diferentes nas classes-objetos de generalização para cada parte de um entrelaçamento (herança múltipla)?	1 ☐ 0 ☐
8. São utilizados discriminadores para indicar o critério de divisão da estrutura de hierarquia?	1 ☐ 0 ☐
9. As características de uma classe de generalização são aplicáveis a todas as classes-objetos especializações?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

35. Coesão Semântica das Classes-Objetos (CSC)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia o grau de relacionamento semântico entre os conceitos definidos na classe-objetos.	
Processo de Avaliação:	**Valores**
• A classe-objetos está composta por atributos e serviços, que caracterizam adequadamente a abstração, visando o cumprimento de suas responsabilidades?	0 ☐ 0,25 ☐ 0,50 ☐ 0,75 ☐ 1 ☐
Comentários:	**Valor Obtido:**

36. Coesão Semântica dos Serviços (CSS)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia o grau de relacionamento semântico entre os elementos de informação definidos no serviço.	
Processo de Avaliação: • Cada serviço especificado na classe-objetos realiza uma única operação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

37. Coesão Semântica dos Clusters (CSCL)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia o grau de relacionamento semântico entre os conceitos especificados nas classes-objetos que compõem o *cluster.*	
Processo de Avaliação: • O *cluster* está composto por um conjunto de classes-objetos que realizam funções relacionadas de acordo com o objetivo definido?	**Valores** 0 0,25 0,50 0,75 1 ☐ ☐ ☐ ☐ ☐
Comentários:	**Valor Obtido:**

38. Acoplamento de Herança (ACH)	**Técnica de Avaliação:** Inspeção Individual
Definição: Característica que avalia o grau em que são utilizados os atributos e serviços herdados.	
Processo de Avaliação: • Numa hierarquia de classes-objetos, as subclasses utilizam os atributos e serviços herdados, os redefinem ou adicionam outros atributos ou serviços?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

39. Correção dos Cenários (CCE)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação está definido de forma correta a seqüência de interações entre classes-objetos	
Processo de Avaliação:	**Valores**
1. Cada cenário está associado a uma funcionalidade requerida pelo produto?	1 0 ☐ ☐
2. Cada cenário tem identificado seus eventos iniciais?	1 0 ☐ ☐
3. Cada cenário tem identificado seus dados iniciais?	1 0 ☐ ☐
4. Cada cenário tem identificada a classe-objetos que recebe o evento inicial (entrada) e a resposta (saída)?	1 0 ☐ ☐
5. Cada cenário tem identificado os caminhos de comunicação entre as classes-objetos envolvidas?	1 0 ☐ ☐
Comentários:	**Valor Obtido: (Média)**

40. Correção do Comportamento das Classes-objetos (CCC)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação está definido de forma correta o comportamento de cada classe-objetos.	
Processo de Avaliação: • Para cada classe-objetos considerada com comportamento complexo, é gerado um diagrama de transição de estado?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

41. Correção do Comportamento dos Clusters (CCL)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação está definido de forma correta o comportamento de cada *cluster*.	
Processo de Avaliação: • O conjunto de comportamentos individuais das classes-objetos fornecem o comportamento desejado em relação à funcionalidade definida para o *cluster?*	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

42. Necessidade das Classes-objetos (NEC)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação estão definidas as classes-objetos consideradas imprescindíveis.	
Processo de Avaliação: • A classe-objetos é considerada imprescindível no contexto do domínio da aplicação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

43. Necessidade dos Atributos (NEA)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se em cada classe-objetos estão definidos os atributos considerados imprescindíveis.	
Processo de Avaliação: • Os atributos definidos na classe-objetos **são** considerados imprescindíveis no contexto do domínio da aplicação?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

44. Necessidade dos Serviços (NES)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se em cada classe-objetos estão definidos os serviços considerados imprescindíveis.	
Processo de Avaliação: • Os serviços definidos na classe-objetos são considerados imprescindíveis no contexto do domínio da aplicação?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

45. Necessidade de Relacionamentos (NER)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação estão definidos os relacionamentos considerados imprescindíveis.	
Processo de Avaliação: • Os relacionamentos definidos entre pares de classes-objetos são considerados imprescindíveis no contexto do domínio da aplicação?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

46. Não Redundância de Classes-objetos (NRC)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia a duplicidade de classes-objetos na especificação.	
Processo de Avaliação: • *Cluster* adjacentes não apresentam redundância de classes-objetos?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

47. Completitude com Relação aos Requisitos (CMM)	**Técnica de Avaliação:** Reunião de Inspeção
Definição: Característica que avalia se na especificação estão definidos todos os requisitos do produto.	
Processo de Avaliação: a) Todas as funções a serem desempenhadas pelo software estão modeladas?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

