GERENCIAMENTO DE PROJETOS BASEADO EM CENÁRIOS: UMA ABORDAGEM DE MODELAGEM DINÂMICA E SIMULAÇÃO

Márcio de Oliveira Barros

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

Aprovada por:

Prof. Cláudia Maria Lima Werner, D.Sc.

Prof. Guilherme Horta Travassos, D.Sc.

Prof. Ana Regina Cavalcanti da Rocha, D.Sc.

Prof. Arndt von Staa, Ph.D.

Prof. Paulo César Masiero, Ph.D.

Prof. Hermano Perrelli de Moura, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2001

BARROS, MÁRCIO DE OLIVEIRA

Gerenciamento de Projetos Baseado em Cenários: Uma Abordagem de Modelagem Dinâmica e Simulação [Rio de Janeiro] 2001

XII, 238 p., 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2001)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Gerenciamento de Projetos de Software

I. COPPE/UFRJ II. Título (série)

A meus pais.

AGRADECIMENTOS

Ao longo de mais de quatro anos de trabalho, fico feliz em ter uma grande lista de pessoas a quem agradecer.

Antes de mais nada, agradeço a Deus por ter possibilitado o início e a conclusão desta importante conquista em minha vida e pela inspiração nos momentos decisivos.

À Prof. Cláudia Werner e ao Prof. Guilherme Travassos, pela orientação impecável ao longo de todo o processo de confecção desta tese. Em especial à Prof. Cláudia Werner, pela confiança no meu trabalho, aceitando um tema de tese tão diferente de sua linha principal de trabalho. Em especial ao Prof. Guilherme Travassos, pelas críticas (melhor aceitas no final do que no início do processo) e por me mostrar que os estudos experimentais podem funcionar na Engenharia de Software.

À Prof. Ana Regina, pela contribuição ao meu aprendizado no decorrer dos cursos de Mestrado e Doutorado, pelas críticas construtivas recebidas desde o exame de qualificação e pela participação na banca examinadora da tese.

Ao Prof. Arndt von Staa, pela participação na banca do exame de qualificação que precedeu esta tese e na banca examinadora da tese.

Ao Prof. Paulo Masiero e ao Prof. Hermano Moura, pela participação na banca examinadora da tese.

Ao Prof. Júlio Salek Aude, figura que sempre inspirou meu desenvolvimento na área acadêmica e que deixou um grande vazio nesta Universidade.

A meus pais, pelo incentivo constante e pela compreensão no decorrer deste período.

À Viviane, que acompanhou os dois últimos anos deste processo, pelo seu carinho e compreensão.

À Prof. Shari Lawrence Pfleeger, por suas críticas, suas revisões e pelo apoio oferecido a esta tese.

Aos participantes do estudo experimental realizado como parte deste trabalho, por sua preciosa colaboração.

Ao pessoal do SIGA 2000, aos que ainda estão na COPPE e aos que já se formaram, em especial ao Alexandre Corrêa, Alexandre Dantas, Dênis Silveira, Gustavo Veronese, Hugo Vidal, José Ricardo, Leonardo Murta, Marcelo Costa, Marcos Mangan, Marcos Kalinovsky, Nelson Miller, Robson Pinheiro e Sômulo Mafra, pelo excelente ambiente de trabalho que me proporcionaram e pelos amigos que criei.

Ao pessoal do grupo de reutilização de software, em especial à Prof. Regina Braga, Alessandréia de Oliveira e Luciana.

Aos amigos da COPPE, pelo incentivo e motivação.

À Franklin Gonçalves, por me mostrar a importância da modelagem quantitativa e de "pensar nos fundamentos".

A Eduarda la Rocque, por me proporcionar conhecer o interessante domínio da análise de riscos, mesmo que não em software.

À Ana Paula, por sempre estar disposta a ajudar.

Ao pessoal da secretaria do Departamento de Engenharia de Sistemas e Computação, pelo auxílio sempre que necessário.

À CAPES, pelo apoio financeiro ao desenvolvimento deste trabalho.

Ao pessoal da RiskControl, em especial à César Aragão, Daniel Serman, Felipe Lourenço, Gustavo Machado, Luiz Eduardo, Marcelo de Paula, Rafael Cruz, Rosane Bensusan, pelas informações sobre a análise de riscos no mercado financeiro.

Aos meus amigos Cláudio Miguel, Mário João, Roberto Nunes, Rodolfo Lima, Luiz Otávio, Roberta Cardim, pelos ótimos momentos de amizade e pelos incentivos.

A todos que, ao longo deste período, estiveram torcendo por mim.

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

GERENCIAMENTO DE PROJETOS BASEADO EM CENÁRIOS: UMA ABORDAGEM DE MODELAGEM DINÂMICA E SIMULAÇÃO

Márcio de Oliveira Barros

Dezembro / 2001

Orientadores: Cláudia Maria Lima Werner
Guilherme Horta Travassos

Programa: Engenharia de Sistemas e Computação

O gerenciamento de projetos é uma atividade fortemente baseada em conhecimento. Os gerentes utilizam suas habilidades e sua experiência para tomar decisões durante a execução de um processo de desenvolvimento de software. Geralmente, os gerentes experientes obtém mais sucesso que os gerentes novatos em termos de atingir metas de cronograma, custos e funcionalidade. Estes melhores resultados são atingidos devido à experiência acumulada ao longo de situações ocorridas em projetos no passado e no conhecimento derivado destas experiências.

Nesta tese, abordamos a criação e reutilização de conhecimento relacionado com o gerenciamento de projetos de desenvolvimento de software. Apresentamos uma representação para este tipo de conhecimento, os *modelos de cenários*, que permitem que um gerente verifique o impacto de teorias, procedimentos, ações e estratégias gerenciais que podem ser aplicadas ou impostas sobre um projeto de software. Os modelos de cenário são descritos através de formulações matemáticas e avaliados através de simulações. Os modelos de cenário compõem um repositório de conhecimento reutilizável para os gerentes de projeto, que podem utilizar os cenários para determinar o impacto de suas incertezas sobre seus projetos. Apresentamos os resultados de um estudo experimental para análise da viabilidade das técnicas propostas e uma aplicação da abordagem proposta em um processo de gerenciamento de riscos.

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

SCENARIO BASED PROJECT MANAGEMENT: A DYNAMIC MODELING AND SIMULATION BASED APPROACH

Márcio de Oliveira Barros

December / 2001

Advisors: Cláudia Maria Lima Werner
Guilherme Horta Travassos

Department: Computer and Systems Engineering

Project management is a knowledge intensive activity. Managers use their skills and experience to make decisions during the execution of a software development process. Usually experienced managers perform better than novice managers in terms of attending to project schedule, budget, and functionality. Such better results are achieved due senior managers' accumulated experiences and the knowledge that they can derive from situations faced in past projects.

We address the creation and reuse of software project management knowledge. We present a knowledge representation, namely *scenario models*, which allows a manager to verify the impact of theories, procedures, actions, and strategies that can be applied or imposed upon a software project. Scenario models are represented by mathematical formulations and evaluated through simulations. Scenario models compose a reusable knowledge base useful for project managers, who can reuse scenarios to assess the impact of their assumptions upon their projects. We present an application of the proposed approach within a risk management process and the results from an experimental analysis of the techniques.

ÍNDICE:

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

"Are we then creatures of action? Do we say that we desire those accepted clichés of comfort when, in fact, it is the challenge and the adventure that truly give us life?"

– Drizzt Do'Urden (Legacy of the Drow)

1 Introdução

1.1 Motivação

O estado da prática e a literatura contemporânea comprovam a persistência de um conhecido problema no desenvolvimento de software: muitos projetos consomem mais recursos do que o planejado, demoram mais tempo para serem realizados, possuem menos funções e menor qualidade do que o esperado. Relatos de insucesso na produção de sistemas de software podem ser encontrados em diversos estudos de casos e experimentos documentados ao longo dos últimos anos (STANDISH GROUP, 1994) (CHARETTE, 1996) (LAFLEUR, 1996) (GROSS et al., 1999). Este panorama é particularmente perceptível em projetos complexos, de larga escala ou inovadores.

Duas linhas de estudo apresentam opiniões distintas sobre os fatores responsáveis pelos insucessos no desenvolvimento de software na indústria. A primeira linha de estudos associa estas falhas a problemas tecnológicos, agravados pela crescente complexidade dos produtos desenvolvidos (AUGUSTINE, 1982). A segunda linha de estudos transfere a responsabilidade para os problemas de gerenciamento, falta ou excesso de comunicação e dificuldades no tratamento das incertezas que se apresentam nos projetos de software (BROWN, 1996, ROYCE, 1998) (REEL, 1999).

As técnicas atualmente aplicadas no gerenciamento de projetos de software foram desenvolvidas nas décadas de 1950 e 1960, para possibilitar a realização de projetos avançados da marinha e da força aérea americanas. Estas técnicas incluem as estruturas de decomposição de projetos, os diagramas de PERT/CPM e os diagramas de alocação de recursos. Estas técnicas assumem algumas premissas que devem ser atendidas pelos projetos para que seu gerenciamento através das técnicas seja eficaz. Estas premissas exigem que os projetos possuam objetivos claros e bem delimitados, que exista pelo menos uma solução conhecida para os problemas abordados, que o cronograma e os recursos necessários para a realização do projeto possam ser estabelecidos antes de seu início, que o ambiente operacional do projeto seja conhecido e que métricas para quantificar o sucesso do projeto possam ser estabelecidas.

As premissas das técnicas de gerenciamento atuais não são facilmente encontradas em projetos de software. O desenvolvimento de projetos em domínios de aplicação inovadores, em domínios com alta volatilidade de requisitos, a necessidade de

integração de diversos domínios para atender aos requisitos do projeto, incertezas, ambigüidades, complexidade, quebra de continuidade, falta de economia de escala, a existência de relações não lineares e de ciclos complexos de realimentação (*feedback loops*) são características de projetos de software de larga escala.

Estas características invalidam os requisitos para a aplicação das técnicas tradicionais de gerenciamento de projetos, que podem funcionar na teoria, mas não são diretamente aplicáveis na prática. Entretanto, devido ao sucesso de alguns projetos que utilizaram estas técnicas, os gerentes tendem a aceitar suas premissas como presentes em todos os projetos (CHARETTE, 1996). Esta falha de interpretação é comum em projetos de software.

O gerenciamento de projetos de software é uma atividade fortemente baseada em conhecimento, onde os gerentes utilizam suas habilidades e sua experiência para tomar decisões enquanto a equipe executa o processo de desenvolvimento de software. Observa-se que os gerentes mais experientes geralmente obtém mais sucesso no controle de projetos de software do que os gerentes inexperientes. Devido às experiências acumuladas ao longo de diversos projetos realizados no passado e ao conhecimento adquirido através destas experiências, os gerentes mais experientes são capazes de perceber determinados aspectos do processo de desenvolvimento que permanecem invisíveis para outros gerentes. O planejamento e controle do projeto passam a considerar estes aspectos, que de outra forma seriam desprezados.

O "modelo de decisão baseado em reconhecimento de padrões"[1], proposto por KLEIN (1998), sugere que os gerentes mantêm uma coleção de padrões em suas mentes e comparam estes padrões ao contexto em que se encontram quando uma decisão se faz necessária. O conhecimento gerencial, contido no repositório mental de experiências dos gerentes experientes, é um recurso valioso. Sua importância para o sucesso de projetos de desenvolvimento de software cria a demanda por uma técnica para representação, armazenamento e reutilização destes padrões mentais, que permita que gerentes menos experientes compartilhem o conhecimento dos especialistas.

Os gerentes de projeto mais experientes geralmente realizam este processo de tomada de decisão mentalmente. Eles constróem um modelo mental do projeto sob análise, criam modelos mentais para os problemas e oportunidades que estão sendo investigados e buscam, dentre os padrões de seu repositório mental, ações adequadas

[1] No original, *recognition-primed decision model.*

para serem tomadas para resolver os problemas e para explorar as oportunidades. Entretanto, como um modelo mental não pode ser diretamente reutilizado por gerentes menos experientes, precisamos de uma representação explícita para o conhecimento tácito dos gerentes experientes, assim como uma técnica que permita a avaliação do impacto destes problemas e oportunidades sobre um projeto de desenvolvimento de software.

1.2 Objetivos desta Tese

O principal objetivo desta tese é a definição de uma representação para o conhecimento aplicável no gerenciamento de projetos de desenvolvimento de software. Pretende-se definir as bases de um paradigma de gerenciamento de projetos, onde o conhecimento dos gerentes mais experientes é descrito segundo a representação proposta e reutilizado por gerentes novatos, que o aplicam em seus projetos.

Para tanto, torna-se necessário que a representação a ser definida permita o auxílio automatizado ao processo de transferência de conhecimento entre gerentes experientes e gerentes novatos. Neste contexto, definimos a transferência de conhecimento como o processo que transforma a informação, que é o elemento representado sob algum formato computacional, em conhecimento de fato, que se reflete no entendimento do gerente dos efeitos decorrentes da aplicação desta informação sobre seu projeto. Pretende-se definir mecanismos que permitam que os gerentes novatos avaliem o impacto da aplicação do conhecimento gerencial reutilizado no contexto de seus projetos, antes de esta aplicação seja feita no projeto real.

A necessidade de automação do processo de transferência de conhecimento exige que a representação a ser proposta seja formal. Do contrário, esta representação poderia permitir ambigüidades, limitando assim os resultados que poderiam ser esperados dos mecanismos de avaliação do seu impacto sobre um projeto (contexto do gerente).

Dois objetivos complementares suportam o principal objetivo desta tese: a construção de ferramentas de suporte ao paradigma proposto e a avaliação de sua viabilidade através da realização de estudos experimentais. O primeiro requisito visa a construção de ferramentas que, em um primeiro estágio, teriam como objetivo permitir a utilização em laboratório da representação e das técnicas de transferência de conhecimento sendo propostas. O segundo requisito visa determinar, ainda que com limitado poder de conclusão, se o objetivo principal foi atingido em algum grau.

Um último objetivo desta tese se refere à aplicação das técnicas propostas em um processo de análise de riscos (HALL, 1998; ABNT, 2000). A análise de riscos foi selecionada devido à importância da existência de conhecimento (na forma de especialistas, experiências anteriores, literatura, entre outras) para sua efetiva aplicação. Assim, pretende-se definir um processo de análise de riscos baseado na descrição dos riscos que podem afetar um projeto e de suas estratégias de resolução segundo a representação de conhecimento gerencial proposta.

1.3 Soluções Propostas

Propomos o gerenciamento de projetos baseado em cenários, um paradigma para a gerência de projetos de desenvolvimento de software que sugere que um gerente especifique o comportamento esperado para um projeto de forma independente dos problemas e oportunidades que podem afetar este projeto. Entretanto, como estes eventos incertos podem ocorrer durante a execução do projeto, afetando o seu comportamento, o gerente deve testar a sensibilidade do projeto à ocorrência de combinações destes problemas e oportunidades.

No paradigma do gerenciamento de projetos baseado em cenários, os problemas e oportunidades que podem afetar o comportamento de um projeto são representados por cenários. Os cenários contêm conhecimento gerencial reutilizável por diversos projetos. Técnicas de modelagem formal e de simulação suportam o paradigma proposto.

O projeto e os cenários são representados por modelos dinâmicos de projetos de software. Estes modelos descrevem as leis e políticas que regem a evolução de um projeto através de postulações lógicas e algébricas, representadas em um conjunto de equações. Estas equações permitem a estimação dos resultados de um projeto, com seu custo e tempo de desenvolvimento, a partir de seus parâmetros, como o tempo necessário para a realização de cada atividade, as relações entre as atividades e a equipe disponível. Os modelos propostos descrevem formalmente as relações, lineares e não lineares, e os ciclos de *feedback* existentes entre estes elementos. Estes modelos também são capazes de representar as incertezas inerentes a estes elementos na forma de distribuições de probabilidade.

Entretanto, as técnicas que são aplicadas atualmente na modelagem dinâmica de projetos de software possuem algumas limitações que inibem seu uso prático na gerência de projetos. Entre estas limitações, podemos citar a dificuldade de construção dos modelos, a incapacidade de representar detalhes e limitações na representação de

conhecimento. Apresentamos um processo de modelagem que procura suprir estas limitações. Este processo de modelagem é utilizado na definição dos modelos de projeto e de cenários.

A sensibilidade de um projeto a um cenário é avaliada através da integração do modelo que representa o projeto com o modelo que representa o cenário. Técnicas de simulação permitem a análise do comportamento dos modelos. Primeiro, o modelo que representa o projeto é simulado isoladamente, apresentando o comportamento do projeto sem cenários. Em seguida, o modelo é integrado ao modelo que representa o cenário, sendo novamente simulado. A segunda simulação, quando comparada a primeira, apresenta os efeitos da ocorrência do cenário sobre o projeto.

Para avaliar a viabilidade e utilidade das técnicas de integração e simulação de modelos de projeto e cenário realizamos um estudo experimental. Os participantes do estudo foram solicitados a gerenciar um projeto de desenvolvimento de software, controlado por um emulador de projetos. Um grupo de participantes aplicou as técnicas de integração e simulação de cenários para apoiar suas decisões de projeto, enquanto os demais participantes se basearam apenas em sua experiência para a tomada de decisões.

Finalmente, apresentamos um processo de análise de riscos que utiliza modelos de cenários para descrever o impacto dos riscos, de seus planos de contenção e contingência sobre um projeto. Os riscos que podem afetar um projeto são documentados através de uma estrutura de dados padronizada, denominada *arquétipo de risco*. Esta estrutura organiza as informações disponíveis sobre cada risco, tais como mecanismos para identificação em um projeto, seu contexto e possíveis estratégias de contenção e contingência. O processo de análise de riscos é dividido em dois sub-processos: um para identificação dos riscos e outro para determinação de riscos que podem afetar uma aplicação. Os dois processos de análise de risco são organizados como um processo genérico de reutilização, onde o primeiro processo gera artefatos reutilizáveis – os arquétipos de risco – que são consumidos pelo segundo processo.

1.4 Justificativa

A justificativa para a realização deste trabalho pode ser rastreada de origens distintas. A principal delas talvez seja a constatação das dificuldades relacionadas com o gerenciamento de projetos de desenvolvimento de software e sua importância para o sucesso dos projetos. Esta importância foi verificada ao longo de diversos projetos em que tivemos a oportunidade de participar ou acompanhar na indústria de software.

Mesmo nos sistemas mais complexos, os blocos formadores do software resultante podiam ser descritos de forma relativamente simples. Por que então tantos problemas surgiam quando estes blocos tinham que ser desenvolvidos e integrados? Por que os desenvolvedores tinham que trabalhar 14 horas por dia para atender ao cronograma previamente planejado por eles próprios? Aparentemente, os principais problemas vinham da coordenação de um conjunto de desenvolvedores, da integração dos blocos construídos e, principalmente, do volume de incertezas que assaltavam os projetos ao longo de seu desenvolvimento. A necessidade de organização e a dificuldade em obter maior controle sobre as incertezas utilizando as técnicas tradicionais de gerenciamento de projetos foi assim constatada.

Pela importância dada ao tratamento das incertezas que se apresentam ao longo de um projeto de software, consideramos que a análise de riscos, ou seja, o processo que auxilia na identificação e resolução das incertezas que podem causar prejuízos a um projeto, é fundamental para o sucesso dos projetos de software. Entretanto, analisando a literatura sobre o tema, descobrimos que diversas abordagens adotam simplificações que dificultam sua aplicação prática em projetos reais. As limitações e vantagens de diversas técnicas de análise de riscos no desenvolvimento de software são analisadas nesta tese.

Passamos então a estudar como a análise de riscos é realizada em outras áreas do conhecimento. Uma área que chamou nossa atenção foi a análise de riscos no mercado financeiro. Empresas de médio e grande porte, que possuem dívidas em moedas estrangeiras, indexadas a taxas que flutuam de acordo com o mercado internacional ou que negociam seus produtos com empresas em outros países, compram instrumentos financeiros para se protegerem dos prejuízos potenciais advindos de variações nestas taxas de juros e de câmbio. Os instrumentos financeiros são oferecidos por bancos de investimento e seguradoras, interessados em assumir o risco da operação contratada pela empresa em troca de uma taxa de compensação (preço do seguro). Entretanto, como o risco é o produto negociado entre as duas partes, ambas devem estar cientes do que estão negociando. Assim, a avaliação de riscos no mercado financeiro evoluiu muito nas últimas décadas.

As técnicas de análise de riscos no mercado financeiro se baseiam na modelagem formal dos instrumentos envolvidos, na determinação dos fatores que podem impor riscos a estes instrumentos, na modelagem da incerteza destes fatores e na avaliação do efeito conjunto de diversos fatores sobre os instrumentos financeiros. Em síntese, as

técnicas de análise de riscos se baseiam em modelagem formal, análise estatística e simulação.

Existem algumas dificuldades na aplicação destas técnicas no desenvolvimento de software. O corpo de conhecimento disponível sobre os riscos que podem afetar um projeto de software é muito subjetivo e disperso, quando comparado aos riscos que podem afetar os instrumentos do mercado financeiro. Nos últimos, existem contratos não ambíguos descrevendo os instrumentos, o que permite sua modelagem matemática e sua avaliação em diferentes contextos de mercado (taxas de juros mais altas, taxas de câmbio mais baixas, entre outros). O conhecimento subjetivo e descrito de forma ambígua dificulta a modelagem das incertezas que se apresentam para um projeto de desenvolvimento de software. Entretanto, acreditamos que, mesmo com estas dificuldades, a modelagem formal é extremamente importante.

É difícil apresentar justificativas para a modelagem formal sem citar a célebre frase de Lorde Kelvin sobre a necessidade de formalização do conhecimento.

> *"Quando se pode medir um elemento sob análise e expressar este elemento em números, é possível demonstrar algum conhecimento sobre o elemento. Mas quando não se pode medir o elemento sob análise, nem expressar suas propriedades em termos numéricos, o conhecimento sobre ele é reduzido e insatisfatório: este pode ser o início de algum conhecimento, mas ainda está muito distante do estágio de ciência."*[2]

Se ainda não temos um nível de formalização de conhecimento requerido para a construção de modelos fiéis das incertezas que podem afetar um processo de desenvolvimento de software, talvez possamos atingir este patamar começando por modelos simples, que seriam refinados ao longo de diversos anos de pesquisa. Entretanto, para começar a trilhar este caminho, precisamos de uma representação formal para o conhecimento sobre a gerência de projetos de software.

O percurso deste caminho já havia começado quando realizamos nossos primeiros passos. Muitos autores já haviam desenvolvido modelos complexos para projetos de software, notoriamente ABDEL-HAMID e MADNICK (1991), com seu modelo baseado na Dinâmica de Sistemas. Entretanto, percebemos algumas dificuldades na interpretação e extensão destes modelos. Estas dificuldades e as soluções propostas para elas, assim como as limitações destas soluções, são apresentadas nesta tese.

---

[2] Tradução livre.

Nossa precedente formação em reutilização de software também influenciou fortemente este trabalho. Estamos tratando de modelos de conhecimento e na utilização destes modelos de conhecimento em contextos distintos dos quais eles foram originalmente projetados. O gerenciamento do conhecimento disponível em uma organização é fundamental, mas este conhecimento deve estar expresso de forma que possa ser transferido para seus interessados. Acreditamos que as técnicas de reutilização de software podem auxiliar neste processo de transferência, formando a base do processo com que os modelos de conhecimento são construídos e utilizados.

1.5 Organização deste Documento

Este documento está organizado em sete capítulos e sete apêndices. O primeiro capítulo contém esta introdução.

O Capítulo 2 apresenta uma visão sistêmica do gerenciamento de projetos de desenvolvimento de software, enfatizando a necessidade de modelos formais para a representação das relações complexas existentes entre os elementos que compõem um projeto. O capítulo contém uma revisão bibliográfica sobre algumas técnicas de modelagem de sistemas e suas aplicações na descrição de projetos de software, identificando alguns requisitos desejáveis para que uma técnica de modelagem seja aplicada neste domínio.

O Capítulo 3 apresenta uma revisão bibliográfica sobre o estado da arte no gerenciamento de riscos de projetos de desenvolvimento de software, identificando as principais características dos processos de gerenciamento de risco encontrados na literatura da Engenharia de Software e ressaltando alguns pontos em que estes processos podem ser aprimorados.

O Capítulo 4 descreve o paradigma do gerenciamento de projetos de software baseado em cenários, apresentando os modelos utilizados por ele e o processo de modelagem que procura atender aos requisitos apresentados no Capítulo 2.

O Capítulo 5 apresenta o planejamento e os resultados do estudo experimental de viabilidade da aplicação das técnicas propostas no Capítulo 4 no gerenciamento de projetos de desenvolvimento de software. O estudo foi realizado em meio acadêmico e contou com a participação de 18 desenvolvedores de software, que atuaram como gerentes de um projeto proposto pelo estudo.

O Capítulo 6 apresenta a aplicação das técnicas propostas no Capítulo 4 na definição de um processo de análise de riscos. Neste processo, os modelos formais do

gerenciamento de projetos baseado em cenários são utilizados para descrever o impacto de um risco sobre um projeto, assim como o impacto de planos de contenção e contingência para este risco.

O Capítulo 7 apresenta as considerações finais desta tese, ressaltando suas contribuições, limitações e perspectivas futuras.

O Apêndice A apresenta um resumo dos diagramas com que são representados os modelos da Dinâmica de Sistemas e a interpretação matemática destes modelos.

O Apêndice B apresenta um exemplo de modelo de projeto e diversos exemplos de modelos de cenários, ambos utilizando o processo de modelagem apresentado no Capítulo 4. Estes modelos foram utilizados no estudo experimental do Capítulo 5.

O Apêndice C apresenta uma descrição textual da dinâmica do modelo de projeto e dos modelos de cenário descritos no Apêndice B.

O Apêndice D apresenta a sintaxe BNF para a construção de modelos de domínio, para instanciação destes modelos e para a construção de cenários.

O Apêndice E apresenta as ferramentas *Illium*, *Hector* e *Manager Máster*, que foram implementadas como parte desta tese e utilizadas no estudo experimental apresentado no Capítulo 5.

O Apêndice F apresenta os dois questionários utilizados no estudo experimental apresentado no Capítulo 5.

O Apêndice G apresenta exemplos de arquétipos de risco, que seguem a estrutura padrão para documentação de riscos que possam afetar projetos de software apresentada no Capítulo 6.

2 Uma Visão Sistêmica do Gerenciamento de Projetos de Software

O gerenciamento de projetos é uma das áreas mais importantes e menos compreendidas da administração (STERMAN, 1992). Problemas relacionados a atrasos e custos acima do planejamento não podem ser considerados exceções em projetos nas mais diversas áreas do conhecimento humano. Tais problemas são comuns em projetos de desenvolvimento de software, conforme constatado por diversos autores (GILB, 1988, DAVIS, 1990, CHARETTE, 1996, STANDISH GROUP, 1994, GOMES, 2001).

Grande parte dos problemas enfrentados por gerentes de projetos de software não são intuitivos. É comum encontrar projetos que sofram da "síndrome dos 90%", onde um projeto é considerado 90% completo quando ainda necessita de 100% do tempo consumido para ser concluído (DEMARCO, 1982). Outros projetos podem sofrer os efeitos da "Lei de Brooks", que indica que quando um projeto está atrasado, adicionar mais desenvolvedores somente atrasará ainda mais o projeto (BROOKS, 1974). Outros exemplos de desvios do comportamento intuitivo também podem ser encontrados na literatura (HART, 1982, DEMARCO, 1982, ABDEL-HAMID e MADNICK, 1991).

Confiar somente na experiência dos gerentes e da equipe de desenvolvimento é uma política que tem se mostrado ineficiente, dado o número de projetos de software encerrados sem sucesso. Dada a complexidade do processo de desenvolvimento, são necessários mecanismos de apoio à decisão para guiar os gerentes no planejamento e controle de projetos de software. Neste capítulo, apresentamos uma visão sistêmica do gerenciamento de projetos de desenvolvimento de software, exploramos os mecanismos utilizados para representação de sistemas complexos e avaliamos sua aplicabilidade no apoio à decisão na gerência de projetos.

Este capítulo está organizado em seis seções. Na seção 2.1, apresentamos a modelagem como mecanismo para representação de sistemas, apresentando as diversas categorias de modelos de sistemas e suas características. Na seção 2.2, apresentamos uma visão sistêmica do processo de desenvolvimento de software, enfatizando a necessidade de modelos formais no apoio à decisão. Na seção 2.3, discutimos algumas aplicações para modelos formais relacionadas com projetos de desenvolvimento de software. Na seção 2.4, analisamos diversas técnicas de modelagem e sua aplicação na construção de modelos de desenvolvimento de software. Na seção 2.5, abordamos a representação de incerteza em modelos formais, distinguindo os tipos de incerteza que

podem ocorrer em um projeto de software e como estes tipos podem ser mapeados sobre os modelos formais. Finalmente, concluímos este capítulo na seção 2.6, apresentando os requisitos para que um modelo formal seja utilizado na gerência operacional de projetos de software.

2.1 Sistemas e Modelos

Um sistema representa uma parte da realidade. Um sistema é formado por um conjunto de componentes, que interagem entre si para realizar tarefas que não podem ser cumpridas pelos componentes isoladamente (MADACHY e BOEHM, 1999). Os relacionamentos entre os componentes de um sistema definem a estrutura do sistema (MARTIN, 1997a).

Um sistema pode ser classificado como aberto ou fechado. Sistemas abertos funcionam como uma seqüência acíclica de ações e reações: uma ação provocada por algum componente do sistema provoca a reação de outros componentes, que por sua vez provocam reações de outros componentes sucessivamente, sem retornar ao componente origem da ação. Em um sistema fechado, a seqüência de ações e reações formam ciclos. Assim, uma ação provocada por algum componente do sistema inicia uma cadeia de reações, que podem vir a alterar as condições que provocaram a ação original. Sistemas fechados são chamados de sistemas retro-alimentáveis, ou sistemas de *feedback*. *Feedback*, ou realimentação, é o processo através do qual uma mudança no sistema provocará uma seqüência de reações que, em última instância, afetará a mudança original (ROBERTS et al., 1983).

Um modelo é uma representação de um sistema. Um modelo captura o conhecimento sobre os componentes de um sistema e as relações entre estes componentes. Modelos podem ser classificados como mentais e explícitos. Os modelos mentais representam a percepção que um indivíduo forma a respeito das interações entre os componentes de um sistema e do comportamento provocado por estas interações (MARTIN, 1997a). Parte do conhecimento de um indivíduo é formada por um conjunto de representações próprias do mundo real, ou seja, por um conjunto de modelos mentais. As decisões tomadas por um indivíduo, ou por um grupo de indivíduos, são baseadas em suas visões particulares do mundo real, ou seja, em seus modelos mentais (STERMAN, 1988, FORRESTER, 1991). Os modelos mentais atuam como filtros, fornecendo o conhecimento sobre os componentes do sistema para que o indivíduo

selecione estratégias de ações que possam influenciar estes componentes, alterando o comportamento do sistema para que um objetivo seja atingido.

Modelos mentais são flexíveis, podendo ser adaptados para situações imprevistas ou expandidos com a chegada de novas informações acerca de uma situação. Entretanto, os modelos mentais são limitados pela incapacidade da mente humana em lidar com um grande número de fatores distintos simultaneamente. Em virtude desta limitação, os modelos mentais são normalmente simples (STERMAN, 1988), considerando apenas um número reduzido de componentes e de relações entre eles. Em sistemas mais complexos, especialmente em sistemas fechados com múltiplos ciclos de realimentação, esta simplificação pode gerar interpretações incorretas do mundo real, acarretando em decisões incorretas. A dificuldade de examinar as premissas de um sistema complexo cria um contexto para que ambigüidades e contradições passem desapercebidas e permaneçam sem solução nos modelos mentais. Assim, a simplicidade dos modelos mentais justifica o comportamento não intuitivo apresentado por alguns sistemas, visto que diversos componentes do sistema e relações entre eles foram desconsiderados. Além disto, modelos mentais residem no conhecimento tácito de cada indivíduo, sendo difíceis de compartilhar e validar.

Modelos explícitos são representações de modelos mentais em uma linguagem que possa ser compartilhada com outros indivíduos. Em um modelo explícito, as premissas de um sistema são declaradas em documentos que podem ser revisados. Entretanto, os modelos explícitos são limitados a influenciar os modelos mentais, visto que nenhuma decisão é tomada a partir deles, mas a partir dos modelos mentais (STERMAN, 1988, FORRESTER, 1991). Sob este ponto de vista, os modelos explícitos são úteis para:

- **Representação do conhecimento**: modelos explícitos promovem o entendimento dos sistemas reais. A descrição de sistemas complexos através de modelos explícitos facilita a transposição da barreira imposta pela limitação da mente humana em tratar a complexidade inerente a estes sistemas. Um modelo explícito ajuda a organizar as informações sobre um sistema, facilitando seu entendimento, ou seja, a construção de um modelo mental;
- **Validação do conhecimento**: segundo FORRESTER (1991), o conhecimento sobre um sistema pode ser dividido em suas premissas, nos resultados esperados para o sistema e em seus resultados reais. Normalmente, o conhecimento a respeito das

premissas de um sistema é mais confiável que o conhecimento sobre seus resultados. O consenso sobre as premissas de um sistema também é mais comum que o consenso sobre seus resultados. Por outro lado, os resultados reais de um sistema podem não replicar os resultados intuitivamente implícitos em suas premissas, especialmente em sistemas complexos. O processo de modelagem separa a parte confiável do conhecimento, ou seja, as premissas, do comportamento esperado do sistema (FORRESTER, 1991). Quando um modelo construído a partir das premissas de um sistema não se comporta como o sistema real, ele pode ser estudado para identificar as condições que fizeram com que seu comportamento se desviasse do que era esperado intuitivamente;

- **Difusão do conhecimento**: por residirem na experiência de cada indivíduo, os modelos mentais não podem ser compartilhados. Modelos explícitos, por estarem descritos em documento acessíveis a um grupo, permitem a difusão do conhecimento expresso no modelo;
- **Previsão**: um modelo explícito captura o conhecimento acerca de um sistema real. Este conhecimento pode ser utilizado para projetar os resultados esperados do sistema de acordo com novas condições iniciais.

2.1.1 Classificação dos Modelos Explícitos

Modelos explícitos representam sistemas através de um conjunto finito de símbolos e de conexões entre estes símbolos. O conjunto de símbolos disponíveis para a construção de um modelo e as possíveis conexões entre estes símbolos formam uma linguagem. Os modelos de sistemas são construídos a partir de uma linguagem de modelagem, que impõe restrições e fornece semântica aos componentes de um modelo.

Uma primeira classificação dos modelos explícitos deve considerar seu formalismo. Esta classificação define um espectro contínuo, onde figuram desde modelos formados por símbolos abstratos até modelos formais, cuja linguagem não permite a expressão de ambigüidades. As linguagens de construção de modelos formais definem precisamente a semântica de cada símbolo e das conexões entre os símbolos que podem ocorrer em um modelo. A precisão com que a semântica de um modelo é descrita determina o universo de inferências que podem ser realizadas acerca do comportamento do sistema representado pelo modelo.

Um modelo formal é um conjunto de postulações matemáticas e lógicas com detalhes suficientes para descrever os objetivos e as limitações de um sistema. Um modelo formal expressa as relações entre os componentes de um sistema através de suas postulações. Um trecho de código em uma linguagem de programação é um exemplo de modelo formal. Nos modelos formais, um sistema é caracterizado por seus parâmetros e suas postulações. Os parâmetros de um modelo são medidas independentes que configuram os valores de entrada e a estrutura do sistema (MADACHY e BOEHM, 1999). Estes parâmetros são informações exógenas, ou seja, estão fora dos limites do sistema. As postulações de um modelo determinam a estrutura interna do sistema, geralmente representada por um conjunto de variáveis. As postulações são informações endógenas, ou seja, estão contidas dentro dos limites do sistema. As configurações de valores assumidos pelas variáveis manipuladas pelas postulações definem os estados do sistema. A forma com que estes valores se alteram ao longo do tempo é denominada de comportamento do sistema (MARTIN, 1997a).

De acordo com a variação do seu estado, os modelos formais podem ser classificados como estáticos ou dinâmicos. Os modelos estáticos não são influenciados pela passagem do tempo, ou seja, permanecem no mesmo estado ao longo de toda a sua existência. Os modelos dinâmicos mudam de estado ao longo do tempo, ou seja, suas variáveis podem assumir valores distintos em diferentes instantes do tempo.

Modelos dinâmicos podem ser classificados como modelos discretos ou contínuos, de acordo com a forma com que ocorrem suas mudanças de estado. Em um modelo discreto, estas mudanças ocorrem através de eventos. O tratamento de um evento modifica os valores das variáveis do modelo, podendo gerar novos eventos. O intervalo de tempo entre dois eventos distintos não pode ser previamente determinado, provocando mudanças de estado sem um período constante. Em um modelo contínuo, os valores das variáveis mudam em intervalos de tempo constantes, previamente determinados e, geralmente, infinitesimais.

A capacidade de expressão de incerteza nos modelos formais permite classificá-los como determinísticos ou estocásticos. Modelos determinísticos não permitem a expressão de incerteza. Apresentados os mesmos parâmetros em avaliações distintas, os modelos determinísticos sempre apresentam comportamento idêntico. Modelos estocásticos permitem a representação de incerteza em suas postulações. Apresentados os mesmos parâmetros em avaliações distintas, não se pode afirmar que os modelos

estocásticos produzem os mesmos resultados. A Tabela 2.1 apresenta um resumo da classificação de modelos formais apresentada nesta seção.

Quanto ao Comportamento	Estáticos Dinâmicos Discretos Dinâmicos Contínuos
Quanto à Representação de Incerteza	Determinísticos Estocásticos

Tabela 2.1- Classificação dos modelos formais

2.1.2 Avaliação do Comportamento de Modelos Formais

O comportamento de modelos baseados em formulações matemáticas pode ser avaliado através de simulações. Uma simulação é um mecanismo que reproduz o comportamento de um sistema através de operações numéricas realizadas pelo computador (MARTIN, 1997a). A simulação pode ser utilizada para avaliação do modelo quando este não possuir uma solução analítica, ou seja, uma solução através de uma fórmula fechada (MADACHY e BOEHM, 1999). Este mecanismo facilita o entendimento das interações entre os componentes de um sistema e do sistema como um todo (BELLINGER, 1999a). O papel da simulação é fundamental em modelos onde os efeitos destas interações estão separados de suas origens no tempo ou no espaço.

Simulações podem ser utilizadas para determinar o efeito que uma alteração em uma parte do sistema provoca no sistema como um todo. Este tipo de análise permite avaliar a resposta de um sistema a uma situação diferente daquela para a qual o sistema foi planejado. Este estudo é denominado análise de cenário. Simulações também são um mecanismo de comunicação eficiente, demonstrando como um processo evolui ao longo do tempo e estimulando o desenvolvimento de mecanismos que melhorem esta evolução (MADACHY e BOEHM, 1999).

As simulações podem ser classificadas como discretas ou contínuas. Em uma simulação discreta uma ação provoca um ou mais eventos que ocorrerão no futuro. Por exemplo, em um modelo que descreva um projeto de desenvolvimento de software, a ação de selecionar um conjunto de desenvolvedores para a realização de uma tarefa resultará no evento de conclusão da tarefa em algum momento no futuro. Um simulador possui uma fila de eventos ordenada pelo tempo para ocorrência de cada evento. Em cada iteração do processo de simulação, o simulador trata o primeiro evento da fila, adiantando um relógio interno até o instante de ocorrência do evento e atualizando as variáveis do modelo de acordo com o evento. O tratamento de um evento também pode

gerar novos eventos, que serão inseridos na fila do simulador. Como o intervalo de tempo entre cada par de eventos não pode ser previamente determinado, a simulação ocorre sem um período constante. A simulação é encerrada em um tempo previamente determinado ou quando não existirem eventos para serem tratados.

A simulação de modelos contínuos ocorre em intervalos de tempo infinitesimais, constantes e previamente determinados. Em uma simulação contínua, as variáveis do modelo são alteradas a cada intervalo de simulação. Por exemplo, a taxa de produtividade da equipe resultará em uma taxa de conclusão de tarefas. Ambas as taxas podem variar ao longo do tempo. Em cada iteração do processo de simulação contínua, o simulador adianta um relógio interno por um tempo equivalente ao intervalo de simulação. Em seguida, os valores das variáveis do modelo são recalculados. A simulação termina após a realização de um número de intervalos previamente estabelecido.

Existe uma ampla discussão sobre o tipo de simulação mais adequado para modelos de projetos de software (POWELL e CLARK, 1999, DRAPPA e LUDEWIG, 1999, RAFFO et al., 1999). Não acreditamos que seja possível determinar um melhor tipo de simulação para todos os modelos de projetos. Em geral, aspectos discretos ocorrem em conjunto com aspectos contínuos em projetos de desenvolvimento de software. Portanto, acreditamos em técnicas de simulação que permitam a representação de ambos aspectos, valendo-se dos avanços obtidos por pesquisas nas duas linhas de simulação. Em um simulador contínuo, eventos discretos podem ser modelados condicionando-se sua ativação a determinados intervalos de tempo. De forma análoga, em um simulador discreto, os aspectos contínuos podem ser tratados por um evento disparado em intervalos constantes de tempo.

2.2 O Pensamento Sistêmico e o Processo de Desenvolvimento de Software

O pensamento sistêmico (SENGE, 1990) – *system thinking* – é uma filosofia que procura elucidar o comportamento de um sistema através de sua estrutura. O pensamento sistêmico analisa as relações entre os componentes de um sistema, descrevendo o comportamento do sistema a partir destas relações (RICHMOND, 1993). Em sistemas com dinâmica complexa, esta análise se beneficia da utilização de modelos formais, uma vez que o comportamento dos sistemas não pode ser previsto de forma eficiente apenas por modelos mentais.

FORRESTER (1991) ressalta o alto grau de evolução alcançado nos campos de engenharia e de tecnologia nas últimas décadas, em contraponto com a reduzida evolução dos campos de economia e de administração no mesmo período. Forrester associa esta diferença de evolução a uma resistência em aceitar que as organizações sociais, como famílias, corporações e governos, são sistemas, pertencendo a um mesmo grupo genérico, onde também se encontram refinarias e pilotos automáticos para aeronaves. Esta resistência, junto com a complexidade crescente dos sistemas sociais, faz com que estes repitam sucessivamente os mesmos erros. Como as leis de formação do sistema não são compreendidas, seus participantes tentam violá-las. Este processo resulta em frustração, a medida que as leis se impõem, neutralizando o esforço aplicado contra elas (BELLINGER, 1999b).

Sistemas de dinâmica complexa se caracterizam pela existência de múltiplos componentes interrelacionados, características variando de forma complexa ao longo do tempo, ciclos de realimentação, relações não lineares e manipulação de informações qualitativas (STERMAN, 1992). As próximas seções analisam a existência destas propriedades em projetos de desenvolvimento de software de larga escala, caracterizando-os como sistemas de dinâmica complexa, que podem se beneficiar da utilização de modelos formais.

2.2.1 Componentes Interrelacionados

Os sistemas de software são formados por diversos componentes interrelacionados. Por exemplo, algumas tarefas de um projeto somente podem ser realizadas após a conclusão de outras tarefas (dependência entre tarefas). Tarefas produzem artefatos, consumindo artefatos produzidos por outras tarefas (dependências entre tarefas e artefatos). Tarefas são realizadas por desenvolvedores (dependência entre tarefas e desenvolvedores) que utilizam ferramentas (dependência entre tarefas e ferramentas) e consomem recursos (dependência entre tarefas e recursos). Outras dependências podem ser traçadas relacionando as tarefas e as tecnologias aplicadas, o projeto e o domínio da aplicação, entre outros. DE ALMEIDA et al. (1998) e KITCHENHAM et al. (1999) apresentam modelos descritivos de processos de desenvolvimento de software onde diversos outros tipos de relacionamentos podem ser identificados.

2.2.2 Dinâmica

Um processo de desenvolvimento de software envolve muitas informações que variam ao longo do tempo, como o tamanho da equipe, o volume de código produzido, o número de tarefas pendentes, o número de requisitos testados, entre outros. Outra característica dinâmica muito importante e presente no processo de desenvolvimento de software é que o sistema reage de maneira distinta a curto e longo prazo. A "lei de Brooks" (BROOKS, 1974) é um exemplo claro deste efeito. A curto prazo o ingresso de novos integrantes em uma equipe de desenvolvimento atrasa um projeto, devido ao tempo que os desenvolvedores experientes deverão dedicar ao treinamento dos novatos. Entretanto, após o treinamento, os desenvolvedores novatos contribuirão de forma positiva para a produtividade da equipe. Assim, a "lei de Brooks" vale localmente, mas não globalmente em um sistema (ABDEL-HAMID e MADNICK, 1991).

2.2.3 Ciclos de Realimentação

O desenvolvimento de software contém diversos ciclos de realimentação. Por exemplo, se um projeto está atrasado em relação a seu planejamento, a gerência pode incentivar a equipe a trabalhar horas extras. Entretanto, embora o esforço da equipe reduza momentaneamente o atraso do projeto, a qualidade dos artefatos produzidos durante as horas extras é geralmente inferior aos artefatos produzidos durante a jornada normal de trabalho (DEMARCO, 1982). A redução de qualidade se reflete em um maior número de erros, que exigirão maior esforço para detecção e correção. O atraso provocado pela detecção e correção dos erros derivados do trabalho de baixa qualidade pode ser maior que o atraso que originalmente provocou a decisão pelo incentivo ao trabalho em horas extras. Assim, ao longo do processo de desenvolvimento, uma decisão pode ter efeito sobre ela mesma, o que representa um ciclo de realimentação.

2.2.4 Relações Não-Lineares

O desenvolvimento de software possui diversas relações não-lineares. O efeito de uma decisão geralmente não pode ser mapeado por uma relação proporcional a ação original. Considerando-se o exemplo da seção 2.2.3, não podemos assumir que o tempo ganho com o incentivo das horas extras é linear ao esforço necessário para a correção dos erros gerados pelo trabalho de qualidade inferior.

2.2.5 Aspectos Qualitativos

O desenvolvimento de software envolve diversos aspectos qualitativos, devido a sua estreita relação com as ciências humanas. Esta relação se deve à influência que os desenvolvedores exercem sobre o comportamento do processo e a qualidade dos produtos. Desta forma, o desenvolvimento de software não deve ser visto como uma tarefa de pura engenharia, onde o componente social possa ser negligenciado. O relacionamento entre os integrantes da equipe, a motivação da equipe em relação ao projeto e as características específicas da cultura da organização de desenvolvimento são fatores sociais que influenciam diretamente o sucesso ou insucesso de um projeto de software.

2.3 Aplicações de Modelos no Processo de Desenvolvimento de Software

As características identificadas na seção 2.2 indicam que projetos de software podem se beneficiar da utilização de modelos formais. Esta percepção não é recente, considerando-se que os primeiros modelos formais de projetos de software surgiram em meados da década de 1970 (HOUSTON, 1996). As próximas subseções apresentam algumas aplicações para estes modelos.

2.3.1 Planejamento e Controle

A capacidade preditiva de um modelo formal pode ser explorada no planejamento e controle de projetos de desenvolvimento de software. No planejamento de um projeto, o gerente insere as características do projeto como parâmetros do modelo, avaliando os resultados esperados para o projeto real de acordo com o comportamento indicado nas postulações do modelo. Se o modelo apresentar propriedades estocásticas, as incertezas acerca dos parâmetros também podem ser expressas, permitindo que o gerente analise a variância (risco) dos resultados esperados para o projeto.

No controle de um projeto, os parâmetros do modelo podem ser atualizados a medida que novas informações sobre o estado do projeto estiverem disponíveis. Com o conhecimento destas informações, os resultados do modelo podem ser reavaliados, estimando-se a distância do estado atual do projeto em relação ao estado planejado. Ações corretivas, se necessárias, também podem ser avaliadas no modelo antes de serem executadas no projeto real.

Diversos modelos foram construídos com o objetivo de predizer o comportamento de projetos de desenvolvimento de software (ABDEL-HAMID e MADNICK, 1991,

LIN e LEVARY, 1989, LIN et al., 1997). Estes modelos têm como característica um desligamento das ferramentas tradicionais de gerenciamento de projetos, sendo descritos por extensos sistemas de equações, que são difíceis de entender e alterar. Assim, a utilização destes modelos na gerência operacional de projetos de software é limitada, dadas as dificuldades de sua construção.