Viabilidade de Implementação

1. Aceitabilidade de Custos (ACC)	Produto: Especificação de Requisitos
Definição: Característica que avalia se as estimativas de custos, para o desenvolvimento e/ou produção do software, são aceitas por usuários e desenvolvedores.	
Processo de Avaliação: 1. O custo estimado para o desenvolvimento do software é aceitável?	**Valores** 1 0 ☐ ☐
2. O custo estimado para operação do software é aceitável?	1 0 ☐ ☐
Comentários:	**Valor Obtido: (Média)**

2. Relevância de Benefícios (REB)	Produto: Especificação de Requisitos
Definição: Característica que avalia se as estimativas de benefícios tangíveis e intangíveis são aceitas como relevantes, por usuários e desenvolvedores.	
Processo de Avaliação: 1. As estimativas de benefícios tangíveis são aceitáveis?	**Valores** 1 0 ☐ ☐
2. As estimativas de benefícios intangíveis (atribuindo valores) são aceitáveis?	1 0 ☐ ☐
Comentários:	**Valor Obtido: (Média)**

3. Compatibilidade Custo/Benefício (CCB)	Produto: Especificação de Requisitos
Definição: Característica que avalia se os custos estimados para o desenvolvimento e operação do produto são compatíveis com os benefícios esperados com sua utilização.	
Processo de Avaliação: • A relação custo/benefício é aceitável?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

4. Existência de Capital (ECT)	Produto: Especificação de Requisitos
Definição: Característica que avalia se a organização possui capital suficiente para custear o desenvolvimento.	
Processo de Avaliação: • A empresa possui capital suficiente para custear o desenvolvimento?	**Valores** 1 0 ☐ ☐
Comentários:	**Valor Obtido:**

5. Disponibilidade de Capital (DCT)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se a organização é capaz de tornar disponível o capital necessário para o desenvolvimento.	
Processo de Avaliação: 1. Existem recursos financeiros disponíveis para o desenvolvimento?	**Valores** 1 ☐ 0 ☐
2. Caso não estejam disponíveis os recursos, existe a possibilidade de serem obtidos?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

6. Existência de Tecnologia (ETE)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se existe o nível de tecnologia necessário para conduzir o desenvolvimento.	
Processo de Avaliação: • Existe a tecnologia necessária para desenvolver o software especificado?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

7. Disponibilidade de Tecnologia (DTE)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se a equipe encarregada do desenvolvimento tem disponível a tecnologia necessária para conduzir o desenvolvimento.	
Processo de Avaliação: 1. A equipe encarregada pelo desenvolvimento dispõe da tecnologia que precisa em termos de hardware?	**Valores** 1 ☐ 0 ☐
2. Caso a tecnologia de hardware não esteja disponível, ela pode ser adquirida?	1 ☐ 0 ☐
3. A equipe encarregada pelo desenvolvimento dispõe da tecnologia que precisa em termos de software?	1 ☐ 0 ☐
4. Caso a tecnologia de software não esteja disponível, ela pode ser adquirida ou desenvolvida?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

8. Existência de Mão de Obra (EMO)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se existe na instalação a mão de obra necessária para o desenvolvimento.	
Processo de Avaliação: • Existe a mão de obra para construir o software?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

9. Disponibilidade de Mão de Obra (DMO)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se estão disponíveis os recursos humanos com o conhecimento e experiência necessários para realizar o desenvolvimento e operação do software.	
Processo de Avaliação:	**Valores**
1. A organização dispõe de mão de obra para o desenvolvimento e operação do software em termos de conhecimento e domínio da tecnologia?	1 ☐ 0 ☐
2. Caso não disponha de mão de obra, para o desenvolvimento e operação é possível adquirí-la por meio de treinamento?	1 ☐ 0 ☐
3. Caso não disponha de mão de obra, para o desenvolvimento e operação é possível adquirí-la por meio de contratação de pessoal?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

10. Adequabilidade de Cronograma (ACR)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se o software pode ser construído no tempo previsto pelo cronograma, considerando possíveis ocorrências de imprevistos e sem descuidar da qualidade definida para o produto.	
Processo de Avaliação:	**Valores**
• O software é possível de ser construído no tempo previsto pelo cronograma?	1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

11. Flexibilidade de Cronograma (FCR)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se o cronograma aceito para o desenvolvimento pode atender, na medida do possível, fatores tais como introdução de atividades não projetadas, contingências etc..	
Processo de Avaliação:	**Valores**
• Existe flexibilidade de cronograma?	1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