Percebendo esta limitação, alguns autores visaram aproximar os modelos de projetos de software às técnicas tradicionais de planejamento e controle de projetos. RODRIGUES e WILLIANS (1996) apresentam um modelo dinâmico de projetos de software integrado a modelos de redes de tarefas. PFHAL e LEBSANFT (1999) apresentam um modelo de projetos que associa características dinâmicas a modelos descritivos de processos de desenvolvimento de software.

Acreditamos que o caminho para disseminar a utilização de modelos formais na gerência operacional de projetos de software passa por facilitar a construção destes modelos e ampliar o universo de conhecimento que eles são capazes de representar. Assim, devido a sua integração com ferramentas tradicionais de gerência, consideramos promissoras as propostas de RODRIGUES e WILLIANS (1996) e PFAHL e LEBSANFT (1999). Entretanto, estas propostas ainda são limitadas na capacidade de ampliar o conhecimento expresso: os modelos consideram um determinado conjunto de aspectos do processo de desenvolvimento de software e não existem mecanismos definidos para que outros aspectos sejam analisados, senão pela alteração das equações que compõem o modelo. Acreditamos que uma melhor alternativa seria o desenvolvimento de modelos separados, contemplando os aspectos desejados, e sua posterior integração com o modelo original, sem a exigência de alteração da composição do modelo original.

2.3.2 Treinamento

PRESSMAN (2000) aponta o treinamento inadequado de gerentes e desenvolvedores como um dos fatores que contribuem para a baixa qualidade no desenvolvimento de software. JONES (1994) ressalta a importância de mecanismos de treinamento, apontando a falta de padrões de avaliação e treinamento de gerentes como riscos para processos de desenvolvimento de software.

O aprendizado tradicional não parece eficaz no campo da gerência de projetos, dada a dificuldade de se replicar um processo de desenvolvimento com o intuito de um aluno vivenciar a experiência de um gerente. STATZ (1994) observa que é comum que

desenvolvedores assumam os postos de gerência sem treinamento específico, esperando-se que eles desenvolvam as proficiências necessárias ao longo de diversos projetos. A autora ressalta que um passo importante para reduzir o número de projetos encerrados sem sucesso é a criação de modelos do processo de desenvolvimento de software, que permitam a avaliação e o treinamento adequado dos gerentes de projeto.

Simuladores não são apenas úteis para modelar sistemas de difícil observação no mundo real, mas também constituem uma ferramenta poderosa para aprimorar o processo de aprendizado quando combinados com experimentação real (MARTIN, 1997a). FORRESTER (1991) prevê que os mecanismos de treinamento do futuro incluirão uma biblioteca de situações de gerenciamento, combinando estudos de caso com modelos dinâmicos de sistemas. Embora esta previsão não tenha sido feita no contexto da gerência de projetos de software, nada impede que tais mecanismos sejam aplicados neste contexto. Entretanto, as técnicas de modelagem ainda devem evoluir para que este estágio seja atingido, visto que os mecanismos de integração de modelos presentes nas linguagens de modelagem atuais ainda são precários.

Um bom processo de treinamento de gerentes é comparado ao processo de aprendizado dos pilotos de avião, realizado em simuladores de vôo. MERRIL (1995) desenvolveu um simulador de vôo baseado em um modelo de gerência de projetos de software e um processo de treinamento baseado neste simulador. MAIER (1996), em um experimento inicial, demonstra a validade de simuladores de vôo baseados em modelos de gerência como ferramenta de treinamento. DRAPPA e LUDEWIG (2000) apresentam os resultados de um estudo de caso e um experimento controlado que avaliaram o aprendizado apoiado por simulações no campo da gerência de projetos de software. Embora os estudos não tenham sido conclusivos quanto a melhoria do desempenho dos alunos após as simulações, observações relevantes sobre o comportamento dos alunos foram traçadas a partir destes estudos experimentais.

Acreditamos que modelagem e simulação podem ser úteis no treinamento em Engenharia de Software. Entretanto, estas técnicas não são suficientes, nem substituem o ensino tradicional, servindo como apoio e motivação para este. Tornam-se necessários novos métodos de ensino e técnicas para extrair o conhecimento de um conjunto de simulações realizadas sobre um modelo.

2.3.3 Políticas de Aprimoramento de Processo

O conceito de aprimoramento contínuo de processo, embora não seja inovador em diversas áreas da produção, somente passou a ser aplicado no desenvolvimento de software a partir do final da década de 1980 (RAFFO, 1993). Desde então, diversas organizações de desenvolvimento vêm promovendo melhorias em seu "*modus operandi*", visando melhor enquadramento em padrões de qualidade de processo, como o modelo CMM (PAULIK et al., 1993) e os padrões derivados da ISO 9000.

Entretanto, pela diversidade de alternativas de aprimoramento de processos e pelo custo e risco relacionados com a implantação destas tecnologias, seria desejável que modelos quantitativos de processos de desenvolvimento de software oferecessem suporte ao estabelecimento de prioridade entre as alternativas, mostrando seus benefícios e suas desvantagens. Estes modelos poderiam ser utilizados para justificar a aplicação de recursos em determinadas alternativas, projetando seu retorno no desempenho do processo de desenvolvimento (RAFFO, 1993).

RAFFO et al. (1999) apresentam um modelo estocástico de projetos de software construído com o intuito de avaliar propostas de alterações em processos antes que estas propostas sejam aplicadas em projetos reais. O modelo captura 71 diferentes atividades do processo de desenvolvimento de software, cada qual complementada com informações sobre custos, cronograma e taxa de erros capturadas de projetos passados.

TVEDT (1996) apresenta um modelo que permite testar políticas de aprimoramento de processos de desenvolvimento de software antes que estas sejam implantadas em um processo real. O modelo representa um processo incremental evolutivo, realizado por equipes concorrentes. A construção do modelo visou a flexibilidade, permitindo o acoplamento de modelos complementares que representem políticas de aprimoramento de processo como, por exemplo, inspeções. O modelo principal pode ser simulado com ou sem os modelos complementares. As simulações permitem a análise dos resultados proporcionados por cada política de aprimoramento de processo, permitindo a seleção das melhores políticas para um processo específico.

A extensão de modelos de projeto através de modelos complementares é importante para que os modelos de projeto não tenham que ser reconstruídos a cada política de aprimoramento avaliada. Entretanto, o custo de integração dos modelos complementares na proposta de Tvedt ainda é alto, exigindo a revisão do modelo original para determinar as postulações afetadas pelo modelo complementar. Seria

desejável que os modelos complementares operassem de forma "plug-&-play" sem exigir o conhecimento de detalhes do modelo principal para que se integrassem a este.

2.4 Técnicas de Modelagem Aplicadas em Software

Nesta seção discutimos diversas técnicas de modelagem e sua aplicação em projetos de desenvolvimento de software. As técnicas analisadas incluem os modelos de eventos discretos (ROSS, 1990), os modelos baseados em diagramas de estado (TRIVEDI, 1982) e os modelos dinâmicos de sistemas (FORRESTER, 1961). Focalizamos também nos aspectos referentes à simulação em cada uma destas técnicas.

2.4.1 Modelos de Eventos Discretos

Os elementos chave de um modelo de eventos discretos são suas variáveis e seus eventos (ROSS, 1990, BANKS et al., 1996). As variáveis descrevem o estado do sistema ao longo do tempo, capturando sua dinâmica. Os valores das variáveis mudam em intervalos de tempo discretos e não constantes, determinados pela ocorrência de eventos. O ajuste nos valores das variáveis pode provocar a geração de novos eventos. A ocorrência destes eventos renova o processo, forçando uma nova atualização das variáveis e conseqüente geração de novos eventos. A simulação prossegue até que nenhum evento seja gerado ou até um limite de tempo previamente determinado.

Exemplos de eventos relevantes em um projeto de desenvolvimento de software incluem o início da execução de uma tarefa, a conclusão de uma tarefa, a contratação de um desenvolvedor, a chegada de um recurso, entre outros. Cada evento relevante para um modelo contém um procedimento associado que indica as alterações realizadas sobre os valores das variáveis do modelo e as características de novos eventos gerados em decorrência destas alterações.

Diversos modelos de projetos de desenvolvimento de software foram construídos utilizando o paradigma de eventos discretos. RUS e COLLOFELO (1998) desenvolveram um modelo com o objetivo de avaliar a eficiência de políticas de aprimoramento de processos e seus efeitos na confiabilidade dos produtos de software construídos através destes processos. Entre as políticas de aprimoramento analisadas, encontram-se a especificação formal de requisitos, inspeções, revisões, reutilização de software, entre outras. A viabilidade de cada política de aprimoramento depende de um conjunto de fatores relacionados com o processo e com o produto, como o tamanho do código fonte, a disponibilidade da equipe para atividades de verificação e validação, o

tempo planejado de testes, a disponibilidade de componentes reutilizáveis, entre outros. A complexidade destas relações, principalmente quando um grande número de políticas de aprimoramento são combinadas, torna necessária a realização de simulações do processo. As simulações permitem a análise das políticas de aprimoramento de confiabilidade, determinando as políticas que resultam em maiores benefícios para o projeto em questão.

O modelo de RUS e COLLOFELO (1998) é dividido em duas seções. A seção de produção captura os aspectos referentes à geração de novos artefatos de software e novos defeitos, que são gerados durante a produção de artefatos. A taxa de geração de defeitos depende de vários fatores, como a relação entre o tempo esperado e o tempo disponível para a conclusão do projeto, a taxa de produção de novos artefatos e a taxa de correção de erros. A seção de gerenciamento divide o processo de desenvolvimento em três etapas: a produção de artefatos, a verificação dos artefatos e a correção de defeitos. Esta seção contabiliza o custo e o tempo do projeto, avaliando os fatores que afetam a taxa de geração de erros, como a data esperada para a conclusão do projeto.

HANSEN (1996) apresenta um modelo que focaliza as atividades de retrabalho (*rework*). O modelo demonstra como o volume de retrabalho e os momentos em que as atividades de retrabalho são inseridas no processo de desenvolvimento afetam os resultados do projeto, como seu custo e o esforço necessário para a realização do projeto. O modelo utiliza um processo de desenvolvimento em cascata, focalizando a atividade de projeto detalhado e analisando o impacto de diversas estratégias de alocação de retrabalho.

O modelo de HANSEN (1996) decompõe um projeto em módulos, determinando o tempo médio para a construção de cada módulo. O modelo pode ser decomposto nas seções de produção, verificação e retrabalho. A seção de produção determina o módulo sendo produzido e contabiliza o tempo consumido e o custo desta construção. Depois de construído, o módulo passa para a seção de verificação, que determina se o módulo está correto ou se precisa de algum tipo de correção. Se correções são necessárias, o módulo passa para a seção de retrabalho, onde seus erros serão reparados.

HANSEN (1996) apresenta os resultados de simulação do modelo com diferentes estratégias de alocação de retrabalho, como retrabalho escalonado após a conclusão do projeto, priorização do retrabalho de cada categoria de módulos e priorização preemptiva de retrabalho. Os resultados demonstram que a estratégia de organização do

retrabalho de um projeto afeta o tempo médio de produção dos módulos, afetando assim o tempo e custo do projeto.

MARTÍNEZ-GARCÍA e WARBOYS (1998) apresentam uma técnica capaz de traduzir modelos descritivos de projetos de software para modelos de eventos discretos. Os modelos descritivos devem ser desenvolvidos segundo a metodologia RAD – *"Role Activity Diagram"* – (WARBOYS, 1998), que captura as atividades que compõem um projeto e os papéis que os agentes devem assumir para realizarem estas atividades. A modelagem por eventos discretos complementa os modelos descritivos com informações acerca do tempo necessário para a realização de cada atividade, assim como outros atributos utilizados durante a simulação. O modelo de eventos discretos permite a simulação do modelo descritivo, oferecendo ao engenheiro de processos dados sobre gargalos, tempos de interação dos agentes, entre outras informações dinâmicas do processo.

SCHACCI (1999) apresenta um modelo de projetos de software descrito através de regras e apoiado por simulação discreta. O estado do modelo é representado por um conjunto de fatos em uma base de conhecimento, enquanto suas postulações são representadas por um conjunto de predicados. Os predicados possuem condições de disparo dependentes do estado do modelo e, quando disparadas, executam um procedimento associado a elas. A simulação por regras permite que o sistema armazene um histórico dos fatos acontecidos ao longo do processo de desenvolvimento. A simulação por eventos discretos, que considera o tempo e o custo de cada atividade do projeto, é utilizada para predição do custo e tempo de realização do projeto.

Dentre os modelos apresentados, o modelo de RUS e COLLOFELLO (1998) e o modelo de HANSEN (1996) permitem a análise de diferentes combinações de atividades no processo de desenvolvimento de software. Entretanto, estes modelos não são facilmente extensíveis: suas equações devem ser inspecionadas e alteradas para que as atividades complementares sejam consideradas na análise. Encontra-se aqui o mesmo problema citado nas seções 2.3.1 e 2.3.3, onde modelos são utilizados para análise de atividades complementares, mas esta análise tem custo elevado, devido a necessidade de alteração nos modelos.

Os modelos de MARTÍNEZ-GARCÍA e WARBOYS (1998) e SCHACCI (1999) trazem como principal característica a possibilidade de descrição dos modelos em uma linguagem mais próxima das ferramentas tradicionalmente utilizadas na gerência de projetos. Esta característica é desejável, pois facilita a construção dos modelos,

permitindo sua futura simulação. Entretanto, assim como os modelos de RODRIGUES e WILLIANS (1996) e PFAHL e LEBSANFT (1999), discutidos na seção 2.3.1, os modelos aqui apresentados não possuem mecanismos definidos para contemplarem aspectos do processo de desenvolvimento de software que não se caracterizam em suas representações de mais alto nível. Se outras características forem necessárias, o modelo deve ser alterado em sua representação de baixo nível, reduzindo-se assim as vantagens da tradução do modelo entre os níveis de abstração. Conforme discutido na seção 2.3.1, seria desejável a representação da dinâmica complementar em modelos separados e integráveis ao modelo original em sua representação de alto nível, sem exigir que o desenvolvedor de modelos trabalhe diretamente sobre as equações do modelo.

2.4.2 Modelos Baseados em Diagramas de Estado

Um diagrama de estados é uma ferramenta de modelagem que apresenta os diversos estados em que um conjunto de entidades pode se encontrar ao longo de sua existência, e as possíveis transições entre estes estados. As transições entre estados são disparadas por eventos, cuja ocorrência é determinada por condições de guarda. Uma condição de guarda é uma expressão lógica geralmente dependente do estado do sistema e avaliada periodicamente durante sua simulação.

Os diagramas de estado podem ser decompostos hierarquicamente. Um sistema pode ser decomposto em diversos componentes com dinâmica independente, cada qual com um determinado estado ao longo do tempo. Os estados dos componentes são representados por diagramas de estados independentes. O conjunto destes diagramas forma o diagrama de estados do sistema. Os diagramas de estado também permitem a descrição de ciclos de realimentação através de ciclos de transição. Assim, pode existir um caminho no grafo formado pelos estados e suas transições, que permita que um estado seja alcançado a partir de si próprio.

HUMPHREY e KELLNER (1989) apresentam uma técnica para modelagem de processos de desenvolvimento de software baseada em diagramas de estado. Esta técnica sugere a identificação das entidades manipuladas pelo processo de desenvolvimento e a construção de diagramas de estado que representem a dinâmica destas entidades. Ao longo de sua existência, as entidades criadas pelo processo podem sofrer diversos tipos de manipulação. Entre estas manipulações as entidades passam por diversos estados, permanecendo por um período positivo e não-nulo em cada estado. Um estado é considerado ativo quando a entidade sofre alguma transformação enquanto

permanece nele. Os autores observam que o processo não é descrito pelas tarefas que o compõem, como nos modelos de redes tradicionais (WIEST e LEVY, 1977), mas pela dinâmica e evolução das entidades produzidas.

O comportamento dinâmico de um modelo de estados pode ser acompanhado através de simulações. Estas simulações podem ser realizadas com ou sem restrições de recursos. Nas simulações sem limitação de recursos, consideram-se apenas os tempos de permanência das entidades em seus estados. Nas simulações com restrições de recursos, considera-se que uma transição entre estados consume um determinado conjunto de recursos, que devem estar disponíveis para que a transição seja realizada. Neste caso, ao fim do seu tempo de permanência em um estado, uma entidade deve aguardar a disponibilidade de recursos para realizar uma transição. Políticas de aplicação de recursos devem ser definidas para resolução de conflitos, que ocorrem quando duas ou mais entidades estão prontas mas os recursos disponíveis permitem apenas que parte das entidades realizem suas transição de estado.

Os modelos de HUMPHREY e KELLNER (1989) foram utilizados por RAFFO (1993) no desenvolvimento de uma técnica para avaliação quantitativa de políticas de aprimoramento de processo. Esta técnica utiliza três medidas de desempenho: custo do projeto, qualidade do código produzido e tempo de desenvolvimento. Os modelos do processo de desenvolvimento são utilizados para avaliar possíveis políticas de aprimoramento de processo, antes do investimento de um esforço substancial na aplicação destas políticas em projetos reais. RAFFO et al. (1999) apresentam um caso prático de aplicação de um modelo de estado na seleção de políticas de aprimoramento de processo. Entretanto, os modelos apresentados por Raffo não trazem detalhes suficientes para permitir a avaliação do custo de integração de modelos descrevendo atividades complementares.

2.4.3 Modelos Baseados em Dinâmica de Sistemas

Dinâmica de Sistemas é uma disciplina de modelagem criada na década de 1960 por FORRESTER (1961), no Massachusetts Institute of Technology. As técnicas da Dinâmica de Sistemas podem ser aplicadas para entender e influenciar a forma com que os elementos de um sistema variam ao longo do tempo. Estas técnicas utilizam conceitos do campo de controle com retroalimentação (ciclos de *feedback*) para organizar as informações disponíveis a respeito de um sistema, criando modelos que podem ser simulados em um computador (FORRESTER, 1991).

As técnicas da Dinâmica de Sistemas foram inicialmente aplicadas nos campos de administração e engenharia. Entretanto, progressivamente, estas técnicas estão sendo aplicadas na análise de sistemas sociais, econômicos, agrícolas, físicos, químicos, biológicos e ecológicos (MARTIN, 1997a). O Apêndice A apresenta os diagramas que representam os modelos da Dinâmica de Sistemas e sua interpretação matemática.

Os modelos produzidos através das técnicas da Dinâmica de Sistemas permitem a descrição de características que não são facilmente expressas em outros modelos. Estas características e sua aplicação na modelagem de processos de desenvolvimento de software são discutidas a seguir.

- **Comportamento Endógeno**: a Dinâmica de Sistemas assume que o comportamento de um sistema é provocado pela estrutura formada pela conexão entre seus componentes (FORRESTER, 1991). Assim, os modelos são construídos para representar a estrutura do sistema, enquanto a simulação demonstra seu comportamento. A Dinâmica de Sistemas assume que o comportamento de fatores internos do modelo determinam seu comportamento visível. No desenvolvimento de software, por exemplo, o aumento da taxa diária de trabalho dos desenvolvedores aumenta sua produtividade, mas pode reduzir a qualidade do trabalho. Esta redução de qualidade provoca um maior número de erros, que, por sua vez, aumenta o esforço total para conclusão do projeto, em função da correção dos erros. Desta forma, se o projeto se encontra atrasado, aumentar a taxa de trabalho pode aumentar ainda mais o atraso do projeto;

- **Integração**: pela filosofia de procurar as causas de um desvio de comportamento na estrutura do sistema, os modelos da Dinâmica de Sistemas integram seus diversos "microcomponentes". A integração destes elementos permite a inferência de seu papel no comportamento do sistema como um todo. Exemplos de "microcomponentes" do processo de desenvolvimento de software são as atividades do processo, os desenvolvedores que realizarão estas atividades, a data esperada para conclusão do projeto, a flexibilidade do cronograma, entre outros;

- **Sistemas Fechados**: a Dinâmica de Sistemas modela com clareza sistemas fechados, ou seja, sistemas caracterizados por ciclos de realimentação. Sistemas fechados assumem que uma decisão provoca uma cadeia de reações, que pode vir a influenciar as condições que exigiram a decisão original. Ciclos de realimentação são a estrutura

básica de todos os sistemas, sendo a causa de grande parte do seu comportamento dinâmico (WHELAN, 1996). Conforme apresentado na seção 2.2.3, há evidências da ocorrência de ciclos de realimentação em projetos de desenvolvimento de software;

- **Causas e Conseqüências Distantes no Tempo**: em diversos sistemas, particularmente em sistemas complexos, as conseqüências de uma decisão podem se ramificar e transparecer apenas muito tempo depois da tomada da decisão. Esta distância no tempo dificulta a percepção da verdadeira origem de um problema, visto que podem existir diversas explicações conflitantes (WEICK, 1979). A simulação de modelos da Dinâmica de Sistemas, por considerar fortemente os ciclos de realimentação e permitir o isolamento de variáveis do modelo, tem a capacidade de acompanhar as conseqüências de cada decisão no sistema;
- **Mapeamento de Modelos Mentais**: os modelos da Dinâmica de Sistemas não se baseiam apenas em informações numéricas. Embora a principal ferramenta de diagramação e as técnicas de simulação exijam que os modelos sejam descritos através de equações, estas equações podem ser geradas a partir do conhecimento contido nos modelos mentais dos indivíduos. Esta propriedade, conforme observado por MERRIL (1995), permite que um modelo da Dinâmica de Sistemas represente comportamentos genéricos, visto que ele pode gerar modos de referência que ocorrem em condições para as quais não existem dados para a construção de modelos estatísticos.

Em meados da década de 1980, ABDEL-HAMID e MADNICK (1991) desenvolveram um modelo de projetos de desenvolvimento de software utilizando as técnicas da Dinâmica de Sistemas. Este modelo foi construído para facilitar o entendimento dos fatores envolvidos nos projetos de software e para permitir a análise dos efeitos provocados por diferentes políticas de gerenciamento sobre seu custo e esforço necessário para a conclusão dos projetos. O modelo se aplica a projetos de média escala, com tamanho entre 16K e 64K linhas de código, e aborda as atividades de projeto, implementação, verificação de qualidade e testes. O modelo foi construído com base em uma extensa revisão da literatura, complementada com 27 entrevistas de campo.

O modelo foi dividido em quatro seções: gerenciamento de pessoal, produção de software, controle e planejamento. A seção de gerenciamento de pessoal inclui as

políticas de contratação, treinamento e transferência de recursos humanos do projeto. A integração do modelo faz com que estas políticas não sejam afetadas apenas pelas informações desta seção, mas também por características descritas em outras seções do modelo. Por exemplo, a política de contratação depende do esforço necessário para a conclusão do projeto, da equipe e do tempo disponível.

O tamanho da equipe e o nível de experiência de seus integrantes variam com o tempo. Estes elementos influenciam a produção de software, controlada pela segunda seção do modelo. Esta seção foi dividida em quatro subsistemas: alocação de pessoal, desenvolvimento de software, controle de qualidade e testes. O subsistema de alocação de pessoal divide o esforço da equipe de desenvolvimento entre as atividades que devem ser realizadas. Este subsistema determina o esforço necessário para o desenvolvimento de software, para o treinamento de novos integrantes da equipe, para o controle de qualidade, para a realização de testes e para a correção de erros. A medida que as tarefas de desenvolvimento são concluídas, seus resultados são submetidos ao controle de qualidade, representado no subsistema de mesmo nome. O controle de qualidade poderá encontrar erros nas tarefas analisadas, passando para a correção destes erros. Entretanto, alguns erros podem passar desapercebidos pelo controle de qualidade, sendo encontrados e corrigidos pelas atividades de teste. Estas atividades são representadas no subsistema de testes.

A medida que o projeto prossegue, seu estágio é comparado com o plano original de desenvolvimento. Esta comparação, realizada pela seção de controle, poderá alterar o planejamento original, alterando o número de integrantes da equipe, o cronograma e os custos. O plano original do projeto é gerado pela seção de planejamento.

O modelo de Abdel-Hamid foi validado pela repetição do comportamento de um projeto de sistema de controle de satélite, desenvolvido pela NASA no início da década de 1980. O termo repetição é utilizado para denotar que as variáveis do modelo mantiveram, ao longo do tempo, valores compatíveis com as medidas observadas no projeto real. O modelo repetiu a evolução no tempo dos custos do projeto, seu cronograma e o tamanho de sua equipe. Um segundo projeto de médio porte foi utilizado como base de um conjunto de experimentos com o modelo. Estes experimentos analisaram a economia do controle de qualidade (ABDEL-HAMID, 1988a), problemas com métodos de estimação de esforço de desenvolvimento (ABDEL-HAMID e MADNICK, 1983, ABDEL-HAMID e MADNICK, 1986, ABDEL-HAMID,

1989, ABDEL-HAMID, 1990), a "síndrome dos 90%" (ABDEL-HAMID, 1988b) e os efeitos da "Lei de Brooks" (ABDEL-HAMID e MADNICK, 1991).

O modelo de Abdel-Hamid e Madnick foi criticado e ampliado por diversos autores. LIN e LEVARY (1989) apresentam um modelo dinâmico de projetos de desenvolvimento de software desenvolvido para o *Jet Propulsion Labs* da NASA. O modelo, denominado SLICS, complementa o modelo de Abdel-Hamid e Madnick considerando as atividades de análise de requisitos, possíveis mudanças nos requisitos ao longo do projeto e a existência de múltiplos marcos de projeto (*milestones*) ao longo do processo de desenvolvimento. Este modelo foi posteriormente estendido (LIN et al., 1997) para considerar detalhes acerca da formação da equipe do projeto e a possibilidade de se eliminar atividades do processo devido a falta de recursos.

Os modelos baseados em Dinâmica de Sistemas apresentados nesta seção possuem uma característica comum: eles não descrevem precisamente os elementos envolvidos nos projetos de desenvolvimento de software. Por exemplo, ao descrever a produtividade dos desenvolvedores participando de um projeto, medida em KLOC[1]/dia, o modelo de ABDEL-HAMID e MADNICK (1991) assume que existem apenas dois tipos de desenvolvedores – experientes e inexperientes – determinando a produtividade média de cada tipo de desenvolvedor. Assim, duas postulações – a produtividade dos desenvolvedores experientes e dos desenvolvedores inexperientes – descrevem a taxa de produção da equipe. O modelo desconsidera variações de produtividade entre os desenvolvedores pertencentes a um mesmo grupo.

Acreditamos que esta simplificação se deve a incapacidade dos modelos construídos segundo a Dinâmica de Sistemas de representar atributos dos elementos componentes do modelo. Estes modelos tendem sempre a considerar médias dos valores destes atributos, ao invés de descrevê-los precisamente. Acreditamos também que, se um modelo deve ser utilizado na gerência operacional de projetos de software, este modelo deve capturar os detalhes acerca das atividades componentes do projeto, os desenvolvedores realizando estas atividades, os artefatos produzidos e os recursos consumidos durante o processo. Assim, as limitações da capacidade de representação dos modelos baseados na Dinâmica de Sistemas devem ser vencidas para que estes modelos sejam efetivamente utilizados na gerência de projetos.

---

[1] KLOC = milhares de linhas de código

TVEDT (1996) apresenta um modelo de projeto que contempla um ciclo de desenvolvimento incremental evolutivo realizado por diversas equipes concorrentes. O modelo foi desenvolvido para permitir a avaliação da eficácia, em termos de tempo e custo, da utilização de políticas de aprimoramento de projetos, como inspeções, por exemplo. As políticas de aprimoramento de processo são descritas em modelos complementares, que podem ser integrados ao modelo principal para que seu impacto sobre este seja avaliado. Entretanto, conforme observado na seção 2.3.3, a integração destes modelos exige a revisão e alteração das equações do modelo principal, o que dificulta a análise de alternativas de aprimoramento de processo, exigindo o conhecimento detalhado do modelo principal e incorrendo em alto custo a cada política avaliada.

RODRIGUES e WILLIANS (1996) apresentam um modelo dinâmico de projetos de software, denominado PMIM, que distribui as propriedades dinâmicas do projeto ao longo da rede de tarefas do projeto. Cada tarefa da rede é representada por um pequeno modelo dinâmico, responsável pelo gerenciamento, controle e acompanhamento da realização da tarefa. Os modelos dinâmicos de tarefas se comunicam entre si, estabelecendo as dependências entre as atividades. Estes modelos também se comunicam com um modelo de gerenciamento central, responsável por decisões que afetam todo o projeto, como o remanejamento de recursos, contratação de desenvolvedores, tempo de treinamento, entre outras.

LEBSANFT e PFAHL (1999) apresentam um modelo dinâmico de projeto de software, denominado PSIM, desenvolvido com objetivo para auxiliar no planejamento e controle de projetos de software e na seleção de políticas de aprimoramento de processo. O modelo foi construído com base em modelos descritivos de processo, capturando as relações entre as fases do processo de desenvolvimento. Ele compreende as atividades de projeto arquitetural, projeto detalhado, implementação e testes.

Os modelos de projetos de RODRIGUES e WILLIANS (1996) e LEBSANFT e PFAHL (1999) apresentam a preocupação em facilitar a construção de modelos dinâmicos de projetos, aproximando estes modelos das técnicas tradicionais de gerência de projetos. Este é um fator relevante para a utilização de modelos dinâmicos na gerência operacional de projetos, reduzindo o custo de construção dos modelos dinâmicos. Entretanto, repetem-se aqui os comentários feitos na seção 2.3.1 sobre a incapacidade de extensão destes modelos sem a alteração direta de suas equações, ou

seja, sem a intervenção do desenvolvedor de modelos na representação de baixo nível de abstração do modelo.

2.5 Representação de Incerteza em Modelos

Ao planejar um projeto de desenvolvimento de software, um gerente geralmente não possui detalhes suficientes para estabelecer precisamente o comportamento futuro do projeto. Assim, como o projeto está sujeito a incertezas, um modelo que represente este projeto também deve ser capaz de representar estas incertezas. Observamos dois tipos de incerteza que devem ser representadas em modelos de projeto de software: incertezas acerca do valor assumido por um parâmetro e incertezas sobre a ocorrência de determinados eventos no futuro.

O primeiro tipo de incerteza ocorre quando um gerente é incapaz de estabelecer um valor preciso sobre uma característica do projeto. Por exemplo, o tempo necessário para a conclusão de uma tarefa é desconhecido até que esta tarefa seja concluída. Entretanto, o gerente pode muitas vezes indicar uma duração esperada, uma duração mínima e uma duração máxima para a tarefa. Esta estratégia é utilizada nas redes de PERT/CPM (WIEST e LEVY, 1977), onde a duração de cada tarefa da rede é descrita por uma distribuição de probabilidade beta PERT (VOSE, 1996), caracterizada pelos três parâmetros apresentados acima.

O segundo tipo de incerteza ocorre quando um gerente é incapaz de determinar se um evento ocorrerá ou não no futuro. Por exemplo, o gerente pode não estar certo sobre a necessidade de uma atividade de retrabalho após a realização de uma inspeção, ou sobre a disponibilidade de um recurso para a realização de uma tarefa. Neste tipo de incerteza, o gerente deve ser capaz de traçar cenários sobre as "realidades" que podem ser encontradas no futuro, determinando a sensibilidade de seu projeto em relação a estes cenários.

Os modelos formais de projetos de software devem auxiliar o gerente na determinação dos dois tipos de incerteza. O modelo de LIN e LEVARY (1989) permite que o gerente descreva seus parâmetros através de um valor esperado, um valor mínimo e um valor máximo, indicando a probabilidade de ocorrência de cada um destes valores. O modelo calcula o valor médio desta distribuição discreta, submetendo este valor ao simulador. Este mecanismo de simulação é muito simples, pois assume que as equações do modelo representam apenas relações lineares.

Uma alternativa seria utilizar simulações de Monte Carlo (VOSE, 1996). A simulação de Monte Carlo é um método numérico que pode ser aplicado na resolução de modelos estocásticos. Ela envolve a geração de números aleatórios segundo as distribuições de probabilidade que descrevem as variáveis de um modelo, produzindo centenas ou milhares de cenários de cálculo do modelo, cada qual com resultados particulares. Os valores calculados para um determinado resultado podem ser organizados em um histograma de freqüência, que é um gráfico ou tabela apresentando os valores assumidos pelo resultado e o número de vezes que cada um destes valores foi calculado nos cenários aplicados sobre o modelo. Normalizado, tal que a área sob sua curva seja unitária, este histograma de freqüência pode ser entendido como uma distribuição de probabilidade para o resultado. ARTHUR e EBERLEIN (1996) apresentam uma técnica que permite a realização de simulações de Monte Carlo sobre modelos dinâmicos de sistemas.

A simulação de Monte Carlo é uma técnica extremamente flexível. Ela permite que as variáveis componentes de um modelo possuam distribuições de probabilidade de qualquer natureza, inclusive distribuições correlacionadas entre si. A simulação não restringe as operações utilizadas no modelo, permitindo a utilização de funções contínuas e discretas. Logaritmos, potências, cálculos condicionais e qualquer operação podem ser utilizados nas equações. Entretanto, a simulação de Monte Carlo é muito onerosa em termos computacionais: a precisão dos resultados produzidos pela simulação está relacionada com a qualidade dos geradores de números aleatórios disponíveis e com a quantidade de simulações realizadas.

2.6 Conclusão

A gerência operacional de projetos de desenvolvimento de software tem entre seus objetivos o planejamento e o controle dos projetos. No planejamento, os recursos necessários, as características dos desenvolvedores que formarão a equipe e o tempo esperado para a conclusão do projeto são algumas das informações relevantes para a definição do plano do projeto. No controle, o gerente precisa comparar as informações provenientes das frentes de desenvolvimento com o planejamento original, determinando a distância do estado atual do projeto para o estado esperado e ativando, quando necessário, soluções corretivas para reduzir esta distância.

Tendo em vista os objetivos de planejamento e controle, ambos com principal exigência de predição de comportamento, os seguintes requisitos são necessários para

que modelos formais sejam utilizados na gerência operacional de projetos de desenvolvimento de software:

- Facilidade de desenvolvimento: os modelos formais de projetos de software devem ser fáceis de construir, entender e validar. Modelos complexos, envolvendo um grande número de equações, são difíceis de depurar e validar, podendo esconder erros que podem levar a decisões gerenciais incorretas. Os atuais modelos de projeto, independente da tecnologia utilizada (eventos, estados, dinâmica) são muito complexos e, embora algum trabalho tenha sido realizado para aproximá-los das técnicas tradicionais de gerenciamento de projetos, seus resultados parecem exigir muito conhecimento do usuário em relação às técnicas de modelagem. Os modelos formais de projetos de software devem ser legíveis e simples. Acreditamos que uma abordagem gerativa, onde as informações são capturadas em uma linguagem mais próxima dos conceitos do domínio (no caso, elementos envolvidos na gerência de projetos de software) e os modelos são gerados a partir destas informações, possa auxiliar na construção dos modelos, transferindo parte da complexidade do processo de construção dos modelos para os responsáveis pela definição da linguagem de modelagem para um determinado domínio;
- Representação detalhada: modelos formais de projeto de software devem ser capazes de armazenar detalhes sobre cada componente do projeto, como seus desenvolvedores, suas tarefas e seus recursos. Os modelos da Dinâmica de Sistemas sofrem particularmente de limitações em relação a este requisito, tendendo a modelar as informações acerca dos componentes de um modelo utilizando médias. Modelos baseados em eventos e diagramas de estado são mais sensíveis a este requisito, modelando as informações acerca de suas entidades através de atributos. A representação detalhada traz ainda um *trade-off*: por um lado, o modelo deve ser legível, conforme especificado no requisito anterior; por outro, ele não pode ser simplista, ignorando os detalhes dos componentes do projeto. Acreditamos que a definição de linguagens de modelagem para domínios, citada no requisito anterior, também pode auxiliar na determinação de um ponto de equilíbrio neste antagonismo. Teríamos então uma descrição complexa para a linguagem de modelagem de um domínio, cujo modelo deve ser legível apenas para o reduzido grupo responsável por sua manutenção e evolução, e modelos mais simples, descritos com base nos

construtores definidos na linguagem de domínio e legíveis para o público maior de potenciais desenvolvedores de modelos para este domínio;

- Representação de conhecimento: é necessário que os modelos tenham capacidade de representar uma ampla gama de conhecimento sobre gerência de projetos de software. Este conhecimento deve ser independente do modelo de um projeto específico, devendo ser aplicável em diversos projetos desenvolvidos por uma organização. O conhecimento deve ser representado em modelos complementares, que possam ser integrados a um modelo principal, este último descrevendo as particularidades de um projeto. Embora diversos modelos de projeto apresentados neste capítulo tenham apresentado o requisito de integração de modelos complementares, especialmente os modelos baseados em estados e na Dinâmica de Sistemas, as técnicas de integração parecem não oferecer auxílio automatizado à integração, exigindo que o responsável inspecione e altere manualmente as equações componentes do modelo;
- Representação de incertezas: o modelo deve permitir a representação de incertezas acerca dos elementos envolvidos em um projeto de software. Pelo menos os dois tipos de incerteza apresentados na seção 2.5 devem ser abordados pelo modelo. Alguns modelos apresentados neste capítulo abordaram a representação de incertezas acerca dos valores de seus parâmetros. Entretanto, poucos modelos fornecem suporte à representação de incertezas acerca da ocorrência de eventos no futuro.

Nenhum dos modelos apresentados neste capítulo cumpre todos estes requisitos, A Tabela 2.2 apresenta um resumo da discussão dos parágrafos anteriores, indicando o grau em que cada tipo de modelo cumpre os requisitos apresentados.

	Eventos Discretos	Diagramas de Estado	Dinâmica de Sistemas
Fac. Desenvolvimento	parcial	parcial	parcial
Detalhamento	✓	✓	
Rep. Conhecimento	parcial	parcial	parcial
Rep. Incerteza	parcial	parcial	parcial

Tabela 2.2 – Grau em que os tipos de modelos atendem aos requisitos

A facilidade de desenvolvimento é uma medida relativa. Assumimos que as técnicas de modelagem atual estão em um estágio intermediário com relação a este requisito. Assim, embora diversos modelos tenham sido construídos com sucesso utilizando estas técnicas, acreditamos que uma representação mais próxima dos

elementos que formam o domínio do problema a ser modelado possa facilitar a construção de modelos.

Em relação à capacidade de detalhamento, observamos que as técnicas de modelagem através de eventos discretos e diagramas de estado são, geralmente, capazes de representar mais detalhes acerca das informações dos elementos que formam um domínio de problema do que os modelos construídos com a Dinâmica de Sistemas. Isto ocorre porque, em geral, os modelos da Dinâmica de Sistemas tendem a representar as características de uma categoria de elementos através de uma média, ao invés de descrever precisamente a característica de cada elemento pertencente à categoria.

Assim como no requisito que avalia a facilidade de desenvolvimento de modelos, classificamos as três técnicas em um nível intermediário quanto a sua capacidade de representação de conhecimento. Isto ocorre porque diversos modelos foram construídos com estas técnicas, cada qual representando algum conhecimento sobre a gerência de projetos de desenvolvimento de software. Entretanto, nenhuma das três técnicas oferece mecanismos para que o conhecimento representado em um modelo possa ser expandido através da construção de modelos complementares. Modelos construídos através de qualquer uma das três técnicas devem ser alterados sempre que a representação de um novo conhecimento se faz necessária.

Finalmente, na avaliação do último requisito, as três técnicas permitem a representação de incertezas acerca de valores em seus modelos. Entretanto, não existe suporte definido nestas técnicas para a representação de eventos incertos. O modelo deve ser alterado sempre que um novo evento incerto deve ser avaliado ou deve prever todos os possíveis eventos incertos que podem ser avaliados no futuro. Idealmente, os eventos incertos seriam representados através de modelos complementares, integrados ao modelo original para sua avaliação.

Os requisitos descritos nesta seção estabelecem a base para a proposta desta tese. Procuramos atingir estes requisitos através de uma representação para o conhecimento gerencial baseada em modelos, que é apresentada no Capítulo 4. Neste capítulo e em capítulos posteriores, discutimos como esta representação pode ser utilizada na gerência operacional de projetos de software, analisamos a extensão em que os requisitos foram atendidos e as limitações da proposta.

3 Análise de Riscos no Gerenciamento de Projetos de Software

A análise de riscos é uma das atividades integrantes do planejamento e controle de projetos de desenvolvimento de software. Os responsáveis por esta atividade procuram antecipar os problemas potenciais que podem ocorrer ao longo do processo de desenvolvimento, avaliando seu impacto sobre o projeto e formulando planos para evitar sua ocorrência ou reduzir seu impacto.

A palavra **risco** deriva do termo italiano *risicare*, que significa "a descobrir". Seu tratamento científico teve início no século XVI, durante a Renascença (HALL, 1998). Genericamente, risco é definido como a possibilidade de perigo, perda ou prejuízo (FERREIRA, 1993). Em uma definição especulativa, risco é definido como a possibilidade de perda ou ganho. A principal diferença entre as duas definições está na consideração do risco como uma fonte de oportunidade.

No contexto do desenvolvimento de software, CONROW e SHISHIDO (1997) definem risco como a probabilidade de um projeto não atingir objetivos particulares de custos, desempenho[1] e cronograma, e as conseqüências de não atingir estes objetivos. HALL (1998) define risco em software como uma medida da probabilidade e das perdas acarretadas pelo acontecimento de um evento que afete negativamente o projeto, o processo ou o produto em um desenvolvimento de software. Em qualquer circunstância, um risco sempre está associado a uma incerteza: se não existe incerteza, não existe risco. A medida em que as incertezas se concretizam, os riscos diminuem.

Um processo de análise de riscos transforma um conjunto de incertezas e dúvidas existentes em um projeto em riscos aceitáveis, com base no conhecimento existente sobre o projeto. HALL (1998) define a análise de riscos como um conjunto de procedimentos que, quando aplicados a um risco específico, fazem com que as conseqüências de sua concretização sejam aceitáveis em qualquer circunstância. GOLDMAN (1998) define a análise de riscos como um conjunto completo de normas e procedimentos que as organizações utilizam para gerenciar, monitorar e controlar sua exposição aos riscos.

As técnicas de análise de riscos foram enfatizadas no contexto de desenvolvimento de software no final da década de 1980, com a definição do modelo de ciclo de vida espiral de BOEHM (1988), no qual um projeto de software é

---

[1] *Performance*

implementado em ciclos. Cada ciclo acrescenta novas funcionalidades ao produto desenvolvido no ciclo anterior. No início de cada ciclo, a análise de riscos, realizada com base nos requisitos iniciais do sistema ou na aceitação de uma versão pelos usuários, define se um novo ciclo de desenvolvimento deve ser realizado.

Ao longo da década de 1990, o trabalho de Boehm foi estendido em diversas direções, passando pela criação de novos processos de análise de riscos qualitativa (CHARETTE, 1989, KAROLAK, 1996, KONTIO, 1997, HALL, 1998), processos de análise de riscos quantitativa (FAIRLEY, 1994, GREY, 1995), processos baseados em conhecimento (MADACHY, 1997) e repositórios de informação sobre riscos (CARR et al., 1993, GARVEY et al., 1997).

Em meados da década de 1990, o departamento de defesa americano reuniu os mais conceituados gerentes de projetos da indústria de software americana com o objetivo de definir linhas de pesquisa para o aprimoramento das técnicas de gerenciamento de projetos de software (BROWN, 1996). O grupo redigiu um documento explicitando os fatores encontrados em grande parte dos projetos bem sucedidos e um guia com nove princípios de gerenciamento. A análise de riscos figura como primeiro princípio neste documento, em conjunto com inspeções formais, gerenciamento centrado na equipe de desenvolvimento, acompanhamento de projeto baseado em métricas, controle de qualidade de granularidade fina, entre outros.

A relevância do gerenciamento de riscos para o sucesso de projetos de desenvolvimento de software é ainda enfatizada por sua presença em diretrizes de qualidade no gerenciamento de projetos (ISO/IEC, 1999; ABNT, 2000) e processos padrão para este gerenciamento (PMI, 1996).

Este capítulo fornece um panorama do estado da arte na área de análise de riscos em projetos de desenvolvimento de software. Inicialmente, na seção 3.1, apresenta-se diversas categorias de risco encontradas na literatura de Engenharia de Software. Na seção 3.2, são descritas as etapas de um processo genérico de análise de riscos, ressaltando as abordagens particulares de diversos processos encontrados na literatura. A seção 3.3 conclui o capítulo, apresentando um resumo das principais características dos processos discutidos na seção 3.2.

3.1 Categorias de Risco em Software

Diversas classificações de riscos que podem afetar projetos de desenvolvimento de software foram propostas na literatura de Engenharia de Software. Uma breve análise

de um conjunto de categorias de riscos torna evidente a diversidade de fontes de incerteza que permeia um projeto de software. Esta diversidade dificulta a análise de riscos, exigindo a criação de processos e ferramentas para viabilizar esta tarefa.

PRESSMAN (2000) classifica os riscos de um projeto de desenvolvimento de software em três grupos: riscos de projeto, riscos de negócio e riscos técnicos. Os riscos de projeto se referem a equipe de desenvolvimento, custos, cronograma, recursos, relações com clientes e problemas com os requisitos do software. Os riscos de negócio se referem a fatia de mercado esperada para o produto (*market share*), a mudanças nas estratégias de mercado da empresa, a falta de apoio do alto gerenciamento e a falta de apoio financeiro ao projeto. Finalmente, os riscos técnicos se referem a problemas que possam vir a ocorrer durante o processo de desenvolvimento do software, como desempenho inadequado, um alto número de erros por artefato, a incapacidade da equipe de desenvolvimento definir um algoritmo para a resolução de um problema, entre outros.

Projeto	Negócio	Técnicos
Equipe	*Market share*	Processo de desenvolvimento
Custos	Mudança de estratégia de mercado	
Cronograma	Falta de apoio gerencial	
Recursos	Falta de apoio financeiro	
Requisitos		
Relações com clientes		

Tabela 3.1 - As categorias de risco definididas por PRESSMAN (2000)

KAROLAK (1996) divide os riscos de um projeto de software em duas categorias: riscos estratégicos e riscos operacionais. Os riscos estratégicos estão relacionados com o negócio da organização de desenvolvimento, contemplando os riscos de mercado, riscos financeiros, riscos de pessoal e riscos de produção. Riscos operacionais se referem aos problemas técnicos, como cronograma, custos e alocação de recursos, tais como a equipe de desenvolvimento, locais físicos, equipamentos e ferramentas. Os riscos técnicos estão relacionados aos aspectos de capacidade de implementação da funcionalidade desejada, qualidade, confiabilidade, facilidade de utilização, desempenho e reusabilidade. Riscos de cronograma, custos e recursos estão relacionados com a realização do projeto com recursos limitados, seja de tempo, financeiro ou outros recursos necessários para o desenvolvimento. O autor aponta para o fato de que as áreas de risco estratégico e operacional não podem ser avaliadas em conjunto sem a definição de uma ponte entre elas. Esta ponte é criada por uma

metodologia holística de controle de risco que analise o processo de desenvolvimento sob as perspectivas da técnica e do negócio.

Estratégicos	Operacionais	
Mercado	Técnicos	Projeto
Financeiro	Funcionalidade	Cronograma
Pessoal	Qualidade	Custos
Produção	Confiabilidade	Alocação de recursos
	Utilizabilidade	
	Desempenho	
	Reusabilidade	

Tabela 3.2 - As categorias de risco definidas por KAROLAK (1996)

Tendo como prioridade os riscos técnicos e de projeto, HYATT e ROSEMBERG (1996) identificam um conjunto de riscos comuns a diversos projetos. Estes riscos compreendem o número de erros latentes em um software, sua confiabilidade, sua manutenibilidade, sua reusabilidade e a capacidade do software ser desenvolvido dentro do tempo especificado.

CONROW e SHISHIDO (1997) identificam cinco grupos de risco no contexto do desenvolvimento de software: riscos de projeto, riscos de atributos de projeto, riscos de gerenciamento, riscos de engenharia e riscos de ambiente de trabalho. Os riscos de projeto se referem à estabilidade dos requisitos, ao grau de envolvimento do usuário no processo de desenvolvimento e a correção das estimativas de complexidade do projeto. Os riscos de atributos de projeto se referem a viabilidade de realização do projeto com relação as limitações de qualidade, cronograma e custos. Os riscos de gerenciamento se referem a eficácia da gerência do projeto em seus diversos níveis. Os riscos de engenharia se referem as técnicas utilizadas no desenvolvimento, aos riscos de especificação e implementação. Os riscos de ambiente de trabalho se referem a capacitação da equipe nas técnicas utilizadas, ao treinamento e aos critérios de seleção de ferramentas, métricas e planos de trabalho.