12. Aceitabilidade de Engenharia Humana (AEH)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se o software que vai ser construído leva em consideração o grau de satisfação e o desenvolvimento do potencial humano previsto para os usuários.	
Processo de Avaliação:	**Valores**
1. Poderá o software especificado proveer uma forma satisfatória para que o usuário possa desempenhar a sua função operacional?	1 ☐ 0 ☐
2. Poderá o software especificado satisfazer as necessidades humanas em seus diversos níveis?	1 ☐ 0 ☐
3. Poderá o software especificado ajudar ao desenvolvimento das capacidades humanas?	1 ☐ 0 ☐
Comentários:	**Valor Obtido: (Média)**

13. Aceitabilidade dos Impactos Sociais (AIS)	**Produto:** Especificação de Requisitos
Definição: Característica que avalia se o software que vai ser construído leva em consideração seus impactos sobre o sistema social ao qual deverá servir.	
Processo de Avaliação: • O software especificado leva em consideração seus impactos sobre o sistema social ao qual deverá servir?	**Valores** 1 ☐ 0 ☐
Comentários:	**Valor Obtido:**

Apêndice III

Instrumento para Hierarquizar Critérios de Qualidade para Especificações Orientadas a Objetos

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
COPPE - PROGRAMA DE ENGENHARIA DE SISTEMAS E COMPUTAÇÃO
Linha de Pesquisa - Engenharia de Software

Instrumento para Hierarquizar Critérios de Qualidade para Especificações Orientadas a Objetos

I. Objetivo

Determinar o grau de importância de um conjunto de atributos de qualidade para especificações desenvolvidas segundo o paradigma de orientação a objetos.

II. Produtos

Os seguintes produtos são considerados neste instrumento:

- Especificações em Geral
- Especificações de Requisitos Orientadas a Objetos - Modelo de Análise
- Especificações de Projeto Orientadas a Objetos - Modelo de Projeto

III. Instruções

Atribua valores de 0 a 4, segundo a escala apresentada na Tabela I, aos critérios de qualidade definidos para ***Especificações de Requisitos e Especificações de Projeto*** de acordo com o grau de importância na sua opinião e/ou experiência.

Grau de Importância	Explicação
0	Indica que a característica que está sendo apresentada não tem nenhuma importância.
1	Indica que a característica que está sendo apresentada tem pouca importância.
2	Indica que a característica que está sendo apresentada tem importância em algumas circunstâncias mas nem sempre.
3	Indica que a característica que está sendo apresentada é muito importante.
4	Indica de maneira absoluta que não há dúvida que a característica que está sendo apresentada é imprescindível.

Tabela I - Escala de Valores

IV. Perfil do Avaliador

Nome do Avaliador: _____

Área de Atuação: ☐ **Análise e Projeto** ☐ **Programação**

V. Hierarquização dos Critérios de Qualidade de Especificações em Geral

OBJETIVO: CONFIABILIDADE DA REPRESENTAÇÃO	
Critérios	**Importância**
1. Correção da Notação	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
2. Correção Sintática	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
3. Correção Semântica	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
4. Correção no Uso do Formato de Documentação	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
5. Uniformidade de Termos	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
6. Uniformidade de Notação	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
7. Uniformidade de Detalhes da Documentação	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
8. Independência de Restrições de Projeto	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
9. Coesão das Informações	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
10. Acoplamento entre Seções	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
11. Estrutura da Documentação	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
12. Complementabilidade	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
13. Aderência às Normas da Organização Desenvolvedora	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
14. Aderência às Normas Estabelecidas pelo Contratante	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
15. Acessibilidade	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
16. Estar Atualizada	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
17. Organização da Documentação	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
18. Localizabilidade Interna	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑
19. Localizabilidade Externa	0 1 2 3 4 ❑ ❑ ❑ ❑ ❑

OBJETIVO: CONFIABILIDADE CONCEITUAL	
Critérios	**Importância**
1. Consistência Interna	0 1 2 3 4
2. Consistência Externa	0 1 2 3 4
3. Ser Explícita	0 1 2 3 4
4. Precisão	0 1 2 3 4
5. Necessidade dos Requisitos	0 1 2 3 4
6. Não Redundância das Informações	0 1 2 3 4
7. Completitude com Relação ao Roteiro definido pela Organização Desenvolvedora	0 1 2 3 4
8. Completitude com Relação ao Método de Desenvolvimento	0 1 2 3 4
9. Completitude com Relação aos Requisitos	0 1 2 3 4