Projeto	Atributos de Projeto	Gerenciamento
Estabilidade dos requisitos	Qualidade	Eficiência da gerência
Grau de envolvimento do usuário	Cronograma	
Estimativas de complexidade	Custos	
Engenharia	**Ambiente**	**Outros**
Técnicas de desenvolvimento	Conhecimento das técnicas	Problemas legais
Dificuldades de especificação	Treinamento	Subcontratação
Dificuldades de implementação	Ferramentas	Documentação
	Métricas	Subestimação de riscos
	Planos de trabalho	Subestimação de complexidade

Tabela 3.3– As categorias de risco definidas por CONROW e SHISHIDO (1997)

Além destas cinco categorias, Conrow e Shishido apontam uma lista complementar de possíveis origens de risco em projetos de média e alta complexidade, que envolvem a possibilidade de problemas legais, de subcontratação, de documentação, subestimação de riscos, de complexidade e de custos.

Para facilitar a identificação de riscos em projetos de desenvolvimento de software, o SEI - *Software Engineering Institute* – definiu uma taxonomia para identificação de riscos. A taxonomia, apresentada na Tabela 3.4, provê uma estrutura para identificação de riscos técnicos e programáticos no desenvolvimento de software (CARR et al., 1993). A taxonomia é dividida em três categorias: produto, ambiente e organização. Os riscos de produto envolvem os aspectos técnicos do trabalho a ser realizado. Os riscos de ambiente envolvem os métodos, ferramentas e técnicas utilizadas no projeto. Os riscos de organização envolvem as restrições externas a que o projeto está sujeito, sejam elas contratuais, organizacionais ou operacionais.

Produto	Ambiente	Organização
1. Requisitos	**1. Processo de Desenvolvimento**	**1. Recursos**
Estabilidade	Formalismo	Cronograma
Completitude	Aplicabilidade	Equipe
Clareza	Controle do processo	Recursos financeiros
Validação	Familiaridade	Ferramentas
Viabilidade	Controle do produto	
Precedência		
Escala		
2. Projeto	**2. Sistema de Desenvolvimento**	**2. Interfaces do Projeto**
Viabilidade	Capacidade	Clientes
Dificuldade	Aplicabilidade	Contratados associados
Interfaces	Utilizabilidade	Subcontratados
Desempenho	Familiaridade	Apoio do alto gerenciamento
Testabilidade	Confiabilidade	Vendedores
Restrições de hardware	Suporte	Política
Reusabilidade	Capacidade de instalação	
3. Implementação e Testes	**3. Gerenciamento**	**3. Contrato**
Viabilidade	Planejamento	Tipo de contrato
Capacidade de teste de unidade	Organização do projeto	Restrições ao desenvolvimento
Capacidade de implementação	Experiência em gerenciamento	Dependências de terceiros
	Interfaces do projeto	
4. Integração e Testes	**4. Métodos de Gerenciamento**	
Ambiente	Monitoração	
Produto	Gerenciamento de pessoal	
Sistema	Verificação de qualidade	
	Gerenciamento de configuração	
5. Especializações	**5. Ambiente de Trabalho**	
Manutenibilidade	Atitude em relação à qualidade	
Confiabilidade	Cooperação	
Segurança	Comunicação	
Fatores humanos	Moral	

Tabela 3.4 – Taxonomia de risco do Software Engineering Institute (CARR et al., 1993)

Cada categoria de riscos contém um conjunto de elementos que, por sua vez, são divididos em atributos. Os atributos servem como guia para formulação de um questionário para identificação de riscos, o TBQ – *Taxonomy Based Questionnaire.* Este questionário é especializado para cada projeto, removendo-se as perguntas referentes a atributos que não são relevantes no contexto do projeto (CARR et al., 1993). O TBQ está organizado em dois níveis. O primeiro nível contém perguntas abrangentes, cujas respostas selecionam perguntas mais específicas do segundo nível. Este filtro permite o foco nos aspectos relevantes de um projeto, sem perda de visão em suas outras características. As perguntas do questionário contêm listas de exemplos que definem seu escopo. Estes exemplos facilitam a comunicação com o usuário, reduzindo a chance de interpretação incorreta das perguntas.

FAIRLEY (1994) identifica um conjunto de fatores de risco relacionados com o desenvolvimento baseado em componentes comerciais *off-the-shelf.* Estes fatores envolvem dificuldades com a integração de componentes, devido a diferentes formatos de dados e protocolos, diferenças entre versões do componente, inexistência de código fonte, dificultando adaptações do componente, e problemas na produção do componente, por exemplo, por falência do vendedor.

HALL (1998) separa os riscos de desenvolvimento de software em três categorias: riscos de projeto, riscos de processo e riscos de produto. Os riscos de projeto compreendem problemas contratuais, relações com fornecedores, restrições de recursos e de interfaces externas à organização de desenvolvimento. Os riscos de processo compreendem problemas de planejamento, equipe, acompanhamento do projeto, controle de qualidade e gerenciamento de configuração. Os riscos de produto estão relacionados com a estabilidade dos requisitos, desempenho, complexidade e confiabilidade. Hall utiliza estas categorias para dividir as responsabilidades por cada risco dentro de um projeto. Assim, os riscos de projeto são delegados para a equipe de gerenciamento, enquanto os riscos de produto são delegados para a equipe de desenvolvimento. Os riscos de processo são compartilhados pelas duas equipes.

MOYNIHAN (1997) realizou um conjunto de entrevistas com gerentes de projeto de pequenas e médias organizações de desenvolvimento de software, identificando os fatores de risco que os gerentes consideram no planejamento de um novo projeto. O conjunto de entrevistas resultou em uma lista de 113 fatores, classificados em 23 categorias. Um importante resultado do trabalho de Moynihan partiu da comparação de seus fatores de risco com a taxonomia de risco do SEI (CARR et al., 1993). A

comparação comprovou que alguns fatores identificados nas entrevistas não possuem correlatos entre os atributos do SEI, especialmente fatores referentes ao conhecimento do usuário no domínio da aplicação e ao controle do projeto. Moynihan justifica as diferenças entre as taxonomias pelos contextos distintos em que os estudos foram realizados. Existindo estas diferenças, a criação de uma relação de fatores de risco única e completa para qualquer domínio de aplicação é provavelmente irrealista.

Projeto	Processo	Engenharia
Riscos contratuais	Equipe	Estabilidade dos requisitos
Relações com fornecedores	Planejamento	Desempenho
Recursos	Acompanhamento	Complexidade
Interfaces externas	Controle de qualidade	Confiabilidade
	Gerenciamento de configuração	

Tabela 3.5 – As categorias de risco definidas por HALL (1998)

As categorias de riscos servem como referência para os riscos que podem ser encontrados em um projeto de software e como um alerta para a diversidade destas incertezas. MOYNIHAN (1997) mostra a dependência existente entre os riscos considerados relevantes por gerentes e os contextos em que se realizam seus projetos. Sendo o contexto definido pelos elementos envolvidos no projeto, tais como seu domínio de aplicação, as tecnologias utilizadas, as características da equipe, entre outros, os riscos que podem afetar um projeto também devem ser associados a estes elementos. Esta associação, além de servir como classificação, pode auxiliar na reutilização dos riscos, determinando os riscos relevantes para um projeto a partir dos elementos que o compõem.

3.2 Processos de Análise de Riscos

CHARETTE (1996) identifica fatores que facilitam ou inibem a realização de análise de riscos em projetos de desenvolvimento de software. Os principais fatores catalisadores são o reconhecimento do valor do software para o crescimento da organização onde este será implantado, o volume de material teórico e prático publicado a respeito de análise de riscos em software, a alta correlação existente entre projetos que praticam técnicas de análise de riscos e projetos bem sucedidos e a existência de regulamentos de clientes específicos, como o departamento de defesa americano, exigindo a prática de análise de riscos em seus projetos.

Os principais fatores que inibem a utilização de técnicas de análise de riscos são a aversão a risco, que pode ser pessoal ou cultural em uma organização, e a aplicação de

técnicas demasiadamente simplificadas de análise de riscos. GEMMER (1997) analisa os aspectos culturais de aversão a risco, apresentando os problemas de comportamento enfrentados durante a implantação de técnicas de análise de riscos em uma empresa.

CHARETTE (1996) observa que as técnicas de análise de riscos são tratadas de forma simplista em diversos projetos, o que resulta na subestimação do número, impacto e severidade dos riscos. Esta subestimação facilita a concretização dos riscos, aumentando a probabilidade de insucesso do projeto e reduzindo a credibilidade das técnicas de análise de riscos. Para evitar este efeito, a análise de riscos deve seguir padrões claramente definidos na forma de um processo.

Geralmente, os processos de análise de riscos são compostos por quatro etapas (FAIRLEY, 1994, WILLIANS et al., 1997, HALL, 1998; ABNT, 2000): a identificação, a avaliação, o planejamento e o controle de riscos. A identificação de riscos explora o conhecimento, as dúvidas e as incertezas existentes em um projeto, identificando os riscos pertinentes ao projeto. A avaliação de riscos agrega informações sobre a possibilidade de ocorrência e o impacto provocado pelos riscos identificados para o projeto. O planejamento de riscos define mecanismos para a percepção da ocorrência de riscos, planos de contenção e contingência para os riscos considerados significativos. O controle de riscos acompanha os mecanismos de percepção previamente definidos, implantando os planos de contenção e contingência e avaliando sua eficácia no controle dos riscos.

Nas próximas subseções discutimos diversos processos de análise de riscos apresentados na literatura, considerando as técnicas utilizadas por estes processos nas etapas acima descritas.

3.2.1 Identificação de Riscos

A identificação de riscos é a primeira etapa do processo de análise de riscos. Esta etapa tem como objetivo listar os riscos que podem afetar um projeto a partir das dúvidas, das incertezas e do conhecimento existente sobre o projeto. A lista de riscos, acompanhados por sua descrição, é o produto gerado por esta etapa. Estes riscos podem ser identificados por uma variedade de técnicas, entre elas:

- Listas de Quesitos: são utilizadas em diversos processos de análise de riscos (CARR et al., 1993, KÄNSÄLÄ, 1997, HALL, 1998) como guias para a identificação de riscos. Diversos autores (GREY, 1995, MOYNIHAN, 1997) alertam que nenhuma

lista de quesitos[2] é completa, portanto, a utilização de listas de quesitos não garante que todos os riscos de um projeto podem ser identificados. As características específicas de um projeto podem não estar capturadas por nenhuma lista de quesitos. Um percurso pelas diversas categorias de risco identificadas na literatura demonstra este fato;

- Questionários: uma extensão das listas de quesitos, os questionários definem uma abordagem para a identificação de riscos, podendo ser utilizados como guias de entrevistas. Diferente das listas de quesitos, questionários formulam questões objetivas sobre os fatores de risco. As listas de quesitos oferecem apenas guias para discussão de riscos, sem oferecer um método para esta discussão. Questionários são utilizados em diversos processos de análise de riscos (BOEHM, 1989, KAROLAK, 1996);

- Fatores de sucesso: a identificação dos fatores que podem levar um projeto ao sucesso permite a identificação dos eventos que possam comprometer este sucesso. Estes eventos caracterizam os riscos do projeto. O processo de análise de riscos de GREY (1995) possui etapas para a identificação dos patrocinadores e dos fatores de sucesso do projeto. Em relação à importância destas etapas, Grey afirma que é crucial reconhecer as expectativas das pessoas que darão suporte ao projeto e que somente através da realização destas expectativas o projeto pode atingir sucesso;

- Geradores de custos: motivados pelos constantes erros nas estimativas de custo de projetos, alguns autores criaram técnicas de identificação de riscos a partir de geradores de custos. MADACHY (1997) apresenta uma técnica de identificação e avaliação de riscos baseada em padrões de respostas ao questionário de avaliação dos geradores de custos do método COCOMO2 (BOEHM et al., 1995). O princípio desta técnica é que uma situação de risco pode ser descrita como uma combinação de valores extremos para um conjunto de geradores de custo, indicando que uma margem de esforço foi reservada para a resolução de problemas potenciais nesta área. O autor argumenta que as práticas de análise de riscos podem ser aprimoradas, explorando-se o conhecimento utilizado durante o processo de estimação de custos;

- Reuniões e entrevistas: exploram o conhecimento no domínio da aplicação sendo desenvolvida para identificar as possíveis falhas em um projeto. Reuniões de

[2] Em inglês, *checklists*.

identificação de riscos devem prosseguir ao longo de todo o desenvolvimento, reavaliando os riscos previamente identificados e capturando novos riscos;

- Revisões: revisões de produtos e processos podem auxiliar na identificação dos riscos referentes a eles, além dos riscos que residem em sua integração com demais componentes do projeto;
- *Brainstorming*: os riscos referentes a um projeto podem ser elucidados pela combinação e organização das idéias originárias de uma sessão de *brainstroming*. Uma vantagem desta abordagem reside na forma não-estruturada com que as idéias podem ser apresentadas. Assim, um participante que contemple um risco, mas não consiga descrevê-lo de forma estruturada, pode ser auxiliado pelos demais participantes da sessão;
- Protótipos: protótipos podem ser utilizados para exploração de áreas críticas de um projeto. Esta exploração pode ser fonte de informações para a identificação de riscos nos componentes analisados e em suas integrações com outros elementos do sistema.

FAIRLEY (1994) alerta que, na identificação de riscos, o analista deve distinguir os riscos dos sintomas gerados por eles. Por exemplo, um atraso de cronograma pode ser um sintoma de dificuldades técnicas ou de falta de recursos. Neste exemplo, o risco não é o atraso de cronograma, mas os fatos que poderão gerar este atraso – dificuldades técnicas e recursos insuficientes.

Esta constatação é relevante visto que as causas e efeitos podem estar distantes no tempo em um projeto de software. Enquanto uma análise superficial pode revelar aparentes causas para a ocorrência de um problema, a verdadeira causa pode estar obscurecida por uma série de transformações ocorridas ao longo do processo de desenvolvimento. Acreditamos que a formalização das relações entre os diversos elementos componentes de um projeto de software pode elucidar estas relações de causalidade, permitindo a identificação e, eventualmente, o tratamento das verdadeiras causas dos riscos envolvidos com o desenvolvimento de software.

3.2.2 Representação de Riscos

Após sua identificação, os riscos devem ser documentados. Geralmente, esta documentação ocorre através de formulários que, quando automatizados, podem formar um banco de dados sobre riscos. WILLIANS et al. (1997), GALLAGHER et al. (1997)

e HALL (1998) apresentam exemplos de formulários para identificação e acompanhamento de riscos em projetos de software. O preenchimento destes formulários começa na etapa de identificação, prosseguindo ao longo de todo o processo de análise de riscos: cada nova etapa acrescenta novas informações ao formulário.

Padrões também são utilizados para a documentação de riscos. Um padrão é uma estrutura de dados padronizada que identifica, documenta, avalia e propõe soluções para um problema recorrente ao longo de diversas aplicações. Embora originalmente propostos para documentar soluções para as atividades de projeto (GAMMA et al., 1994), os padrões foram utilizados em diversas outras atividades do processo de desenvolvimento de software. Sua principal vantagem reside na estrutura comum, que facilita a leitura, a comunicação e a disseminação de soluções sobre os riscos entre os participantes do projeto. JONES (1994) apresenta uma estrutura de dados padronizada para a documentação de riscos, oferecendo diversos exemplos de riscos documentados segundo esta estrutura.

A representação dos riscos identificados para um projeto de software ainda é um tema de pesquisa: quase todos os processos de análise de riscos utilizam documentação textual para descrever seus riscos. Em alguns processos, dados geralmente qualitativos sobre a probabilidade de ocorrência ou o impacto do risco complementam esta informação. Entretanto, estes dados carregam um alto grau de subjetividade e não permitem uma análise agregada do impacto de diversos riscos acontecendo sobre o mesmo projeto. Percebemos a necessidade de melhores mecanismos de representação dos riscos, formalizando as relações entre os diversos elementos componentes de um projeto de software e as incertezas acerca destes elementos. Estes mecanismos devem permitir uma avaliação mais completa sobre seu impacto isolado ou em combinações sobre um projeto.

3.2.3 Reutilização de Riscos

KONTIO (1997) apresenta o processo de análise de riscos **RiskIt**, no qual os riscos são modelados através de cenários, que relacionam eventos, fatores e efeitos de risco. Um evento de risco representa a ocorrência de um evento que provoca um efeito negativo em um projeto. Um fator de risco é uma característica que afeta a probabilidade de ocorrência deste evento. Um efeito de risco representa o impacto do

evento no projeto, medido como uma perda nas expectativas dos patrocinadores[3] do projeto. Estas expectativas são previamente modeladas como funções de utilidade (FRENCH, 1989) para cada patrocinador.

GARVEY et al. (1997) apresentam a ferramenta **RAMP,** que mantém uma base de dados relacionando os projetos realizados em uma organização de desenvolvimento, os riscos referentes a estes projetos e as respectivas estratégias de controle aplicadas. RAMP auxilia a identificação dos riscos de um novo projeto, permitindo a análise de projetos similares. A ferramenta opera em uma *intranet*, permitindo o acesso de desenvolvedores em diferentes locais geográficos. Os riscos são documentados através de formulários padrão, sendo apresentados no formato de páginas Web. RAMP se baseia em agentes "inteligentes" para a recuperação de informações e para a manutenção da base de dados. Seu agente de busca permite que o analista de risco especifique os atributos de seu projeto, identificado projetos similares através de um algoritmo de comparação de padrões. A manutenção da base de dados sobre riscos pode ser realizada através de um agente, que verifica periodicamente a base, requisitando atualizações dos responsáveis por cada projeto.

Os processos de análise de riscos de KONTIO (1997) e GARVEY et al. (1997) possuem em comum a ênfase dedicada à reutilização de riscos. Consideramos que este tipo de reutilização é muito importante para o sucesso dos projetos: as informações sobre os riscos que podem afetar um projeto, assim como qualquer conhecimento relacionado com o desenvolvimento de software, é um artigo de suma importância, que as organizações de desenvolvimento de software devem estar preparadas para reutilizar. Os processos de análise de riscos devem oferecer suporte para este tipo de reutilização. KONTIO e BASILI (1996) apresentam a reutilização de cenários de risco apoiada pela *Experience Factory*, proposta por BASILI et al. (1992), enquanto os agentes "inteligentes" oferecem suporte à reutilização de riscos na ferramenta RAMP (GARVEY et al., 1997). Entretanto, estas abordagens são limitadas em sua capacidade de avaliação dos riscos reutilizados no contexto de um novo projeto, conforme será abordado na próxima subseção.

---

[3] *Stakeholders*, em inglês.

3.2.4 Avaliação de Riscos

A etapa de avaliação de riscos determina a possibilidade de concretização de cada risco previamente identificado, assim como o impacto de sua ocorrência no processo ou nos produtos desenvolvidos. Esta etapa recebe como entrada a lista de riscos identificados para o projeto, produzindo uma lista com informações sobre a probabilidade de ocorrência e o impacto dos riscos.

A possibilidade de ocorrência de um risco define sua chance de concretização em um determinado projeto. O impacto de um risco determina a perda potencial no produto ou no processo de desenvolvimento após a concretização de um risco. Exemplos destes fatores incluem os custos do projeto, o tempo para conclusão, os níveis de desempenho, a reusabilidade dos componentes desenvolvidos durante o projeto, entre outros.

3.2.4.1 Medidas de Riscos

A possibilidade de ocorrência e o impacto de um risco podem ser medidas quantitativas ou qualitativas. Medidas qualitativas assumem a forma de termos de linguagem natural ou valores discretos dentro uma escala com conotação subjetiva. Para facilitar a análise destas medidas, os termos de linguagem natural utilizados podem ser catalogados em um dicionário, limitando o universo de valores da medida. Este universo pode variar desde medidas booleanas, onde apenas dois termos são válidos ("sim" e "não"), até medidas mais refinadas, com termos intermediários (como "moderado", "pouco provável" e "muito provável"). O uso da linguagem natural facilita a expressão e a comunicação do conhecimento da equipe de desenvolvimento, visto que sua experiência não está expressa em números (PRESSMAN, 2000).

Técnicas de quantificação são mecanismos de mapeamento de termos qualitativos em informações quantitativas. Estas técnicas podem ser utilizadas para unir as vantagens das abordagens quantitativas e qualitativas. Assim, informações qualitativas podem ser utilizadas durante a identificação e o início da avaliação de riscos, sendo convertidas para informações quantitativas posteriormente. Exemplos de técnicas de quantificação incluem o mapeamento de termos de linguagem natural em intervalos numéricos ou números fuzzy (ZADEH, 1988).

Intervalos de conversão são faixas de números, que podem ser associados a cada termo de linguagem natural utilizado na avaliação de riscos. Sempre que estes termos forem utilizados, eles podem ser quantificados para o elemento central de seu intervalo

de conversão. Termos modificadores, geralmente advérbios (muito, pouco, entre outros), podem ser utilizados junto aos termos de linguagem natural para que a quantificação tenda para um extremo do intervalo.

Números fuzzy também podem ser utilizados para converter termos de linguagem natural em informações quantitativas. Operadores e técnicas de agregação fuzzy (BARDOSSY et al., 1993) podem ser utilizados para manipular estes números. HAPKE et al. (1994) apresentam uma abordagem de análise de riscos que utiliza números fuzzy para quantificação de termos qualitativos.

Riscos avaliados quantitativamente envolvem incertezas acerca de elementos do mundo real, como tempo ou dinheiro. Estes riscos podem ser modelados e avaliados em termos numéricos, sendo medidos a partir de informações dos produtos ou processos de desenvolvimento. Medidas quantitativas são expressas na forma de números, funções ou expressões matemáticas, que determinam a probabilidade de ocorrência de um evento ou condição. A avaliação quantitativa de riscos é particularmente interessante quando existem informações quantitativas de projetos passados. Através de procedimentos estatísticos, as medidas podem ser extraídas destes dados, dissociando-se a análise de riscos de opiniões pessoais. Entretanto, a insistência na quantificação de algumas categorias de riscos pode levar a paralisia da análise e a dificuldades de comunicação do risco. Alguns autores (WILLIANS et al., 1997) defendem inclusive que nem todas as categorias de risco, como riscos legais ou políticos, podem ser medidas quantitativamente, devendo ser expressas de forma qualitativa.

Observa-se na literatura a existência de duas classes de processos de análise de riscos: os processos qualitativos e os processos quantitativos. A diferença fundamental entre estes processos reside na forma com que os riscos de um projeto são avaliados. Entretanto, uma diferença de escopo também pode ser observada entre estas classes de processos. Em geral, processos qualitativos (CARR et al., 1993, HALL, 1998) possuem amplo escopo, avaliando subjetivamente diversos aspectos do desenvolvimento de software, cobrindo desde aspectos técnicos até aspectos políticos. Por outro lado, os processos quantitativos trabalham em escopo reduzido, definindo modelos sofisticados para atividades específicas do gerenciamento de projetos (GREY, 1995, FAIRLEY, 1994), como a modelagem do tempo de desenvolvimento, custo e consumo de recursos.

Poucos processos de análise de riscos reúnem as abordagens qualitativa e quantitativa. Alguns destes processos quantificam as informações subjetivas mapeando-as em intervalos numéricos e aplicando operações aritméticas sobre estes intervalos

(KAROLAK, 1996). Outros processos quantificam o impacto da concretização de riscos em unidades do mundo real, enquanto os riscos são avaliados qualitativamente (KONTIO, 1997).

Técnicas qualitativas são geralmente guiadas por questionários ou estudos dirigidos. GREY (1995) observa que estas técnicas evitam que o analista esqueça de riscos óbvios, mas não suportam a medição da possibilidade de ocorrência e do impacto dos riscos. Segundo o autor, os questionários podem limitar o universo de análise, sendo úteis apenas em domínios estáveis e repetitivos.

Técnicas de ranqueamento permitem a medição dos riscos, que são avaliados a partir de respostas a questionários dadas em determinadas escalas. As respostas são agregadas, através de processos de quantificação, gerando um termo que representa o grau de risco de um projeto. GREY (1995) defende que estes processos de quantificação, geralmente simples, não são capazes de representar as relações potencialmente complexas existentes entre os riscos e os componentes de um projeto de desenvolvimento de software.

Em seu processo de análise de riscos, KAROLAK (1996) utiliza árvores baesianas de três níveis para calcular o risco de um projeto, baseado nas respostas de um questionário com 83 perguntas. Cada pergunta do questionário é respondida quantitativamente por um número entre 0 e 1. As respostas são agregadas em dez fatores de risco, que geram impacto em três categorias de risco: riscos técnicos, riscos de custos e riscos de cronograma.

O nível mais interno da árvore de avaliação de riscos calcula a chance de sucesso em um fator de risco, que é definida como a probabilidade de não ocorrerem problemas referentes ao fator durante o processo de desenvolvimento. As arestas deste nível da árvore possuem pesos iguais e o risco em um fator é calculado pela média das respostas referentes a ele no questionário. No segundo nível da árvore, as probabilidades de sucesso nas categorias de risco são calculadas a partir das probabilidades de sucesso nos fatores. Os pesos das arestas deste nível são definidos por KAROLAK (1996), de acordo com a influência dos fatores nas categorias de risco. Finalmente, o último nível da árvore calcula o risco de sucesso do projeto, de como uma média ponderada das probabilidades de sucesso nas categorias de risco. Os pesos das arestas deste nível da árvore são definidos pelo gerente, de acordo com a relevância de cada categoria para o projeto. Este método de avaliação simplifica as relações existentes entre os fatores de

risco, assumindo que estas são lineares e que não existem relações entre fatores de diferentes categorias de riscos.

Técnicas quantitativas modelam os riscos de um projeto em unidades usuais do planejamento, como tempo e dinheiro. GREY (1995) apresenta um processo de análise de riscos que parte da estrutura de decomposição de um projeto – *work breakdown structure (WBS)* – e das relações de dependência entre suas tarefas – definidas através de redes de *PERT/CPM* – para modelar o risco financeiro e de cronograma de um projeto. O custo e tempo necessário para a realização de cada tarefa do WBS são modelados por funções de distribuição de probabilidade. Simulações de Monte Carlo (VOSE, 1996) são utilizadas para a análise dos riscos nas categorias de custo e cronograma, de acordo com as agregações impostas pelas dependências entre tarefas expressas na rede de PERT.

FAIRLEY (1994) também utiliza modelagem estatística em seu processo de análise de riscos. Cada fator de risco é modelado como uma função de distribuição de probabilidade e simulações de Monte Carlo são utilizadas para calcular a exposição agregada a risco de um projeto. KÄNSÄLÄ (1997) apresenta o processo de análise de riscos RiskMethod, onde cada risco é associado a uma probabilidade de ocorrência e uma magnitude de impacto. A probabilidade é representada por um número entre zero e um. A magnitude do impacto é representada por um número em uma escala definida para o risco. Quando diversos resultados distintos são possíveis para um mesmo risco, cada resultado é associado com uma probabilidade e uma medida de impacto. O impacto de um risco é calculado pelo somatório dos produtos das probabilidades dos riscos por suas medidas de impacto.

Embora as técnicas quantitativas sejam desejáveis por oferecerem medidas objetivas de impacto dos riscos, elas também possuem suas limitações. GEMMER (1997) observa que a conotação de um número ou de um termo qualitativo pode variar entre diferentes pessoas. A diferença de interpretação pode levar o analista por uma linha de raciocínio distinta da planejada, incorrendo em um possível erro. O autor ressalta que as evidências e o contexto do risco são mais relevantes que os números associados a suas características.

Embora sofram das limitações apresentadas por GEMMER (1997), consideramos que a análise de riscos deve prezar as técnicas quantitativas, devido a sua objetividade. Entretanto, por sua abrangência, as técnicas qualitativas não devem ser descartadas, podendo auxiliar na identificação dos riscos. Acreditamos que uma forte limitação das

técnicas atuais de avaliação quantitativa de riscos reside na utilização de modelos de projeto de software que não podem ser expandidos para contemplar outros elementos senão as tarefas que devem ser realizadas no projeto. A utilização de um modelo de projeto mais genérico, que possibilite a representação dos elementos envolvidos no projeto e a quantificação de variáveis qualitativas, pode apoiar a análise de riscos quantitativa.

3.2.4.2 Priorização

Diversos processos de análise de riscos (KÄNSÄLÄ, 1997, WILLIANS et al., 1997, KONTIO, 1997, HALL, 1998, PRESSMAN, 2000) encerram a etapa de avaliação de riscos com a priorização dos riscos analisados. Nesta etapa, os riscos são ordenados segundo algum critério e apenas os riscos mais significativos prosseguem para as próximas etapas.

Entre os critérios utilizados na priorização, podemos destacar a exposição e a severidade dos riscos. A exposição a risco é definida como o produto da possibilidade de ocorrência de um risco por seu impacto no projeto. Riscos com alta exposição possuem alta probabilidade de ocorrência, alto impacto ou ambos, devendo estar sujeitos a maior controle. A severidade de um risco é definida como a exposição do risco dividida pelo tempo estimado até sua ocorrência. Um risco com alta severidade está na eminência de se concretizar, pode provocar um grande impacto sobre os resultados do projeto ou ambos, demandando maior atenção.

Risk Radar (MOORE, 1998) utiliza uma abordagem muito simples na avaliação dos riscos de um projeto. Na base de dados manipulada pela ferramenta, a probabilidade de ocorrência de um risco é descrita por uma porcentagem simples. O impacto de um risco é descrito por um número entre 1 e 5, onde um número maior indica um impacto mais significativo no projeto. O intervalo de tempo esperado até a concretização de um risco é descrito por uma faixa de datas. De posse destas informações, a Risk Radar calcula a exposição e a severidade dos riscos, permitindo a priorização dos riscos pelos critérios básicos - probabilidade, impacto e horizonte de tempo – ou pelos critérios compostos - exposição e severidade.

Em diversas situações, somente os riscos mais críticos, segundo o critério de priorização selecionado, serão considerados nas próximas etapas do processo de análise de riscos. Esta priorização se justifica pela inviabilidade econômica de controlar todos os riscos de um projeto. WILLIANS et al. (1997) apresentam uma experiência onde o

custo de gerenciar riscos de pequeno impacto custa mais do que o impacto provocado pela concretização destes riscos. Baseado na lei de Pareto, que determina que 80% dos problemas são gerados por 20% das origens de problemas, PRESSMAN (2000) justifica a priorização e seleção dos riscos mais relevantes para o projeto, investindo recursos para planejar apenas estes riscos.

Entre as técnicas utilizadas na priorização de riscos podemos destacar o ranqueamento, o ranqueamento parcial, a diversificação e a multivotação. O ranqueamento ordena os riscos por um critério previamente selecionado, considerando apenas os principais riscos da lista ordenada. O ponto de corte na lista pode ser selecionado impondo-se um limite mínimo no critério de ordenação ou um limite máximo no número de riscos controlados.

O ranqueamento parcial (GALLAGHER et al., 1997) é realizado em duas etapas. Inicialmente, cada participante elege e ordena os N riscos que considera mais significativos. Em seguida, os riscos são organizados em N conjuntos, tal que o i-ésimo conjunto contém os riscos presentes na i-ésima posição da lista ordenada de cada participante. Riscos duplicados são eliminados, permanecendo apenas a cópia que pertence à lista de maior prioridade. O resultado desta técnica é uma lista parcialmente ordenada dos riscos mais prioritários de um projeto.

KONTIO (1997) apresenta uma técnica para ranqueamento parcial que utiliza uma matriz para gerar a ordenação parcial dos riscos que podem afetar um projeto. Inicialmente, os riscos devem ser ordenados de acordo com sua probabilidade de ocorrência e seu impacto, formando duas listas ordenadas de riscos. As listas são utilizadas para construir a matriz supracitada, que contém uma coluna para cada categoria da ordenação de probabilidade de ocorrência dos cenários de risco e uma linha para cada categoria da ordenação do impacto dos cenários. Os cenários de risco são dispostos nesta matriz de acordo com sua probabilidade de ocorrência e seu impacto. A lista priorizada dos riscos é construída iterativamente. Em cada iteração, os riscos que não possuem outros riscos acima ou a sua esquerda na matriz são inseridos na lista e removidos da matriz. A lista resultante é apenas parcialmente ordenada, visto que não é possível determinar a prioridade entre dois riscos que possam ser removidos da matriz na mesma iteração.

A diversificação prioriza riscos que maximizem a diversificação de projetos, ou seja, riscos que tenham ação complementar a riscos presentes em outros projetos em

andamento. Riscos complementares são definidos como riscos que, em uma situação normal, se compensam.

A multivotação (GALLAGHER et al., 1997, HALL, 1998) é uma técnica utilizada para ordenar uma lista extensa de riscos. Um número fixo de votos são distribuídos a cada participante da votação. Cada participante deve distribuir seus votos entre os riscos do projeto, dedicando um maior número de votos aos riscos que considera mais significativos. Ao fim da votação, os riscos são ordenados de acordo com o número de votos recebidos.

Conforme apresentado, existem diversos métodos para a priorização de riscos. Entretanto, consideramos que mais importante que o método de priorização são os mecanismos utilizados para a determinação da relevância de um risco. Os relacionamentos complexos que podem existir entre os componentes de um projeto de software, conforme indicado no Capítulo 2, podem esconder situações onde as causas e os efeitos estão distante no tempo. Assim, um risco considerado irrelevante, dado seu pequeno impacto no curto prazo, pode provocar um grande efeito a longo prazo no projeto. Os mecanismos de avaliação de riscos devem focalizar em uma análise holística do impacto dos riscos, focalizando detalhadamente seus efeitos sobre os projetos.

3.2.5 Planejamento de Riscos

A etapa de planejamento de riscos determina as estratégias que serão aplicadas para eliminar ou reduzir os riscos de um projeto. O planejamento define mecanismos para percepção da concretização de cada risco, planos de contenção e planos de contingência.

Mecanismos de percepção variam de acompanhamento manual até a definição de métricas e valores limite para considerar um risco como concretizado. Por exemplo, o risco de módulos de alta complexidade pode ser controlado medindo-se a complexidade ciclomática (MCCABE, 1976) de cada módulo. Se esta métrica ultrapassar um limite pré-estabelecido, o módulo é considerado de alto risco, sendo o fato apresentado para a equipe de desenvolvimento. HALL (1998) apresenta um conjunto de métricas para planejamento e controle de riscos. GREY (1995) também observa a importância da determinação dos responsáveis pelo tratamento de cada risco, pois a responsabilidade pela realização de ações determinadas no planejamento não pode ser dividida.

Planos de contenção têm o objetivo de evitar a ocorrência de um risco ou reduzir seu impacto no projeto, sendo aplicados antes da ocorrência do risco. Estes planos

podem se realizar através de treinamento, da contratação de especialistas, da realização de etapas de verificação, do alargamento do cronograma de tarefas críticas e na reserva de recursos além do originalmente planejado, entre outras formas. Como os planos de contenção são aplicados antes da ocorrência do risco, o planejamento de riscos deve especificar sua data de implantação e os recursos necessários para esta.

Planos de contingência têm o objetivo de reduzir os efeitos da concretização de um risco, sendo postos em execução após sua ocorrência. Planos de contingência podem se concretizar como rotas alternativas às tarefas definidas no WBS, a utilização de recursos de um fundo de emergência e a criação de pequenas equipes para a resolução de aspectos críticos do software, entre outros.

A criação de planos de contenção e contingência depende dos recursos disponíveis em cada projeto. Não existem regras para criação destes planos, apenas guias e heurísticas (BOEHM, 1989, KONTIO, 1997). Os planos de contingência e contenção, normalmente, incorrem em custo extra para o projeto. A equipe de gerenciamento de risco é responsável pela análise da relação custo-benefício de cada plano. Esta análise depende da estratégia selecionada para o planejamento de riscos. HALL (1998) apresenta seis estratégias distintas para o planejamento de riscos.

- Aceitação: quando a relação custo-benefício dos planos de contenção e contingência não for propícia, a equipe de gerenciamento pode decidir por não aplicar estes planos, convivendo com as conseqüências da concretização do risco;
- Rejeição: em algumas circunstâncias, quando a concretização de um risco pode ser muito prejudicial, a equipe de gerenciamento pode optar pela rejeição do risco. Esta rejeição pode se realizar, por exemplo, por uma negociação junto ao cliente para remoção das funcionalidades que concentrem o risco;
- Proteção: a equipe de gerenciamento pode requisitar recursos extras, que serão utilizados para amenizar ou corrigir os efeitos da ocorrência de um risco. Esta estratégia privilegia os planos de contingência;
- Redução: a equipe de gerenciamento pode acrescentar recursos ao plano de projeto, utilizando estes recursos para reduzir a probabilidade ou o impacto da ocorrência de riscos. Esta estratégia privilegia os planos de contenção;
- Pesquisa: a equipe de gerenciamento pode buscar mais informações para aprimorar a estratégia de tratamento dos riscos. Estas informações podem ser adquiridas de

diversas formas, como através de experimentos, protótipos ou consultas aos usuários;

- Transferência: a equipe de gerenciamento pode optar por delegar a terceiros o desenvolvimento de parte do projeto, transferindo assim o risco ou transformando-o em um risco de outra natureza (por exemplo, risco técnico para risco contratual).

Baseado na hipótese que se um projeto independente tivesse que se proteger de todos os seus riscos seu preço seria proibitivo, KITCHENHAN e LINKMAN (1997) propõem a criação de fundos de contingência de riscos. Estes fundos seriam controlados pela organização de desenvolvimento e disponibilizados a todos os projetos. Além de capturar a diversificação, os fundos de contingência facilitam a existência de projetos de maior risco, que podem se apropriar dos fundos de contingência de projetos de menor risco.

Muitos processos de análise de riscos se concentram nas duas primeiras atividades do processo (identificação e avaliação de riscos), não abordando as atividades de planejamento e controle. Acreditamos que estas duas atividades, especialmente a atividade de planejamento, onde os planos de controle dos riscos são traçados, são de grande importância. Os planos de contenção e contingência, desenvolvidos durante o planejamento de riscos e avaliados durante o controle de riscos, são parte do conhecimento associado aos riscos que podem afetar um projeto, devendo ser motivo de reutilização. Além disso, o impacto dos planos de contenção e de contingência sobre o comportamento de um projeto também deve ser avaliado. Esta avaliação, conforme a avaliação do impacto dos riscos, deve capturar as relações complexas existentes entre os diversos componentes de um projeto. Assim, faz-se relevante a contemplação das atividades de planejamento e controle pelos processos de análise de riscos.

3.2.6 Controle de Riscos

O controle de riscos começa após o planejamento do projeto. Esta etapa tem como objetivos o acompanhamento da evolução dos riscos, implantação, correção e avaliação dos planos de contenção e contingência. O controle de riscos monitora os mecanismos de percepção de ocorrência dos riscos identificados no plano de gerenciamento, verificando se os limites para aplicação dos planos de contenção e contingência foram ultrapassados. Quando os limites forem ultrapassados, os referidos planos são implantados.

A verificação dos limites deve ser uma atividade contínua ao longo do processo de desenvolvimento. Entretanto, esta atividade normalmente é realizada em reuniões semanais ou quinzenais (KONTIO, 1997). Regimes de controle mais estritos podem ser implantados em períodos específicos do processo de desenvolvimento, quando determinados fatores de risco se tornam determinantes para o sucesso do projeto.

A eficácia dos planos de contenção e contingência é avaliada, assim como o impacto do evento indesejado sobre o processo de desenvolvimento e o produto. Estas informações são armazenadas para aplicação em futuros projetos ou para auditoria do projeto atual.

WILLIANS et al. (1997) dividem as etapas realizadas durante o controle de riscos em alterações nas estratégias de saneamento, aplicação de ações nos riscos mais importantes, aplicação de planos de contingência após a concretização dos riscos, relaxamento dos planos de contingência, quando o risco for reduzido, e encerramento do controle sobre um risco, quando sua concretização não for mais possível. Todas as ações tomadas no controle de um risco devem ser justificadas, permanecendo a justificativa catalogada para futuros projetos.

FAIRLEY (1994) analisa a possibilidade de falha dos planos de contingência, incluindo etapas em seu processo de análise de riscos para o gerenciamento e recuperação de crise. Na etapa de gerenciamento de crise, os recursos do projeto são redistribuídos com o objetivo de amenizar ou corrigir o efeito gerador da crise. Uma data de avaliação é demarcada para o projeto, quando este é encerrado se o estado de crise não tiver sido recuperado. Na etapa de recuperação de crise, a equipe é recompensada por seus esforços durante a crise e os limites de tempo e custo de desenvolvimento do projeto são reavaliados, determinando-se se este deve prosseguir.

3.3 Conclusão

A Tabela 3.6 apresenta um resumo das principais características de algumas abordagens de análise de riscos apresentadas neste capítulo. Em seguida, sintetizamos as críticas e observações a respeito das abordagens, conforme os quesitos relacionados na tabela.

A linha **Detalhamento** indica uma preocupação da abordagem de análise de riscos em coletar informações complementares sobre um risco, como o contexto em que ele ocorre, as estratégias de controle do risco, os responsáveis pelo controle, entre outros. Em geral, as abordagens de análise de riscos quantitativas (FAIRLEY, 1994, GREY,

1995, MADACHY, 1997), por atuarem sobre fatores de risco, tais como custo, cronograma e qualidade, não armazenam detalhes sobre cada risco específico. Em geral, estes processos de análise de riscos focalizam a etapa de avaliação de riscos, ignorando as etapas subsequentes. O processo de (FAIRLEY, 1994) é uma exceção dentre os processos quantitativos, armazenando informações sobre as estratégias de contenção e contingência dos riscos.

	Hall 98	Willians 97	Grey 95	Madachy 97	Fairley 94	Kontio 97	Garvey 97	Gemmer 97
Detalhamento	✓	✓			parcial	✓	✓	✓
Reutilização						✓	✓	
Agregação Complexa			✓	✓	✓	parcial		
Planejamento	✓	✓			✓	✓	✓	
Controle	✓	✓			✓	✓		

Tabela 3.6 – Resumo das principais características das abordagens de análise de riscos

Consideramos que os riscos identificados para um projeto devem ser precisamente descritos, incluindo informações acerca do contexto de sua ocorrência, dos fatores que inibem e promovem esta ocorrência, dos planos de contenção e contingência, entre outras informações. A capacidade de armazenamento destas informações permite que os riscos atuem como artefatos centralizadores do processo de análise de riscos: todas as etapas do processo recebem as informações criadas pelas etapas anteriores, refinando esta informação e produzindo uma nova lista de riscos. Além disso, o detalhamento dos riscos também enriquece o conhecimento expresso no artefato, potencialmente aumentando seu valor em uma futura reutilização.

A linha **Reutilização** indica as abordagem de análise de riscos que oferecem suporte à reutilização de informações coletadas sobre riscos em novos projetos. O processo RiskIt (KONTIO, 1997, KONTIO e BASILI, 1996) permite a reutilização de riscos através de sua integração com a *Experience Factory* (BASILI et al., 1992). RAMP (GARVEY et al., 1997) armazena informações sobre os riscos encontrados por diversos projetos, permitindo sua reutilização por analogia. Os demais processos não contemplam diretamente a reutilização de riscos.

Riscos representam o conhecimento dos gerentes sobre os problemas potenciais que podem afetar um projeto de desenvolvimento de software. Quando as informações sobre os riscos são detalhadas, armazenando os planos de contenção e contingência utilizados para evitar e sanear os risco, estes também representam o conhecimento sobre estratégias eficazes de resolução dos problemas. Este conhecimento, como qualquer

conhecimento relacionado com o desenvolvimento de software, é muito valioso e deve ser reutilizado. Assim, acreditamos que os processos de análise de riscos devem suportar a reutilização de riscos.

A linha **Agregação Complexa** determina se os processos de análise de riscos são capazes de avaliar relações complexas, possivelmente não lineares, existentes entre os riscos e os componentes do projeto. As abordagens que utilizam simulações (FAIRLEY, 1994, GREY, 1995), ou sistemas de regras (MADACHY, 1997), geralmente, são capazes de capturar estas relações. O processo RiskIt (KONTIO, 1997) atende parcialmente este requisito, pois consideramos que existe um limite na capacidade das funções de utilidade descreverem relações complexas entre riscos.

Os processos de análise de riscos quantitativos se baseiam em modelos. Como os modelos utilizados nos processos atuais de gerenciamento de riscos são limitados a um determinado conjunto de fatores, não podendo ser estendidos para considerarem outros componentes do projeto, os processos de análise de riscos estão limitados a estes fatores. Acreditamos que a utilização de um modelo de projeto extensível, que possa contemplar diversos fatores relacionados com projetos de software, sendo enriquecido com informações a medida que estes fatores são acrescentados ao modelo, pode auxiliar na avaliação das relações complexas existentes entre os riscos, suas estratégias de resolução e os componentes do projeto.

As linhas **Planejamento** e **Controle** indicam a participação das abordagens nas etapas de planejamento e controle de riscos. Conforme indicado na Tabela 3.6, nem todos os processos de análise de riscos apresentados na literatura contemplam estas etapas. Acreditamos que os processos de análise de riscos devem considerar as quatro etapas do processo genérico de análise de riscos, de forma a produzir e avaliar as estratégias de resolução de riscos, que enriquecem os detalhes acerca dos problemas potenciais que podem afetar um projeto de software.

Os requisitos analisados nesta seção direcionam o processo de análise de riscos baseado em simulação e integração de modelos de projeto e cenários apresentado no Capítulo 6. Este processo aborda as quatro principais atividades de um processo de análise de riscos (identificação, avaliação, planejamento e controle). Muitas das técnicas apresentadas nesta revisão literária foram consideradas satisfatórias, sendo utilizadas no processo de análise de riscos proposto. As principais contribuições do processo proposto se referem à reutilização de riscos e avaliação de seu impacto sobre um projeto.

4 Gerenciamento de Projetos Baseado em Cenários

A seguinte heurística descreve brevemente a visão tradicional de gerenciamento de projetos de desenvolvimento de software: planeje o vôo e voe segundo o plano[1]. Esta heurística está fortemente relacionada com as premissas das técnicas atualmente aplicadas no gerenciamento de projetos de software. Ela assume que o gerente de projeto pode definir uma estratégia detalhada para conduzir a equipe de desenvolvimento das especificações iniciais de um produto até a sua implantação no ambiente operacional, considerando determinadas restrições de cronograma, recursos e qualidade. A realização desta tarefa requer um conhecimento completo do esforço de desenvolvimento a ser realizado, o que pode não ser possível em projetos inovadores e complexos.

Eventos inesperados, como a incapacidade dos desenvolvedores resolverem determinados problemas, reduzido envolvimento dos usuários com o projeto e erros surgindo de produtos "concluídos" provocam turbulências no vôo planejado pelo gerente de projetos. Em diversas situações, o sistema que se deseja construir não pode ser precisamente definido no início do projeto, sendo seus requisitos descobertos ao longo do processo de desenvolvimento. Nestes casos, a criação de uma estratégia detalhada nas primeiras etapas do projeto se torna ainda mais difícil.

Como o conhecimento preciso sobre o comportamento de um projeto em suas primeiras etapas não é comum em diversas disciplinas, inclusive no desenvolvimento de software, as técnicas tradicionais de gerenciamento de projetos foram adaptadas para tratar algum grau de incerteza. Por exemplo, as redes de atividades foram estendidas para permitir a descrição do tempo necessário para a execução de cada atividade como uma função de distribuição de probabilidade. As tabelas de alocação de recursos foram adaptadas para condicionar a distribuição de recursos à ocorrência de determinados eventos incertos. Enfim, o plano de projeto passou a ser observado como um artefato dinâmico, suscetível a mudanças ao longo do processo de desenvolvimento, quando as incertezas do projeto se transformam em problemas ou são resolvidas.

Entretanto, as técnicas tradicionais de gerenciamento não podem capturar todos os tipos de incerteza que podem ocorrer durante a execução de um projeto de software. Estas técnicas são incapazes de descrever outras fontes de incerteza além daquelas

---

[1] Em inglês, "*plan the flight and fly the plan*".

relacionadas com os elementos para os quais as técnicas foram originalmente desenvolvidas. As incertezas acerca destes elementos estão ainda limitadas às informações que as técnicas armazenam para descrevê-los. Por exemplo, as redes de atividade podem descrever incertezas a respeito da duração das atividades e das dependências entre elas. As incertezas sobre a motivação da equipe, sobre as taxas de geração, descoberta e correção de erros decorrentes da utilização de um determinado método de desenvolvimento de software, por exemplo, não podem ser capturadas pelas redes de atividades tradicionais. De fato, acreditamos que nenhum modelo estático possa permitir a avaliação de qualquer tipo de incerteza. Entretanto, um modelo extensível poderia descrever as informações conhecidas sobre um projeto de software, permitindo que modelos complementares armazenem as informações necessárias para representar os diversos tipos de incertezas.

Neste capítulo, apresentamos o **gerenciamento de projetos baseado em cenários**, um conjunto de técnicas que permitem que um gerente de projetos defina o comportamento esperado para um projeto e um conjunto de eventos que podem acontecer ao longo de seu desenvolvimento. Os eventos representam as incertezas do gerente sobre os rumos de seu projeto no futuro. Eles são representados através de cenários, que são utilizados para avaliar o impacto dos eventos sobre o comportamento do projeto (custo, cronograma, qualidade, entre outros). Simulação e modelagem formal de processos são ferramentas que suportam este paradigma.

O gerenciamento de projetos baseado em cenários não é a primeira abordagem a observar a necessidade de um plano de projeto dinâmico. Abordagens previamente apresentadas na literatura da Engenharia de Software focalizaram as dificuldades e os problemas existentes na definição de um plano estático nas etapas iniciais do processo de desenvolvimento. HIGHSMITH (1992) observa que o desenvolvimento bem sucedido de projetos de software inclui uma cuidadosa avaliação dos objetivos a serem atingidos, preparação adequada, observação contínua do ambiente e da evolução do projeto, a definição e implantação de táticas de ajuste. O autor observa que o plano inicial deve ser periodicamente avaliado face aos eventos que se apresentam ao longo do projeto, ajustando-se este plano às novas condições. Esta estratégia é denominada "ancoramento e ajuste": o plano inicial é utilizado como uma referência (âncora) para o projeto, mas pode ser alterado ao longo do tempo pelas táticas de ajuste.

O gerenciamento de projetos baseado em cenários adota uma estratégia similar. Entretanto, a maior diferença entre o paradigma proposto e as abordagens apresentadas

na literatura reside na formalização do comportamento esperado para um projeto de software, na formalização das incertezas que podem ocorrer ao longo do processo de desenvolvimento e na avaliação do impacto destas incertezas sobre o comportamento esperado para o projeto. O gerenciamento de projetos baseado em cenários se sustenta nos seguintes princípios:

- Gerenciamento baseado em análise de riscos: a identificação e documentação das incertezas que podem ocorrer ao longo de um processo de desenvolvimento é fundamental para a construção de cenários. A análise de riscos é utilizada para caracterizar, entender e descrever estas incertezas. O desenvolvimento progressivo de uma cultura organizacional que promova a discussão dos riscos que podem afetar um projeto, a seleção de projetos baseada em riscos e a criação de mecanismos que permitam aproveitar as oportunidades reveladas pelos riscos de um projeto são condições para a implantação do paradigma de gerenciamento de projetos baseado em cenários;
- Gerenciamento pró-ativo: a participação dos gerentes no ciclo de desenvolvimento se estende além da criação do plano de projeto, ocorrendo ao longo de todo o processo. Os gerentes são responsáveis pelo acompanhamento do projeto, promovendo e observando a eficácia de aprimoramentos no processo, criando situações de ganho mútuo[2] (BOEHM e ROSS, 1988) e otimizando a distribuição e utilização dos recursos reservados para o projeto;
- Modelagem de projetos: modelos dinâmicos de projetos de software, que permitam a análise de cenários e a avaliação quantitativa da sensibilidade do projeto a estes cenários, são utilizados para avaliar o impacto de ações, decisões e da ocorrência de riscos ao longo do processo de desenvolvimento. Os modelos servem como uma "maquete" do projeto real, onde os cenários podem ser aplicados, testados e avaliados antes de serem implantados no projeto;
- Acompanhamento quantitativo de projetos: métricas são coletadas ao longo do processo de desenvolvimento e utilizadas como mecanismos de alerta, amplificando a visibilidade do projeto para a equipe. As métricas guiam a atualização dos modelos de projeto a medida que o projeto se desenrola e a definição dos parâmetros de

[2] No original, *"win-win situations"*.

projetos futuros, formando uma base de informações históricas na organização de desenvolvimento;

- Análise estatística: a avaliação de séries históricas de métricas coletadas durante o processo de desenvolvimento e ao longo de diversos projetos determina os processos estocásticos que governam os atributos dos produtos e dos processos de desenvolvimento. HUMPHREY e SINGPURWALLA (1991) utilizaram métodos de análise temporal de séries históricas para modelar a produtividade de um desenvolvedor ao longo de uma seqüência de atividades. O modelo resultante desta análise foi utilizado para projetar os resultados do mesmo programador ao desenvolver três atividades distintas. Esperamos que este tipo de análise possa ser aplicado para descrever a volatilidade dos requisitos de um sistema, as taxas de detecção e correção de erros, entre outras características do processo ou dos produtos construídos.

A Figura 4.1 apresenta uma visão geral da arquitetura do paradigma de gerenciamento de projetos baseado em cenários. Um gerente de projetos e um gerente de projetos sênior são os principais papéis envolvidos nesta arquitetura. O gerente de projetos é responsável pela descrição do projeto em desenvolvimento e pela recuperação de cenários. Os cenários são organizados em um repositório, que atua como uma base centralizada de conhecimento.

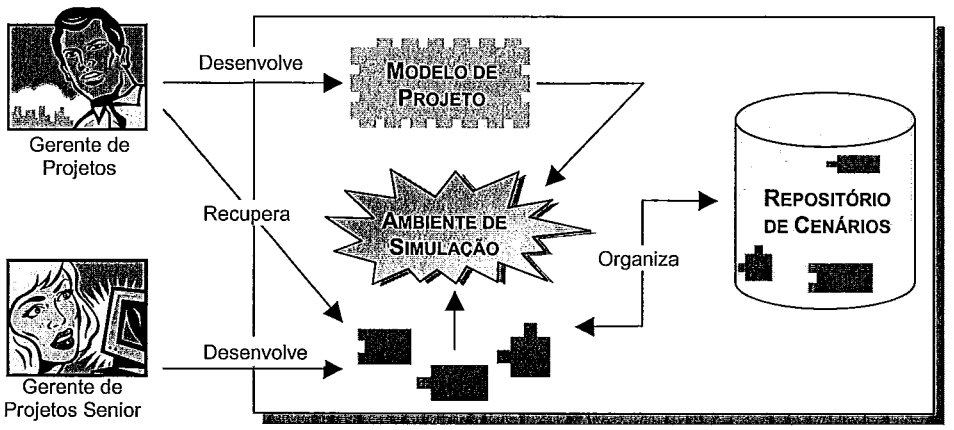

Figura 4.1 – A arquitetura do paradigma de gerenciamento de projetos de software baseado em cenários

Os gerentes de projeto sênior são os principais responsáveis pela criação de cenários, baseados em sua experiências pessoais. Eventualmente, os gerentes de projeto também poderão desenvolver cenários próprios, para capturarem incertezas específicas

de seus projetos. Depois da seleção dos cenários que podem afetar um projeto, estes cenários são integrados à descrição do projeto dentro de um ambiente de simulação, que avalia o impacto dos cenários sobre o comportamento esperado para o projeto.

Em um artigo comemorativo pela passagem do 35º aniversário da primeira apresentação das técnicas de Dinâmica de Sistemas, FORRESTER (1991) afirma que, "... no futuro, o treinamento em projeto de organizações incluirá o estudo de uma biblioteca de situações genéricas de gerenciamento que combinarão estudos de casos com modelos dinâmicos e simulação computacional". Embora o autor estivesse focalizando apenas as atividades de treinamento, acreditamos que a reutilização de modelos de situações genéricas de gerenciamento tem aplicação mais ampla, podendo auxiliar no planejamento e controle de uma organização. No gerenciamento de projetos baseado em cenários, as situações genéricas de gerenciamento são representadas por cenários, enquanto o repositório de cenários atua como sua biblioteca.

O paradigma de gerenciamento de projetos baseado em cenários é centralizado em dois artefatos: o modelo de projeto e os modelos de cenários. O modelo de projeto define o comportamento esperado para um projeto, enquanto os modelos de cenários descrevem rotas alternativas que o projeto pode seguir devido a ocorrência de eventos inesperados. Cenários contêm conhecimento gerencial genérico e reutilizável (BARROS et al., 2000d). Eles mantêm as representações de ações, teorias, políticas e procedimentos gerenciais que podem ser aplicados ou impostos a um projeto de desenvolvimento de software. Existem diversas categorias de cenários, apresentadas e discutidas na seção 4.3. Os cenários podem ser integrados ao modelo de projeto, modificando seu comportamento para representar as conseqüências de sua ocorrência sobre o projeto. O modelo de projeto descreve a interface de integração de cenários.

Os modelos de projeto e cenários são modelos formais de desenvolvimento de software. Um modelo formal descreve as relações existentes entre os elementos do mundo real através de postulações matemáticas ou lógicas. No Capítulo 2, discutimos a aplicação de modelos formais de projetos de desenvolvimento de software. A seção 2.1 apresenta as vantagens de se utilizar modelos formais, enquanto a seção 2.3 apresenta possíveis aplicações da modelagem formal de projetos de software. O gerenciamento de projetos baseado em cenários utiliza modelos de projetos baseados na Dinâmica de Sistemas (apresentada na seção 2.4.3).

A Dinâmica de Sistemas é uma técnica e linguagem de modelagem utilizada para descrever sistemas complexos que focaliza nos aspectos estruturais destes sistemas

(FORRESTER, 1961). Esta técnica identifica e modela relações de causa e efeito e ciclos de realimentação[3] através de diagramas de fluxos, compostos por quatro elementos básicos: repositórios, taxas, fluxos e processos. Os repositórios representam elementos que podem ser acumulados ou consumidos ao longo do tempo. Um repositório é descrito por um nível, que indica o número de elementos que se encontram no repositório em um determinado instante. As taxas descrevem a variação do nível de um repositório ao longo do tempo, formulando seus aumentos e reduções através de equações matemáticas. Processos e fluxos representam variáveis intermediárias, que podem ser utilizadas nas equações que descrevem as taxas. O Apêndice A apresenta os diagramas que representam os modelos da Dinâmica de Sistemas, assim como a interpretação matemática de seus construtores.

Simulação é a técnica utilizada para avaliar o comportamento dos modelos construídos a partir dos blocos básicos da Dinâmica de Sistemas. A integração dos modelos de cenários com o modelo de projeto pode revelar alterações no comportamento original do modelo de projeto. A simulação de um modelo de projeto com e sem um determinado cenário demonstra o impacto potencial da ocorrência do cenário sobre o comportamento do projeto. O modelo de projeto é denominado um **modelo concreto**, podendo ser simulado sem exigir a integração com outros modelos. Modelos de cenário que somente podem ser simulados quando integrados a um modelo de projeto são denominados **modelos abstratos**.

4.1 Os Modelos de Projeto

Na seção 2.6 discutimos as deficiências das técnicas utilizadas na modelagem de projetos de desenvolvimento de software. As principais deficiências estão relacionadas com a dificuldade de construção dos modelos, sua capacidade de representação de conhecimento, sua capacidade de detalhamento e de representação de incertezas. Os modelos de projeto do gerenciamento de projetos baseado em cenário utilizam a Dinâmica de Sistemas que, como as outras técnicas de modelagem apresentadas no capítulo 2, não cumpre todos os requisitos listados na seção 2.6. Para que estes requisitos sejam atingidos, propomos adaptações para a modelagem com a Dinâmica de Sistemas.

---

[3] No original, "*feedback loops*".

Tradicionalmente, os modelos desenvolvidos segundo a Dinâmica de Sistemas são descritos por diversas equações matemáticas (ALBIN, 1997), o que dificulta seu entendimento e adaptação. Os elementos componentes do sistema no mundo real não são facilmente identificados no labirinto de equações formado pelos construtores da Dinâmica de Sistemas. A representação destes elementos está geralmente dispersa ao longo de diversas equações, o que força os desenvolvedores a analisar grande parte do modelo para determinar as equações que descrevem o comportamento de um elemento. Estas dificuldades inibem a utilização prática da Dinâmica de Sistemas na modelagem de projetos de software. Idealmente, os modelos deveriam ser construídos a partir dos conceitos do domínio do problema, sendo posteriormente traduzidos para equações da Dinâmica de Sistemas, para sua avaliação através de simulação. No caso dos modelos de projeto, os conceitos do domínio são especificados em modelos descritivos de processos de desenvolvimento de software.

Ao analisarmos diversos modelos descritivos de processos de desenvolvimento apresentados na literatura de Engenharia de Software (LIU e HOROWITZ, 1989, HUMPHREY e KELLNER, 1989, CHRISTIE, 1995, FALBO, 1998), visando identificar os elementos comuns entre estes modelos, pudemos definir os componentes do modelo de projeto utilizado no gerenciamento de projetos baseado em cenários.

O modelo DesignNet (LIU e HOROWITZ, 1989) se baseia em grafos E/OU e redes de Petri para representar, respectivamente, a estrutura de decomposição de atividades de um projeto e as relações de precedência entre estas atividades. O modelo captura as relações entre as atividades e os artefatos de software criados e consumidos por elas, mas não contempla os desenvolvedores ou outros tipos de recursos.

HUMPHREY e KELLNER (1989) propuseram um modelo de processo baseado nas transformações sofridas por artefatos ao longo de um projeto. Estas transformações são representadas em diagramas de estado, que capturam as características dinâmicas dos artefatos. Estas características podem ser avaliadas através de simulações, que podem ou não considerar limites de disponibilidade de recursos. Este modelo, entretanto, não considera as particularidades de cada tipo de recurso. Por exemplo, o modelo é capaz de requisitar um determinado número de programadores para realizar uma atividade, mas não considera as habilidades específicas de cada programador.

CHRISTIE (1995) descreve um modelo de processo centralizado no fluxo das atividades componentes de um projeto de software. Este modelo descreve os artefatos que devem estar disponíveis, as condições que devem ser verificadas antes que a

atividade tenha início, os agentes participantes e as restrições que uma atividade deve respeitar. Estas dependências são representadas por relacionamentos entre as atividades e outros elementos do projeto.

FALBO (1998) apresenta uma ontologia para processos de desenvolvimento de software. Esta ontologia descreve as relações de precedência entre as atividades, a decomposição de atividades e a classificação destas atividades entre construção, gerenciamento e controle de qualidade. A ontologia captura os artefatos criados e utilizados por cada atividade e os recursos consumidos por ela. Artefatos de software podem ser compostos por outros artefatos e são classificados como componentes, código e documentos. Recursos incluem software, hardware e desenvolvedores. A ontologia foi utilizada por KITCHENHAM et al. (1999) na construção de uma ontologia de manutenção de projetos de software.

A partir da análise destes modelos descritivos de processos, extraímos um modelo simplificado que contém os elementos comuns encontrados em grande parte desses modelos. O modelo simplificado sofreu algumas adaptações que visaram aumentar seu foco nos aspectos dinâmicos que ocorrem em projetos de desenvolvimento de software.

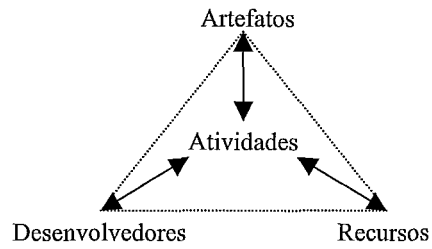

Figura 4.2 – A representação espacial do modelo de projeto

A representação dos modelos de projeto se baseia em uma figura de quatro vértices (Figura 4.2), uma analogia geométrica que focaliza as principais características do modelo: suas atividades, seus desenvolvedores, seus artefatos e seus recursos. As atividades ocupam o vértice central, representando pacotes de trabalho que devem ser desenvolvidos para a conclusão do projeto. Os desenvolvedores representam as pessoas que trabalham no projeto. Eles realizam as atividades utilizando os recursos. Os recursos representam elementos físicos ou lógicos que são utilizados ao longo do processo de desenvolvimento, como computadores, ferramentas de software, componentes reutilizáveis, entre outros. As atividades criam e evoluem artefatos de software, eventualmente utilizando outros artefatos como insumo.

Um projeto de software é dividido em diversas atividades, que podem ser decompostas em atividades mais simples. Em cada nível de decomposição as atividades formam um grafo direcionado acíclico, cujas arestas representam as dependências entre as atividades. Uma dependência é uma relação direcionada entre duas atividades, que indica que uma atividade (destino da relação) somente pode ter início após a conclusão da outra atividade (origem da relação). A disponibilidade de recursos e desenvolvedores também são restrições ao início de uma atividade.

A duração de cada atividade é expressa como uma função de distribuição de probabilidade beta-PERT (VOSE, 1996). Esta distribuição de probabilidade captura a incerteza do gerente acerca do tempo necessário para concluir cada atividade. O gerente determina os parâmetros da distribuição beta-PERT, que indicam o tempo mínimo, o tempo esperado e o tempo máximo para a conclusão da atividade, ao invés de indicar um valor rígido para a sua duração. Simulações de Monte Carlo (VOSE, 1996) propagam as incertezas expressas em cada atividade ao longo do modelo, determinando a distribuição de probabilidades que descreve a duração do projeto.

As conexões entre as atividades e os desenvolvedores representam os desenvolvedores responsáveis pela realização de uma atividade. As conexões entre as atividades e os artefatos denotam os artefatos utilizados como insumos das atividades e os artefatos produzidos por elas. As conexões entre as atividades e os recursos demonstram os recursos utilizados pelas atividades.

A separação entre os desenvolvedores e os recursos focaliza a distinção entre recursos humanos e recursos de software ou hardware. Recursos, conforme especificado no modelo de projeto, representam periféricos, ferramentas de software, componentes reutilizáveis, bibliotecas, que são utilizados para a realização das atividades. Diversos modelos descritivos de processos descrevem estes elementos junto com os recursos humanos (HUMPHREY e KELLNER, 1989, CHRISTIE, 1995). Entretanto, separamos os recursos humanos dos demais recursos no modelo de projeto para focalizar suas propriedades dinâmicas relevantes, geradas pelas diferentes habilidades de cada desenvolvedor e sua importância na relação com os demais elementos do projeto.

O modelo permite o acompanhamento do estado do projeto ao longo de seu desenvolvimento. A medida que novas informações acerca do estado dos elementos do projeto estão disponíveis, estas informações podem ser inseridas no modelo, ajustando seu comportamento a seu estado corrente. Ao inserir novas informações no modelo descritivo, um novo modelo dinâmico pode ser gerado e submetido a simulação. Como

este modelo foi atualizado com o estado do projeto, novas análises de sensibilidade a cenários podem ser executadas, permitindo que o gerente ajuste suas expectativas a respeito do comportamento do projeto. Por exemplo, se o gerente de projetos está incerto sobre a capacidade de sua equipe em desenvolver um projeto em uma determinada linguagem de programação, a codificação de alguns módulos do projeto dentro do cronograma estabelecido (informação sobre o estado do projeto) pode alterar as expectativas do gerente sobre o comportamento futuro do projeto, reduzindo suas incertezas sobre a capacitação da equipe.

4.2 Representação dos Modelos de Projeto

Na seção anterior apresentamos a descrição dos modelos de projetos, indicando que estes seriam representados em uma linguagem de maior nível de abstração que os construtores da Dinâmica de Sistemas. Esta linguagem facilitaria o desenvolvimento do modelo, que seria então representado por conceitos mais próximos ao domínio do problema, posteriormente traduzidos para os construtores da Dinâmica de Sistemas para que o modelo seja analisado através de simulações. A representação em maior nível de abstração procura atender ao primeiro requisito da seção 2.6, facilitando o desenvolvimento de modelos dinâmicos de projetos de software.

Outro problema dos modelos construídos com base na modelagem tradicional da Dinâmica de Sistemas reside na tendência a descrever uniformemente todos os elementos pertencentes a uma mesma categoria em um domínio. Os modelos geralmente utilizam valores médios para quantificar as propriedades destes elementos. Por exemplo, diversos modelos de projeto de software assumem que todos os desenvolvedores experientes possuem a mesma produtividade, ou que todos os desenvolvedores estão igualmente motivados (ABDEL-HAMID e MADNICK, 1991, LIN e LEVARY, 1989, LIN et al., 1997). Assumimos que esta simplificação se deve às limitações da Dinâmica de Sistemas na representação das propriedades dos elementos, uma vez que deve existir pelo menos uma variável independente no modelo para cada atributo de cada elemento sendo modelado, o que incorre em um grande número de equações para um modelo complexo. Esta limitação dos modelos dinâmicos está relacionada com o terceiro requisito especificado na seção 2.6, que se refere à capacidade de detalhamento do modelo.

Além disso, um modelo geralmente têm o objetivo de descrever um problema específico dentro de um domínio. Entretanto, o modelo contém conhecimento do

domínio embutido em suas equações e parte deste conhecimento é genérico e reutilizável. Este conhecimento não é facilmente identificado entre as equações do modelo, o que inibe a criação de modelos em domínios complexos, uma vez que cada desenvolvedor de modelos deve adquirir, aprender e representar o mesmo conhecimento de domínio em cada novo modelo. Esta característica também não oferece economia de escala quando diversos modelos são construídos para o mesmo domínio porque cada modelo deve repetir a descrição do conhecimento do domínio.

Por um lado precisamos de modelos simples, que facilitem a sua validação e entendimento. Por outro lado, queremos modelos que capturem os detalhes acerca dos elementos que os compõem. Assim, precisamos de técnicas de modelagem que aprimorem o desenvolvimento de modelos complexos. Se o conhecimento sobre um domínio é claramente separado das informações de um problema específico, cada novo modelo desenvolvido para o domínio pode reutilizar esta informação. A reutilização do conhecimento sobre o domínio, representado em um modelo criado por especialistas do domínio treinados na técnica de modelagem, pode reduzir os custos de desenvolvimento de novos modelos.

Assim, propomos uma abordagem que trata a complexidade dos modelos dinâmicos pelo aumento do nível de abstração de seus construtores. O maior nível de abstração é atingido descrevendo-se o modelo através de construtores que representem conceitos mais próximos do mundo real (elementos do domínio do problema) e mais distantes das postulações matemáticas (elementos do domínio da solução). As equações continuam necessárias para a realização de simulações e análises, mas deixam de ser utilizadas na descrição do modelo. O modelo deve ser expresso em uma linguagem mais próxima de seus usuários, sendo posteriormente transformado em sua representação matemática. Assim, ao invés de desenvolver um modelo a partir dos construtores da Dinâmica de Sistemas (repositórios, taxas, processos e fluxos), os desenvolvedores de modelos utilizam construtores de alto nível, que contém o conhecimento do domínio.

A abordagem proposta é composta por um metamodelo e um processo de tradução para a Dinâmica de Sistemas. O metamodelo é uma representação em alto nível de abstração dos construtores da Dinâmica de Sistemas, que permite a definição de linguagens de modelagem específicas para determinados domínios. Estas linguagens definem os construtores que contém o conhecimento do domínio e são utilizados para descrever os modelos. O processo de tradução compila os modelos descritos nas linguagens de modelagem definidas a partir do metamodelo, traduzindo sua

representação de alto nível para os construtores básicos da Dinâmica de Sistemas, que podem ser utilizados em simulações e na análise de modelos. O comportamento dos modelos é expresso através de uma extensão dos construtores da Dinâmica de Sistemas, que são descritos em separado para cada categoria de elementos que compõe o modelo de domínio.

Analisando o metamodelo da Dinâmica de Sistemas sob o prisma da reutilização de software, a abordagem pode ser comparada ao paradigma DRACO (NEIGHBORS, 1981) (LEITE et al., 1994). Em DRACO, os autores propõem a criação de linguagens que descrevam um domínio de aplicações, capturando as principais características e funcionalidades de um conjunto de aplicações desenvolvidas para cada domínio. No desenvolvimento de uma nova aplicação para o domínio, o conhecimento representado na linguagem de domínio é reutilizado e transformado, pela aplicação de sucessivos refinamentos, em um sistema executável. De forma análoga, o metamodelo da Dinâmica de Sistemas permite a captura do comportamento de um conjunto de modelos em uma representação de alto nível e sua futura tradução para os construtores tradicionais da Dinâmica de Sistemas, que podem ser simulados.

Acreditamos que esta abordagem não se aplica apenas na descrição de modelos de projeto de desenvolvimento de software, podendo ser aplicada na modelagem de outros domínios (BARROS et al. 2001a). O gerenciamento de projetos baseado em cenários explora o metamodelo da Dinâmica de Sistemas para descrever os modelos de projeto a partir de suas atividades, seus desenvolvedores, seus recursos e seus artefatos, sem perder sua capacidade de simulação e análise. Os benefícios da aplicação do metamodelo da Dinâmica de Sistemas em outros domínios ainda não foram avaliados.

4.2.1 Um Exemplo para Modelagem

Para facilitar a apresentação das definições contidas nas próximas seções, consideremos o trecho de projeto apresentado na Figura 4.3. Esta figura apresenta os artefatos consumidos e produzidos por duas atividades, assim como o desenvolvedor responsável por cada atividade. Os retângulos com bordas arredondadas representam as atividades, cujo nome é indicado em negrito dentro do retângulo. A duração estimada de cada atividade, expressa em número de dias de trabalho, é indicada abaixo do seu nome na figura. As linhas conectando as atividades representam os relacionamentos de precedência entre elas, cuja direção é indicada por pequenas setas ao lado das linhas.

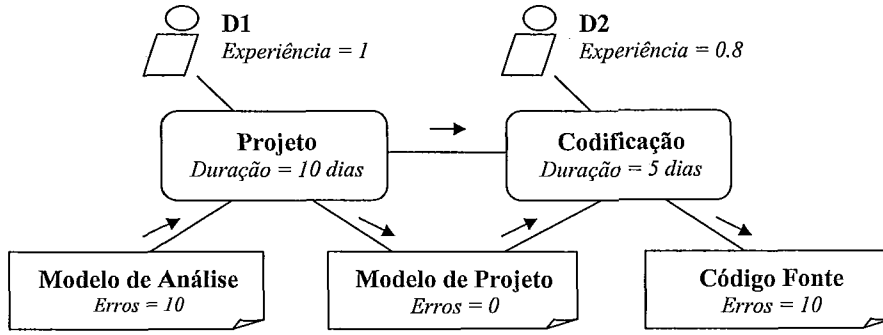

Figura 4.3 – Trecho de projeto utilizado nas definições das próximas seções

Dois desenvolvedores, chamados D1 e D2, projetam e codificam um módulo de software. As figuras acima das atividade representam os desenvolvedores. Os nomes dos desenvolvedores são apresentados em negrito ao lado de suas figuras. Os desenvolvedores possuem experiência, que influencia a qualidade de seu trabalho. A experiência de cada desenvolvedor é apresentada abaixo de seu nome. As linhas que conectam os desenvolvedores e as atividades indicam os desenvolvedores responsáveis por cada atividade do projeto.

As figuras abaixo das atividades representam os artefatos que são consumidos ou produzidos por elas. Os nomes dos artefatos são apresentados em negrito dentro de suas figuras. Os artefatos carregam erros latentes, gerados durante a execução das atividades. Artefatos produzidos a partir de insumos errôneos também conterão erros. O número de erros latentes em um artefato é indicado abaixo de seu nome. As linhas conectando os artefatos e as atividades indicam o consumo ou a produção de artefatos pelas atividades. O consumo ou produção podem ser diferenciados por pequenas setas, apresentadas ao lado das linhas. A atividade de projeto utiliza o modelo de análise como insumo para produzir o modelo de projeto. A atividade de codificação utiliza o modelo de projeto para produzir o código fonte do módulo.

É importante observar que o trecho de projeto apresentado na Figura 4.3 é muito simples e será utilizado apenas para efeito de exemplo. Para um exemplo mais completo de modelo de projeto, consulte o Apêndice B. A dinâmica do modelo mais completo está descrita no Apêndice C. O modelo apresentado no Apêndice B não foi utilizado nos exemplos das próximas seções devido à sua extensão.

4.2.2 Definições

Nesta seção definimos os conceitos relacionados com o metamodelo de Dinâmica de Sistemas. Estes conceitos são referenciados ao longo das próximas seções.

Uma **classe** representa um conjunto de elementos que podem ser descritos pelas mesmas propriedades e exibem comportamento similar. Por exemplo, no trecho de projeto apresentado na Figura 4.3, uma classe descreve todo o conjunto de desenvolvedores. Cada desenvolvedor é denominado uma **instância** da classe. D1 e D2 são instâncias da classe Desenvolvedor.

Uma classe define as propriedades que descrevem suas instâncias. Uma **propriedade** é uma informação relevante sobre uma classe que pode assumir valores distintos para cada instância, de acordo com as características do elemento do mundo real representado por ela. No trecho de projeto apresentado na Figura 4.3, a experiência de um desenvolvedor é uma propriedade da classe Desenvolvedor. D1 e D2 possuem diferentes valores para esta propriedade, respectivamente 1 e 0,8.

Uma classe também define o comportamento de suas instâncias. O **comportamento** de uma classe é uma formulação matemática das respostas produzidas por ela decorrentes de alterações em outras instâncias ou no ambiente. Este comportamento pode depender das propriedades da classe, o que permite que diferentes instâncias de uma mesma classe reajam de forma distinta a uma mesma alteração. O comportamento de uma classe é descrito por um sistema de equações baseado nos construtores da Dinâmica de Sistemas (FORRESTER, 1961).

Uma instância de uma classe pode se relacionar com outras instâncias. Um **relacionamento** representa uma conexão estrutural entre duas ou mais instâncias de classes. Os relacionamentos podem ocorrer entre instâncias de classes distintas ou entre instâncias de uma mesma classe. Este último tipo de relacionamento é denominado um **auto-relacionamento**.

Um **papel** representa a conotação que uma instância de classe assume ao participar de um relacionamento. O papel indica as responsabilidades e o comportamento esperado da instância. No trecho de projeto apresentado na Figura 4.3, o artefato Modelo de Projeto assume o papel de insumo no seu relacionamento com a atividade de codificação.

Finalmente, um **modelo de domínio** contém as classes de elementos que colaboram entre si dentro de um domínio, descrevendo suas propriedades, seu

comportamento e os relacionamentos entre suas instâncias. O modelo de domínio não descreve um modelo para um problema específico, mas uma área de conhecimento para a qual modelos podem ser construídos. O modelo de domínio é uma descrição genérica do domínio, que pode ser especializada para um problema específico. No contexto do gerenciamento de projetos baseado em cenários, existe um modelo de domínio, cujas classes são reutilizadas em todos os modelos de projeto. Este modelo de domínio contém as definições das propriedades, do comportamento e dos relacionamentos entre as classes que representam as atividades, os recursos, os desenvolvedores e os artefatos pertencentes a um projeto de desenvolvimento de software.

4.2.3 O Processo de Modelagem

Nesta seção apresentamos o processo de modelagem baseado nas definições da seção 4.2.2. Este processo pode ser dividido em três passos, apresentados na Figura 4.4: a modelagem de domínio, a instanciação e a compilação de modelos.

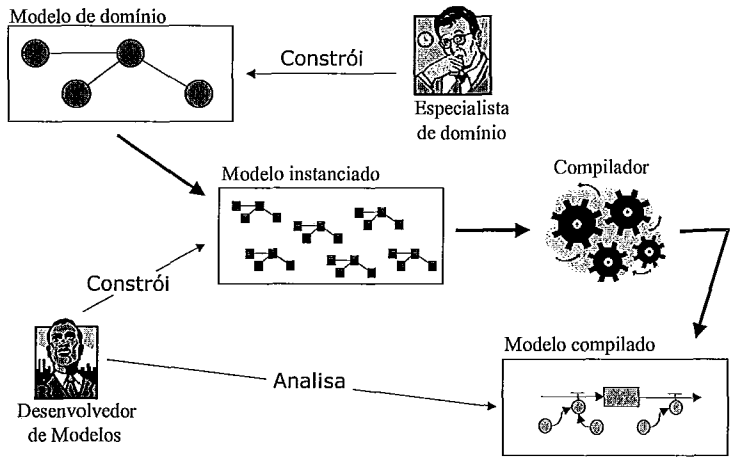

Figura 4.4 – Processo de modelagem baseado no metamodelo da Dinâmica de Sistemas

Inicialmente, um especialista em um domínio desenvolve um modelo de domínio. Este modelo contém as classes dos elementos que compõem o domínio, especificando suas propriedades, seu comportamento e os tipos de relacionamento que podem ocorrer entre as classes. Este modelo não pode ser simulado, uma vez que ele não especifica quantas instâncias de cada classe existem no modelo nem determina os valores de suas propriedades. Este passo é denominado **modelagem de domínio**.

A criação de um modelo específico a partir de um modelo de domínio é o segundo passo do processo de modelagem, onde um desenvolvedor de modelos especifica quantas instâncias de cada classe do modelo de domínio existem no modelo de

interesse. O desenvolvedor de modelos também especifica os valores das propriedades destas instâncias e os relacionamentos existentes entre elas, de acordo com os tipos de relacionamentos definidos entre as classes no modelo de domínio. O modelo resultante contém informações acerca de suas instâncias, sem apresentar construtores da Dinâmica de Sistemas. Estes construtores são herdados do comportamento das classes, que foram previamente descritas no modelo de domínio. Este passo é denominado **instanciação de modelo**.

Finalmente, o modelo construído a partir do modelo de domínio é traduzido para construtores da Dinâmica de Sistemas no terceiro passo do processo de modelagem. Esta tradução resolve as referências entre as instâncias de classes do modelo, realizadas através de seus relacionamentos, permitindo que o modelo seja simulado e analisado em simuladores padrão da Dinâmica de Sistemas. Este passo é denominado **compilação de modelo**.

A aplicação do processo de modelagem proposto exige o levantamento das classes que participam de um domínio, assim como suas propriedades, seu comportamento e os relacionamentos que podem ocorrer entre as instâncias destas classes. Diversas linhas de pesquisa propõem alternativas para a elicitação deste conhecimento, como a análise de domínios (ARANGO, 1988) (BRAGA, 2000), o gerenciamento de conhecimento (FISCHER e OSTWALD, 2001) (PREECE et al., 2001), ontologias (GUARINO, 1998) (FALBO, 1998) (OLIVEIRA, 1999), entre outros. A determinação de uma melhor tecnologia dentre as anteriores ou a criação de uma nova tecnologia para o levantamento do conhecimento necessário para a aplicação do processo de modelagem está fora do escopo deste trabalho.

4.2.4 Modelagem do Domínio

A Tabela 4.1 apresenta a sintaxe BNF[4] simplificada para a linguagem de descrição de modelos de domínio, que são representados através de uma linguagem de modelagem. A sintaxe completa desta linguagem pode ser encontrada no Apêndice D. As palavras reservadas e símbolos terminais utilizados pela linguagem de modelagem são apresentados entre aspas, enquanto os símbolos não terminais são apresentados em fonte itálica.

---

[4] BNF ou ("Backus Naum Form") é uma notação formal para descrever a sintaxe de uma linguagem de programação.

model	= "MODEL" *model_name* "{" {*model_item*} "}" ";"
model_item	= (*class* | *relation*) ";"
class	= "CLASS" *class_name* "{" {*class_item*} "}"
class_item	= (*property* | *behavior* | "PUBLIC" | "PRIVATE") ";"
property	= "PROPERTY" *property_name default_value*
behavior	= (*stock* | *rate* | *proc* | *table*)
stock	= "STOCK" *stock_name expression*
rate	= "RATE" "(" *affected_stock_name* ")" *rate_name expression*
proc	= "PROC" *proc_name expressions*
table	= "TABLE" *table_name table_values*
relation	= (*multi_relation* | single_*relation*)
multi_relation	= "MULTIRELATION" *relation_name source_class* "," *target_ class* [*target_role*]
single_relation	= "RELATION" *relation_name source_class* "," *target_class* [*target_role*]
target_role	= "(" *role_name* ")"

Tabela 4.1- Sintaxe BNF para os modelos de domínio

A Tabela 4.2 apresenta um modelo simplificado para o domínio de gerenciamento de projetos de software. Este modelo se baseia nos elementos utilizados no trecho de projeto da Figura 4.3. Um exemplo mais completo de modelo para este domínio pode ser encontrado no Apêndice B.

O modelo na Tabela 4.2 não especifica nenhum comportamento. Ele apresenta apenas a sintaxe para declaração de classes e dos relacionamentos dentro do modelo de domínio. A palavra reservada MODEL introduz o modelo de domínio, denominado ProjectModel. O modelo contém três classes, cada qual declarada através da palavra reservada CLASS. As classes são declaradas dentro do contexto do modelo de domínio, delimitado por um par de chaves. Cada classe contém seu próprio contexto, também delimitado por chaves, onde suas propriedades e seu comportamento são declarados.

```
MODEL ProjectModel
{
        CLASS Developer
        {
                PROPERTY experience 1;
        };
        CLASS Artifact
        {
                PROPERTY latent_errors 0;
        };
        CLASS Activity
        {
                PROPERTY duration 0;
        };
        MULTIRELATION Precedence Activity, Activity (NextActivities);
        MULTIRELATION Team Activity, Developer;
        MULTIRELATION Income Activity, Artifact;
        RELATION Outcome Activity, Artifact;
};
```

Tabela 4.2 – Modelo simplificado para o domínio de gerenciamento de projetos de software

A palavra reservada PROPERTY especifica uma propriedade de uma classe. O modelo do domínio de gerenciamento de projetos define a propriedade Experience para a classe Developer. Esta propriedade representa o conhecimento e a experiência em projetos de software de um desenvolvedor. Estas características podem afetar a produtividade e a qualidade do trabalho do desenvolvedor, afetando indiretamente a duração do projeto e o número de erros latentes nos artefatos produzidos por ele.

O desenvolvedor de modelos deve determinar quantos desenvolvedores são necessários para o modelo e especificar a experiência de cada desenvolvedor. No trecho de projeto da Figura 4.3 temos dois desenvolvedores, D1 e D2, e seus respectivos graus de experiência, 1 e 0,8. Se o valor de uma propriedade não for especificado para uma instância, esta propriedade assume o valor *default*, que é especificado ao lado do nome da propriedade no modelo de domínio.

As classes Artifact e Activity possuem uma propriedade cada. A propriedade Latent_Errors indica o número esperado de erros em um artefato. Para efeito de exemplo, consideramos que os erros latentes em um artefato utilizado como insumo por uma atividade são propagados para os artefatos produzidos por esta atividade. A propriedade Duration especifica quanto tempo um desenvolvedor com experiência mediana necessita para realizar uma atividade.

O modelo de domínio permite dois tipos de relacionamentos entre classes: relacionamentos múltiplos (ou associações 1:N), onde uma instância da classe origem é associada a diversas instâncias da classe destino, e relacionamentos simples (ou associações 1:1), onde uma instância da classe origem é associada a uma única instância da classe destino. Relacionamentos N:N não são permitidos pelo modelo de domínio, devendo ser decompostos em dois relacionamentos 1:N conectados através de uma classe intermediária.

A palavra reservada RELATION representa um relacionamento simples. O modelo de domínio contém apenas um relacionamento simples, o relacionamento Outcome, que ocorre entre uma atividade e um artefato. Este relacionamento indica que uma atividade produz um único artefato. Ao criar um modelo a partir do modelo de domínio, um desenvolvedor deve indicar o artefato que será produzido por cada atividade. A palavra reservada MULTIRELATION representa um relacionamento múltiplo. O relacionamento Team, que representa os desenvolvedores associados a uma atividade, é um relacionamento múltiplo: diversos desenvolvedores podem estar associados a uma mesma atividade.

Um relacionamento entre duas instâncias de classes permite que as equações que descrevem o comportamento de uma instância acessem e modifiquem o comportamento da outra instância. O modelo de domínio limita as referências ao longo de relacionamento por construtores de visibilidade. Uma instância pode acessar informações de suas instâncias relacionadas apenas quando esta informação for disposta em uma área pública. A palavra reservada PUBLIC marca o início de uma área pública entre as equações de comportamento de uma classe. Informações privativas podem ser utilizadas apenas por outras equações dentro da mesma instância da classe. A palavra reservada PRIVATE marca o início de uma área privativa em uma classe.

A princípio, os relacionamentos do modelo de domínio são unidirecionais, ou seja, apenas a instância origem tem acesso às equações de comportamento da instância destino. Os relacionamentos podem ser alterados para bidirecionais especificando-se um papel para a instância destino. Através do nome deste papel, a instância destino pode manipular o comportamento da instância origem. O modelo de domínio exige que os auto-relacionamentos sejam bidirecionais, ou seja, que um papel seja especificado para sua classe destino.

O relacionamento Team é um relacionamento unidirecional, onde apenas a instância da classe Activity possui acesso às informações acerca de seus desenvolvedores. Para que a associação seja bidirecional, um papel deve ser especificado para a classe destino (neste caso, os desenvolvedores). O nome do papel é indicado entre parênteses, após o nome da classe destino na declaração do relacionamento. O relacionamento Income é similar ao relacionamento Team, sendo múltiplo e unidirecional. Ele representa os artefatos utilizados como insumos por uma atividade.

O relacionamento Precedence é um auto-relacionamento que conecta instâncias da classe Activity. Este relacionamento é múltiplo e bidirecional, utilizando o papel NextActivities em sua atividade destino, conforme requisitado pelo modelo de domínio. A classe destino pode utilizar os nomes dos papéis para acessar informações de suas instâncias associadas.

As referências entre as instâncias de um modelo, representadas por seus relacionamentos e suas equações de comportamento, são resolvidas durante a tradução do modelo para os construtores da Dinâmica de Sistemas. Estas referências podem ser utilizadas para definir o comportamento de uma instância de acordo com o comportamento das instâncias a que esta se encontra associada. Considere, por exemplo,

que após ter observado o comportamento de um modelo, um gerente de projeto decide tentar outras distribuições de desenvolvedores para as atividades do projeto. Um modelo escrito sem a utilização do modelo de domínio precisaria ser profundamente alterado para refletir as mudanças na distribuição da equipe, uma vez que estes relacionamentos estariam embutidos nas equações do modelo. Utilizando o modelo de domínio, o gerente teria apenas que alterar os relacionamentos entre as instâncias para refletir as novas distribuições de desenvolvedores e atividades, traduzindo e simulando o modelo em seguida, sem ter feito nenhuma alteração explícita em suas equações.

```
MODEL ProjectModel
{
   CLASS Developer
   {
        PROPERTY experience 1;
        PROC Productivity experience;
   };
   CLASS Artifact
   {
        PROPERTY latent_errors 0;
        STOCK Errors latent_errors;
   };
   CLASS Activity
   {
        PROPERTY duration 0;
        STOCK TimeToConclude duration;
        RATE (TimeToConclude) Work if(DependOk, -Min(Prod*TimeToConclude/DT, Prod), 0);
        PROC DependOk GROUPSUM (Precedence, TimeToConclude) < 0.001;
        STOCK ExecutingOrDone 0;
        RATE (ExecutingOrDone) RTExecuting if(AND(ExecutingOrDone<0.001, DependOk),1,0);
        RATE (Outcome.Errors) ErrorsTransmit if (RTExecuting>0.001, GROUPSUM(Income,
              Errors)/DT, 0);
        PROC Prod GroupMax (Team, Productivity);
        PROC ErrorsPerDay 5;
        RATE (Outcome.Errors) ErrorsCommitted -1 * ErrorsPerDay * Work / Prod;
   };
   MULTIRELATION Team Activity, Developer;
   MULTIRELATION Precedence Activity, Activity (NextActivities);
   MULTIRELATION Income Activity, Artifact;
   RELATION Outcome Activity, Artifact;
};
```

Tabela 4.3 – Um modelo para o domínio de gerenciamento de projetos de software

A Tabela 4.3 apresenta um modelo de domínio completo para representar os elementos do trecho de projeto da Figura 4.3, contendo classes, propriedades, relacionamentos e comportamento. As equações de comportamento do modelo de domínio estão distribuídas ao longo das classes. As classes Developer e Artifact possuem comportamento muito simples. A classe Developer define apenas um processo para representar o valor de sua propriedade Experience. Este processo torna o valor de

sua propriedade disponível para outras instâncias, uma vez que o valor da propriedade somente pode ser acessado pela própria instância. A classe Artifact contém um único repositório que representa o número esperado de erros latentes no artefato. Os erros podem ser gerados durante a realização da atividade ou recebidos de artefatos errôneos utilizados como insumo. As equações de comportamento que regem a geração de erros durante a realização da atividade e a propagação de erros entre artefatos errôneos estão localizadas na classe Activity.

A classe Activity contém grande parte do comportamento do modelo de domínio. O repositório TimeToConclude indica o tempo necessário para a conclusão de uma atividade. A taxa Work é responsável por reduzir o nível deste repositório. Sua equação utiliza o processo DependOk, que determina se as atividades precedentes da atividade atual foram concluídas. Este processo utiliza o operador GROUPSUM, que soma os valores de uma propriedade para cada instância associada à instância atual através de um relacionamento. No processo DependOk, o operador GROUPSUM soma o nível dos repositórios TimeToConclude para cada atividade que deve ser concluída antes do início da atividade atual. O processo verifica se o valor resultante do operador é próximo de zero, determinando se as atividades já foram concluídas.

As duas equações seguintes, ExecutingOrDone e RTExecuting, são utilizadas para criar uma variável que assume o valor unitário quando a atividade tem início, permanecendo com valor zero no restante do tempo de simulação. Esta variável é utilizada pela taxa ErrorsTransmit, que propaga os erros latentes dos artefatos utilizados como insumos por uma atividade para o artefato produzido pela atividade. Para efeito deste exemplo, assumimos que todos os erros latentes nos artefatos utilizados como insumos serão reproduzidos no artefato produzido por uma atividade. Embora esta assertiva não seja apropriada para todos os modelos de gerenciamento de projetos, ela é utilizada aqui para apresentar um mecanismo denominado **multitaxa**, que permite que uma taxa afete diversos repositórios ao mesmo tempo.

Uma multitaxa é uma taxa que pode ser conectada a diversos repositórios através de um relacionamento. Se o nome do repositório afetado por uma taxa é composto pelo nome de um relacionamento seguido pelo nome de um repositório na classe destino, os repositórios de cada instância associada à instância atual através do relacionamento serão afetados pela taxa. Se o relacionamento for simples, apenas um repositório será afetado. A multitaxa ErrorsTransmit utiliza um relacionamento simples com a classe Artifact para acessar o repositório Errors do artefato utilizado como insumo, somando o

número de erros latentes propagados entre os artefatos no passo de simulação em que a atividade tem início. Uma estratégia similar é utilizada pela taxa ErrorsCommitted para adicionar erros ao artefato produzido a medida que a atividade se realiza.

Finalmente, o processo Prod calcula a produtividade da equipe que realizará a atividade. Para efeito de exemplo, assumimos a equipe é tão produtiva quando seu desenvolvedor mais produtivo. O processo utiliza o operador GROUPMAX para determinar o valor máximo de um comportamento em instâncias associadas com a instância corrente através de um relacionamento. O operador GROUPMIN determina o valor mínimo de um comportamento nas instâncias associadas com a instância corrente através de um relacionamento.

Nesta seção, apresentamos a sintaxe para a definição das classes e dos relacionamentos entre classes que compõem um modelo de domínio. As classes servirão como construtores para o desenvolvimento de modelos a partir do modelo de domínio. O comportamento definido para as classes no modelo de domínio é reutilizado sempre que um desenvolvedor de modelos constrói um modelo para o domínio. Na construção deste modelo, o desenvolvedor apenas especifica as instâncias das classes relacionadas com o problema sendo modelado e especifica os valores de suas propriedades. O modelo será posteriormente traduzido para os construtores tradicionais da Dinâmica de Sistemas, trazendo-se, neste momento, o comportamento definido nas classes para cada uma de suas instâncias. Na próxima seção, apresentamos como um modelo pode ser construído a partir das classes e do conhecimento expresso no modelo de domínio.