OBJETIVO: UTILIZABILIDADE	
Critérios	**Importância**
1. Aceitabilidade de Custos	0 1 2 3 4
2. Relevância dos Benefícios	0 1 2 3 4
3. Compatibilidade Custo/Benefício	0 1 2 3 4
4. Existência de Capital	0 1 2 3 4
5. Disponibilidade de Capital	0 1 2 3 4
6. Existência de Tecnologia	0 1 2 3 4
7. Disponibilidade de Tecnologia	0 1 2 3 4
8. Existência de Mão de Obra	0 1 2 3 4
9. Disponibilidade de Mão de Obra	0 1 2 3 4
10. Adequabilidade de Cronograma	0 1 2 3 4
11. Flexibilidade de Cronograma	0 1 2 3 4
12. Aceitabilidade dos Impactos Sociais	0 1 2 3 4
13. Aceitabilidade da Engenharia Humana	0 1 2 3 4

VI. Hierarquização dos Critérios de Qualidade de Especificações Orientadas a Objetos

OBJETIVO: CONFIABILIDADE DA REPRESENTAÇÃO **ER = Especificação de Requisitos** **EP = Especificação de Projeto**		
Critérios		**Importância**
1. Uniformidade de Abstração das Classes-objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
2. Uniformidade de Abstração dos Atributos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
3. Uniformidade de Abstração dos Serviços	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
4. Uniformidade de Abstração dos *Clusters*	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
5. Nível de Fatoração das Classes-Objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
6. Nível de Profundidade da Hierarquia de Classes-objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
7. Coesão Estrutural das Classes-Objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
8. Coesão Estrutural dos *Clusters*	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
9. Acoplamento de Relacionamento das Classes-Objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
10. Acoplamento de Relacionamento dos *Clusters*	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
11. Acoplamento de Interação das Classes-Objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐

OBJETIVO: CONFIABILIDADE DA REPRESENTAÇÃO (cont.)		
ER = Especificação de Requisitos **EP = Especificação de Projeto**		
Critérios		**Importância**
12. Acoplamento de Interação dos *Clusters*	ER	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
	EP	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
13. Tamanho das Classes-Objetos	ER	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
	EP	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
14. Tamanho dos Serviços	ER	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
	EP	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
15. Tamanho dos *Clusters*	ER	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
	EP	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
16. Tamanho das Interfaces	ER	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
	EP	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
17. Simplicidade dos Serviços	ER	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
	EP	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
18. Simplicidade dos Atributos	ER	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒
	EP	0 1 2 3 4 ❒ ❒ ❒ ❒ ❒

OBJETIVO: CONFIABILIDADE CONCEITUAL		
ER = Especificação de Requisitos **EP = Especificação de Projeto**		
Critérios		**Importância**
1. Correção das Classes-Objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
2. Correção dos Atributos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
3. Correção dos Serviços	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
4. Correção dos Relacionamentos de Associação	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
5. Correção dos Relacionamentos de Composição	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
6. Correção da Hierarquia de Classes-objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
7. Coesão Semântica das Classes-objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
8. Coesão Semântica dos *Serviços*	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
8. Coesão Semântica dos *Clusters*	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
9. Acoplamento de Herança	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐

OBJETIVO: CONFIABILIDADE CONCEITUAL (cont.)		
ER = Especificação de Requisitos **EP = Especificação de Projeto**		
Critérios		**Importância**
10. Correção dos Cenários	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
11. Correção do Comportamento das Classes-Objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
12. Correção do Comportamento dos *Clusters*	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
13. Necessidade das Classes-objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
14. Necessidade dos Atributos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
15. Necessidade dos Serviços	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
16. Necessidade dos Relacionamentos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
18. Não Redundância de Classes-objetos	ER	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐
	EP	0 1 2 3 4 ❐ ❐ ❐ ❐ ❐

VII. Observações

Apêndice IV

Formulários de Coleta de Dados

Formulário de Coleta de Dados

Sistema ABC

A

Subsistema A ☐

Subsistema B ☐

Págs __ de

Fase ☐ Especificação de Requisitos
☐ Especificação de Projeto

Assunto: _____
Tarefa: _____

Nome da Classe-Objetos	C	A	D	I	G	R	nfc	npc	nanc	na	nad	nar	ns	nsd	nsr	nme	nmr

Formulário de Coleta de Dados

B

Sistema ABC

Subsistema A ☐
Subsistema B ☐

Págs __ de ___

Fase ☐ Especificação de Requisitos
☐ Especificação de Projeto

Tarefa: _____
Nome da Classe-Objetos: _____

Nome do Serviço	D	S	nos	cxa	if's	case	ciclos	cxs	nac	naoc