4.2.5 Instanciação de Modelo

A Tabela 4.4 apresenta a sintaxe BNF simplificada para os modelos criados a partir de modelos de domínio. A sintaxe BNF completa pode ser encontrada no Apêndice D. Estes modelos também são representados através de uma linguagem de modelagem. As palavras reservadas da linguagem de modelagem são apresentadas entre aspas, enquanto os símbolos não terminais são apresentados em fonte itálica.

model_instance	= "DEFINE" *model_instance_name model_name* "{" {*class_instance*} "}" ";"
instance	= *instance_name* "=" "NEW" *class_name instance_items*
instance_items	= { instance_item ";" }
instance_item	= (*instance_prop* | *instance_link*) ";"
instance_prop	= "SET" *property_name* "=" *property_value*
instance_link	= "LINK" *relation_name links*

Tabela 4.4 – Sintaxe BNF para modelos utilizando um modelo de domínio

Para exemplificar e instanciação de modelos consideremos o trecho de projeto apresentado na Figura 4.3, onde dois desenvolvedores projetam e codificam um módulo a partir de sua especificação. A Tabela 4.5 apresenta um modelo para o projeto.

Sem a utilização do modelo de domínio, o trecho de projeto apresentado na Figura 4.3 deveria ser modelado com equações da Dinâmica de Sistemas. Entretanto, utilizando o processo de modelagem proposto, um especialista em projetos de software desenvolve um modelo de domínio, conforme apresentado na seção 4.2.4, contendo as classes que compõem o projeto (atividade, artefato e desenvolvedor) e definindo suas propriedades, seu comportamento e os relacionamentos que podem ocorrer entre suas instâncias. Através da instanciação, desenvolvedores de modelos reutilizam um modelo de domínio criado pelo especialista, descrevendo apenas as instâncias específicas de cada classe do domínio que fazem parte do problema a ser modelado. Em seguida, os desenvolvedores de modelos determinam os valores das propriedades das instâncias e os relacionamentos existentes entre elas, conforme descrito nas classes do modelo de domínio.

```
DEFINE MyProject ProjectModel
{
        D1 = NEW Developer
                SET Experience = 1;
        D2 = NEW Developer
                SET Experience = 0.8;
        AnalysisModel = NEW Artifact
                SET latent_errors = 10;
        DesignModel = NEW Artifact
                SET latent_errors = 0;
        SourceCode = NEW Artifact
                SET latent_errors = 0;
        Designing = NEW Activity
                SET duration = 10;
                LINK Team D1;
                LINK Income AnalysisModel;
                LINK Outcome DesignModel;
        Coding = NEW Activity
                SET duration = 5;
                LINK Team D2;
                LINK Precedence Designing;
                LINK Income DesignModel;
                LINK Outcome SourceCode;
};
```

Tabela 4.5 – Um modelo para o projeto proposto na Figura 4.3

O modelo na Tabela 4.5 contém as instâncias de classes do modelo de domínio necessárias para descrever o projeto proposto. A palavra reservada DEFINE introduz o modelo de projeto, seguida pelo nome do modelo (MyProject) e pelo modelo de domínio no qual este se baseia (ProjectModel). As instâncias participantes do modelo são representadas dentro de seu contexto, delimitado por chaves. Os desenvolvedores,

D1 e D2, são as primeiras instâncias apresentadas no modelo. A palavra reservada NEW cria uma instância da classe identificada pelo nome apresentado após a palavra reservada. A nova instância é associada ao identificador apresentado no lado esquerdo da atribuição. Após a criação das instâncias dos desenvolvedores, o modelo cria as instâncias dos artefatos e das atividades.

A palavra reservada SET especifica os valores das propriedades para uma instância. Os valores das propriedades são definidos imediatamente após a criação da instância. Conforme indicado no modelo de domínio, valores distintos podem ser atribuídos a mesma propriedade de diferentes instâncias. Por exemplo, a experiência do desenvolvedor D1 é igual a 1, enquanto a experiência do desenvolvedor D2 é igual a 0,8. Esta característica permite que o modelo da Dinâmica de Sistemas considere precisamente as diferenças relevantes entre as instâncias de uma mesma classe, o que exige muitas equações nos modelos tradicionais.

Os relacionamentos são especificados pelas instâncias da classe Activity. Isto ocorre porque esta classe é a única origem de relacionamentos no modelo de domínio, sendo assim responsável pela definição dos relacionamentos nos modelos desenvolvidos a partir do modelo de domínio. A palavra reservada LINK determina as instâncias de classes que são associadas em cada relacionamento. Por exemplo, a atividade de codificação (Coding) utiliza o modelo de projeto (DesignModel) como insumo, é precedida pela atividade de projeto (Designing), é desenvolvida pelo desenvolvedor D2 e produz o código fonte (SourceCode) do módulo.

Os modelos não especificam qualquer comportamento ou construtor da Dinâmica de Sistemas: estes elementos são herdados de suas classes no modelo de domínio, sendo utilizados para traduzir as descrições de alto nível do modelo para equações da Dinâmica de Sistemas, que podem ser simuladas. Este processo de transformação é denominado **compilação de modelo** e é descrito na próxima seção.

4.2.6 Compilação do Modelo para Construtores da Dinâmica de Sistemas

As técnicas apresentadas nas seções anteriores permitem a construção de modelos de domínio e a instanciação de modelos para problemas particulares a partir dos modelos de domínio. Estas técnicas auxiliam na construção de modelos detalhados e complexos, mas perdem sua utilidade se os modelos não podem ser simulados. Para permitir sua simulação, descrevemos nesta seção o processo que traduz a representação de modelos baseada em classes para construtores da Dinâmica de Sistemas, que podem

ser analisados em um simulador convencional (HIGH PERFORMANCE SYSTEMS, 1995, VENTANA SYSTEMS, 1999, BARROS et al., 2000a, BARROS, 2001).

Este processo é denominado **compilação de modelos**. Ele pode ser visto como um processo de geração de artefatos (FRANCA e STAA, 2001), onde uma ferramenta de software produz um artefato (modelo compilado) a partir de sua especificação de alto nível (modelo baseado em conceitos do domínio). Utilizaremos o modelo apresentado na seção 4.2.5 como exemplo. A Tabela 4.6 apresenta sua versão compilada.

```
# Code for object "D1"
PROC  D1_experience 1.0000;
PROC  D1_Productivity D1_experience;

# Code for object "D2"
PROC  D2_experience 0.8000;
PROC  D2_Productivity D2_experience;

# Code for object "AnalysisModel"
PROC  AnalysisModel_latent_errors 10.0000;
STOCK AnalysisModel_Errors AnalysisModel_latent_errors;

# Code for object "DesignModel"
PROC  DesignModel_latent_errors 0.0000;
STOCK DesignModel_Errors DesignModel_latent_errors;

# Code for object "SourceCode"
PROC  SourceCode_latent_errors 0.0000;
STOCK SourceCode_Errors SourceCode_latent_errors;

# Code for object "Designing"
PROC  Designing_duration 10.0000;
STOCK Designing_TimeToConclude Designing_duration;
RATE  (SOURCE, Designing_TimeToConclude) Designing_Work IF (Designing_DependOk,  -MIN
      (Designing_Prod * Designing_TimeToConclude / DT, Designing_Prod), 0.000);
PROC  Designing_DependOk 0 < 0.001;
STOCK Designing_ExecutingOrDone 0.000;
RATE  (SOURCE, Designing_ExecutingOrDone) Designing_RTExecuting IF (AND
      (Designing_ExecutingOrDone < 0.001, Designing_DependOk), 1.000, 0.000);
RATE  (SOURCE, DesignModel_Errors) Designing_ErrorsTransmit1 IF (Designing_RTExecuting
      > 0.001, (AnalysisModel_Errors) / DT, 0.000);
PROC  Designing_Prod MAX (D1_Productivity);
PROC  Designing_ErrorsPerDay 5.000;
RATE  (SOURCE, DesignModel_Errors) Designing_ErrorCommited1  -1.000 *
      Designing_ErrorsPerDay * Designing_Work / Designing_Prod;

# Code for object "Coding"
PROC  Coding_duration 5.0000;
STOCK Coding_TimeToConclude Coding_duration;
RATE  (SOURCE, Coding_TimeToConclude) Coding_Work IF (Coding_DependOk,  -MIN
      (Coding_Prod * Coding_TimeToConclude / DT, Coding_Prod), 0.000);
PROC  Coding_DependOk (Designing_TimeToConclude) < 0.001;
STOCK Coding_ExecutingOrDone 0.000;
RATE  (SOURCE, Coding_ExecutingOrDone) Coding_RTExecuting IF (AND
      (Coding_ExecutingOrDone < 0.001, Coding_DependOk), 1.000, 0.000);
RATE  (SOURCE, SourceCode_Errors) Coding_ErrorsTransmit1 IF (Coding_RTExecuting >
      0.001, (DesignModel_Errors) / DT, 0.000);
PROC  Coding_Prod MAX (D2_Productivity);
PROC  Coding_ErrorsPerDay 5.000;
RATE  (SOURCE, SourceCode_Errors) Coding_ErrorCommited1  -1.000 * Coding_ErrorsPerDay *
      Coding_Work / Coding_Prod;
```

Tabela 4.6 – Modelo tradicional da Dinâmica de Sistemas gerado a partir do modelo da seção 4.2.5

A versão compilada do modelo contém apenas construtores tradicionais da Dinâmica de Sistemas (repositórios, taxas, processos e fluxos), que são representados utilizando a linguagem de simulação adotada pela ferramenta *Illium* (BARROS et al., 2000a, BARROS, 2001). A ferramenta *Illium* é um simulador de modelos da Dinâmica de Sistemas que oferece diversos mecanismos de simulação, incluindo análise no tempo e simulações de Monte Carlo. A ferramenta *Illium* é apresentada no Apêndice E.

Cada construtor possui um nome único, utilizado para identificá-lo nas equações do modelo. Os processos e taxas são descritos por uma expressão, que é avaliada em cada passo de simulação. As taxas são também associadas a dois repositórios, que representam a origem e o destino de seu fluxo. Provedores infinitos, representados pela palavra reservada SOURCE, e receptores infinitos, representados pela palavra reservada SINKER, podem substituir estes repositórios. As tabelas são descritas por uma lista de constantes separadas por vírgulas. Tabelas e receptores infinitos não foram utilizados neste exemplo.

Para tornar mais clara a discussão, doravante chamaremos a representação baseada em classes apenas por modelo, enquanto a versão baseada em construtores da Dinâmica de Sistemas será chamada de modelo compilado. O modelo compilado possui sete blocos de equações, cada qual contendo as equações de uma instância do modelo. Considere as equações geradas para a instância D1 da classe Developer. Estas equações, ressaltadas na Tabela 4.7, contém a declaração de uma propriedade e uma equação de comportamento.

```
PROC  D1_experience 1.0000;
PROC  D1_Productivity D1_experience;
```

Tabela 4.7 – Modelo gerado para a instância do desenvolvedor D1

A primeira equação declara a propriedade Experience da instância do desenvolvedor D1. As propriedades de uma instância são declaradas como processos no modelo compilado, sendo inicializadas com o valor especificado para elas no modelo ou com o valor *default*, indicado no modelo de domínio. O nome do processo que representa a propriedade no modelo compilado é composto pelo nome da instância seguido do nome da propriedade. Os dois nomes, separados por um caractere de sublinhado[5], compõem um nome único para o processo dentro do modelo compilado,

[5] Ou *underscore* ("_").

uma vez que o modelo não permite a declaração de duas instâncias com o mesmo nome e o modelo de domínio não permite a declaração de dois comportamentos com o mesmo nome na mesma classe. Esta composição permite a declaração de diversas instâncias, cada qual com seus valores de propriedades. Processos distintos representam as propriedades de diferentes instâncias no modelo compilado. O mesmo ocorre nas equações geradas para a instância D2.

A segunda equação representa a única equação de comportamento definida para a classe Developer, especializada para a instância D1. A referência à propriedade Experience na equação de comportamento é conectada ao processo que representa a propriedade. Assim como na declaração das propriedades, o nome da instância também é utilizado como prefixo do construtor que representa a equação de comportamento no modelo compilado. A equação de comportamento é repetida para cada instância do modelo compilado. Isto também pode ser observado ao analisarmos as equações geradas para a instância D2.

A Tabela 4.8 focaliza o modelo gerado para a instância da atividade de codificação. A primeira equação do modelo representa a declaração da propriedade Duration da atividade. As duas equações seguintes apresentam as duas primeiras equações de comportamento da classe, parametrizadas por outras equações de comportamento e pela propriedade Duration da instância. A quarta equação apresenta a versão compilada do operador GROUPSUM, que é substituído por uma lista de somas, cujos operandos são participantes do relacionamento selecionado no operador.

Na instância da atividade de codificação, que é precedida por uma única atividade, o operador GROUPSUM é compilado para uma referência a uma equação da instância Designing. Nesta instância, que por sua vez não possui atividades precedentes, o operador GROUPSUM é compilado para zero, pela inexistência de operadores para a soma.

O comportamento DependOk da atividade de codificação utiliza um repositório declarado por outra instância (neste caso, a instância Designing). Este uso decorre do relacionamento entre as atividades, que permite que uma instância consulte ou altere o comportamento de outras instâncias. O comportamento DependOk na classe Activity utiliza o relacionamento Precedence. O processo de compilação percebe o acesso externo a classe através do nome do relacionamento, utilizado nas equações de comportamento. O nome do relacionamento precede o nome do comportamento acessado pela classe origem na classe destino.

```
PROC  Coding_duration 5.0000;

STOCK Coding_TimeToConclude Coding_duration;

RATE  (SOURCE, Coding_TimeToConclude) Coding_Work IF (Coding_DependOk,  -MIN
      (Coding_Prod * Coding_TimeToConclude / DT, Coding_Prod), 0.000);

PROC  Coding_DependOk (Designing_TimeToConclude) < 0.001;

STOCK Coding_ExecutingOrDone 0.000;

RATE  (SOURCE, Coding_ExecutingOrDone) Coding_RTExecuting IF (AND
      (Coding_ExecutingOrDone < 0.001, Coding_DependOk), 1.000, 0.000);

RATE  (SOURCE, SourceCode_Errors) Coding_ErrorsTransmit1 IF (Coding_RTExecuting >
      0.001, (DesignModel_Errors) / DT, 0.000);

PROC  Coding_Prod MAX (D2_Productivity);

PROC  Coding_ErrorsPerDay 5.000;

RATE  (SOURCE, SourceCode_Errors) Coding_ErrorCommited1  -1.000 * Coding_ErrorsPerDay *
      Coding_Work / Coding_Prod;
```

Tabela 4.8 - Modelo gerado para a instância da atividade de codificação

As duas equações seguintes apenas repetem o comportamento especificado na classe para a instância atual. A taxa Coding_ErrorsTransmit contém o modelo gerado para a multitaxa ErrorsTransmit declarada na classe Activity. A multitaxa é compilada para diversas equações, cada qual representando uma taxa distinta que afeta um repositório associado à multitaxa. Como a taxa ErrorsTransmit está associada a um relacionamento simples no exemplo, o modelo gerado para ela contém apenas uma taxa, que afeta o repositório que representa o número de erros latentes no artefato produzido (no caso, o código fonte do módulo de software). Um número é adicionado ao nome da taxa no modelo gerado para a multitaxa: ele é utilizado para diferenciar as diversas taxas que podem ser geradas para representar a multitaxa. O acesso ao repositório Errors na instância DesignModel também resulta da utilização do operador GROUPSUM.

A próxima equação utiliza o operador GROUPMAX. O código gerado para este operador é similar ao código gerado para o operador GROUPSUM: ao invés de somas aritméticas entre seus operandos, o operador GROUPMAX calcula o valor máximo destes operandos utilizando o operador MAX. As últimas duas equações apresentam, respectivamente, a especialização de uma equação de comportamento para a instância e o modelo gerado para outra multitaxa.

4.2.7 Uma Comparação com uma Abordagem Orientada a Objetos

O metamodelo para a Dinâmica de Sistemas pode ser comparado com as extensões para modelagem orientada a objetos propostas recentemente pela ferramenta PowerSim (MYRTVEIT, 2000). As duas abordagens representam tentativas distintas para aumentar o nível de abstração do processo de modelagem da Dinâmica de Sistemas,

através da construção de modelos que possam ser reutilizados por outros desenvolvedores de modelos e conectados entre si para a construção de modelos mais complexos. Entretanto, existem claras diferenças entre as duas abordagens.

Primeiro, acreditamos que o metamodelo da Dinâmica de Sistemas é mais simples que as extensões propostas pela ferramenta PowerSim. O metamodelo define poucos conceitos novos e utiliza os construtores tradicionais da Dinâmica de Sistemas para descrever o comportamento das classes representadas no modelo de domínio. A proposta de PowerSim envolve diversos conceitos novos, como conectores, conexões e construtores de visibilidade complexos.

Além disso, o metamodelo se propõe a descrever um domínio de modelagem, focalizando nas classes de elementos que compõem o domínio, seus relacionamentos, suas propriedades e seu comportamento. O metamodelo permite que um especialista no domínio defina uma linguagem de modelagem alto nível, enquanto desenvolvedores de modelos menos experientes constróem modelos específicos para o domínio a partir do conhecimento expresso pelo especialista. As extensões de PowerSim permitem a construção de componentes reutilizáveis independentes de domínio, que podem ser agregados a construtores tradicionais da Dinâmica de Sistemas externos ou a outros componentes através de uma interface.

Finalmente, embora as extensões de PowerSim permitam a definição de componentes de alto nível para a construção de modelos, estas extensões não definem claramente os relacionamentos existentes entre estes componentes. Estes relacionamentos precisam ser estabelecidos através de construtores tradicionais da Dinâmica de Sistemas, em um nível de abstração muito inferior ao dos componentes. O foco do domínio de modelagem sugerido pelo metamodelo da Dinâmica de Sistemas, ao invés da definição de componentes independentes, permite que os desenvolvedores de modelos trabalhem apenas com conceitos mais próximos ao domínio do problema, em um nível de abstração mais alto. Os relacionamentos entre as classes dos modelos de domínio e os papéis destas classes são claramente estabelecidos, não exigindo a intervenção do desenvolvedor de modelos nas equações de comportamento para a conexão destes conceitos.

Nesta seção e nas seções anteriores apresentamos o modelo de projeto, sua representação e o processo de modelagem que permite que este modelo seja construído a partir de construtores de alto nível, mais próximos ao domínio do problema que as

equações da Dinâmica de Sistemas. Na próxima seção, apresentamos os modelos de cenários, sua representação e integração com os modelos de projeto.

4.3 Os Modelos de Cenários

Cenários representam eventos, políticas, procedimentos, ações e estratégias gerenciais que não podem ser considerados parte de um projeto de desenvolvimento de software, mas práticas impostas ou aplicadas sobre o projeto ou situações excepcionais que o gerente pode encontrar ao longo do projeto. O paradigma do gerenciamento de projetos baseado em cenários determina que o gerente de projetos deve planejar o comportamento do projeto separadamente dos cenários que podem afetá-lo, utilizando simulações para testar o impacto de combinações destes cenários sobre o comportamento do projeto.

Os cenários estão fortemente relacionados com elementos de projeto, ou seja, as atividades, desenvolvedores, recursos e artefatos utilizados em um projeto de desenvolvimento de software. Os cenários podem ser classificados em quatro categorias:

- Eventos potenciais: estes cenários descrevem os eventos incertos que podem afetar um projeto de desenvolvimento de software. Estes eventos podem alterar o comportamento original do projeto, afetando seus resultados, como sua data de conclusão, seu custo total de desenvolvimento e sua qualidade. Cenários de eventos potenciais são associadas a elementos de projeto específicos, como tecnologias, papéis assumidos por desenvolvedores, artefatos e domínios de aplicação. Por exemplo, a utilização de técnicas de JAD pode apresentar cenários, como um aumento no tempo dedicado à análise de requisitos e redução na volatilidade de requisitos ao longo do projeto;

- Políticas gerenciais: estes cenários representam políticas e procedimentos de gerenciamento de projetos que podem ser impostos a um projeto por padrões organizacionais ou pelo alto gerenciamento da organização. Como nos cenários de eventos potenciais, os eventos de políticas gerenciais estão associados a elementos de projeto específicos. Exemplos de cenários de políticas gerenciais incluem atrasos na contratação de novos funcionários, alta rotatividade na equipe de desenvolvimento, atrasos na aquisição de recursos necessários para a continuidade do projeto e propensão da organização em contratar novos desenvolvedores. Políticas e

procedimentos gerenciais representam padrões de comportamento táticos, que influenciam um grande número de projetos de software construídos no mesmo ambiente de desenvolvimento;

- Teorias: cenários de teorias representam padrões de comportamento que o gerente acredita que possam ter impacto sobre o projeto. Estes cenários representam teorias estabelecidas (raras na ciência de gerenciamento) e teorias hipotéticas, resultantes de senso comum ou da experiência de especialistas. Estes cenários representam o comportamento resultante das interações entre diversos elementos de projetos de software. Por exemplo, ABDEL-HAMID e MADNICK (1991) apresentam uma teoria de propagação de erros que pode ser representada como um cenário de teoria. Teorias representam padrões de comportamento tático que podem influenciar um grande número de projetos, independente do seu ambiente de desenvolvimento;
- Estratégias: estes cenários representam decisões que o gerente pode tomar em relação a um projeto específico. Eles são úteis para testar o impacto de curto e longo prazo das decisões sobre o comportamento do projeto. Exemplos de cenários estratégicos incluem a substituição do desenvolvedor responsável por uma atividade, a imposição de marcos de projeto e a criação de uma equipe de exploração para tratar uma incerteza. Estratégias representam decisões específicas para um projeto, que são fortemente acopladas aos elementos que compõem este projeto.

Os cenários representam um repositório de conhecimento reutilizável para os gerentes de projeto. Eles permitem a documentação das assertivas e informações conhecidas sobre os elementos de projeto e seus relacionamentos. Esta informação pode ser reutilizada por projetos que utilizem os mesmos elementos. Os cenários podem ser organizados em uma base de conhecimento centralizada da organização. Durante o desenvolvimento de uma aplicação, quando o gerente determina o domínio da aplicação, as tecnologias a serem aplicadas, os desenvolvedores, os artefatos a serem construídos, entre outros, os cenários associados a estes elementos auxiliam o gerente na exploração das incertezas que podem afetar seu projeto.

Os cenários também podem ser utilizados nas atividades de treinamento. A combinação de diversas políticas de gerenciamento, teorias e eventos pode ser aplicada sobre um modelo de projeto. Participantes do treinamento podem selecionar as estratégias de gerenciamento, aplicá-las sobre o modelo de projeto e analisar seu

impacto sobre o comportamento do projeto no curto e no longo prazo. Acreditamos que a separação dos fatos conhecidos de um projeto de seus cenários, que são incertos ou representam assertivas particulares, facilita a integração "plug-&-play" de diversas estratégias de gerenciamento e a análise de suas conseqüências em um projeto.

Modelos de cenário são modelos abstratos. Eles não podem ser analisados isoladamente, devendo ser integrados com um modelo de projeto antes de simulados. Esta integração ocorre através de uma interface de integração, representada pelas equações do modelo de projeto que o cenário pode consultar ou alterar. Na próxima seção apresentamos a representação dos modelos de cenário, enfatizando sua interface de integração com os modelos de projeto.

4.4 Representação de Modelos de Cenários

Os modelos de cenários são representados através da sintaxe estendida da Dinâmica de Sistemas, que se baseia no processo de modelagem descrito na seção 4.2.3. A sintaxe BNF para a representação de cenários pode ser encontrada no Apêndice D. Os modelos de cenários são desenvolvidos para afetar ou consultar o comportamento das instâncias de um modelo. Assim, um modelo de cenário está sempre associado a um modelo de domínio, contendo referências para as classes deste domínio.

```
SCENARIO ProductivityDueLearning ProjectModel
{
    CONNECTION TheDeveloper Developer
    {
       STOCK DaysInProject 0;
       RATE  (DaysInProject) DaysInProjectCounter 1;

       PROC  DIPFactor MIN (DaysInProject / 42, 1.0);
       PROC  DIPModifier LOOKUP (LearningTable, DIPFactor, 0, 1);
       TABLE LearningTable 1.0, 1.0125, 1.0325, 1.055, 1.09, 1.15, 1.2, 1.22, 1.245,
             1.25, 1.25;

       AFFECT Productivity Productivity * DIPModifier;
    };
};
```

Tabela 4.9 – Cenário que representa o efeito do tempo de participação no projeto sobre a produtividade

A Tabela 4.9 apresenta um modelo de cenário que descreve o efeito do tempo de participação de um desenvolvedor em um projeto sobre sua produtividade. As equações de comportamento do cenário foram adaptadas do modelo de ABDEL-HAMID e MADNICK (1991), assumindo um domínio de aplicação simples, onde o desenvolvedor leva cerca de dois meses (42 dias úteis) para atingir o total aprendizado dos conceitos do

domínio. A palavra reservada SCENARIO introduz um modelo de cenário. Ela é seguida do nome do cenário (ProductivityDueLearning) e do nome do modelo de domínio ao qual o cenário está associado (ProjectModel). O nome do cenário deve ser único entre os cenários desenvolvidos para o domínio. O modelo de cenário possui um contexto, delimitado por um par de chaves, onde são declaradas suas conexões e suas restrições.

Uma conexão representa um relacionamento entre o cenário e uma classe do modelo de domínio. Quando um cenário é integrado a um modelo construído a partir de um modelo de domínio, suas conexões devem ser especificadas entre as instâncias do modelo. A palavra reservada CONNECTION indica uma conexão, sendo seguida pelo nome da conexão (TheDeveloper) e pelo nome da classe a que a conexão se refere (Developer). O nome da conexão deve ser único entre as conexões do cenário.

Dentro do contexto de uma conexão, também delimitado por um par de chaves, um modelo de cenário pode definir propriedades, comportamentos e ajustes provocados pela conexão. Uma propriedade de uma conexão indica uma nova característica da instância associada a ela, que deve ser especificada pelo desenvolvedor de modelos. Como nas propriedades do modelo de domínio, uma propriedade de conexão define um valor *default*, que será utilizado quando o desenvolvedor de modelos não indicar o valor da propriedade na instância associada à conexão.

Ao definir uma propriedade de uma conexão, o modelo de cenário permite a especificação de uma característica não contemplada originalmente no modelo de domínio para uma instância. Este é um dos mecanismos que permite que os modelos de cenários sejam utilizados como extensão dos modelos de domínio. O modelo de domínio, a partir do qual serão construídos os modelos de projetos, define apenas as características fundamentais dos elementos do domínio, delegando a definição das características utilizadas apenas para analisar determinadas incertezas para os modelos de cenário.

Um mecanismo de extensão similar ao apresentado para as propriedades também é utilizado na definição dos comportamentos das conexões. Uma equação de comportamento definida para uma conexão será repetida em cada instância do modelo que atenda a esta conexão. Uma equação de comportamento de uma conexão pode referenciar outras equações de comportamento da conexão, equações da classe, relacionamentos da classe e propriedades da conexão. A definição de novas equações para uma instância permite que o modelo de cenário possua uma dinâmica própria. Esta

dinâmica pode considerar as demais equações de comportamento da instância, podendo inclusive afetar este comportamento. Entretanto, como o comportamento de uma conexão não pode substituir equações previamente definidas para a instância, a única forma deste comportamento afetar as equações da instância ocorre através da definição de novas taxas para um repositório definido na classe da instância.

Entretanto, em algumas situações pode ser necessário que o modelo de cenário altere uma equação de comportamento original da instância. Estas equações encontram-se nos processos e nas taxas, ou seja, não podem ser afetadas pelo recurso apresentado no parágrafo anterior, que somente é efetivo em repositórios. Os ajustes de uma conexão permitem que o modelo de cenário redefina as equações de processos e taxas de uma classe do modelo de domínio. Um ajuste é declarado através da palavra reservada AFFECT, que é seguida pelo nome do comportamento (processo ou taxa) afetado pelo ajuste e pela sua nova equação. Uma referência ao comportamento ajustado na equação de ajuste faz com que o comportamento anterior seja utilizado na nova equação. No cenário apresentado na Tabela 4.9, por exemplo, a equação de produtividade de um desenvolvedor é afetada por um ajuste, seu novo valor sendo calculado por uma transformação não linear do tempo de participação do desenvolvedor no projeto e da produtividade original.

Antes do cenário	`PROC Productivity Experience;`
Depois do cenário	`PROC DIPModifier LOOKUP (LearningTable, DIPFactor, 0, 1);` `PROC Productivity (Experience) * DIPModifier;`

Tabela 4.10 – Equação de produtividade de um desenvolvedor após a integração do cenário

É importante observar que, devido à precedência dos operadores aritméticos e funções que podem ser utilizados nas equações de comportamento ajustadas (ver sintaxe BNF na seção 4.2.4), a ordem em que os modelos de cenário são ativados sobre uma instância é relevante. Isto ocorre porque as equações de ajustes das conexões dos cenário podem referenciar a equação original de um comportamento da classe. Assim, se diversas conexões de cenários são impostas sobre uma mesma instância, a ordem em que as equações originais são referenciadas nos cenários pode provocar diferentes comportamentos no modelo compilado.

Por exemplo, consideremos dois cenários hipotéticos que afetem a duração de uma atividade de codificação em um modelo de projeto. Consideremos que o primeiro cenário represente a necessidade de adicionar assertivas de depuração no código e que

ele indique que a duração da atividade deve ser aumentada de X horas. O segundo cenário, por sua vez, representa a influência da experiência da equipe na utilização da linguagem de programação selecionada. Consideremos que o segundo cenário sugira que a duração da atividade deve ser corrigida por um fator multiplicativo Y, que é função da experiência da equipe na linguagem. A Tabela 4.11 apresenta os resultados da aplicação dos dois cenários, em diferentes ordens, sobre uma atividade de codificação cuja duração foi originalmente estimada como D dias. Existem diversas combinações de valores para X e Y que fazem com que os resultados apresentados para D' sejam dependentes da ordem de aplicação dos cenários.

Ordem de Aplicação dos Cenários	Duração Resultante para a Atividade
Cenário 1 antes, cenário 2 depois	D' = (D + X) * Y
Cenário 2 antes, cenário 1 depois	D' = D * Y + X

Tabela 4.11 – Efeito da ordem de aplicação em dois cenários hipotéticos

Em diversas situações as equações de comportamento de uma conexão de modelo de cenário podem assumir que o mesmo ou outro cenário foi aplicado sobre a mesma instância ou sobre instâncias associadas à instância atual. Considere os cenários da Tabela 4.12, que representam o efeito do trabalho em horas extras sobre a produtividade dos desenvolvedores e o tempo que os desenvolvedores aceitam trabalhar em horas extras, devido a sua exaustão.

O primeiro cenário indica o ganho de produtividade e o crescimento da taxa de produção de erros a medida que um desenvolvedor trabalha em regime de horas extras. O cenário assume um regime normal como oito horas de trabalho por dia. O cenário, cujas equações foram adaptadas do modelo de ABDEL-HAMID e MADNICK (1991), representa a constatação de DEMARCO (1982), de que a qualidade do trabalho produzido em horas extras é geralmente inferior à qualidade do trabalho produzido por desenvolvedores trabalhando em regime normal.

O segundo cenário analisa a exaustão de um desenvolvedor trabalhando em regime de horas extras durante um intervalo de tempo. O cenário indica que, durante um período de trabalho em horas extras, o desenvolvedor progressivamente se exaure. Ao atingir um determinado grau de exaustão, o desenvolvedor retorna a trabalhar no máximo oito horas por dia, independente de pressões gerenciais para a manutenção do regime de horas extras. O desenvolvedor continua trabalhando em regime normal até que esteja descansado do período de horas extras. Somente então o desenvolvedor aceita

novamente um regime de horas extras. As premissas e equações deste modelo também foram adaptadas do modelo de ABDEL-HAMID e MADNICK (1991).

```
SCENARIO Overworking ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      PROPERTY WorkHours 8;          # 8 to 12 hours

      STOCK DailyWorkHours WorkHours;
      PROC  WHModifier 1 + (DailyWorkHours - 8) / (12 - 8);

      PROC  SEModifier LOOKUP (SchErrorsTable, WHModifier-1, 0, 1);
      TABLE SchErrorsTable 0.9, 0.94, 1, 1.05, 1.14, 1.24, 1.36, 1.5;

      AFFECT Productivity Productivity * WHModifier;
   };
};

SCENARIO Exhaustion ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      STOCK Exhaustion 0;
      PROC  MaxExhaustion 50;
      RATE  (Exhaustion) ExhaustionRate if(Resting = 1, -MaxExhaustion / 20.0,
            EXModifier);

      PROC  EXModifier LOOKUP (ExaustionTable, DedicationFactor, 0, 1.5);
      PROC  DedicationFactor 1 - (1 - Dedication) / 0.4;
      PROC  Dedication 0.6 + (WHModifier - 1) * (1.2 - 0.6);
      TABLE ExaustionTable 0.0, 0.0, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1.15, 1.6, 1.9,
            2.2, 2.5;

      STOCK Resting 0;
      RATE  (Resting) RestingRate1 IF (InitResting, 1 / DT, 0);
      RATE  (Resting) RestingRate2 IF (QuitResting, -1 / DT, 0);
      RATE  (DailyWorkHours) DWHRate IF (Resting = 1, (8 - DailyWorkHours) / DT, 0);

      PROC  InitResting AND(Resting = 0, Exhaustion > MaxExhaustion);
      PROC  QuitResting AND(Resting = 1, Exhaustion < 0.1);
   };

   CONSTRAINT TheDeveloper, Overworking.TheDeveloper;
};
```

Tabela 4.12 – Cenários que representam o trabalho em horas extras e a exaustão de desenvolvedores

As restrições de um cenário permitem que este indique as conexões de outro ou do mesmo cenário que são esperadas nas instâncias afetadas por suas próprias conexões. O segundo cenário da Tabela 4.12 define uma restrição que exige que o desenvolvedor associado a única conexão deste cenário também esteja associado a uma conexão do cenário de trabalho extra (primeiro cenário da mesma tabela). A palavra reservada CONSTRAINT indica uma restrição em um cenário. A restrição possui dois parâmetros, que indicam, respectivamente, uma instância ou um conjunto de instâncias que serão verificadas e o cenário e respectiva conexão que devem ser aplicados sobre estas instâncias. Assim, o primeiro parâmetro da restrição indica o nome de uma conexão do cenário ou o nome de uma conexão seguido de um nome de relacionamento da classe associada à conexão no modelo de domínio. O segundo parâmetro indica o

nome do cenário e da conexão que deve ser ativada sobre as instâncias indicadas no primeiro parâmetro.

No caso do segundo modelo de cenário da Tabela 4.12, a restrição indica que todas as instâncias onde a conexão TheDeveloper do cenário Exhaustion for ativada também devem ser alvos da conexão TheDeveloper do modelo de cenário Overworking. Esta exigência de conexão ocorre porque a taxa DWHRate do segundo cenário afeta o repositório DailyWorkHours declarado pelo primeiro cenário. Se o primeiro cenário não for ativado sobre o desenvolvedor no qual o segundo cenário foi ativado, o modelo ficará inconsistente, acessando um repositório que não foi declarado. A restrição declarada no segundo cenário permite que o processo de tradução dos modelos verifique a inconsistência antes de gerar o modelo compilado.

4.5 A Integração Entre Modelos de Projeto e Cenários

A integração entre os modelos de projeto e os modelos de cenários é representada nos modelos de projeto. Após a definição das instâncias, conforme especificado na seção 4.2.5, um modelo de projeto pode indicar as ativações de cenários sobre estas instâncias. A Tabela 4.13 apresenta a ativação do cenário ProductivityDueLearning sobre os desenvolvedores D1 e D2.

A palavra reservada ACTIVATE indica a ativação de um cenário sobre um modelo de projeto. Ela é seguida pelo nome do cenário a ser ativado e por suas conexões no modelo. Cada conexão é representada pela palavra reservada CONNECT, seguida do nome da conexão do cenário e do nome da instância do modelo de projeto que será afetada pela conexão. Na primeira ativação de cenário do modelo da Tabela 4.13, temos o desenvolvedor D1 sendo alvo da conexão TheDeveloper definida pelo cenário ProductivityDueLearning.

```
DEFINE MyProject ProjectModel
{
        ...

        ACTIVATE ProductivityDueLearning
                CONNECT TheDeveloper D1;

        ACTIVATE ProductivityDueLearning
                CONNECT TheDeveloper D2;
};
```

Tabela 4.13 – Ativação de cenários sobre instâncias do modelo de projeto

Quando um cenário é ativado sobre um modelo, as equações de suas conexões são repetidas em cada instância afetada por estas conexões. Além disso, os ajustes das

conexões alteram as equações originais definidas na classe da instância ou em conexões de cenários previamente ativados sobre a instância. O processo de tradução dos modelos é responsável por tratar as conexões de cenários, gerando as equações das conexões parametrizadas para as instâncias afetadas. A Tabela 4.14 apresenta as novas equações geradas para o desenvolvedor D1 após a integração do cenário ProductivityDueLearning.

```
PROC D1_Experience 1.000000;
PROC D1_Productivity (D1_Experience) * D1_DIPModifier;
STOCK D1_DaysInProject 0.000000;
RATE (SOURCE, D1_DaysInProject) D1_DaysInProjectCounter 1.000000;
PROC D1_DIPFactor MIN (D1_DaysInProject / 20.000000, 1.000000);
PROC D1_DIPModifier LOOKUP (D1_LearningTable, D1_DIPFactor, 0.000000, 1.000000);
TABLE D1_LearningTable 1.0, 1.0125, 1.0325, 1.055, 1.09, 1.15, 1.2, 1.22, 1.245, 1.25,
      1.25;
```

Tabela 4.14 – Equações que descrevem o desenvolvedor D1 após a integração do cenário da Tabela 4.9

A primeira equação gerada para o desenvolvedor D1 representa sua única propriedade, sem apresentar alterações com relação ao modelo gerado anteriormente (ver seção 4.2.6). A segunda equação representa o ajuste realizado pelo cenário sobre a equação que originalmente representava a produtividade do desenvolvedor. A equação original aparece entre parênteses, multiplicada pelo fator de ajuste especificado no cenário. As equações seguintes representam as equações definidas na conexão do cenário e especializadas para a instância que representa o desenvolvedor D1. Os nomes destas equações receberam o nome da instância como prefixo.

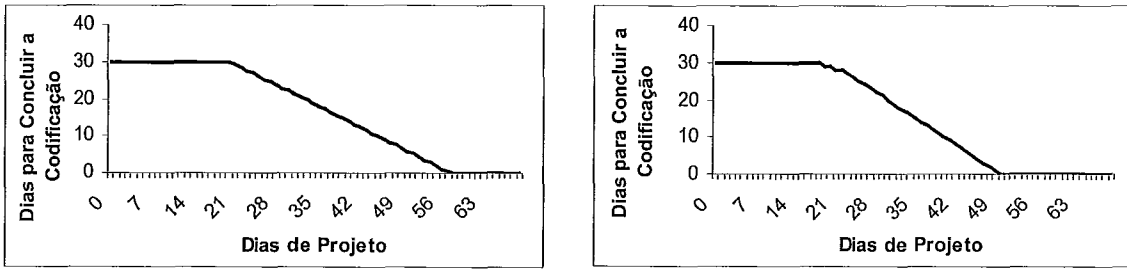

Figura 4.5 - Gráficos de simulação com e sem a ativação de cenários sobre um modelo de projeto

As equações do modelo gerado com a ativação dos cenários são diferentes do modelo sem os cenários. Assim, o comportamento dos dois modelos pode ser diferente. A Figura 4.5 contém dois gráficos de simulação que apresentam a variação ao longo do tempo de simulação da equação de comportamento TimeToConclude da atividade Coding. Este comportamento indica o tempo de trabalho esperado para a conclusão da atividade de codificação. A atividade é considerada concluída quando o valor deste

comportamento atinge zero, durante o período de simulação. Como a atividade de codificação é a última atividade do trecho de projeto apresentado na Figura 4.3, a conclusão desta atividade também representa a conclusão do trecho de projeto.

O gráfico da esquerda na Figura 4.5 apresenta o modelo de projeto da seção 4.2.5 sem as ativações de cenários. O gráfico da direita apresenta o mesmo modelo com a ativação de cenários indicadas na Tabela 4.13. Observe a mudança de comportamento do modelo: enquanto o modelo de projeto não integrado aos cenários leva cerca de 63 dias para ser concluído, o modelo de projeto integrado aos cenário é concluído em apenas 50 dias. Este encerramento mais rápido do projeto se deve ao ganho de produtividade dos desenvolvedores modelado pelo cenário. Assim, se o gerente de projeto acredita que este ganho pode ser obtido, ou seja, que a teoria representada pelo cenário ocorre na prática, a análise de cenários demonstra o efeito do cenário sobre a descrição específica de seu projeto.

4.5.1 O Processo de Otimização do Modelo Compilado

Após a integração dos cenários ao modelo de projeto, a tradução do modelo resultante para os construtores da Dinâmica de Sistemas gera as equações que compõem o modelo compilado. Devido ao processo de repetição das equações das classes e das conexões para cada instância do modelo, o modelo compilado pode ser muito extenso e conter diversas equações redundantes. Esta redundância não altera o comportamento do modelo, mas prejudica o desempenho do simulador, aumentando o tempo necessário para a realização de um passo de simulação do modelo.

Para extrair a redundância dos modelos compilados, propomos um otimizador que analise as equações provenientes do processo de tradução dos modelos, removendo as equações duplicadas e simplificando as expressões das demais equações. O processo de otimização deve ser invisível ao usuário e realizar os seguintes passos:

- Substituição de expressões constantes: as referências a equações cuja expressão seja descrita apenas por uma constante podem ser substituídas pela constante utilizada na expressão da equação original. Assim, após todas as substituições de suas referências, as equações constantes podem ser eliminadas, reduzindo o tamanho do modelo a ser simulado;
- Substituição de expressões de referência: diversas expressões de equações do modelo podem ser descritas apenas por uma referência a outra equação do modelo.

Considere, por exemplo a equação que descreve a produtividade do desenvolvedor D1 no modelo apresentado na seção 4.2.6. A expressão desta equação apenas referencia a equação que descreve a experiência do desenvolvedor. Referências à equação de produtividade podem ser transformadas em referências à equação de experiência, removendo-se a equação de produtividade do modelo;

- Eliminação de tabelas: referências a duas ou mais tabelas com os mesmos valores podem ser substituídas por referências a uma única tabela, removendo-se as tabelas duplicadas;
- Redução de expressões: diversas reduções podem ser aplicadas sobre os operadores utilizados nas expressões das equações do modelo. Por exemplo, se uma referência a equação ou uma constante é apresentada entre parênteses, os parênteses podem ser removidos. Uma seqüência de somas (ou qualquer outra operação aritmética) de constantes pode ser resolvida e substituída pela constante resultante da operação. Funções, como máximo, mínimo, exponencial e logaritmo, utilizando apenas parâmetros constantes podem ser resolvidas e substituídas pelas constantes resultantes.

Como a redução de expressões pode gerar novas expressões constantes ou descritas apenas por referências a outras equações, o processo de otimização é executado iterativamente até que nenhuma nova redução possa ser realizada ou até que um limite de iterações seja atingido.

A otimização geralmente reduz de forma significativa o número de equações do modelo compilado. Por exemplo, a versão compilada do modelo apresentado na seção 4.2.5, quando integrado ao cenário de ganho de produtividade devido ao tempo de participação no projeto, apresentado na Tabela 4.9, apenas no desenvolvedor D1 possui 36 equações. Sua versão otimizada possui apenas 23 equações, o que representa um ganho de cerca de 36%.

```
STOCK D1_DaysInProject 0.000000;
RATE (SOURCE, D1_DaysInProject) D1_DaysInProjectCounter 1.000000;
PROC D1_DIPFactor MIN (D1_DaysInProject / 20.000000, 1.000000);
PROC D1_DIPModifier LOOKUP (D1_LearningTable, D1_DIPFactor, 0.000000, 1.000000);
TABLE D1_LearningTable 1.0, 1.0125, 1.0325, 1.055, 1.09, 1.15, 1.2, 1.22, 1.245, 1.25,
      1.25;
```

Tabela 4.15 – Equações do modelo apresentado na Tabela 4.14 após a otimização

A Tabela 4.15 apresenta a versão otimizada do modelo gerado para o desenvolvedor D1 após a integração do cenário que modela seu ganho de produtividade de acordo com o tempo de projeto. Neste modelo restaram apenas as equações da conexão do cenário. A equação que representava a experiência do desenvolvedor foi substituída da equação que representa sua produtividade. Esta equação, por sua vez, foi reduzida ao produto de uma expressão constante (o valor unitário que representa a experiência do desenvolvedor) pelo fator de produtividade do cenário. Tendo a expressão constante sido reduzida e a equação restante sendo descrita somente pela referência ao fator de produtividade, as referências a esta expressão também foram substituídas por referências ao fator e a equação foi removida do modelo.

4.6 Conclusão

Neste capítulo apresentamos uma representação para o conhecimento gerencial que captura o efeito de teorias, ações, políticas e procedimentos gerenciais sobre um projeto de desenvolvimento de software. O projeto é descrito por um modelo formal, representado a partir de suas atividades, desenvolvedores, recursos e artefatos. As teorias, ações, procedimentos e políticas gerenciais que podem afetar o projeto são descritas por modelos separados, denominados modelos de cenários.

É importante frisar que não estamos gerando novos conhecimentos gerenciais: nosso interesse se restringe a representar este conhecimento de uma forma que ele possa ser aplicado sobre uma representação de um projeto, mostrando os efeitos de sua ocorrência sobre o projeto. Acreditamos que uma representação com esta funcionalidade pode ser superior às tradicionais heurísticas de gerenciamento de projetos, cuja representação ambígua – geralmente textual – não permite que seus efeitos sobre um projeto específico sejam avaliados formalmente.

Acreditamos que os modelos de cenário, enquanto formato para representação de conhecimento gerencial, possuem uma característica fundamental: a capacidade de transferir conhecimento. Muitas representações de conhecimento se baseiam em formatos ambíguos, como documentos textuais, imagens, vídeo e som. Consideramos que estas representações carregam informação, mas que o conhecimento somente pode ser adquirido através da interpretação destas informações em um contexto, ou seja, através de uma análise do impacto das novas informações sobre o contexto no qual elas podem ser aplicadas. Representações informais de conhecimento, como aquelas

baseadas em formatos ambíguos, raramente podem prover auxílio automatizado na transferência de conhecimento, deixando sua interpretação para o usuário interessado. Os modelos de cenários, através de sua integração com os modelos de projeto e análise através de simulações, apresentam formalmente os efeitos de sua ocorrência sobre um projeto de software. A análise deste efeito pode auxiliar na interpretação das informações contidas nas equações do cenário mediante o contexto do usuário, provendo recursos para a transferência de conhecimento.

O conhecimento a ser representado em um modelo pode ser gerado de diversas formas, como através de experimentos, entrevistas, consenso e revisão da literatura da Engenharia de Software. Este último mecanismo foi utilizado na construção dos cenários utilizados neste capítulo (com as simplificações citadas ao longo do texto) e nos cenários do modelo apresentado no Apêndice B. O Apêndice C apresenta a dinâmica dos cenários descritos no Apêndice B, assim como as referências bibliográficas de onde suas equações foram extraídas.

A aplicação das técnicas descritas neste capítulo na gerência de projetos de desenvolvimento de software depende fortemente da existência de conhecimento sobre a categoria dos sistemas a serem desenvolvidos. Em assim sendo, acreditamos que sua aplicação mais imediata sejam os sistemas de informação de médio porte, entre 100 e 1000 pontos de função. Acreditamos que esta categoria de sistemas possui o maior corpo de conhecimento na literatura de Engenharia de Software (ABDEL-HAMID e MADNICK, 1991; JONES, 1994; IFPUG, 1999; MADACHY e BOEHM, 1999; JONES, 2000).

No próximo capítulo apresentamos um estudo experimental que analisa a viabilidade de aplicação das técnicas propostas neste capítulo no gerenciamento de projetos de desenvolvimento de software.

5 Estudo Experimental do Gerenciamento Baseado em Cenários

Após o estabelecimento das bases do gerenciamento de projetos baseado em cenários, através das técnicas de integração e simulação de modelos de projeto e cenários, um estudo empírico foi planejado e executado para identificar indícios de viabilidade destas técnicas. Neste capítulo, apresentamos o plano, os resultados e as lições aprendidas neste estudo. A estrutura utilizada para descrição do plano do estudo experimental foi inspirada em (WOHLIN et al., 2000) e (SHULL et al., 2001).

Em um estudo de viabilidade, os dados são coletados de acordo com algum planejamento experimental, embora o controle sobre todas as possíveis variáveis não seja atingido (SHULL et al., 2001). Este tipo de estudo empírico tem como objetivo oferecer aos pesquisadores informações que possibilitem, de alguma forma, justificar o aprimoramento contínuo das técnicas em desenvolvimento. Neste estudo, enfatizamos a funcionalidade e utilidade das técnicas de integração e simulação de modelos de projeto e cenário, colocando a usabilidade como um objetivo secundário.

Além disso, ressaltamos a aplicação de cenários na gerência de projetos de desenvolvimento de software, ao invés da construção de novos cenários. Tal decisão foi tomada porque gostaríamos de alguma evidência sobre a eficácia dos cenários no gerenciamento de projetos antes de investir esforço na construção de melhores ferramentas para o desenvolvimento de modelos de cenário e de projeto.

Alguns apêndices complementam a descrição do estudo experimental. O Apêndice B contém o modelo de projeto e os modelos de cenário utilizados durante este estudo. O Apêndice C descreve a dinâmica destes modelos, apresentando as fontes de onde foram extraídas suas equações. O Apêndice E apresenta a ferramenta *Hector*, que foi utilizada durante o estudo. Finalmente, o Apêndice F apresenta os modelos de questionários preenchidos pelos participantes.

Este capítulo está dividido em oito seções. Na seção 5.1, apresentamos a definição do estudo experimental, ressaltando seus objetivos. Na seção 5.2, mostramos o planejamento do estudo experimental, considerando que este possa ser empacotado e replicado em diversos ambientes. Na seção 5.3, discutimos o sistema utilizado no estudo experimental. Na seção 5.4, detalhamos a instanciação do experimento, ou seja, a especialização do plano de experimento para o ambiente onde o estudo foi executado. Na seção 5.5, apresentamos a execução do estudo experimental, explicitando as

características e a organização de seus participantes. Na seção 5.6, listamos os resultados da análise sobre os dados colhidos durante o experimento. Na seção 5.7, apresentamos algumas lições aprendidas durante a realização e análise dos resultados do estudo experimental. Finalmente, na seção 5.8 concluímos o capítulo, resumindo os resultados obtidos através do estudo experimental.

5.1 Definição do Estudo Experimental

Objeto de Estudo: a utilização das técnicas de integração e simulação de modelos de projeto e cenário no gerenciamento de um projeto de desenvolvimento de software para avaliação do comportamento deste projeto.

Objetivo: identificar a viabilidade de utilização das técnicas de integração e simulação de modelos no gerenciamento de projetos de desenvolvimento de software.

Foco de Qualidade: os ganhos obtidos pela utilização das técnicas propostas, medidos através do custo e tempo de desenvolvimento de projetos gerenciados com o apoio destas técnicas, e as dificuldades encontradas pelos usuários em seu entendimento e utilização.

Perspectiva: o estudo será desenvolvido sob a ótica do pesquisador, avaliando a viabilidade de utilização das técnicas de integração e simulação de modelos, tendo em vista a continuidade do desenvolvimento das pesquisas relacionadas com estas técnicas. Neste estudo, não estamos interessados no tempo necessário para a utilização das técnicas propostas. Entretanto, consideramos que melhorias neste tempo podem ser identificados com análises e estudos futuros.

Contexto: o gerenciamento de um projeto de desenvolvimento de software definido em laboratório com comportamento determinado por um gerador de eventos aleatórios baseado em projetos reais. O estudo será conduzido no formato de múltiplos testes sobre um objeto.

A estrutura a seguir, baseada no método GQM (BASILI et al., 1994), descreve o objetivo do estudo experimental.

Analisar *a utilização das técnicas de integração e simulação no gerenciamento de projetos*
Com o propósito de *caracterizar a viabilidade de uso e continuidade de desenvolvimento*
Referente ***aos*** *ganhos obtidos por sua utilização e as dificuldades de sua utilização*
Do ponto de vista do *pesquisador*
No contexto do *gerenciamento de projetos de desenvolvimento de software.*

5.2 Planejamento do Estudo Experimental

Contexto Global: as técnicas de integração e simulação de modelos de projeto e cenário são um subconjunto do paradigma de gerenciamento de projetos baseado em cenários. A modelagem dinâmica de projetos de software (FORRESTER, 1961), a utilização do modelo de domínio baseado em classes e relacionamentos (BARROS et al., 2001a), a construção e validação de modelos de cenários, a construção e validação de arquétipos de risco (BARROS et al., 1999; BARROS et al., 2001b) se unem às técnicas sob avaliação para compor o paradigma. Estas técnicas podem ser classificadas em dois grupos: as técnicas de construção e as técnicas de utilização. As técnicas de construção se referem à representação de processos de software como modelos de projeto, à criação de conhecimento gerencial e sua representação sob a forma de cenários. Estas técnicas são aplicáveis em um ambiente de pesquisa (ambientes acadêmicos e grupos de pesquisa dentro de empresas), onde o conhecimento será elucidado e formalizado em modelos de cenários. As técnicas de utilização se referem ao uso de modelos de projeto e cenário no gerenciamento de projetos de software. Estas técnicas são utilizadas na indústria (por gerentes de projetos em empresas de desenvolvimento de software), integrando o conhecimento gerado no ambiente de pesquisa com as descrições dos projetos industriais para avaliação de seu comportamento.

Contexto Local: este estudo tem como intenção avaliar a viabilidade da utilização das técnicas de integração e simulação de modelos de projeto e cenário no gerenciamento de projetos de software. Os participantes do estudo serão requisitados a atuar como gerentes de um mesmo projeto de desenvolvimento de software, cujo comportamento será determinado por uma máquina de geração de eventos aleatórios. Devido às características estocásticas desta máquina, cada participante observará um comportamento diferente, embora o projeto seja o mesmo para todo o estudo. Um conjunto dos participantes será treinado na utilização das técnicas de integração e

simulação de modelos de projeto e cenário, enquanto os demais participantes utilizarão apenas as técnicas de gerência aprendidas nos cursos de graduação e pós-graduação.

Treinamento: o treinamento dos participantes que utilizarão as técnicas de integração e simulação de modelos de projeto e cenário será realizado em uma sala de aula, em sessão única com duração estimada de duas horas. O treinamento será composto de duas fases, que poderão ser interrompidas a qualquer momento pelos participantes para perguntas. Na primeira fase, realizaremos uma exposição sobre simulação e sobre as técnicas propostas, utilizando um conjunto de transparências que também serão distribuídas para os participantes. Na segunda fase, apresentaremos um caso de uso das técnicas de integração e simulação de modelos de projeto e cenário. Neste caso de uso, os participantes poderão acompanhar o processo de integração de um cenário a um modelo de projeto, a simulação do modelo integrado e a interpretação dos gráficos resultantes da simulação. O segundo grupo, que não utilizará as técnicas propostas, não receberá treinamento na utilização destas técnicas.

Projeto Piloto: antes da execução do estudo, realizaremos um projeto piloto com a mesma estrutura descrita neste planejamento. Para o projeto piloto, selecionaremos apenas dois participantes. Um participante será treinado na utilização das técnicas de integração e simulação de modelos de projeto e cenários e utilizará estas técnicas no gerenciamento do projeto proposto. O outro participante realizará o gerenciamento do projeto utilizando apenas os conhecimentos adquiridos durante sua formação acadêmica. Os participantes utilizarão o mesmo material descrito neste documento, entretanto, com acompanhamento integral do realizador do estudo. Sendo assim, o projeto piloto será um estudo baseado em observação. Ele não visa extrair qualquer resultado dos participantes, mas encontrar problemas no material planejado para o estudo, permitindo que este material seja aprimorado antes de sua utilização.

Participantes: os participantes do estudo serão um conjunto de desenvolvedores de software. O estudo não será executado em um ambiente da indústria, mas em um ambiente acadêmico. A capacidade de generalização deste estudo é discutida adiante, quando avaliamos as limitações e problemas que podem ser encontrados durante sua execução.

Instrumentação: todos os participantes receberão um software que apresentará uma representação gráfica do processo que deve ser controlado. Este software conterá o modelo de projeto, a máquina de geração de eventos aleatórios, os mecanismos para contabilizar o tempo e o custo de desenvolvimento e os comandos que o usuário poderá utilizar enquanto estiver atuando como gerente do projeto. Pontos de decisão em que um participante poderá influenciar o projeto incluem a seleção do desenvolvedor responsável por uma atividade, a indicação do número de horas de trabalho diário de um desenvolvedor e a indicação do esforço dedicado a uma atividade de inspeção. Estas decisões afetam o comportamento do projeto, que é regido pelas características dinâmicas apresentadas no Apêndice D. Todos os participantes receberão também o questionário Q1 para levantamento de sua formação e experiência. Os participantes treinados na utilização das técnicas de integração e simulação de modelos de projeto e cenário receberão ainda um kit contendo um conjunto de cenários, um modelo de projeto e um simulador para utilização das técnicas propostas. O kit inclui ainda o questionário Q2 para avaliação da satisfação do usuário na utilização destas técnicas. Os questionários Q1 e Q2 são apresentados no Apêndice F.

Critérios: o foco de qualidade do estudo exige critérios que avaliem os ganhos proporcionados pela utilização das técnicas de integração e simulação de modelos de projeto e cenário e as dificuldades encontradas pelos usuários na utilização destas técnicas. Os ganhos obtidos pela utilização das técnicas serão avaliados quantitativamente através do tempo e do custo de desenvolvimento do projeto gerenciado pelos participantes durante o estudo. A máquina geradora de eventos aleatórios que controla o projeto será responsável por contabilizar estes dados, que determinarão o desempenho dos participantes. O Apêndice D apresenta a dinâmica utilizada no desenvolvimento do modelo de projeto que rege esta máquina, assim como uma discussão sobre a validade desta dinâmica em projetos reais. As dificuldades encontradas pelos usuários na utilização das técnicas serão avaliadas através de dados qualitativos, levantados com base no questionário Q2. Este questionário será distribuído aos participantes que utilizarem as técnicas de integração e simulação de modelos de projeto e cenários depois do gerenciamento do projeto proposto.

Hipótese Nula: a hipótese nula determina que a utilização das técnicas de integração e simulação de modelos de projeto e cenário não produz benefícios no gerenciamento de projetos de desenvolvimento de software. De acordo com os critérios selecionados, esta hipótese se traduz na inexistência de diferenças significativas no custo e tempo para conclusão dos projetos de desenvolvimento de software gerenciados por participantes utilizando estas técnicas em relação a projetos gerenciados utilizando-se técnicas convencionais de gerência.

H_0: $\mu_{\text{custo sem técnicas}} = \mu_{\text{custo com técnicas}}$ e $\mu_{\text{tempo sem técnicas}} = \mu_{\text{tempo com técnicas}}$

Hipótese Alternativa: determina que os participantes do estudo que utilizarem as técnicas de integração e simulação de modelos de projeto e cenário terão resultados superiores aos participantes que utilizarem apenas as técnicas convencionais de gerência de projetos. De acordo com os critérios selecionados, esta hipótese se traduz em menor custo e tempo para a conclusão dos projetos gerenciados por participantes utilizando as técnicas propostas em relação a projetos gerenciados apenas com o auxílio de técnicas convencionais.

H_1: $\mu_{\text{custo sem técnicas}} > \mu_{\text{custo com técnicas}}$ e $\mu_{\text{tempo sem técnicas}} > \mu_{\text{tempo com técnicas}}$

Variáveis Independentes: a principal variável independente do estudo é um indicador que determina se cada participante utilizou ou não as técnicas de gerenciamento baseado em cenários (escala nominal). A formação e a experiência dos participantes, medidas em uma escala nominal, também são informações independentes coletadas durante o estudo (pelo questionário Q1), que poderão ser utilizadas durante a análise para a formação de blocos compostos por participantes com perfis semelhantes.

Variáveis Dependentes: as variáveis dependentes são o custo e tempo de desenvolvimento do projeto, cujo comportamento é determinado pela máquina de geração de eventos aleatórios. Este comportamento é sensível às decisões tomadas pelo gerente de projeto, cujo papel será cumprido pelos participantes do estudo. O custo será medido em reais (escala razão), de acordo com o número de homens/hora consumidos durante o projeto e o custo da hora de trabalho de cada desenvolvedor selecionado pelo gerente. O tempo será medido em dias (escala razão), indicando o tempo necessário para a conclusão do projeto desde a sua data de início.

Análise Qualitativa: tem o objetivo de avaliar a dificuldade de aplicação das técnicas propostas e a qualidade do material utilizado no estudo. A análise qualitativa será realizada através do questionário Q2. A dificuldade de aplicação das técnicas propostas será avaliada por perguntas envolvendo a utilidade das técnicas e a dificuldade de interpretação dos resultados apresentados por elas. A qualidade do material utilizado no estudo será avaliada por perguntas que tratam da utilidade dos cenários entregues aos usuários, da qualidade do treinamento oferecido e da descrição inicial do problema. Nesta avaliação, temos a intenção de verificar se o material está influenciando os resultados do estudo.

Capacidade Aleatória: pode ser exercida na seleção dos participantes do estudo e na distribuição dos objetos de análise entre os participantes. Idealmente, os indivíduos que realizarão o estudo devem ser selecionados aleatoriamente dentre o universo de candidatos a participantes, ou seja, dentre o conjunto das pessoas disponíveis que atendam aos critérios especificados no parágrafo *Participantes*. Entretanto, se a seleção aleatória não for possível, ao menos os objetos de análise devem ser distribuídos aleatoriamente entre os participantes.

Classificação em Bloco: a princípio, não identificamos a necessidade de dividir os participantes em blocos, visto que o estudo avaliará apenas um fator, que é a utilização das técnicas de integração e simulação de modelos de projeto e cenário. Um único bloco será capaz de determinar o efeito deste fator sobre os resultados do estudo. Entretanto, a coleta de dados sobre a formação e a experiência dos participantes permitirá sua futura classificação e a organização de blocos durante a análise dos dados.

Balanceamento: durante a realização do estudo, nos limitaremos apenas a distribuir um número similar de participantes na utilização das técnicas propostas e na utilização de técnicas tradicionais. Durante a análise dos dados, após a eliminação dos casos extremos, procuraremos um balanceamento, se este for possível.

Mecanismos de Análise: o estudo proposto se classifica como um experimento de dois tratamentos sobre um mesmo objeto, onde as variáveis dependentes são representadas na escala razão. Potenciais mecanismos de análise para este estudo são o teste-T (FREUND e PERLES, 1999) (WOHLIN et al., 2000) e o teste de Mann-

Whitney (WOHLIN et al., 2000). O teste-T é um procedimento paramétrico de análise estatística baseado na distribuição T, que verifica, com um determinado grau de certeza (como 95% ou 98%), se dois grupos possuem a mesma média, de acordo com suas variâncias. O teste de Mann-Whitney é um teste não paramétrico baseado no ranqueamento dos valores observados nos dois grupos sob análise.

Validade Interna: a validade interna de um estudo é definida como a capacidade de um novo estudo replicar o comportamento do estudo atual com os mesmos participantes e objetos com que ele foi realizado. A validade interna do estudo é dependente do número de participantes executando o estudo. Prevemos a participação de cerca de oito participantes, o que garante um bom nível de validação interna do estudo. Certamente, um número maior de participantes melhoraria a validade interna. Outro ponto que pode influenciar o resultado do estudo é a troca de informações entre os participantes que já realizaram o estudo e os que ainda não o realizaram. Para evitar este problema, requisitaremos explicitamente que os participantes não troquem informações a respeito do projeto. Tentaremos ainda realizar o estudo em uma única sessão, onde os participantes trabalharão paralelamente, sem trocar informações. Finalmente, o projeto não será apresentado como uma competição, inibindo a comparação de resultados entre os participantes.

Validade Externa: a validade externa do estudo mede sua capacidade de refletir o mesmo comportamento em outros grupos de participantes e profissionais da indústria, ou seja, em outros grupos além daquele em que o estudo foi aplicado. Acreditamos que o maior problema em relação à validade externa do estudo é a falta de interesse dos participantes no estudo. Alguns indivíduos podem realizar o estudo de forma descompromissada, sem um interesse real na realização do projeto em menor custo ou tempo, como aconteceria na indústria. A validade externa do estudo é considerada suficiente, uma vez que este visa avaliar a viabilidade de aplicação das técnicas de gerenciamento baseado em cenários. Demonstrada esta viabilidade, novos estudos podem ser planejados para refinar o universo de aplicação das técnicas propostas.

Validade de Construção: a validade de construção do estudo se refere à relação entre os instrumentos e participantes do estudo e a teoria que está sendo provada por este. Neste caso, escolhemos um domínio de aplicação amplamente conhecido,

neutralizando o efeito da experiência dos participantes no domínio. Esta escolha evita que experiências anteriores gerem uma interpretação incorreta do impacto das técnicas propostas. Os dados que serão utilizados para calibrar o processo de referência dizem respeito a projetos previamente desenvolvidos e a dados presentes na literatura, reduzindo o erro de ajuste no modelo. Além disso, o estudo não visa avaliar a correção do cenário, mas a capacidade de utilização destes cenários no gerenciamento. Assim, os cenários serão compatíveis com os eventos aleatórios gerados pela máquina que controla o projeto.

Validade de Conclusão: a validade de conclusão do estudo mede a relação entre os tratamentos e os resultados, determinando a capacidade do estudo em gerar alguma conclusão. Não encontramos grandes dificuldades em relação à capacidade de conclusão do estudo, visto que esta pode ser traçada a partir de um mecanismo de análise estatística amplamente utilizado, como o teste-T (as escalas das variáveis dependentes e independentes assim permitem). O teste-T possui alto poder estatístico e não assume normalidade ou qualquer outra distribuição nos dados analisados. Além disso, o estudo utiliza medidas objetivas, o que neutraliza a influência humana sobre os dados apurados e analisados.

5.3 O Sistema Utilizado no Estudo Experimental

O sistema selecionado para o experimento foi a primeira versão do sistema de controle acadêmico do Programa de Engenharia de Sistemas e Computação (PESC) da COPPE/UFRJ. Este sistema, denominado CtrlPESC, foi desenvolvido pelo aluno de Mestrado Leonardo Murta sob a coordenação da Prof. Cláudia Werner. O sistema utiliza a plataforma Internet. O lado servidor do sistema foi construído para a plataforma LINUX, utilizando a linguagem PHP3. No contexto do estudo experimental, os aspectos tecnológicos da implementação do sistema não são relevantes.

O sistema CtrlPESC, cujo tamanho pode ser medido com cerca de 63 pontos de função ajustados (AFP), tem como principal objetivo realizar parte do controle acadêmico do PESC. Para tanto, o sistema permite o cadastramento dos alunos, dos professores, das linhas de pesquisa do programa, das disciplinas oferecidas e das inscrições realizadas pelos alunos. Além disso, o sistema dispõe de um cadastro de usuários que permite controlar as pessoas que têm acesso permitido.

Um usuário é identificado no CtrlPESC através de seu nome, um nome curto utilizado para acesso ao sistema (*login*), sua senha e seu endereço de correio eletrônico. Sempre que um usuário deseja acessar o sistema, ele deve se identificar em uma caixa de diálogo. Após a identificação do usuário, o sistema apresenta uma tela de entrada e um menu que oferece acesso aos cadastros disponíveis. Cada cadastro permite a inclusão, remoção, alteração e consulta de seus elementos, embora determinadas operações estejam indisponíveis de acordo com o usuário.

Uma linha de pesquisa é identificada por um nome e por seu professor responsável. O professor responsável é identificado por uma referência para um dos professores cadastrados no sistema. Uma linha contém diversos professores, sendo apenas um deles o seu professor responsável. Um professor é identificado por seu nome completo, um nome curto utilizado para acesso (*login*) ao sistema, uma senha de acesso, um endereço de correio eletrônico e uma linha de pesquisa. A linha de pesquisa é uma referência a uma das linhas cadastradas anteriormente. Cada professor está associado a apenas uma linha de pesquisa e pode oferecer diversas disciplinas.

Uma disciplina é identificada por um código, um nome, tipo, nível, número de créditos e professor responsável. O tipo indica se a disciplina é obrigatória ou opcional. O nível indica se a disciplina pode ser cursada por alunos de mestrado ou apenas de doutorado. Complementando as informações sobre a disciplina, o sistema oferece um campo de observações para qualquer anotação. Os alunos participam das disciplinas através de inscrições. Um aluno é identificado por seu DRE, nome, correio eletrônico, ano e período de ingresso no departamento, nível, regime, tipo de bolsa, tipo de ingresso, linha, orientador e co-orientador. O nível indica se o aluno está cursando o mestrado ou o doutorado. O regime indica se a dedicação do aluno é parcial ou integral. O tipo de bolsa indica a instituição de fomento à pesquisa que fornece a bolsa de estudos do aluno. O tipo de ingresso indica se o aluno é formando (aluno de graduação com pelo menos 80% dos créditos cumpridos) ou se já concluiu a graduação. A linha indica a linha de pesquisa em que o aluno ingressou. O orientador e co-orientador são os professores que orientam a tese do aluno. Finalmente, um campo para observações gerais sobre o aluno também está disponível.

Uma inscrição indica a participação de um aluno em uma disciplina oferecida pelo PESC em um período. A inscrição indica o ano e o período em que a disciplina está sendo oferecida, o aluno inscrito, a disciplina que será cursada e o professor responsável pela disciplina.

5.4 Instanciação do Estudo Experimental

Seleção dos Participantes: para a presente execução do estudo, selecionamos os participantes dentre os alunos dos cursos de Mestrado e Doutorado em Engenharia de Software da COPPE Sistemas e os alunos do curso de graduação em Informática da Universidade Federal do Rio de Janeiro. Estes participantes atendem às restrições previamente indicadas (vide seção 5.2, parágrafo *Participantes*), visto que são desenvolvedores de software. Os participantes foram selecionados por conveniência, representando um subconjunto não aleatório do universo de alunos dos cursos de Engenharia de Software da COPPE Sistemas.

Capacidade Aleatória: a seleção dos participantes do estudo não foi aleatória, pois considerou a disponibilidade dos indivíduos para o estudo. Entretanto, os objetos de análise foram distribuídos entre os participantes de forma aleatória. Assim, a seleção dos participantes que utilizaram as técnicas propostas foi aleatória. De posse dos nomes dos participantes, realizamos um sorteio para determinar aqueles que utilizariam as técnicas de integração e simulação.

Mecanismos de Análise: utilizaremos o teste-T como mecanismo para avaliação das hipóteses do estudo. O teste-T é um dos mecanismos de análise estatísticos mais utilizados, sendo aplicável quando desejamos comparar as médias de dois tratamentos. O teste de Mann-Whitney será utilizado como segundo mecanismo de verificação.

Instrumentação: o emulador de projetos de desenvolvimento de software "Manager Master" foi desenvolvido para permitir a realização do experimento. Este software contém o modelo do projeto de software selecionado para o experimento, assim como os cenários que serão aplicados sobre este modelo. O emulador de projetos foi utilizado por todos os participantes do experimento, recebendo as decisões dos participantes sobre a distribuição dos desenvolvedores nas atividades do processo, o tempo de trabalho diário de cada desenvolvedor e o tempo dedicado às atividades de inspeção. O emulador de projetos calculou os resultados destas decisões sobre o tempo de desenvolvimento e o custo do projeto, apresentando estes resultados para o usuário. O sistema funciona em turnos, ou seja, tomadas todas as decisões desejadas para um determinado instante do projeto, o participante comanda o avanço de um dia

no projeto. A passagem de um dia gera novas informações sobre o comportamento do projeto, que podem ser utilizadas pelo participante para novas decisões. O sistema prossegue desta forma até que o projeto seja encerrado. O emulador de projetos é acompanhado de um manual do usuário, que foi lido pelos participantes antes de sua utilização no experimento. O emulador de projetos é apresentado em maiores detalhes no Apêndice E. Os questionários Q1 e Q2, citados no parágrafo sobre instrumentação da seção 5.2 também foram instrumentos utilizados no estudo.

Procedimentos de Participação: existem dois procedimentos distintos de participação no experimento. O primeiro foi seguido pelos participantes que não utilizaram as técnicas de integração e simulação de modelos de projeto e cenários, enquanto o segundo procedimento foi utilizado pelos participantes que utilizaram estas técnicas.

Procedimento de Participação sem a Utilização das Técnicas Propostas

- Participante chega ao local do experimento
- Participante assina o termo de adesão ao experimento
- Responsável pelo experimento associa um número ao participante
- Participante recebe treinamento com base nas transparências
- Participante lê o manual do usuário do emulador de projeto
- Responsável pelo experimento anota o número do participante no questionário Q1
- Participante preenche o questionário Q1
- Participante entrega o questionário Q1 para o responsável pelo experimento
- Responsável pelo experimento guarda o questionário Q1 na pasta do experimento
- Participante executa o emulador de projeto
- Participante decide a melhor configuração de decisões a serem tomadas
- Participante aplica suas decisões no emulador de projeto
- Responsável pelo experimento anota o desempenho do participante em seu registro

Procedimento de Participação com a Utilização das Técnicas Propostas

- Participante chega ao local do experimento
- Participante assina o termo de adesão ao experimento
- Responsável pelo experimento associa um número ao participante
- Participante recebe treinamento com base nas transparências
- Participante lê o manual do usuário do emulador de projeto
- Responsável pelo experimento anota o número do participante no questionário Q1
- Participante preenche o questionário Q1
- Participante entrega o questionário Q1 para o responsável pelo experimento
- Responsável pelo experimento guarda o questionário Q1 na pasta do experimento
- Participante executa o emulador de projeto

- Participante executa a ferramenta de simulação
- Participante carrega o modelo do projeto CtrlPESC
- Participante abre o ambiente de simulação
- Participante realiza simulações de configurações de decisões a serem tomadas
- Participante decide a melhor configuração de decisões a serem tomadas no projeto
- Participante aplica suas decisões no emulador de projeto
- Responsável pelo experimento anota o desempenho do participante em seu registro
- Responsável pelo experimento anota o número do participante no questionário Q2
- Participante preenche o questionário Q2
- Participante entrega o questionário Q2 para o responsável pelo experimento
- Responsável pelo experimento guarda o questionário Q2 na pasta do experimento

5.5 Execução do Estudo Experimental

Realização: o estudo experimental foi realizado durante um curso de pós-graduação em Engenharia de Software, durante o terceiro trimestre do ano letivo de 2001, no PESC (COPPE/UFRJ). Nove (9) pessoas participaram do estudo no primeiro dia. Os participantes foram divididos em dois grupos que realizaram o experimento em seqüência. O primeiro grupo, composto por quatro participantes, realizou o experimento em uma hora (1 hora). O segundo grupo, composto por cinco pessoas, realizou o experimento em uma hora e vinte minutos (1 hora e 20 minutos). Todos os nove participantes aplicaram as técnicas propostas na realização do experimento. Durante os sete dias seguintes da realização do estudo pelos dois grupos, nove (9) outros participantes realizaram o experimento, em diferentes dias e horários, sem aplicar as técnicas de integração e simulação de modelos de cenário e projeto.

Treinamento: os participantes que realizaram o estudo experimental utilizando as técnicas de integração e simulação de modelos de cenário e projeto foram treinados na utilização destas técnicas antes da realização do estudo. O treinamento se estendeu por uma hora e cinco minutos (1 hora e 5 minutos), ocupando uma das aulas do referido curso de pós-graduação.

Participantes: Os participantes incluíram os alunos do curso citado, alunos do programa de pós-graduação em Engenharia de Software e um aluno do curso de graduação em Informática. Dos 18 participantes, temos 13 alunos de Mestrado, 4 alunos de Doutorado e um aluno de graduação. Oito participantes haviam sido líderes de projetos na indústria, enquanto 3 participantes haviam sido líderes apenas em projetos acadêmicos e 7 participantes nunca haviam sido líderes de projeto. Dos

últimos sete participantes, 3 haviam participado em projetos na indústria, 3 haviam desenvolvido projetos de software como parte de trabalhos de curso e um havia desenvolvido software apenas para uso pessoal. Nenhum participante teve contato anterior com as técnicas propostas, a não ser no treinamento. A Tabela 5.1 apresenta um resumo da formação e da experiência dos participantes do estudo experimental. A Tabela 5.2 apresenta o perfil completo dos participantes.

Formação Acadêmica	
Graduandos	1
Mestrandos	13
Doutorandos	4

Experiência de Liderança e Desenvolvimento		
Líderes (Indústria)		8
Líderes (Academia)		3
Nunca foram Líderes	Des. Software (indústria)	3
	Des. Software (academia)	3
	Des. Software (uso pessoal)	1

Tabela 5.1- Resumo da formação e da experiência dos participantes do estudo experimental

ID	Utilizou as Técnicas	Formação	Experiência de Desenvolvimento	Experiência de Liderança
1	✓	Mestrando	Indústria (3 anos)	Indústria (1 ano)
2	✓	Doutorando	Uso pessoal	Nunca foi líder
3	✓	Mestrando	Academia (6 projetos)	Academia
4	✓	Mestrando	Academia (3 projetos)	Nunca foi líder
5	✓	Mestrando	Indústria (0,5 ano)	Academia
6	✓	Mestrando	Indústria (3 anos)	Academia
7	✓	Mestrando	Indústria (2 anos)	Nunca foi líder
8	✓	Doutorando	Indústria (2 anos)	Indústria (1 ano)
9	✓	Mestrando	Academia (2 projetos)	Nunca foi líder
10	×	Mestrando	Indústria (3 anos)	Nunca foi líder
11	×	Graduando	Academia (1 projeto)	Nunca foi líder
12	×	Doutorando	Indústria (10 anos)	Indústria (4 anos)
13	×	Doutorando	Indústria (3 anos)	Indústria (2 anos)
14	×	Mestrando	Indústria (4 anos)	Indústria (2 anos)
15	×	Mestrando	Indústria (4 anos)	Indústria (2 anos)
16	×	Mestrando	Indústria (1 ano)	Nunca foi líder
17	×	Mestrando	Indústria (4 anos)	Indústria (2 anos)
18	×	Mestrando	Indústria (10 anos)	Indústria (2 anos)

Tabela 5.2 – Perfil completo dos participantes do estudo experimental

Organização: o grupo que aplicou as técnicas propostas era composto por dois (2) alunos de Doutorado e sete (7) alunos de Mestrado. Os demais participantes realizaram o estudo sem a aplicação das técnicas propostas. Cinco (5) participantes do primeiro grupo tinham experiência como líderes de projetos na indústria ou na academia. Seis (6) participantes do segundo grupo tinham este tipo de experiência. A Tabela 5.3 apresenta um resumo da formação dos grupos que realizaram o estudo experimental.

	Participantes que aplicaram as técnicas propostas	Participantes que não aplicaram as técnicas propostas
Graduandos	0	1
Mestrandos	7	6
Doutorandos	2	2
Líderes (Industria)	5	6

Tabela 5.3- Resumo da formação dos grupos no estudo experimental

Custo do Estudo: como os participantes do estudo experimental foram alunos do PESC e os equipamentos utilizados foram os computadores dos laboratórios do Programa, o custo de realização do estudo experimental se concentrou no custo de planejamento e no custo de realização. Estes dois custos estão fortemente relacionados com o tempo necessário para o planejamento e a execução do estudo, conforme descrito abaixo.

Custo de Planejamento: o planejamento é certamente a parte mais custosa para a realização de um estudo experimental. O planejamento do estudo de viabilidade das técnicas de integração e simulação de modelos de projeto e cenários se estendeu por cerca de seis meses, entre março e setembro do ano de 2001. Durante este período, foram geradas sete versões do plano de experimento apresentado na seção 5.2 e foi desenvolvido o emulador de projetos de software. A necessidade deste último e a intenção de construir uma boa interface com o usuário tornaram o planejamento particularmente dispendioso.

Custo de Execução: conforme descrito acima, a execução do estudo foi realizada em duas etapas. O primeiro grupo de participantes, dividido em dois subgrupos, realizou o estudo em cerca de 3 horas e 30 minutos, incluindo-se o tempo de treinamento. Os demais participantes levaram em média 40 minutos para realizar o estudo, estando distribuídos ao longo de uma semana.

5.6 Análise dos Resultados do Estudo Experimental

Avaliação do Treinamento: o treinamento foi aplicado apenas para os participantes que utilizaram as técnicas de integração e simulação de cenários no experimento. O treinamento ocupou cerca de uma hora, ou seja, metade do tempo planejado. O treinamento foi expositivo, utilizando um conjunto de transparências, e seu tempo seria melhor aproveitado se incluísse uma demonstração do simulador de modelos e

do emulador de projetos de software. Diversos participantes perceberam a necessidade de mais treinamento (ID 2, 3, 6, 7 e 9)[1], um dentre eles sugerindo especificamente a apresentação de um exemplo (ID 2).

Avaliação Quantitativa: a análise quantitativa foi separada em duas análises independentes: tempo e custo. Cada uma destas análises foi realizada em três etapas: eliminação de valores extremos, teste paramétrico e teste não paramétrico de média.

Eliminação de Valores Extremos: tendo coletado as informações de tempo e custo através do emulador de projetos, passamos para a etapa de eliminação de valores extremos. Nesta etapa, aplicamos um corte baseado na distribuição T em 98% (FREUND e PERLES, 1999). Este método paramétrico de corte é análogo ao corte pela curva normal, sendo aplicável quando não podemos assumir que a população analisada segue uma distribuição normal e dispomos de um pequeno número de pontos para a análise ($n < 30$). Na análise de tempo de conclusão do projeto, o corte T eliminou dois participantes do grupo que aplicou as técnicas propostas e três participantes do grupo que não aplicou as técnicas. Na análise de custo do projeto, o corte T eliminou dois participantes do primeiro grupo e quatro participantes do segundo grupo. A Tabela 5.4 apresenta o tempo e o custo de realização do projeto para cada participante, ressaltando os participantes eliminados pelo corte de valores extremos (valores cortados e em itálico). Observe que, assim como as análises subsequentes, os cortes de tempo e de custo foram realizados de forma independente: um participante pode ter sido eliminado no corte de tempo e permanecer para análise de custos e vice-versa.

Participantes que aplicaram as técnicas propostas		
ID	Tempo	Custo
1	*~~39 dias~~*	13.761,13 $
2	38 dias	14.614,00 $
3	25 dias	14.634,92 $
4	25 dias	14.487,90 $
5	25 dias	*~~16.207,31 $~~*
6	27 dias	15.940,38 $
7	30 dias	*~~10.950,00 $~~*
8	*~~46 dias~~*	15.213,76 $
9	27 dias	15.031,63 $

Participantes que não aplicaram as técnicas propostas		
ID	Tempo	Custo
10	*~~18 dias~~*	15.301,65 $
11	28 dias	*~~15.951,81 $~~*
12	37 dias	14.549,69 $
13	42 dias	*~~14.217,86 $~~*
14	31 dias	*~~15.610,33 $~~*
15	33 dias	14.643,40 $
16	35 dias	*~~14.175,59 $~~*
17	*~~57 dias~~*	15.169,84 $
18	*~~58 dias~~*	14.999,12 $

Tabela 5.4 – Tempo e custo obtido por cada participante (após o corte de valores extremos)

[1] Ao preencher o questionário Q1, cada participante recebeu um identificador numérico (ID).

Estatísticas Descritivas: a Tabela 5.5 apresenta algumas estatísticas descritivas extraídas dos dados quantitativos coletados durante o estudo experimental, após a eliminação dos valores extremos. As estatísticas apresentam algumas informações relevantes para análise. O desvio padrão dos dois grupos de participantes foram muito similares em relação ao tempo e muito distintos em relação ao custo do projeto. Acreditamos que este fato está relacionado com a utilização de dois critérios independentes de análise (tempo e custo). Analisando o processo de tomada de decisão de alguns participantes[2], observamos que um conjunto de participantes realizou o estudo desprezando uma das variáveis de análise, geralmente o custo, para tentar concluir o projeto no menor tempo possível. Este parece ser o caso de um dos participantes que atingiu o tempo mínimo com a aplicação das técnicas propostas: seu custo foi muito superior ao custo obtido pelos dois outros participantes que também atingiram o tempo mínimo. Parece também haver baixa correlação entre a formação dos participantes, sua experiência como líderes de projeto e seus resultados. Na análise de tempo, 3 alunos de Mestrado obtiveram os valores mínimos aplicando as técnicas propostas, nenhum entre eles havendo indicado experiência na liderança de projetos na indústria. Um (1) aluno de graduação, também inexperiente na liderança de projetos industriais, obteve o tempo mínimo sem a aplicação das técnicas de integração e simulação de modelos de projeto e cenários. Na análise de custo, o valor mínimo com a aplicação das técnicas propostas foi obtido por um aluno de Mestrado com um (1) ano de experiência na liderança de projetos na indústria, enquanto o valor mínimo sem a aplicação destas técnicas foi obtido pelo aluno de Doutorado com maior experiência na liderança de projetos industriais (4 anos). Os valores máximos na análise de tempo foram obtidos por um aluno de Doutorado sem experiência na liderança de projetos na indústria (aplicando as técnicas) e por um aluno de Mestrado com 2 anos de experiência na liderança de projetos industriais. Em relação ao tempo, os valores máximos foram obtidos por alunos de Mestrado sem experiência na liderança de projetos na indústria. A Figura 5.1 apresenta graficamente as informações da Tabela 5.5.

[2] Nem todos puderam ser analisados em detalhes, pois alguns grupos de participantes realizaram o estudo em paralelo.

	Participantes que aplicaram as técnicas		Participantes que não aplicaram as técnicas	
	Tempo	Custo	Tempo	Custo
Média	28,1 dias	14.812,07 $	34,3 dias	14.932,74 $
Máximo	38,0 dias	15.940,38 $	42,0 dias	15.301,65 $
Mínimo	25,0 dias	13.761,13 $	28,0 dias	14.549,69 $
Desvio Padrão	4,7 dias	678,54 $	4,9 dias	326,79 $

Tabela 5.5 – Estatísticas descritivas dos dados quantitativos coletados durante o estudo experimental

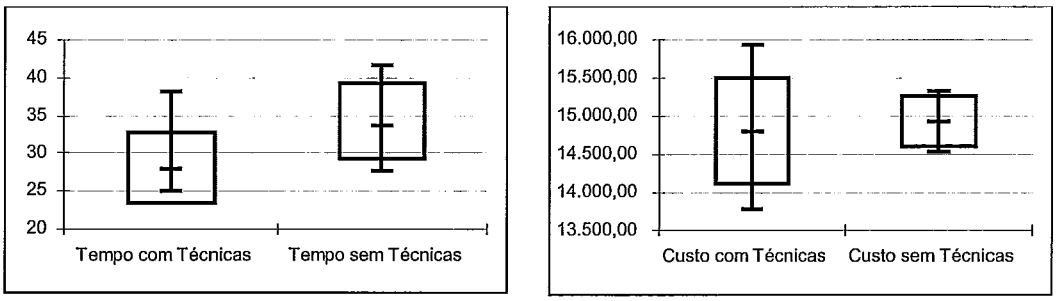

Figura 5.1 – Representação gráfica dos dados da Tabela 5.5

Teste Paramétrico: os dados que permaneceram após a eliminação de extremos foram submetidos a um teste-T com 95% de certeza. Na análise de tempo, o teste T foi conclusivo no sentido de afirmar que o tempo médio que os participantes que utilizaram as técnicas de integração e simulação de modelos levariam para concluir o projeto CtrlPESC foi inferior ao tempo médio que os participantes que não aplicaram estas técnicas levariam para concluir o projeto. A análise de custo foi inconclusiva, o que não nos permite afirmar nada em relação aos custos do projeto. A Tabela 5.6 apresenta os resultados intermediários do teste T, conforme descrito em (FREUND e PERLES, 1999).

	Teste T (95%)	
Variável	Tempo	Custo
Dispersão (Sp)	4,79	564,77
T_0	-2,32	-0,36
Graus de Liberdade	11	10
Distribuição T	2,20	2,23
Resultado ($T_0 < -T$)	$\mu_{s/\text{ técnicas}} > \mu_{c/\text{ técnicas}}$	inconclusiva

Tabela 5.6 – Resultados intermediários do teste T (FREUND e PERLES, 1999)

Teste Não-Paramétrico: com o objetivo de aferir as conclusões do teste T, os dados resultantes do corte de valores extremos foram submetidos a um teste de Mann-Whitney. Este teste, baseado na ordenação dos valores analisados, chegou às mesmas conclusões do teste T. A Tabela 5.7 apresenta os resultados intermediários do teste de Mann-Whitney (WOHLIN et al., 2000).

Variável	Teste de Mann–Whitney	
	Tempo	Custo
Tamanho do primeiro conjunto	7	7
Tamanho do segundo conjunto	6	5
Tamanho do menor conjunto	6	5
Tamanho do maior conjunto	7	7
Soma dos ranks do menor conjunto	57	36
U	6	14
U'	36	21
Min (U, U')	6	14
Tabela Mann-Whitney	6	5
Resultado	$\mu_{\text{s/ técnicas}} > \mu_{\text{c/ técnicas}}$	inconclusiva

Tabela 5.7 – Resultados intermediários do teste de Mann-Whitney (WOHLIN et al., 2000)

Conclusão: embora os testes estatísticos relacionados com o custo não tenham sido conclusivos, o sucesso dos testes relacionados com o tempo para a realização do projeto nos indica que as técnicas de integração e simulação de modelos de cenário e projeto podem ser viáveis e podem auxiliar os gerentes no planejamento e controle de projetos de software. Entretanto, além de outros estudos conforme o descrito neste capítulo, são necessários aprimoramentos no sentido de facilitar a implantação e utilização das técnicas propostas. Abaixo, apresentamos a análise qualitativa, realizada com o intuito de apresentar novas direções de pesquisa e de aprimoramento das técnicas.

Avaliação Qualitativa: a seguir, resumimos os resultados da análise qualitativa sobre a utilidade e usabilidade das técnicas propostas e sobre a qualidade do material utilizado no estudo experimental. Esta análise se baseia nas respostas ao questionário Q2, apresentado no Apêndice F.

Utilidade das Técnicas: dos 9 participantes que aplicaram as técnicas de integração e simulação de modelos de projeto e cenário no gerenciamento do projeto proposto, todos indicaram que as técnicas utilizadas foram úteis, que os cenários apresentados foram úteis e que aplicariam as técnicas novamente. Um participante (ID 3) ressaltou necessidade de pesquisas para calibrar os modelos utilizados pelas técnicas propostas.

Usabilidade das Técnicas: seis (6) participantes julgaram que a interpretação dos resultados gerados pelas técnicas propostas foi simples. Os demais participantes

requisitaram melhorias na apresentação dos resultados das simulações. Estas dificuldades estão relacionadas com a visualização de tempo e custo em um mesmo eixo vertical (ID 2), a incapacidade de perceber a rede de atividades do projeto (ID 6) e de associar um resultado do gráfico a uma determinada atividade ou decisão (ID 9).

Qualidade do Material: apenas 3 participantes julgaram que o treinamento foi suficiente para a aplicação das técnicas propostas. Quatro (4) participantes ressaltaram a necessidade de mais treinamento, um (1) dentre eles ressaltando a necessidade de um exemplo de aplicação das técnicas durante o treinamento. Os demais participantes (2) requisitaram melhores definições para alguns termos utilizados no estudo experimental, como inspeções, por exemplo.

5.7 Lições Aprendidas

Em uma repetição do experimento devemos considerar alguns pontos, vistos como limitações de sua primeira execução.

Descrição do Problema: é preciso descrever o sistema que será desenvolvido de forma mais detalhada. Dois participantes sugeriram que os objetivos da empresa em relação ao projeto (qualidade, tempo e custo) sejam indicados na apresentação do projeto (ID 4 e 9). O objetivo de qualidade foi explicitamente citado por um participante (ID 4), o que pode indicar que as condições de encerramento das atividades de teste devem ser melhor definidas. A apresentação da descrição do sistema deve ser realizada junto com o treinamento, de forma a que todos os participantes estejam sujeitos ao mesmo contexto. A utilização de uma descrição textual do projeto exigiu a resolução de diversas dúvidas dos participantes: seria desejável apresentar a rede de tarefas construída para o projeto, detalhando como a duração de cada tarefa foi estimada. Finalmente, a descrição deve indicar as condições de parada das atividades de teste.

Aprimoramento do Treinamento: Sugere-se, para uma repetição do experimento, que as ferramentas a serem utilizadas no experimento sejam apresentadas aos participantes durante o treinamento. Ao invés de apenas

expositivo, o treinamento deve permitir que os participantes apliquem o processo de análise e tomada de decisão em um projeto menor, diferente do utilizado no experimento, antes de sua realização. Da mesma forma, seria desejável que os participantes que não aplicaram as técnicas propostas tivessem algum treinamento, principalmente na utilização do emulador de projetos de software. Este treinamento reduziria a necessidade de intervenção de um organizador do experimento para resolução de dúvidas durante sua realização.

Folha de Consentimento: devemos reavaliar o cabeçalho da folha de consentimento de participação no experimento, expressando nesta que a participação no experimento não incorrerá em prejuízo na avaliação do participante, independente dos resultados obtidos por ele no experimento.

Participação em Grupos: seria desejável que os participantes treinados com as técnicas propostas realizassem o experimento paralelamente. Na execução descrita na seção 5.5, os participantes que aplicaram as técnicas propostas foram divididos em dois grupos, que executaram o estudo em seqüência. A divisão em dois grupos permite que os participantes do segundo grupo tenham mais tempo para refletir sobre as técnicas recentemente aprendidas antes de aplicá-las no experimento. Embora nenhuma diferença considerável possa ser percebida nos resultados desta execução do experimento, tal pode ocorrer em outro contexto.

Instrumentação: alguns participantes expressaram dificuldades na utilização do simulador de modelos de projetos e cenários (ver seção 5.6). Tais dificuldades já haviam sido antecipadas no projeto piloto e demandam alguns aprimoramentos no simulador como, por exemplo, a utilização de dois eixos independentes nos gráficos de resultado, a limitação do universo dos valores que um parâmetro do modelo pode assumir e a apresentação gráfica da rede de tarefas do projeto. Em relação ao emulador de projetos, foi ressaltada a necessidade de apresentação mais clara da duração das tarefas e de soluções para tornar alguns comandos desnecessários (como a definição da equipe de desenvolvimento).

Depuração dos Resultados: durante a análise dos resultados do estudo, ficou clara a necessidade de captura de mais informações sobre as decisões tomadas por cada participante. Ao invés de capturar apenas o resultado final (tempo e custo do

projeto) de cada participante, como ocorreu na primeira execução do estudo experimental, o emulador de projetos poderia capturar todas as decisões tomadas ao longo do projeto. Este avanço deverá facilitar a "depuração" dos resultados do experimento durante a análise.

5.8 Conclusão

Neste capítulo, apresentamos o planejamento, a execução e a análise dos resultados de um estudo experimental que avaliou a viabilidade das técnicas de integração e simulação de modelos de projeto e cenários. O estudo analisou a capacidade de gerentes de projeto utilizando as técnicas propostas concluírem um projeto hipotético em menor tempo e com menor custo que gerentes controlando o mesmo projeto sem o auxílio das técnicas de integração e simulação de modelos de projeto e cenários.

Apesar da reduzida população de estudo (18 participantes), a análise dos resultados permite identificar que, neste universo, os resultados do testes estatísticos (teste-T e teste de Mann-Whitney) podem ser considerados conclusivos em relação ao tempo para conclusão do projeto hipotético e inconclusivos em relação ao seu custo de desenvolvimento. Desta forma, o estudo mostra indícios de que as técnicas podem ser viáveis, assim como aponta algumas direções em que a pesquisa deve avançar.

Entretanto, repetições deste estudo, em diferentes contextos, além da realização de estudos de observação, são ainda necessárias para a efetiva identificação de problemas e aprimoramento das técnicas.

6 Gerenciamento de Riscos Baseado em Cenários

As técnicas apresentadas nos capítulos anteriores descrevem como os cenários são formulados, integrados com um modelo de projeto e como o comportamento do projeto é avaliado com e sem a integração de combinações de cenários para determinar o impacto destes cenários sobre o projeto. Dentre as diversas categorias de cenários apresentadas no Capítulo 4 (seção 4.4), os cenários de eventos representam eventos incertos que podem ocorrer durante um processo de desenvolvimento de software. Sendo incertos, estes cenários podem ser vistos como riscos enfrentados pelo projeto. Baseado nesta constatação e na possibilidade de aplicar os modelos de cenários na avaliação quantitativa do impacto dos riscos sobre um projeto, utilizamos as técnicas descritas no Capítulo 4 para definir um processo de análise de riscos. Este processo descreve como os riscos são documentados através de modelos de cenários, como são reutilizados ao longo de uma série de projetos similares e como os modelos de cenários são utilizados na avaliação do impacto dos riscos sobre uma aplicação.

O Capítulo 3 apresentou uma revisão do estado da arte em análise de riscos em projetos de desenvolvimento de software. Um processo de análise de riscos transforma um conjunto de incertezas e dúvidas acerca de um projeto de desenvolvimento de software em riscos aceitáveis, de acordo com o conhecimento acumulado sobre o projeto. A análise de riscos não remove os riscos de um projeto, mas os deixa mais visíveis e, eventualmente, quantificáveis. A visibilidade dos riscos serve como um mecanismo de alerta, avisando aos gerentes de projeto sobre as incertezas a que eles estão expostos e permitindo o planejamento para prevenção ou correção dos efeitos da ocorrência destas incertezas.

O paradigma do gerenciamento de projetos baseado em cenários adota a seguinte definição para risco: um risco representa a possibilidade de ocorrência de um evento que cause prejuízos a um processo de desenvolvimento de software ou aos artefatos produzidos por ele. A análise de riscos no paradigma se baseia na representação de riscos através de cenários, na reutilização destes cenários ao longo de diversos projetos e na simulação de combinações de cenários para a avaliação do impacto provocado pela ocorrência dos riscos sobre o comportamento de um determinado projeto.

Estas atividades representam dois diferentes objetivos em um processo de análise de riscos. Primeiro, os riscos são identificados e documentados utilizando-se de

cenários. Em seguida, os riscos são reutilizados ao longo de diversos projetos, permitindo a simulação de seu impacto sobre o comportamento do projeto. Estes objetivos lembram um processo genérico de reutilização (MOORE e BAILIN, 1991), conforme apresentado na Figura 6.1.

O processo genérico de reutilização se baseia em dois sub-processos: um para o desenvolvimento de artefatos reutilizáveis, que identifica e documenta estes artefatos de software, e outro para o desenvolvimento de aplicações com base na reutilização dos artefatos previamente desenvolvidos, que os seleciona e adapta a partir de uma base de artefatos reutilizáveis. O primeiro sub-processo é denominado **desenvolvimento para reutilização**, enquanto o segundo é denominado **desenvolvimento com reutilização**.

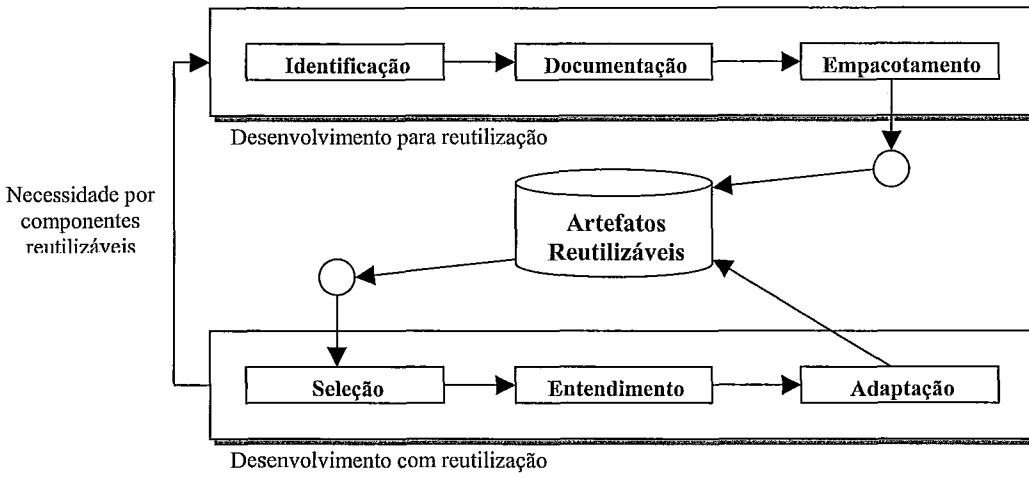

Figura 6.1 – O processo genérico de reutilização de software (MURTA, 1999)

Projetando-se o processo genérico de reutilização sobre o processo de análise de riscos proposto pelo paradigma de gerenciamento de projetos baseado em cenários, este processo também é dividido em dois sub-processos: um para a identificação de riscos e outro para a reutilização destes riscos durante o desenvolvimento de uma aplicação. O artefato reutilizável compartilhado pelos dois processos de análise de riscos é representado pelas informações que descrevem um risco.

O principal objetivo do processo para identificação de riscos é descrever os riscos associados a utilização de um elemento de projeto específico em um projeto de desenvolvimento de software. Os elementos de projeto incluem o domínio da aplicação, as tecnologias, os papéis desempenhados pelos desenvolvedores, os artefatos de software e os recursos consumidos pelo projeto. O processo de análise de riscos organiza as informações sobre os riscos associados a estes elementos e permite sua reutilização em diversos processos de desenvolvimento de aplicações. O primeiro

processo de análise de riscos é executado quando uma organização explora um novo elemento de projeto (como uma metodologia, uma linguagem de programação, um domínio de aplicação, entre outros) que pode impor riscos a diversos projetos de desenvolvimento de software.

O principal objetivo do processo de análise de riscos para o desenvolvimento de aplicações é identificar e avaliar os riscos que podem afetar um projeto de acordo com os elementos de projeto que o compõem. Este processo reutiliza os riscos associados aos elementos de projeto, identificados e documentados pelo processo de análise de riscos anterior. Ele ocorre em paralelo com o processo de desenvolvimento de uma aplicação, acompanhando sua evolução e resolvendo seus riscos.

Os processos de análise de riscos reforçam a análise quantitativa, modelando o impacto do risco e de suas estratégias de resolução como modelos de cenário. Reconhecemos as dificuldades de modelar quantitativamente todos os riscos envolvidos em um processo de desenvolvimento de software, especialmente os riscos não técnicos, como riscos legais, contratuais e políticos. Entretanto, ao sugerir a documentação formal dos riscos através de cenários, esperamos que a equipe de identificação de riscos discuta os efeitos dos riscos sobre o projeto, documentando as premissas que levaram à construção dos seus modelos de impacto. Sem a documentação formal dos riscos, o raciocínio sobre seus efeitos no projeto pode ser muito subjetivo, eventualmente sem utilidade para projetos futuros.

Os riscos são identificados e documentados através de **arquétipos de risco**, que servem como mecanismo de alerta para o gerente de projetos e como suporte para os cenários que modelam o impacto, as estratégias de contenção e de contingência dos riscos. Os arquétipos de risco representam os artefatos reutilizáveis que conectam os dois sub-processos de análise de riscos. Na próxima seção, apresentamos as informações que descrevem um arquétipo de risco.

6.1 Arquétipos de Risco

Um arquétipo de risco é uma descrição de um problema potencial, recorrente ao longo de diversos projetos, cuja ocorrência pode causar prejuízos a um produto ou processo de desenvolvimento de software. Ele inclui uma descrição do contexto em que o problema pode ocorrer e apresenta estratégias que podem ser aplicadas para solucionar o problema potencial antes ou depois de sua ocorrência.

A definição de um arquétipo de risco se baseia no conceito de padrão de projeto, conforme definido por GAMMA et al. (1994). Um padrão de projeto descreve uma estrutura orientada a objetos, composta de classes e objetos, que pode ser utilizada na resolução de um problema genérico em um projeto de software. Cada padrão de projeto identifica, documenta e avalia um problema de projeto recorrente ao longo do desenvolvimento de diversos sistemas de software. O padrão de projeto captura a experiência no desenvolvimento da solução, permitindo que esta experiência seja reutilizada por outros projetos que enfrentem o mesmo problema.

Como os padrões de projeto, os arquétipos de risco identificam, documentam e fornecem informação para a avaliação dos eventos potenciais que podem afetar negativamente os projetos de desenvolvimento de software. A experiência de identificação e resolução de um risco é documentada no arquétipo, que pode ser reutilizado em projetos futuros.

Os arquétipos de risco determinam as informações que devem ser armazenadas junto com a descrição do problema potencial, apresentando estas informações em um formato estruturado e padronizado (BARROS et al., 1999, BARROS et al., 2000c). A representação padronizada visa promover a comunicação dos riscos, seu entendimento e a distinção clara dos diversos eventos incertos. Acreditamos que o formato estruturado promove o preenchimento completo da documentação dos riscos, de forma que um arquétipo contenha todas as informações necessárias para análises de risco no futuro. A existência de informações completas sobre os riscos que podem afetar um projeto é importante para reduzir as barreiras psicológicas contra a análise de riscos: riscos são conceitos abstratos (KONTIO, 1997), e uma descrição incompleta dos riscos tende a reduzir a motivação para a análise de riscos. Além disso, os arquétipos de risco unem as estratégias de resolução (planos de contenção ou contingência) aos problemas potenciais por eles descritos, trazendo a descrição do problema em conjunto com suas possíveis soluções e heurísticas para aplicação destas soluções (BARROS et al., 2001c).

Os arquétipos de risco podem ser associados a qualquer elemento de projeto, como uma tecnologia, uma atividade ou um domínio de aplicação. Atrasos de cronograma, custos acima do previsto, a necessidade de atividades de retrabalho, perdas de qualidade e funcionalidade, atividades de desenvolvimento não planejadas, mudanças nos requisitos, problemas de verificação e validação dos requisitos são exemplos de arquétipos de risco. A estrutura padronizada que representa um arquétipo de risco é composta por cinco blocos, descritos nos parágrafos a seguir:

- **Identificação do Arquétipo**: este bloco descreve o problema potencial representado pelo arquétipo de risco. Ele é composto pelas seguintes informações:
  1. Nome: o nome do arquétipo de risco descreve sucintamente a essência do problema que é representado pelo arquétipo;
  2. Sinônimos: outros termos que identificam o arquétipo de risco. Estes termos são úteis para reforçar o entendimento do problema descrito pelo arquétipo;
  3. Problema Potencial: define objetivamente o problema representado pelo arquétipo de risco, ou seja, o evento que, ao ocorrer em um projeto, causa prejuízos a seus produtos ou ao processo de desenvolvimento;
  4. Efeitos: descreve textualmente os efeitos da ocorrência do problema potencial em um projeto de desenvolvimento de software. A descrição ressalta os produtos e processos afetados pelo arquétipo, determinando os possíveis efeitos sobre cada um destes elementos;
  5. Impacto: descreve, utilizando um modelo de cenário, os efeitos da ocorrência do problema potencial. Este modelo, denominado **modelo de impacto do risco**, pode ser integrado ao modelo de projeto de uma aplicação em construção, modificando o comportamento do projeto para refletir as conseqüências da ocorrência do risco;
  6. Fontes de Dados: indicam a origem das informações utilizadas para descrever a dinâmica do modelo de impacto do risco. Estas fontes de dados podem incluir bancos de dados históricos medidos em uma empresa, referências à literatura de Engenharia de Software, referências a experimentos e estudos de casos previamente realizados, entre outros.

- **Mecanismos de Identificação**: este bloco descreve os mecanismos que são utilizados para determinar se o risco descrito pelo arquétipo pode ocorrer em um projeto de desenvolvimento de software. Este bloco é composto pelas seguintes informações:
  1. Contexto: o contexto do risco define as condições que aumentam as chances de ocorrência do problema descrito pelo arquétipo de risco em um projeto. Estas condições são identificadas através de uma análise baseada em um questionário;
  2. Questionário: cada pergunta do questionário de um arquétipo de risco é composta por uma frase interrogativa e um conjunto de informações de ajuda. O questionário é apresentado ao gerente de projetos durante o processo de análise de riscos para o desenvolvimento de aplicações, visando determinar se o risco

descrito pelo arquétipo pode ocorrer na aplicação. Estas questões estão relacionadas com as incertezas acerca das condições que promovem a ocorrência do risco em um projeto;

3. Casos conhecidos: os casos conhecidos de ocorrência de um arquétipo de risco apresentam uma lista de referências a projetos onde o arquétipo foi anteriormente identificado. Cada vez que o arquétipo é identificado para um projeto, um plano de tratamento de riscos documenta suas informações específicas para o projeto. Os planos de tratamento de riscos descrevem o contexto em que o arquétipo de risco ocorreu, as estratégias de resolução selecionadas e as conseqüências de sua ativação no projeto. As referências a casos conhecidos em um arquétipo de risco são associadas a planos de tratamento de riscos. Durante a identificação dos riscos que podem afetar um aplicação, o gerente de projetos pode navegar através destes planos, analisando as ocorrências dos arquétipos em projetos anteriores.

- **Estratégias de Contenção**: este bloco descreve possíveis estratégias para inibir ou eliminar o problema potencial descrito pelo arquétipo de risco antes de sua ocorrência em um projeto. O bloco também apresenta informações para a avaliação das conseqüências da aplicação de cada estratégia de resolução. Ele é composto por múltiplas ocorrências das seguintes informações:
  1. Descrição: planos de contenção são estratégias de resolução para reduzir as chances de ocorrência do risco descrito pelo arquétipo em um projeto. Estes planos são aplicáveis antes da ocorrência do risco no projeto. A descrição do plano define as incertezas que serão controladas e os mecanismos que serão utilizados neste controle;
  2. Condições: como as incertezas que promovem a ocorrência do risco são identificadas no questionário do bloco de mecanismos de identificação do arquétipo de risco, existe uma forte associação entre as perguntas do questionário e a aplicação dos planos de contenção. As condições de aplicação de um plano de contenção são determinadas por um subconjunto do universo de respostas às perguntas do questionário de identificação do arquétipo. As chances de ocorrência de um risco podem ser reduzidas por diversos planos de contenção. As condições de aplicação filtram as incertezas para selecionar estratégias de contenção específicas de acordo com as características do projeto;

3. Efeitos: descreve textualmente os efeitos do plano de contenção. A descrição ressalta os produtos e os processos afetados pelo plano, determinando os possíveis efeitos do plano sobre cada um destes elementos;
4. Impacto: descreve, através de um modelo de cenário, os efeitos da aplicação do plano de contenção. Como o modelo de impacto de risco, este modelo pode ser integrado ao modelo de projeto de uma aplicação em desenvolvimento para permitir a avaliação dos efeitos do plano de contenção sobre o comportamento do projeto;
5. Fontes de Dados: indicam a origem das informações utilizadas para descrever a dinâmica do modelo de impacto do plano de contenção.

- **Estratégias de Contingência**: este bloco descreve possíveis estratégias de resolução para reduzir o impacto do problema potencial descrito pelo arquétipo de risco depois de sua ocorrência em um projeto. O bloco também contém informações para a avaliação das conseqüências da ativação de cada plano de contingência. Ele é composto por múltiplas ocorrências das seguintes informações:
  1. Descrição: os planos de contingência são possíveis estratégias de resolução aplicáveis após a ocorrência do problema para reduzir seu impacto no projeto. Este parágrafo descreve os mecanismos utilizados por um plano de contingência para o risco descrito pelo arquétipo;
  2. Efeitos: descreve textualmente os efeitos do plano de contingência. Como nos planos de contenção, a descrição ressalta os produtos e os processos afetados pelo plano, determinando os possíveis efeitos do plano sobre cada um destes elementos;
  3. Impacto: descreve, através de um modelo de cenário, os efeitos da aplicação do plano de contingência. Este modelo também pode ser integrado ao modelo de projeto de uma aplicação para permitir a avaliação dos efeitos do plano de contingência sobre o projeto;
  4. Fontes de Dados: indicam a origem das informações utilizadas para descrever a dinâmica do modelo de impacto do plano de contingência.

- **Arquétipos Relacionados**: este bloco referencia os arquétipos de risco que ocorrem em situações similares ou arquétipos de risco que podem substituir o arquétipo atual. Ele é composto pelas seguintes informações:

1. Riscos induzidos: a ocorrência de um risco em um projeto pode induzir a ocorrência de outros riscos no mesmo projeto. Estes relacionamentos entre arquétipos de risco são capturados em uma lista de arquétipos induzidos pelo arquétipo atual. Cada vez que um risco é identificado para um projeto, seus arquétipos induzidos são automaticamente identificados para o projeto;
2. Riscos similares: a lista de riscos similares a um arquétipo de risco contém referências para outros arquétipos que documentam problemas similares ao problema descrito pelo arquétipo atual. Durante a identificação de riscos para uma aplicação, o gerente pode navegar pelos arquétipos similares, determinando o risco que melhor se aplica ao contexto e às características do projeto em desenvolvimento.

A Figura 6.2 apresenta um exemplo de arquétipo para o risco de validação, ou seja, o risco de se construir um sistema a partir de especificações incorretas. A figura mostra o conteúdo dos blocos da estrutura de informação que representa o arquétipo de risco, conforme discutido nos parágrafos anteriores. O Apêndice G apresenta outros exemplos de arquétipos de risco.

Identificação do Arquétipo

1. Nome: Risco de validação
2. Sinônimos: Requisitos incorretos
3. Problema potencial: desenvolvimento de um sistema cuja funcionalidade não é aceita ou não satisfaça o cliente.
4. Efeitos: perda de qualidade do produto
5. Impacto: *modelo de cenário*
6. Fontes de Dados: *referências*

Mecanismos de Identificação

1. Contexto: projetos em que a equipe tem pouca interação com os clientes e com os usuários finais do software em construção; projetos que não desenvolvem protótipos; projetos com diversos clientes distintos; projetos inovadores.
2. Questionário:

   a) Avalie o grau de ambigüidade nos requisitos do sistema.
   b) Avalie o número de requisitos não definidos do sistema.
   c) Avalie o envolvimento do usuário no projeto.
   d) Avalie o grau de aplicação de técnicas de validação, como simulação operacional ou prototipação, no projeto.

3. Casos conhecidos:

   Projeto 105: classificação e análise de polímeros
   Projeto 112: supermercado eletrônico

Estratégias de Contenção

1. Plano: Desenvolvimento de protótipos
2. Condição: Projeto com poucos mecanismos de validação
3. Efeitos: Ajuste de cronograma (construção dos protótipos)
4. Impacto: *modelo de cenário*
5. Fontes de Dados: *referências*

1. Plano: Aplicação de técnicas de JAD
2. Condição: Projetos com pouca participação dos usuários
3. Efeitos: Ajuste de cronograma (sobrecarga de reuniões)
4. Impacto: *modelo de cenário*
5. Fontes de Dados: *referências*

Estratégias de Contingência

1. Plano: extensão do cronograma
2. Efeitos: permite a prototipação e refinamento dos requisitos
3. Impacto: *modelo de cenário*
4. Fontes de Dados: *referências*

Arquétipos Relacionados

3. Riscos Induzidos:
   - Ajuste de cronograma
   - Aumento de custos
   - Reputação
4. Riscos Similares:

Figura 6.2 – Um arquétipo de risco descrevendo o risco de validação

Esperamos que a criação de um arquétipo de risco enriqueça o vocabulário da equipe de desenvolvimento. Assim como ocorre com os padrões de projeto, o nome do arquétipo de risco é utilizado como um sinônimo para o problema recorrente por ele representado. A citação do arquétipo de risco tende a ser associada com sua descrição completa, trazendo consigo todas as informações contidas no arquétipo. Esperamos que a extensão do vocabulário da equipe, com os termos que se referem a identificação de riscos, promova uma cultura de discussão dos riscos que podem afetar os projetos desenvolvidos em uma organização. Conforme indicado no capítulo 4, a criação e expansão desta cultura de risco é um dos princípios do gerenciamento de projetos baseado em cenários.

6.2 O Processo de Identificação de Riscos para Elementos de Projetos

Um domínio de aplicação pode conter riscos que ocorrem em diversos projetos desenvolvidos dentro do domínio. Por exemplo, alguns domínios estão sujeitos a alta volatilidade de requisitos ou objetivos restritos de desempenho, disponibilidade e confiabilidade. Aplicações desenvolvidas utilizando uma determinada tecnologia podem compartilhar alguns riscos. Se uma organização de desenvolvimento de software opta por utilizar C++ em um projeto, este projeto pode estar sujeito a restrições de portabilidade. Se a mesma organização optar por utilizar Java no projeto, este poderá estar sujeito a limitações de desempenho.

Através destes exemplos, observamos que diversos elementos de projetos, como o domínio da aplicação, as tecnologias selecionadas ou os papéis desempenhados pelos desenvolvedores em um projeto, impõem riscos a um projeto de desenvolvimento de software. Estes riscos são conseqüências naturais dos benefícios e desvantagens trazidos por estes elementos para o projeto. Entretanto, as forças positivas e negativas geradas pelos elementos de projeto podem não estar claras para o gerente durante o planejamento e controle de um projeto. Deve existir alguma documentação descrevendo estas forças e apresentando mecanismos para sua consideração durante o planejamento e o controle de um projeto.

Como cada elemento de projeto pode estar relacionado com diversos riscos, a equipe de gerenciamento de riscos deve reutilizar o conhecimento sobre os riscos que podem ocorrer em aplicações que utilizem estes elementos (BARROS et al., 2001d, BARROS et al., 2002). A formalização destes riscos reduz o esforço investido por cada projeto na identificação dos riscos com o intuito de resolver as mesmas dúvidas e

incertezas. Ela também previne que determinados riscos passem desapercebidos nas atividades de identificação de riscos de algumas aplicações.

O primeiro sub-processo de um processo de análise de riscos segundo o paradigma de gerenciamento de projetos baseado em cenários é responsável pela identificação e documentação dos riscos associados a um determinado elemento de projeto. Este sub-processo não considera uma única aplicação, mas um conjunto de aplicações que podem utilizar um determinado elemento de projeto sob análise. Seu resultado é uma lista de arquétipos de risco associados ao elemento de projeto.

O processo para identificação dos riscos associados a um elemento de projeto determina os riscos que podem ocorrer em aplicações relacionadas com este elemento de projeto, especifica as condições em que estes riscos são relevantes para a aplicação, define os mecanismos para a identificação do risco em uma aplicação e define estratégias de contenção e contingência para o risco. O processo tem como objetivos:

1. Evitar que riscos relevantes para uma aplicação passem desapercebidos por sua atividade de identificação de riscos, anotando os riscos que podem ocorrer em diversos projetos relacionados com os mesmos elementos de projeto antes das atividades de identificação de riscos para aplicações específicas;
2. Aprimorar a produtividade da equipe de gerenciamento de riscos, através da reutilização dos riscos associados a elementos de projeto no processo de análise de riscos para o desenvolvimento de aplicações;
3. Oferecer suporte a preparação de sessões de análise de riscos, onde os elementos de projeto atuam como a base para discussão dos riscos e pontos críticos de uma aplicação;
4. Evitar a "paralisia" na análise de riscos, que pode ocorrer durante as atividades de análise de riscos para uma aplicação, pela falta de heurísticas para iniciar a atividade de identificação de riscos. O processo de análise de riscos para o desenvolvimento de aplicações define estas heurísticas e utiliza uma lista de problemas potenciais relacionados com elementos de projeto.

A reutilização de riscos foi proposta previamente na literatura da Engenharia de Software (KONTIO, 1997, GARVEY, 1997, HALL, 1998). As principais diferenças da reutilização de riscos na abordagem proposta pelo processo de análise de riscos segundo o paradigma de gerenciamento de projetos baseado em cenários são:

1. A utilização de modelos de cenários para representar o impacto dos riscos, planos de contenção e de contingência sobre o comportamento do projeto, assim como a utilização de mecanismos de simulação para avaliar este impacto;
2. A associação entre os riscos e os elementos de projeto, que permite a seleção dos riscos a que uma aplicação está sujeita, de acordo com seus elementos de projeto.

A Figura 6.3 apresenta as atividades que compõem o processo de identificação de riscos para elementos de projeto. Devido a sua estrutura de refinamentos, onde cada atividade refina o artefato criado pela atividade anterior, a arquitetura do processo lembra um ciclo de vida em cascata. Entretanto, esperamos que o processo seja aplicado em diversos interações, cada qual descobrindo novos arquétipos de risco e adicionando informações aos arquétipos previamente documentados.

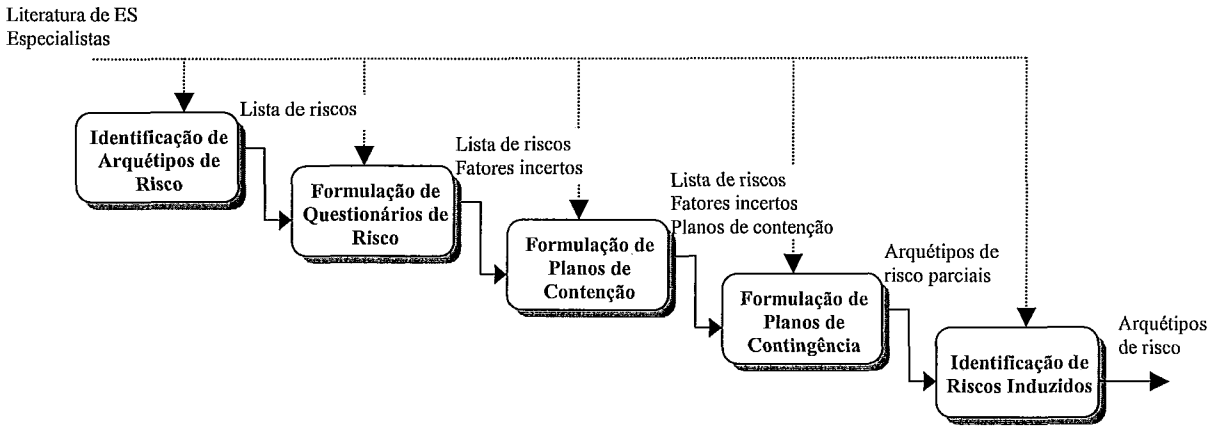

Figura 6.3 – O processo para identificação dos riscos relacionados com um elemento de projeto

A Figura 6.3 segue uma notação adaptada dos diagramas IDEF0 (AIR FORCE, 1981), que são diagramas padronizados para a modelagem de funções de processos de manufatura. A notação original dos diagramas IDEF0 apresenta as entradas, saídas, controles e mecanismos utilizados por cada atividade de um processo. Na notação adotada na Figura 6.3, as atividades são apresentadas em retângulos com bordas arredondadas. As setas conectando as atividades denotam as dependências entre as atividades e os fluxos de controle e artefatos entre elas. Assim, uma seta entrando em uma atividade indica os artefatos que são utilizados por aquela atividade. De forma análoga, uma seta saindo de uma atividade denota os artefatos produzidos ou transformados por aquela atividade. As linhas pontilhadas apontando para todas as

atividades representam mecanismos de controle e artefatos externos ao processo que são utilizados por suas atividades.

A primeira atividade do processo de identificação de riscos para elementos de projeto é a identificação de arquétipos de risco. Esta atividade tem como objetivo descobrir e listar os riscos que ocorrem com freqüência em aplicações que utilizem o elemento de projeto sob análise, ressaltando o contexto que promove a ocorrência destes riscos. Esta atividade utiliza as incertezas, dúvidas, dificuldades e o conhecimento existente sobre o elemento de projeto. Nesta atividade, a equipe de análise de riscos produz uma lista de riscos, sob a forma de arquétipos onde apenas o bloco de identificação de risco está preenchido.

A atividade de identificação de arquétipos de risco também define o modelo de impacto do risco, ou seja, o modelo de cenário que descreve o impacto promovido pela ocorrência do risco sobre o comportamento de um projeto. A dinâmica do modelo de cenário deve ser claramente documentada, suas fontes de dados e referências sendo citadas no arquétipo de risco. Durante o processo de análise de riscos para o desenvolvimento de aplicações, um gerente de projetos deve simular a sensibilidade de seu projeto a diversas combinações de modelos de impacto de riscos em diferentes horizontes de tempo. Estas análises se realizam através da integração dos modelos de impacto dos riscos com o modelo de projeto da aplicação.

Os arquétipos de risco identificados são associados ao elemento de projeto sob análise. As atividades seguintes do processo de identificação de riscos para elementos de projeto preenchem o restante da estrutura de informação dos arquétipos de risco identificados para o elemento de projeto. A próxima atividade, a formulação dos questionários de risco, tem como objetivo listar perguntas que ofereçam auxílio na identificação das condições que promovem a ocorrência dos arquétipos de risco em uma aplicação. Cada arquétipo de risco é analisado de forma independente, formulando-se seu questionário. A identificação de uma nova pergunta deve ser seguida da definição de suas informações de ajuda. Esta atividade resulta em arquétipos de risco refinados, onde se expressam os fatores de incerteza que promovem a ocorrência do risco descrito pelo arquétipo em uma aplicação.

As incertezas que promovem a ocorrência de um risco são expressas no questionário de seu arquétipo e guiam a seleção de estratégias de contenção para o risco. Estas estratégias definem ações que podem controlar as incertezas expressas nas perguntas do questionário, reduzindo a probabilidade ou o impacto da ocorrência do

problema potencial sobre um projeto. A atividade de formulação dos planos de contenção analisa as perguntas do questionário, identificando ações que podem ser utilizadas para reduzir as incertezas expressas nestas perguntas. É importante observar que nenhum projeto pode se prevenir contra todos os seus riscos, mesmo que existam planos de contenção para todas as incertezas que promovam estes riscos. Entretanto, se existem planos preparados para cada situação e os riscos são periodicamente reavaliados, os planos de contenção mais relevantes podem ser ativados ao longo do desenvolvimento do projeto.

Um modelo de cenário é desenvolvido para cada plano de contenção durante a atividade de formulação de planos de contenção. Cada modelo mostra o impacto do plano de contenção sobre o comportamento de um projeto. Como ocorre nos cenários de impacto de risco, um gerente deve simular a sensibilidade de seu projeto aos cenários de contenção durante o processo de análise de riscos para o desenvolvimento de aplicações. Esta análise de sensibilidade é realizada integrando-se os modelos de cenário dos planos de contenção ao modelo de projeto da aplicação em desenvolvimento. Diversos cenários de contenção podem ser combinados com cenários de impacto de riscos, permitindo que o gerente avalie sua eficácia na resolução dos riscos. A atividade de formulação de planos de contenção resulta em arquétipos de risco ainda mais refinados, onde a identificação de riscos, os fatores de incerteza e as ações para o controle destas incertezas estão preenchidos.

Da mesma forma, a atividade de formulação de planos de contingência analisa os arquétipos de risco, identificando ações que reduzam o impacto de sua ocorrência de riscos em um projeto de desenvolvimento de software. Nesta atividade, os cenários de impacto dos planos de contingência são desenvolvidos, podendo ser simulados de forma similar ao impacto dos riscos e dos planos de contenção. Esta atividade resulta em arquétipos de risco parciais, que podem ser utilizados para identificar riscos e preparar suas estratégias de resolução.

A atividade de identificação de riscos induzidos analisa os relacionamentos do arquétipo atual com outros arquétipos de risco previamente desenvolvidos. Nesta atividade, a equipe de análise de riscos identifica riscos similares para cada arquétipo de risco. Além disso, os riscos que são induzidos pela ocorrência de cada arquétipo de risco são identificados. A atividade resulta em uma lista de arquétipos de risco completos, que podem ser recuperados a partir dos elementos de projeto utilizados por uma aplicação.

O resultado do processo de identificação de riscos para elementos de projeto é uma lista de arquétipos de risco completos, porém com a seção de casos de ocorrência conhecidos ainda vazia. Esta seção será preenchida a medida em que os arquétipos de risco forem reutilizados na identificação de riscos em aplicações. Esta reutilização ocorre no segundo sub-processo do processo de análise de riscos do paradigma de gerenciamento de projetos baseado em cenários, que é apresentado na próxima seção.

6.3 O Processo de Análise de Riscos para o Desenvolvimento de Aplicações

O processo para identificação de riscos para elementos de projetos, apresentado na seção anterior, constrói uma base de conhecimento reutilizável descrevendo problemas potenciais que podem afetar uma aplicação que utilize determinados elementos de projeto, como tecnologias, domínios de aplicação, papéis desempenhados por desenvolvedores e artefatos de software.

O processo de análise de riscos para o desenvolvimento de aplicações se refere ao segundo sub-processo do processo genérico de reutilização, que reutiliza os artefatos construídos pelo primeiro sub-processo para identificar os riscos que podem afetar o desenvolvimento de uma aplicação. Este processo explora os elementos de projeto utilizados pela aplicação, rastreando os problemas potenciais associados a eles. Um modelo de projeto, previamente desenvolvido para a aplicação, determina os elementos de projeto relacionados com a aplicação. Os arquétipos de risco, também previamente associados a elementos de projeto, determinam o contexto em que os riscos por ele documentados ocorrem, os mecanismos para avaliar seu impacto no projeto, planos de contenção para inibir ocorrência do risco e planos de contingência para reduzir seu impacto sobre o comportamento do projeto.

Os objetivos do processo de análise de riscos para o desenvolvimento de aplicações são determinar os riscos que podem afetar uma aplicação de acordo com seus elementos de projeto, avaliar o impacto destes riscos sobre o comportamento do projeto de desenvolvimento da aplicação, selecionar os riscos mais importantes para a aplicação, desenvolver planos de tratamento para estes riscos, monitorar estes riscos em relação ao estado do projeto e implantar os planos de contenção e contingência a medida em que estes se façam necessários. Este processo tem como objetivo:

1. Aprimorar a capacidade da equipe de gerenciamento de riscos para analisar os diversos riscos que podem afetar uma aplicação, evitando que riscos relevantes

passem desapercebidos. O processo proposto procura atingir este objetivo através da reutilização de arquétipos de risco que documentem problemas potenciais associados aos elementos de projeto utilizados em uma aplicação. Estes elementos de projeto, que são apresentados no modelo de projeto da aplicação, fazem a ligação entre a aplicação e a base de conhecimento sobre riscos;

2. Aprimorar a precisão na seleção de riscos, ou seja, a capacidade da equipe de gerenciamento para analisar apenas os riscos que são relevantes para a aplicação. Inicialmente, os elementos de projeto utilizados em uma aplicação guiam a reutilização dos arquétipos de risco, focalizando a atenção da equipe de gerenciamento de riscos nos riscos relevantes para a aplicação. Em seguida, este conjunto de riscos é filtrado pelas respostas aos questionários dos arquétipos, que podem sugerir que um determinado risco, mesmo relacionado com um dos elementos de projeto da aplicação, não é relevante para o projeto;
3. Aprimorar a avaliação e planejamento de riscos, através da utilização de modelos dinâmicos, que são capazes de capturar relações complexas entre os elementos de projeto. Modelos de impacto de riscos, integrados ao modelo de projeto de uma aplicação, são utilizados para avaliar o impacto dos riscos através de simulação. Diversos modelos de impacto de planos de contenção e contingência também podem ser integrados ao modelo de projeto, analisando sua eficácia na resolução do risco e seu impacto sobre o comportamento do projeto;
4. Aprimorar o acompanhamento e controle de riscos, através de alertas baseados no modelo de projeto. Estes alertas avisam o gerente da existência de condições que fazem com que um risco ocorra em seu projeto. Eles são calculados sempre que novas informações sobre o estado do projeto estão disponíveis e são atualizadas no modelo de projeto;
5. Evoluir a base de conhecimento sobre riscos, através da criação de novos arquétipos de risco e da evolução de arquétipos previamente desenvolvidos com os novos problemas enfrentados durante o desenvolvimento de uma aplicação.

A Figura 6.4 apresenta as atividades que compõem o processo de análise de riscos para o desenvolvimento de aplicações. As primeiras atividades do processo seguem um ciclo de vida em cascata, enquanto as atividades seguintes são executadas em diversas iterações durante o processo de desenvolvimento da aplicação. As atividades que

compõem o trecho do processo executado como um ciclo de vida em cascata recebem uma primeira versão do modelo de projeto de uma aplicação, selecionando os riscos associados a seus elementos de projeto. As atividades executadas iterativamente controlam os riscos a partir das informações de estado atualizadas no modelo de projeto da aplicação, verificando os alertas dos planos de contenção ao longo do processo de desenvolvimento. A notação utilizada Figura 6.4 é a mesma previamente utilizada na Figura 6.3 e descrita na seção 6.2.

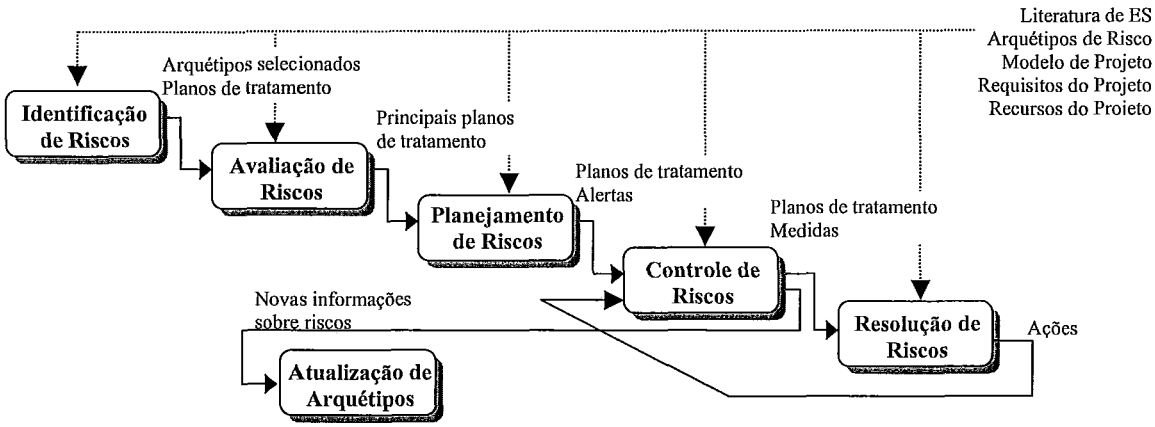

Figura 6.4 – O processo de análise de riscos para o desenvolvimento de aplicações

A primeira atividade do processo de análise de riscos para o desenvolvimento de aplicações é a identificação de riscos, que analisa os elementos de projeto utilizados pelo modelo de projeto de uma aplicação para determinar os riscos que podem afetar o comportamento do projeto. Estes riscos foram previamente documentados como arquétipos de risco e associados aos elementos de projeto pelo processo de identificação de riscos para elementos de projeto. Na atividade de identificação de riscos, a equipe de gerenciamento de riscos consulta os elementos de projeto apresentados no modelo de projeto da aplicação, recuperando os arquétipos de risco associados a estes elementos. Riscos induzidos, indicados nos arquétipos de risco dos elementos de projeto, também são recuperados. O gerente de projeto também pode navegar pelos arquétipos de risco similares e por suas ocorrências, selecionando outros arquétipos.

Ainda na atividade de identificação de riscos, o gerente de projeto responde aos questionários de identificação dos arquétipos de risco. Estas respostas são registradas em **planos de tratamento de riscos**, que são formulários armazenados junto à aplicação que contêm informações específicas sobre a ocorrência dos arquétipos em um projeto. Por exemplo, enquanto os arquétipos de risco contêm questionários para a identificação de riscos em diversos projetos, um plano de tratamento de riscos contém as respostas

dadas pelo gerente de projeto de uma aplicação para este questionário. Um plano de tratamento de riscos está associado a um único arquétipo de risco. A Figura 6.5 apresenta um exemplo de plano de tratamento de risco para o risco de validação.

Identificação	Contenção
Arquétipo de Risco: Risco de Validação Questionário: a) Grau de ambigüidade: baixo b) Requisitos não definidos: médio c) Envolvimento do usuário: médio d) Técnicas de validação: baixo Probabilidade: alta Impacto: Tempo (estimado): 6 meses Custo (estimado): $ 50.000,00	Estratégia: Desenvolvimento de protótipos Eficácia: os protótipos têm se mostrado eficazes na elicitação de requisitos funcionais previamente desconhecidos Impacto: Tempo (avaliado): 3 meses Custo (avaliado): $ 20.000,00 Alertas: o plano será ativado desde o início do projeto
	Contingência
	Estratégia: Extensão do cronograma Eficácia: o plano ainda não foi aplicado Impacto: Tempo (avaliado): 2 meses Custo (avaliado): $ 12.000,00

Figura 6.5 – Um exemplo de plano de tratamento de risco

As respostas aos questionários de identificação dos arquétipos de risco filtram a lista inicial de arquétipos selecionados para o projeto, focalizando as atenções da equipe de gerenciamento nos riscos mais relevantes, de acordo com as características do projeto. Estas respostas serão utilizadas também para a seleção de estratégias de contenção para inibir a ocorrência dos riscos no projeto.

Os planos de tratamento de riscos contêm ainda informações sobre a probabilidade de ocorrência do risco e sobre as estratégias selecionadas para conter ou resolver este risco, de acordo com os planos de contenção e de contingência definidos em seu arquétipo. Finalmente, os arquétipos de risco são associados aos planos de tratamento de riscos em suas listas de casos de ocorrência conhecidos, que indicam ocorrências prévias do risco documentado no arquétipo em projetos de desenvolvimento de software. As atividades que seguem a identificação dos riscos no processo de análise de riscos para o desenvolvimento de aplicações preenchem as informações dos planos de tratamento de riscos.

Na atividade de avaliação de riscos, a equipe de gerenciamento de risco estima a probabilidade de ocorrência e o impacto de cada risco selecionado na atividade anterior sobre o comportamento do projeto, utilizando o modelo de impacto do risco apresentado nos arquétipos de risco selecionados. A equipe também pode avaliar o impacto da

combinação de diversos riscos, através da integração combinada de diversos modelos de impacto de riscos com o modelo de projeto da aplicação. Esta característica permite que o processo de análise de riscos capture os efeitos de diversificação, onde um risco compensa a ocorrência de outros riscos, reduzindo o nível de risco do projeto como um todo quando os dois riscos são analisados em conjunto.

A atividade de avaliação de riscos começa com a avaliação subjetiva da probabilidade de ocorrência de cada risco selecionado no projeto para desenvolvimento da aplicação. As respostas aos questionários de identificação dos arquétipos de risco podem ser utilizadas como suporte para esta estimativa. A equipe de gerenciamento de riscos deve indicar sua estimativa sobre a probabilidade de ocorrência de cada risco, anotando estas estimativas nos respectivos planos de tratamento do risco. A estimativa deve seguir uma escala nominal formada por cinco termos: muito alta, alta, média, baixa e muito baixa. Esta classificação será posteriormente utilizada na priorização dos riscos.

A atividade de avaliação de riscos prossegue com a seleção das variáveis do modelo de projeto da aplicação em desenvolvimento que são consideradas relevantes para a avaliação do impacto dos riscos. A equipe de gerenciamento de riscos seleciona, dentre as variáveis que descrevem o comportamento do projeto, tais como custo, cronograma e qualidade, aquelas que serão utilizadas na análise de impacto dos riscos.

A seleção das variáveis consideradas na análise de impacto dos riscos depende dos requisitos do projeto e dos recursos destinados a sua realização. Por exemplo, se um projeto deve atender a restrições sobre um atributo não funcional, como desempenho ou confiabilidade, uma variável no modelo de projeto que represente este atributo pode ser considerada relevante na análise de impacto dos riscos. Por outro lado, se o projeto deve ser concluído em uma determinada data, uma variável que represente o tempo para conclusão do projeto pode ser selecionada para a análise de impacto dos riscos.

Em seguida, são realizadas análises de impacto para os diversos riscos selecionados na atividade de identificação dos riscos. Estas análises são realizadas integrando-se os modelos de impacto de cada risco ao modelo de projeto da aplicação em desenvolvimento e avaliando-se as alterações sofridas pelas variáveis selecionadas após a integração e simulação de cada modelo de cenário. Cada impacto de cada risco sobre uma variável selecionada no modelo de projeto é anotado nos planos de tratamento do risco. Cada impacto em cada variável também é classificado segundo a mesma escala nominal utilizada na determinação da probabilidade de ocorrência de um risco. Esta

classificação, junto com a classificação determinada para a probabilidade de ocorrência do risco, será utilizada na priorização dos riscos.

Na priorização dos riscos, recomendamos um método similar ao utilizado pelo processo apresentado por KONTIO (1997). Este método dispõe os riscos que podem afetar um projeto em uma matriz, de acordo com suas probabilidades de ocorrência e seu impacto nas variáveis de comportamento de projeto selecionadas, e utiliza um algoritmo para percorrer a matriz, gerando uma lista parcialmente ordenada de riscos.

Antes da priorização dos riscos, o impacto e a probabilidade de ocorrência de cada risco foi avaliados e classificado em uma escala nominal de cinco termos. O processo proposto por KONTIO (1997) considera apenas uma dimensão de impacto de risco, que são mapeados em uma matriz de duas dimensões, uma representando a probabilidade de ocorrência e a outra representando o impacto do risco. Cada célula da matriz pode conter diversos riscos e representa o subconjunto dos riscos com uma mesma probabilidade de ocorrência e um mesmo impacto, conforme a classificação qualitativa realizada anteriormente. A Figura 6.6 apresenta um exemplo da matriz utilizada no processo de análise de riscos proposto por KONTIO (1997).

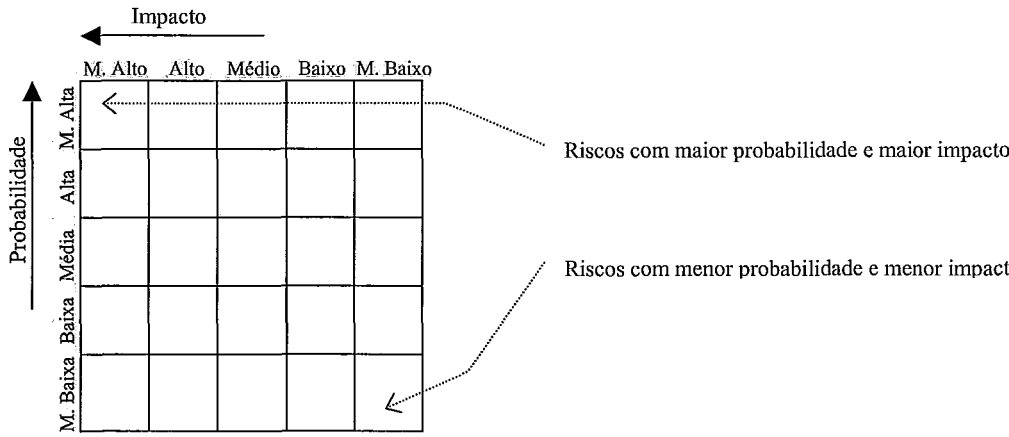

Figura 6.6 – Matriz de priorização de riscos do processo de KONTIO (1997)

O processo de geração da lista parcialmente ordenada de riscos percorre diagonalmente a matriz de priorização, começando no seu canto superior esquerdo e seguindo até o seu canto inferior direito, como se a matriz fosse cortada por uma reta perpendicular a sua diagonal principal. Em cada passo do percurso os riscos das células atravessadas pela reta são inseridos na lista parcialmente ordenada. Nada se pode afirmar sobre a ordem dos riscos inseridos na lista em cada iteração do corte: assim, a lista resultante do processo de priorização é dita parcialmente ordenada. Dados pesos

iguais para a relevância de cada dimensão da matriz, os riscos selecionados em iterações anteriores à iteração atual são mais relevantes para o projeto, por possuírem maior probabilidade de ocorrência ou maior impacto. A Figura 6.7 apresenta os três primeiros estágios do corte de uma matriz hipotética de riscos.

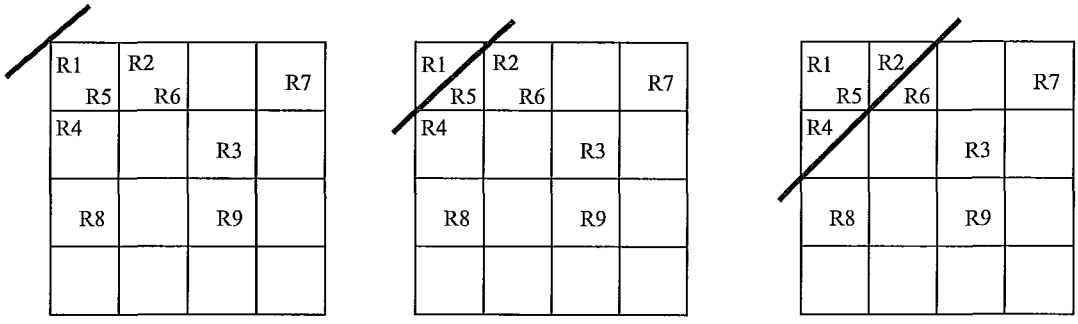

Figura 6.7 - Três primeiros estágios da geração da lista ordenada em uma matriz hipotética

Como o processo de análise de riscos para o desenvolvimento de aplicações analisa o impacto de um risco sobre diversas variáveis, propomos a utilização de uma matriz contendo tantas dimensões quantas variáveis forem selecionadas para a análise de impacto dos riscos, mais uma dimensão para a probabilidade de ocorrência do risco. O processo de corte passa a ser realizado não por uma reta, mas por um hiperplano (N dimensões) cortando o hipercubo (N dimensões) que representaria a matriz.

Após a priorização, os riscos mais relevantes são selecionados, enquanto os planos de tratamento de riscos remanescentes são descartados. A seleção reduz os recursos necessários para o planejamento, controle e acompanhamento dos riscos ao longo do processo de desenvolvimento da aplicação, focalizando os esforços e os recursos existentes nos riscos mais relevantes. O número de planos de tratamento de riscos levados para as próximas atividades depende dos recursos disponíveis e de características específicas do projeto.

A atividade de planejamento de riscos recebe os planos de tratamento de riscos selecionados na atividade anterior. A equipe de gerenciamento de riscos desenvolve estratégias para conter estes riscos e define alertas para indicar quando os planos de contenção dos riscos devem ser ativados. Esta atividade analisa as respostas aos questionários dos arquétipos de risco, selecionando os planos de contenção definidos pelos arquétipos de acordo com as respostas ao questionário. Em seguida, os alertas são definidos para a ativação dos planos de contenção selecionados. Cada alerta é associado

a variáveis do modelo de projeto ou dos cenários de impacto de riscos, cujos valores são comparados a limites de alerta indicados no plano de tratamento de riscos sempre que o modelo de projeto for atualizado com novas informações de estado.

As atividades de controle e resolução de riscos formam a parte iterativa do processo de análise de riscos para o desenvolvimento de aplicações. Na atividade de controle de riscos, a equipe de gerenciamento de riscos reavalia os alertas para ativação dos planos de contenção sempre que o modelo de projeto da aplicação em desenvolvimento for atualizado com novas informações de estado. Os novos valores calculados para os alertas são comparados com limites previamente estabelecidos nos planos de tratamento de riscos.

A atividade de resolução de riscos é executada quando um risco ocorre em um projeto. Estratégias de contingência adequadas são selecionadas e ativadas sobre o projeto. O impacto dos planos de contingência sobre o comportamento do projeto é avaliado nesta atividade.

Finalmente, a atividade de atualização de arquétipos enriquece a base de conhecimento sobre riscos, complementando os arquétipos de risco com problemas que não haviam sido percebidos em aplicações previamente desenvolvidas. Nesta atividade, a equipe de desenvolvimento refina os arquétipos de risco, indicando novos planos de contenção e de contingência e inserindo a aplicação entre os casos conhecidos de ocorrência dos arquétipos. A atividade é executada após o encerramento do processo de desenvolvimento da aplicação. Ela é a conexão dos dois sub-processos de análise de riscos, que atualiza a base de artefatos reutilizáveis e cria a necessidade por novos arquétipos, eventualmente disparando um novo processo de identificação de riscos para elementos de projeto.

6.4 Um Exemplo de Aplicação dos Processos de Análise de Riscos

Nesta seção, apresentamos um exemplo de aplicação dos processos de análise de riscos propostos em projetos reais. Os resultados apresentados aqui derivam de um estudo observacional que utilizou os dois sub-processos em dois projetos desenvolvidos para o mesmo domínio de aplicação. O domínio selecionado foi o gerenciamento de riscos no mercado financeiro. Os projetos selecionados envolvem a especialização de um pacote de gerenciamento de riscos para o caixa, a dívida e o ativo operacional de duas grandes empresas brasileiras nos setores de siderurgia e petróleo.

Antes da realização do primeiro projeto, existia pouca informação sobre os riscos relacionados com o desenvolvimento de um software de gerenciamento de riscos para empresas brasileiras. A equipe responsável pelo projeto tinha experiência prévia no desenvolvimento de um software de gerenciamento de riscos para um banco de investimentos. Entretanto, a especialização deste software para uma empresa apresentou diversas diferenças, entre elas:

1. Os horizontes de análise de riscos são diferentes: em geral, bancos de investimentos estão interessados em flutuações de mercado de curto prazo, como variações de taxas de câmbio e preços de ações dentro de um mesmo dia. Eles precisam saber sua perda potencial de patrimônio devido a variações ocorridas nas informações de mercado até o dia seguinte. Por outro lado, as empresas estão interessadas na suscetibilidade de suas carteiras de médio e longo prazo a variações de mercado. Estas carteiras se estendem por diversos anos e dependem de diversas taxas de câmbio futuras (como dólar, euro e yen), além de outros riscos (juros americanos, risco soberano, entre outros). As empresas geralmente estão mais interessadas na variação de seus lucros e de seu valor de mercado devido a flutuações de mercado ao longo de um ano ou semestre fiscal;
2. Commodities: quando o primeiro projeto foi realizado, o banco de investimentos onde o software de gerenciamento de riscos foi desenvolvido tinha pouco interesse na negociação de commodities, como café, ouro, petróleo ou aço. Entretanto, estes commodities são fundamentais para as empresas, que vendem estes produtos em um mercado globalizado. Como o mercado determina os preços destes commodities, a capacidade da empresa em obter lucro ou prejuízo depende das variações destes preços. Mesmo que a empresa venda seus produtos apenas no mercado local, dentro do país, ela pode utilizar insumos importados para suas linhas de produção. Mercados de contratos futuros, como o *London Mercantile Exchange* (LME), podem regular os preços destes insumos.

Estes são apenas dois exemplos para mostrar as diferentes visões do gerenciamento de riscos para um banco de investimentos e uma empresa. O primeiro projeto foi realizado em uma empresa do setor siderúrgico. Este projeto estava sujeito a alguns riscos relacionados com estas características, além de alguns riscos genéricos de projetos de desenvolvimento de software, como volatilidade de requisitos, por exemplo. Os riscos dependentes de domínio foram documentados como arquétipos de risco e

associados a um elemento de projeto denominado *domínio de software de gerenciamento de riscos no mercado financeiro*. Abaixo, apresentamos exemplos destes riscos:

1. Modelagem de commodities: devido a sua sazonalidade e dependência de fatores externos, como condições de tempo, empresas de provisionamento, pragas e infestações, as incertezas associadas com determinados commodities são difíceis de modelar. Processos estocásticos conhecidos (e geralmente complexos) descrevem o comportamento dinâmico dos preços destes commodities, como petróleo e ouro. A mensuração dos riscos dependentes destes processos também é uma tarefa difícil, especialmente nos derivativos de commodities;
2. Avaliação de riscos baseada em simulações de Monte Carlo: estas simulações são utilizadas com freqüência para mensurar os riscos associados com instrumentos do mercado financeiro que não podem ser avaliados analiticamente. Enquanto as simulações de Monte Carlo são flexíveis e representam uma plataforma confiável para a avaliação de riscos, elas exigem muitos recursos computacionais, reduzindo o desempenho da avaliação de riscos. A dependência em simulações de Monte Carlo para a avaliação de grandes carteiras de instrumentos financeiros pode reduzir o desempenho do software de análise de riscos, visto que seu usuário deverá esperar muito pela resposta de cada análise que deseja executar;
3. Volatilidade da composição das carteiras: definido aqui como a freqüente inclusão de novos tipos de instrumentos financeiros nas carteiras da empresa, a volatilidade de composição das carteiras é uma faceta dependente de domínio dos riscos impostos pela volatilidade de requisitos. Cada vez que um novo instrumento é inserido no software de gerenciamento de riscos, seu mapeamento para fatores de risco deve ser definido, projetado, codificado e testado. Eventualmente, novos fatores de risco e dados históricos sobre sua variação no tempo também precisam ser adicionados.

Mesmo sem uma avaliação formal de seus riscos, o primeiro projeto foi considerado como bem sucedido, sendo concluído com apenas uma semana de atraso em seu cronograma de quatro meses. O projeto foi dividido em quatro fases: elicitação de carteiras, análise estatística, mapeamento de instrumentos para fatores de risco e avaliação do mapeamento. Após a conclusão do projeto, arquétipos de risco foram documentados, através do processo de identificação de riscos para elementos de projeto.

O segundo projeto foi desenvolvido em uma empresa do setor petrolífero. Este projeto executou o processo de análise de riscos para o desenvolvimento de aplicações de forma simplificada, utilizando os arquétipos de risco desenvolvidos após o primeiro projeto. O segundo projeto foi maior do que o primeiro, visto que a empresa possui diversas carteiras de ativo operacional, distribuídas ao longo de suas diversas subsidiárias e diferenciados nos produtos vendidos, comprados ou distribuídos. O projeto foi programado para ser realizado em seis meses, mas se atrasou em uma semana. Acreditamos que o projeto poderia ter se atrasado ainda mais se não fosse a análise dos arquétipos de risco e a experiência obtida na realização do projeto anterior.

Os arquétipos de risco foram utilizados como alerta de problemas no processo de desenvolvimento utilizado no segundo projeto. Quando a equipe de análise de carteiras, composta por engenheiros financeiros, produziu uma lista de instrumentos financeiros para a equipe de mapeamento, composta de analistas de sistemas e programadores, os arquétipos de risco produzidos pelo primeiro projeto auxiliaram os desenvolvedores na identificação dos instrumentos que representavam riscos para o projeto. Os analistas de carteiras puderam, então, se concentrar na decomposição destes elementos, replicando seu comportamento a partir de um conjunto de instrumentos de mercado mais simples sempre que possível. Simulações de Monte Carlo, necessárias para a avaliação dos instrumentos complexos, foram oferecidas como ferramenta complementar, utilizada apenas para análises mais detalhadas, enquanto aproximações analíticas foram utilizadas para o controle de riscos realizado periodicamente pela empresa.

Cinco novos arquétipos de risco também foram gerados a partir do conhecimento adquirido após o encerramento do segundo projeto. Alguns exemplos destes arquétipos de risco são apresentados no Apêndice G. Cenários de impacto de risco não foram utilizados para avaliar o impacto dos riscos sobre o comportamento dos projetos porque as ferramentas necessárias para esta avaliação ainda não se encontravam totalmente desenvolvidas e dados históricos para a construção dos modelos de cenário e projeto não estavam disponíveis. Entretanto, dados quantitativos sobre a duração das atividades, a volatilidade das carteiras, a geração e correção de erros nos projetos foram registrados e estão disponíveis para a construção de modelos de cenários. Está prevista uma análise mais completa em projetos futuros, desta vez aplicando os modelos dinâmicos.

6.4.1 Estudos Futuros

O estudo observacional descrito na seção 6.4 não foi planejado com o mesmo nível de detalhamento do estudo experimental apresentado no Capítulo 5. Parte desta falta de rigor pode ser atribuída aos diferentes momentos em que os estudos foram realizados, ao estágio de desenvolvimento do metamodelo da Dinâmica de Sistemas e ao reduzido controle que poderia ser exercido sobre o ambiente e os participantes do estudo observacional. Entretanto, para melhor sustentar as propostas específicas do processo de análise de riscos do paradigma de gerenciamento baseado em cenários, pretendemos realizar novos estudos precedidos por um maior planejamento.

Dentre os estudos que se pretende realizar, podemos citar a continuação do estudo observacional no domínio do gerenciamento de riscos no mercado financeiro. Após a realização do estudo observacional, o grupo responsável pelos projetos de análise de riscos na empresa em questão continuou seu trabalho, implantando sistemas de análise de riscos em outras empresas de diversos setores da economia. Com o crescimento do negócio, surgiu a necessidade de maior controle, assim como a necessidade de avaliar o riscos das decisões tomadas pela equipe de desenvolvimento e pela gerência da empresa.

Consideremos o seguinte cenário: os dados utilizados para o cálculo de risco são coletados através de planilhas, que são distribuídas para as empresas clientes, mas também para bancos e controladoras de fundos de investimento em que as empresas clientes possuem participação. Sempre que algum instrumento financeiro da carteira da empresa é alterado, seja pela criação ou remoção de seus campos devido a uma mudança de contrato, um avanço tecnológico do sistema de análise de riscos ou a correção de um erro latente no código de mapeamento, as planilhas devem ser modificadas e remetidas aos clientes e seus associados. A substituição freqüente de planilhas altera a rotina dos clientes, eventualmente gerando atrito entre estes e a empresa prestadora de serviços de análise de riscos, o que incorre em risco de reputação para a última. Por outro lado, a inclusão ou remoção de um campo de um instrumento pode facilitar o levantamento de seus dados e oferecer maior precisão aos resultados gerados pelo sistema de risco, trazendo benefícios futuros para os clientes.

A decisão de trocar ou não uma planilha não é trivial, devido a seu relacionamento com diversos outros fatores (como custo e reputação) e à distância no tempo de suas conseqüências (problemas no curto prazo; vantagens no médio e longo prazo). Como

medir o impacto deste *trade-off*? Acreditamos que arquétipos de risco podem ser utilizados para descrever a situação, ressaltando as forças positivas e negativas que atuam sobre a decisão e apresentando modelos de cenário que permitam sua avaliação quantitativa. Uma análise de viabilidade mais formal do processo de análise de riscos passa pela criação de um maior conjunto de arquétipos de risco e pela avaliação de seu impacto em um projeto real. Este tipo de estudo experimental é difícil de realizar, pois suas características inviabilizam a repetição *in vitro* (em laboratório) exigindo situações reais de projeto.

6.5 Uma Comparação com Outros Processos de Análise de Riscos

A Tabela 6.1 compara o processo de análise de riscos proposto pelo paradigma de gerenciamento de projetos baseado em cenários com alguns processos de análise de riscos apresentados no Capítulo 2.

O processo proposto apresenta algumas diferenças quando comparado aos demais processos de análise de riscos. Primeiro, o processo proposto considera as relações entre o gerenciamento de riscos e os elementos de projeto utilizados no desenvolvimento de uma aplicação, tais como seu domínio, as tecnologias aplicadas, os artefatos construídos, os recursos utilizados e os desenvolvedores responsáveis pelo projeto. Os elementos de projeto auxiliam na seleção dos riscos relevantes para um projeto, servindo como um primeiro filtro do universo de arquétipos de risco disponíveis na base de conhecimento da empresa. Os questionários de identificação presentes nos arquétipos atuam como um segundo filtro, selecionando, dentre os arquétipos relacionados com os elementos de projeto utilizados pela aplicação, aqueles que são relevantes para o projeto, de acordo com suas características e seu contexto.

	Hall 98	Willians 97	Grey 95	Madachy 97	Fairley 94	Kontio 97	Garvey 97	Gemmer 97	Processo Proposto
Detalhamento	✓	✓			parcial	✓	✓	✓	✓
Reutilização						✓	✓		✓
Agregação Complexa			✓	✓	✓	parcial			✓
Planejamento	✓	✓			✓	✓	✓		✓
Controle	✓	✓			✓	✓			✓

Tabela 6.1- Comparação do processo proposto com outros processos de análise de riscos

Além disso, os arquétipos de risco representam uma forma estruturada para a documentação de riscos, contribuindo para o atendimento do primeiro requisito da Tabela 6.1, que sugere que a representação utilizada pelo processo de análise de riscos

contenha informações detalhadas sobre o problema potencial sendo documentado. Outros processos de análise de riscos (JONES, 1994, GARVEY, 1997) utilizam estruturas de dados padronizadas para a documentação de riscos, mas as informações contidas nestas estruturas são limitadas em sua capacidade de promover a avaliação quantitativa dos riscos.

Em seguida, o processo de análise de riscos proposto enfatiza a reutilização das informações sobre os riscos, visando atender ao segundo requisito indicado na Tabela 6.1. O processo de análise de riscos segue um processo genérico de reutilização, onde um sub-processo produz informações reutilizáveis sobre os riscos a que um projeto está sujeito, enquanto outro sub-processo guia a utilização destas informações na análise de riscos para o desenvolvimento de uma aplicação.

A utilização de modelos dinâmicos do processo de desenvolvimento de software permite que o gerente analise quantitativamente o impacto dos riscos selecionados sobre o comportamento (custo, tempo, qualidade, entre outros) de seu projeto. Sendo representados como modelos de cenários, os modelos de impacto de riscos podem capturar as relações complexas existentes entre os riscos e outros componentes do projeto. Os modelos de impacto dos riscos podem ser agregados isoladamente ou em conjunto com o modelo de projeto desenvolvido para a aplicação. Quando agregados em conjunto, a integração permite capturar os efeitos de diversificação na avaliação dos riscos.

A utilização de modelos dinâmicos na análise de impacto dos riscos contribui para o terceiro requisito da Tabela 6.1. Consideramos que, dentre as abordagens de análise de riscos de amplo escopo[1] previamente apresentadas na literatura, a proposta de KONTIO (1997) é a que mais se aproxima de atingir este requisito. Entretanto, acreditamos que as funções de utilidade sugeridas na abordagem de KONTIO (1997) podem se tornar extremamente complexas quando diversos aspectos do processo de desenvolvimento de software são considerados em conjunto, devido as relações não-lineares e aos ciclos de realimentação que podem existir entre eles. Acreditamos que a utilização de modelos de cenários representa uma melhor forma de observar a cascata de impactos impostos por

[1] Excluem-se aqui as abordagens de análise de risco quantitativas que focalizam um conjunto limitado e previamente definido de aspectos dos processos de desenvolvimento, como as propostas de GREY (1995), FAIRLEY (1994) e MADACHY (1997). Estas propostas capturam relações complexas entre os aspectos analisados, mas não permitem que a equipe de gerenciamento de riscos formule hipóteses sobre a influência de outros fatores sobre estes aspectos (modelo fixo).

um risco sobre o comportamento esperado para um projeto. Entretanto, o estudo observacional realizado não nos permite verificar esta hipótese.

Os modelos de cenário também são utilizados para avaliar o impacto dos planos de contenção e de contingência de riscos durante a atividade de planejamento. Assim como os modelos de impacto de riscos, os modelos de impacto das estratégias de resolução também permitem capturar as relações complexas existentes entre os planos e os elementos de projeto utilizados pela aplicação. O processo de análise de riscos proposto contempla a atividade de planejamento de riscos, visando atender ao quarto requisito da Tabela 6.1.

Finalmente, o controle de riscos, quinto e último requisito da Tabela 6.1, é realizado pela atualização do modelo de projeto da aplicação com informações sobre o estado do projeto e posterior reavaliação dos modelos de impacto de risco e de estratégias de resolução. Além disso, o processo de gerenciamento de riscos proposto permite o acompanhamento dos alertas previamente definidos para os planos de contenção através de variáveis do modelo.

O processo de análise de riscos proposto também possui suas limitações. Uma das principais limitações está relacionada com a combinação de impacto de cenários. Embora a utilização de modelos de cenário permita que o gerente analise o impacto de combinações destes cenários, o sub-processo de análise de riscos para o desenvolvimento de aplicações não oferece heurísticas para o teste de combinação de modelos de impacto de cenários, restringindo-se à avaliação de cenários independentes. Isto se deve ao fato do número de combinações possíveis de um conjunto de cenários crescer muito rapidamente com o tamanho do conjunto. Assim, algum suporte automatizado de combinação e teste de cenários se faz necessário para que o gerente não tenha que avaliar todas as combinações manualmente. A priorização de cenários, durante a atividade de avaliação de riscos, também deve ser aprimorada para levar em consideração as combinações de cenários de risco.

7 Conclusões

Conforme ressaltamos no Capítulo 2 desta tese, a aplicação de modelagem quantitativa e simulação na descrição e avaliação de projetos de desenvolvimento de software vêm crescendo nas últimas duas décadas (LIN e LEVARY, 1989, HUMPHREY e KELLNER, 1989, ABDEL-HAMID e MADNICK, 1991, RAFFO, 1993, HANSEN, 1996, RODRIGUES e WILLIANS, 1996, TVEDT, 1996, MARTÍNEZ-GARCÍA e WARBOYS, 1998, RUS e COLLOFELO, 1998, SCHACCI, 1999, LEBSANFT e PFAHL, 1999).

Neste contexto, esta tese apresenta mais um passo na direção de facilitar a construção e ampliação do conhecimento representado em modelos de projeto de desenvolvimento de software. Observamos alguns requisitos desejados para as técnicas de modelagem atualmente utilizadas na construção destes modelos e propomos uma técnica de modelagem que busca atender estes requisitos em determinado grau.

Este capítulo está dividido em três seções. Na seção 7.1, apresentamos as principais contribuições desta tese. Na seção 7.2, apresentamos algumas limitações do atual estado de desenvolvimento das técnicas propostas. Finalmente, na seção 7.3 apresentamos algumas perspectivas de trabalho futuro.

7.1 Contribuições da Tese

As principais contribuições desta tese podem ser divididas em quatro categorias: a representação de conhecimento gerencial, o desenvolvimento de ferramentas de suporte, o planejamento e execução de um estudo experimental para analisar a viabilidade de aplicação das técnicas propostas e o processo de análise de riscos. Abaixo, relacionamos as contribuições classificadas segundo estas categorias.

Representação de Conhecimento Gerencial

- A definição das bases do paradigma de gerenciamento de projetos baseado em cenários, incluindo a postulação de seus princípios, a definição de seus principais artefatos (os modelos de projeto e os modelos de cenários), a determinação dos responsáveis pela construção, manutenção e reutilização destes artefatos e a definição do processo em que eles são utilizados durante o planejamento e o controle de projetos de desenvolvimento de software;

- A definição de um metamodelo para a técnica de modelagem da Dinâmica de Sistemas, que permite a construção de modelos em um nível de abstração mais alto, ou seja, mais próximo aos conceitos do domínio do problema sendo modelado. O metamodelo se baseia na construção de um modelo de domínio, que define as classes de elementos pertencentes ao domínio do problema e os relacionamentos que podem ocorrer entre as instâncias destas classes. O comportamento de cada classe é especificado através de construtores tradicionais da Dinâmica de Sistemas. Sempre que um desenvolvedor de modelos construir um modelo para o domínio, este modelo é desenvolvido com base nas classes definidas no modelo de domínio, reutilizando-se o comportamento previamente definido para estas classes;
- A definição de um processo de modelagem que suporta a aplicação do metamodelo da Dinâmica de Sistemas. Este processo se divide em três atividades: a construção de um modelo para o domínio, a construção de um modelo dentro de um domínio e a tradução do modelo representado na linguagem do domínio para os construtores tradicionais da Dinâmica de Sistemas;
- A definição de uma representação para o conhecimento gerencial, denominada modelos de cenários. Estes modelos representam as estratégias, táticas, ações, eventos incertos, procedimentos e políticas gerenciais que podem ser aplicadas ou impostas a um projeto de desenvolvimento de software;
- A definição de uma técnica que permite avaliar os efeitos da aplicação de uma estratégia, ação, procedimento ou política gerencial sobre um projeto de software. Esta técnica se baseia na integração de modelos de cenário a um modelo do projeto de software em questão, na simulação do modelo integrado e na comparação das mudanças de comportamento provocadas pelos cenários sobre o modelo de projeto. Esta técnica focaliza a transferência de conhecimento, ou seja, a capacidade de apresentar ao usuário não somente a informação (o modelo de cenário), mas também os resultados de sua aplicação no contexto do usuário (a alteração de comportamento provocada pelo cenário no modelo de projeto).

Implementação de Ferramentas de Suporte

- A especificação de um compilador que traduz um modelo descrito segundo o metamodelo da Dinâmica de Sistemas para os construtores tradicionais desta técnica

de modelagem. O compilador permite que o modelo descrito a partir das classes definidas no modelo de domínio possa ser simulado e avaliado em ferramentas de simulação tradicionais da Dinâmica de Sistemas;

- A implementação de uma ferramenta de simulação para modelos construídos com base nos construtores tradicionais da Dinâmica de Sistemas. Esta ferramenta, denominada *Illium*, é apresentada em detalhes no Apêndice E. *Illium* oferece diversos mecanismos de simulação para a avaliação dos modelos compilados, incluindo simulação de Monte Carlo, simulações no tempo e simulações com variação de parâmetros. Em (BARROS et al., 2000b) exemplos de utilização destes mecanismos de simulação na avaliação de alternativas de aprimoramento de processos de software são apresentados;
- A implementação de uma primeira versão do compilador que traduz os modelos construídos com base nas classes definidas em um modelo de domínio para os construtores tradicionais da Dinâmica de Sistemas. Este compilador foi desenvolvido como parte integrante da ferramenta *Hector*, também apresentada no Apêndice E.

Planejamento e Execução do Estudo Experimental

- O planejamento de um estudo experimental para análise da viabilidade de aplicação das técnicas de integração e simulação de modelos de projeto e de cenários no gerenciamento de projetos de desenvolvimento de software. O estudo experimental foi realizado no início do segundo semestre de 2001 e seus resultados são descritos e analisados no Capítulo 5;
- O empacotamento do estudo experimental para análise de viabilidade das técnicas propostas. Como um estudo experimental nunca oferece a resposta final para uma questão, torna-se importante facilitar sua repetição (WOHLIN et al., 2000). O pacote do experimento inclui todos os documentos e instrumentos utilizados durante seu planejamento e execução, com o intuito de facilitar sua repetição no futuro.

Processo de Análise de Riscos

- A definição de um processo de análise de riscos, que utiliza modelos de cenário para descrever os eventos incertos que podem ocorrer ao longo de um projeto de

desenvolvimento de software e as estratégias de contenção e contingência para estes eventos. Os modelos de cenário, denominados modelos de impacto, permitem avaliar o impacto dos riscos (eventos incertos) e de seus planos de contenção e contingência antes que estes sejam efetivamente aplicados em um projeto;

- A definição de uma estrutura de dados padronizada, denominada arquétipo de risco, para o armazenamento das informações relevantes acerca dos risco que possam afetar um conjunto de projetos. Os arquétipos de risco representam um repositório de informações reutilizáveis para gerentes de projeto. Eles são descritos em diversas seções, contendo mecanismos para a identificação de riscos em projetos, para seleção de estratégias de contenção e contingência, casos conhecidos de ocorrência do risco, referências para riscos similares e induzidos pelo risco atual, além dos modelos de impacto de riscos e de seus planos de contenção e contingência.

7.2 Limitações das Técnicas Propostas

As técnicas propostas no contexto desta tese possuem algumas limitações que esperamos reduzir em trabalhos futuros.

Consideramos que a principal limitação reside na dificuldade de se construir os modelos de cenário. Esta limitação pode ser decomposta na dificuldade de se obter as informações para a especificação do cenário e na dificuldade de representar este cenário na linguagem de modelagem proposta. Alguns autores (LEBSANFT e PFAHL, 1999) apresentam técnicas e processos para a elicitação do comportamento dinâmico de projetos de desenvolvimento de software. Sendo a construção de cenários um dos pilares das técnicas propostas, consideramos relevantes estas técnicas de elicitação de conhecimento. Por outro lado, as dificuldades no que concerne a representação de cenários talvez possam ser reduzidas pela construção de melhores ferramentas, que ofereçam apoio automatizado à representação dos modelos de cenários sem que o desenvolvedor de modelos tenha que programar diretamente estes modelos. A capacidade de verificação dos modelos também deve ser avaliada. Neste sentido, o estabelecimento de diretrizes de qualidade para a construção destes modelos é um campo aberto para pesquisa.

Outra limitação das técnicas propostas se refere ao processo de análise de riscos. Este processo está definido de forma a permitir que um modelo de impacto de risco seja analisado de cada vez. Entretanto, o efeito da diversificação de riscos nos indica que o

impacto combinado de dois ou mais riscos pode ser diferente dos efeitos isolados destes riscos. Embora as técnicas de integração e simulação de modelos de projeto e cenários permitam a análise conjunta de dois ou mais modelos de cenário, as atividades do processo de análise de riscos não contemplam esta análise em conjunto. Tal ocorre devido ao grande número de combinações de modelos de impacto de riscos que deveriam ser analisados pelo gerente de projeto. Assim, precisamos de apoio automatizado à combinação de cenários, mecanismos de priorização que levem em conta a combinação dos efeitos de um conjunto de cenários e uma representação para os riscos identificados para um projeto que permita anotar os efeitos combinados de dois ou mais riscos, tal como os formulários de risco permitem para cada risco isoladamente.

Finalmente, identificamos algumas limitações no estudo experimental apresentado no Capítulo 5. Uma das limitações diz respeito ao número de participantes do estudo experimental. É reconhecido que quanto maior o número de participantes em um estudo experimental, maior será sua capacidade de conclusão. Assim, para que as conclusões do estudo possam ser estabelecidas para uma maior população com maior grau de certeza, o estudo deve ser repetido outras vezes, com um maior número de participantes e em diferentes contextos.

Outra limitação do estudo experimental refere-se a este concentrar-se apenas na viabilidade de aplicação das técnicas de integração e simulação de modelos de projeto e cenários no gerenciamento de projetos de desenvolvimento de software. Outros estudos experimentais devem ser realizados para analisar outras facetas da abordagem proposta, como as dificuldades relacionadas com a construção de modelos de cenários, com a aquisição de conhecimento para a construção de modelos de cenários, com a construção de modelos de projetos, com a capacidade de seleção de cenários pelos gerentes, entre outros componentes da abordagem proposta.

7.3 Perspectivas Futuras

Como perspectivas de trabalhos futuros, podemos citar:

- A implementação de melhores ferramentas que ofereçam suporte ao paradigma de gerenciamento de projetos baseado em cenários. O estudo experimental apresentado no Capítulo 5 indicou algumas direções em que as ferramentas atuais podem ser aprimoradas. Além disso, acreditamos que a construção de ferramentas que permitam a geração de modelos para outros simuladores da Dinâmica de Sistemas (HIGH

PERFORMANCE SYSTEMS, 1990, VENTANA SYSTEMS, 1999), a construção de ferramentas para permitir a construção gráfica de modelos de domínio, de modelos a partir de modelos de domínios e de modelos de cenários pode ampliar o universo de potenciais usuários das técnicas propostas;

- A aplicação do metamodelo da Dinâmica de Sistemas em outras áreas do conhecimento. Ao longo do processo de desenvolvimento desta tese, o autor teve a oportunidade de participar de duas edições da *System Dynamics Society International Conference*, onde uma versão preliminar do metamodelo da Dinâmica de Sistemas foi apresentada. A participação nas conferências rendeu contatos com grupos de outras universidades que realizam pesquisas em simulação e modelagem dinâmica. Devido ao tempo investido na conclusão desta tese, estes contatos não puderam ser ampliados, mas parecem promissores no sentido de cooperação em futuros projetos de pesquisa;
- A construção de um maior número de modelos de cenários, aumentando assim a gama de conhecimento gerencial representada segundo as técnicas propostas;
- O estudo de aplicações das técnicas propostas em treinamento, através da construção de um ambiente de construção de jogos de estratégia, cuja dinâmica seria descrita por modelos construídos segundo o metamodelo da Dinâmica de Sistemas. A intenção é construir modelos de gerenciamento de projetos de software progressivamente mais refinados e utilizar estes modelos para o treinamento de gerentes menos experientes e alunos de graduação e pós-graduação;
- A repetição do estudo experimental descrito no Capítulo 5, com um maior número de participantes, em diferentes contextos e considerando as evoluções propostas pelos participantes do estudo previamente executado;
- O planejamento e execução de outros estudos experimentais para avaliar outras direções da abordagem proposta, como as dificuldades de construção de modelos de projeto e de cenários, a capacidade de aquisição de conhecimento para a construção de modelos de cenários, a capacidade de seleção de cenários por gerentes novatos, entre outras;
- O planejamento e execução de estudos experimentais mais formais para a avaliação da usabilidade do processo de análise de riscos proposto no Capítulo 6, para a determinação da completude das informações dos arquétipos de riscos, para avaliar

sua influência na construção de uma cultura de livre discussão dos riscos associados a projetos, entre outros elementos envolvidos com o processo de análise de riscos.

Finalmente, gostaríamos de ressaltar que o trabalho apresentado nesta tese foi desenvolvido a partir da constatação do crescente interesse na modelagem quantitativa e simulação de projetos de desenvolvimento de software. Outra motivação para o desenvolvimento deste trabalho foi a crença de seu autor de que o conhecimento deve ser representado de forma não ambígua para que possa ser compartilhado.

A gerência de projetos de software nos oferece heurísticas desde que os projetos de software atingiram um tamanho tal que algum esforço passou a ser necessário para controlar o seu processo de desenvolvimento. Entretanto, grande parte das heurísticas são apresentadas em formatos informais, que não permitem sua avaliação sobre o contexto de um projeto de desenvolvimento de software específico.

Esta tese procurou desenvolver uma representação formal e extensível para o conhecimento gerencial. Se a ambição nos permite ir além das nossas limitadas capacidades de realização, esperamos que o conhecimento sobre o gerenciamento de projetos de software seja progressivamente registrado de maneira mais formal. Para tanto, seguimos caminhos que já haviam sido inicialmente percorridos, tentando simplificar seu percurso por aqueles que virão depois de nós. O percurso de qualquer caminho começa pelo primeiro passo. Modestamente, esperamos estar contribuindo com mais um passo nesta estrada...

Referências Bibliográficas

ABDEL-HAMID, T.K., MADNICK, S.E., 1983, "The Dynamics of Software Project Scheduling: A System Dynamics Perspective", *Communications of the ACM*, (Maio), pp. 340 – 346

ABDEL-HAMID, T.K., MADNICK, S.E., 1986, "Impact of Schedule Estimation on Software Project Behavior", *IEEE Software*, (Maio)

ABDEL-HAMID, T.K., 1988a, "The Economics of Software Quality Assurance: A Simulation Based Study", *MIS Quartely*, (Setembro)

ABDEL-HAMID, T.K., 1988b, "Undestanding the '90% Syndrome' in Software Project Management: A Simulation-Based Study", *Journal of Systems & Software*, Vol. 8, No. 4 (Setembro), pp. 319 - 330

ABDEL-HAMID, T.K., 1989, "An Integrated System Dynamics Perspective of Software Project Management: Examples of Project Estimation and Scheduling Experiments", IN: *The Proceedings of the 1989 Society for Computer Simulation Western Multiconference*, (Janeiro)

ABDEL-HAMID, T.K., 1990, "On the Utility of Historical Project Statistics for Cost and Schedule Estimation", *Journal of Systems and Software*, Vol. 13, No. 1 (Janeiro), pp. 71 – 82

ABDEL-HAMID, T., MADNICK, S.E., 1991, *Software Project Dynamics: An Integrated Approach*, Englewood Cliffs, NJ: Prentice-Hall, Inc.

ABNT, 2000, "NBR ISO 10006: Gestão da Qualidade – Diretrizes para a Qualidade no Gerenciamento de Projetos", Associação Brasileira de Normas Técnicas, Rio de Janeiro, Brasil

AIR FORCE, 1981, *Integrated Computer Aided Manufacturing Architecture*, Part II, vol. IV, Function Modeling Manual (IDEF0), AFWAL-TR-81-4023, Wright-Patterson Air Force Base, OH: Air Force Systems Command (Junho)

ALBIN, S., 1997, *Building an System Dynamics Model Part 1: Conceptualization*, IN: Relatório Técnico D-4597, MIT System Dynamics Group, Cambridge, MA

ARANGO, G., 1988, *Domain Engineering for Software Reuse*, Relatório Técnico UCI-ICS 88-27, Universidade da Califórnia, CA

ARTHUR, W.B., EBERLEIN, R.L., 1996, "Sensitivity Simulations", IN: *The Proceedings of the 1996 International System Dynamics Conference*, Cambridge, MA (Agosto)

AUGUSTINE, N.R., 1982, *Augustine's Laws*, Nova York, NY: American Institute Of Aeronautics and Astronautics

BANKS, J.J., CARSON, J.S., NELSON, B.L., 1996, *Discret-Event System Simulation*, 2 ed, Englewood Cliffs, NJ: Prentice-Hall, Inc.

BARDOSSY, A., DUCKSTEIN, L., BOGARDI, I., 1993, "Combination of Fuzzy Numbers Representing Expert Opnions", *Fuzzy Sets and Systems*, Vol. 57

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 1999, "Risk Analysis: a Key Success Factor for Complex System Development", *Proceedings of the 12th International Conference in Software & System Engineering and their Applications*, Paris, FR (Dezembro)

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2000a, "Illium: Uma Ferramenta de Simulação de Modelos Dinâmicos de Projetos de Software", IN: *Anais da Seção de Ferramentas do XIV Simpósio Brasileiro de Engenharia de Software*, João Pessoa, PB, Brasil (Outubro)

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2000b, "Applying Formal Modeling and Dynamic Simulation to Quality Evaluation, Prediction and Improvement of Software Processes", IN: *Anais do Workshop de Qualidade de Software do XIV Simpósio Brasileiro de Engenharia de Software (WQS'2000)*, João Pessoa, PB, Brasil (Outubro)

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2000c, "L'analyse de risques: un facteur de succès pour le développement de systèmes complexes", *Génie Logiciel*, ISSN 0295-6322, Setembro(24), pp. 14-20

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2000d, "Towards a Scenario Based Project Management Paradigm", IN: *Anais do V Workshop de Teses e Dissertações em Engenharia de Software, XIV Simpósio Brasileiro de Engenharia de Software*, João Pessoa, PB, Brasil (Outubro), pp. 391 – 396

BARROS, M. O., 2001, "ILLIUM - System Dynamics Simulator", *homepage* da ferramenta *ILLIUM*, disponível na URL http://www.cos.ufrj.br/~marcio/Illium.html

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2001a, "From Models to Metamodels: Organizing and Reusing Domain Knowledge in System Dynamics Model Development", IN: *Proceedings of the 19th Conference of the System Dynamics Society*, Atlanta, USA (Julho)

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2001b, *Towards a Scenario Based Project Management Paradigm*, Relatório Técnico do Departamento de Engenharia de Sistemas e Computação 543/01, COPPE/UFRJ (Fevereiro)

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2001c, "Scenario Oriented Project Management Knowledge Reuse within a Risk Analysis Process", IN: *Proceeding of the Software Engineering and Knowledge Engineering Conference, SEKE'01*, Buenos Aires, Argentina (Maio), pp. 37 – 44

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2001d, "Representing Project Management Knowledge as Scenario Models", submetido para publicação na revista *IEEE Software* (Outubro)

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2002, "Project Management Knowledge Reuse Through Scenario Models", submetido para publicação IN: *Proceeding of the 7th*

International Conference on Software Reuse, ICSR-7, Austin, Texas, Estados Unidos (Abril)

BASILI, V.R. et al., 1992, "The Software Engineering Laboratory – an Operational Software Experience Factory", IN; *The Proceedings of the International Conference on Software Engineering*, Melbourne, Austrália, (Maio), pp. 370-381

BASILI, V.R., CALDIERA, G., ROMBACH, H.D., 1994, "Goal Question Metric Paradigm", IN; Marciniak, J.J., *Encyclopedia of Software Engineering*, vol. 1, New York, NY: Wiley, pp. 528 – 532

BELLINGER, G., 1999a, "Modeling & Simutation: an Introduction", disponível na URL http://www.outsights.com/systems/welcome.htm, OutSights Co., Annadale, VA

BELLINGER, G., 1999b, "Change Management: The Columbo Theory", disponível na URL http://www.outsights.com/systems/welcome.htm, OutSights Co., Annadale, VA

BOEHM, B.W., 1988, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, Vol. 21, No. 5 (Maio), pp. 61 – 72

BOEHM, B.W., ROSS, R., 1988, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, Vol. 21 (Maio), No. 5

BOEHM, B.W., 1989, *Software Risk Management*, Los Alamitos, CA: IEEE Computer Society Press

BOEHM, B.W. et al., 1995, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", IN: *Annals of Software Engineering, Special Volume on Software Process and Product Measurement*, Amsterdam, Netherlands: J.D. Arthurs and S.M. Henry Editions, J.C. Baltzer Science Publishers

BRAGA, R., 2000, *Busca e Recuperação de Componentes em Ambientes de Reutilização de Software*, Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil

BROOKS, F.P. Jr, 1974, *The Mythical Man-Month: Essays on Software Engineerin,* Reading, MA: Addison-Wesley Publishing Co.

BROWN, N., 1996, "Industrial-Strength Management Strategies", *IEEE Software*, Vol. 13, No. 4 (Julho)

CARR, M.J. et al., 1993, *Taxonomy-Based Risk Identification*, Relatório Técnico SEI 93-TR-006, Software Engineering Institute, Pittsburgh, PA

CHARETTE, R.N., 1989, *Software Engineering Risk Analysis and Management*, London, UK: McGraw-Hill Co.

CHARETTE, R.N., 1996, "Large-Scale Project Management is Risk Management", *IEEE Software*, Vol. 13, No. 4 (Julho), pp. 110 – 117

CHRISTIE, A., 1995, *Software Process Automation: The Technology and Its Adoption*, Berlin, GR: Springer-Verlag Publishing

CONROW, E.H.; SHISHIDO, P.S., 1997, "Implementing Risk Management on Software Intensive Projects", *IEEE Software*, Vol. 14, No. 1 (Maio/Junho), pp. 83 – 89

DAVIS, A.M., 1990, *Software Requirements Analysis & Specification*, Englewood Cliffs, NJ: Prentice-Hall, Inc.

DE ALMEIDA, F.R., de MENEZES S.C., da ROCHA A.R.C., 1998, "Using Ontologies to Improve Knowledge Integration in Software Engineering Environments", In: *Proceedings of 2nd World Multiconference on Systemics, Cybernetics and Informatics, Volume I / Proceedings of 4th International Conference on Information Analysis and Synthesis, International Institute of Informatics and Systemics*: Caracas, Venezuela; pp. 296–304

DEMARCO, T., 1982, *Controlling Software Projects*. New York, NY: Yourdon Press, Inc.

DRAPPA, A., LUDEWIG, J., 1999, "Quantitative Modeling for the Interactive Simulation of Software Projects", *The Journal of Systems and Software*, Vol. 46, pp. 113-122

DRAPPA, A., LUDEWIG, J., 2000, "Simulation in Software Engineering Training", IN: *Proceedings of the International Conference on Software Engineering*, Limerick, Ireland, pp. 199-208

FAIRLEY, R. 1994, "Risk Management for Software Projects", *IEEE Software*, Vol. 13, No. 3 (Maio), pp. 57 – 67

FALBO, R., 1998, *Integração de Conhecimento em um Ambiente de Desenvolvimento de Software*, Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil

FERREIRA, A.B.H., 1993, *Minidicionário da Língua Portuguesa*, São Paulo, SP: Editora Nova Fronteira

FISCHER, G., OSTWALD, J., 2001, "Knowledge Management: Problems, Promisses, Realities, and Challenges", *IEEE Intelligent Systems* (Janeiro/Fevereiro), pp. 60 – 72

FORRESTER, J.W., 1961, *Industrial Dynamics*, Cambridge, MA: The MIT Press

FORRESTER, J.W., 1991, *System Dynamics and the Lessons of 35 Years*, Relatório Técnico D-4224-4, MIT System Dynamics Group, Cambridge, MA

FRANCA, L.P.A., STAA, A., 2001, "Geradores de Artefatos: Implementação e Instanciação de Frameworks", IN: *Anais do V Simpósio Brasileiro de Engenharia de Software*, Rio de Janeiro, Brasil (Outubro)

FRENCH, S., 1989, *Readings in Decision Analysis*, London: UK, Chapman and Hall

FREUND, J.E., PERLES, B.M., 1998, *Statistics: a First Course*, Seventh Edition, Englewood Cliffs, NJ: Prentice-Hall, Inc.

GALLAGHER, B. P.; ALBERTS, C.J.; BARBOUR, R.E., 1997, *Software Acquisition Risk Management: Key Process Area (KPA) – A Guidebook*, Relatório Técnico SEI/CMU-97-HB-002, Pittsburgh, PA

GAMMA, E. et al., 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, MA: Addison-Wesley Publishing Co.

GARCIA, R.E., MASIERO, P.C., 1999, "Dinâmica de Sistemas: Um Apoio ao Gerente de Projetos", IN: *Anais da X Conferência Internacional de Tecnologia de Software*, Curitiba, Brasil (Maio), pp. 135 – 147

GARVEY, P.R., PHAIR, D.J., WILSON, J.A., 1997, "An Information Architecture for Risk Assessment and Management", *IEEE Software*, Vol. 14, No. 3 (Maio/Junho), pp. 25 – 34

GEMMER, A., 1997, "Risk Management: Moving Beyond Process", *IEEE Computer*, (Maio)

GILB, T., 1988, *Principles of Software Engineering Management*, Reading, MA: Addison Wesley Publishing Co.

GOLDMAN, SACHS & Co, SWISS BANK CORPORATION, 1998, *The Practice of Risk Management: Implementing Processes for Managing Firmwide Market Risk*, London, UK: Euromoney Publications PLC

GOMES, A., 2001, *Avaliação de Processos de Software Baseada em Medições*, Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil

GREY, S., 1995, *Practical Risk Assessment for Project Management*, IN: Wiley Series in Software Engineering Practice, Nova York, NY: John Wiley & Sons Inc.

GROSS, N., STEPANEK, M., PORT, O., CAREY, J., 1999, "Software Hell", *Business Week* (Dezembro).

GUARINO, N., 1998, "Formal Ontology and Information Systems", IN: *Proceedings of the First International Conference in Formal Ontology in Information Systems*, IOS Press, Trento, Itália (Junho)

HALL, E.M., 1998, *Managing Risk: Methods for Software Systems Development*, IN: SEI Series in Software Engineering, Reading, MA: Addison Wesley Longman Inc.

HANSEN, G.A., 1996, "Simulating Software Development Processes", *IEEE Computer*, Vol. 29, No. 1 (Janeiro), pp. 73 – 77

HAPKE, M., JASZKIEWICZ, A., SLOWINSKI, R., 1994, "Fuzzy Project Scheduling System for Software Development", *Fuzzy Sets and Systems*, Vol. 67

HART, J.J., 1982, "The Effectiveness of Design and Code Walkthrough", IN: *The Proceedings of the 6th International Computer Software and Applications Conference (COMSPAC)*, (Novembro)

HIGH PERFORMANCE SYSTEMS, 1990, *STELLA II Users Guide.* Hanover, NH: High Performance Systems Inc.

HIGHSMITH, J., 1992, "Software Ascents", *American Programmer Magazine* (Junho)

HOUSTON, D., 1996, *System Dynamics Modeling and Simulation of Software Development: A Tutorial*, Relatório Técnico do Grupo de Modelagem com Dinâmica de Sistemas, disponível na URL http://www.eas.asu.edu/~sdm, Arizona State University, Tempe, AZ

HUMPHREY, W.S., KELLNER, M.I., 1989, *Software Process Modeling: Principles of Entity Process Models*, Relatório Técnico CMU/SEI-89-TR-2, ESD-89-TR-2, Pittsburgh, PA

HUMPHREY, W.S., SINGPURWALLA, N.D., 1991, "Predicting (Individual) Software Productivity", *IEEE Transactions on Software Engineering*, Vol. 17, No. 2 (February)

HYATT, L.E., ROSENBERG, L.H., 1996, "Software Metrics Program for Risk Assessment", IN: *The Proceedings of the 47th International Astronautical Congress & Exhibition, 29th Safety and Rescue Symposium, Risk Management and Assessment Session*, Beijing, China (Outubro)

IFPUG, 1999, *Function Point Counting Practices Manual, Release 4.1*, Westerville, OH: International Function Point Users Group

ISO/IEC, 1999, "Software Engineering – Guide for the Application of the ISO/IEC 12207 to project management", Relatório Técnico ISO/IEC DTR 16326, International Organization for Standardization, Genebra, Suiça

LAFLEUR, R., 1996, "Project Management: Getting Control and Keeping Control of Complex Projects", *American Programmer*, (Abril), pp. 23 – 28

LEBSANFT, K., PFAHL, D., 1999, "Knowledge Acquisition for Building System Dynamics Simulation Models: An Experience Report from Software Industry", IN: *Proceeding of the Software Engineering and Knowledge Engineering Conference, SEKE'99*, Kaiserslautern, Alemanha (Junho), pp. 378 - 387

LEITE, J., SANT'ANNA, M., et al., 1994, "Draco-Puc: A Technology Assembly for Domain Oriented Software Development", IN: *Proceedings of the Third International Conference on Software Reuse*, IEEE Computer Society Press, Rio de Janeiro, Brasil

LIN, C.Y., LEVARY, R.R., 1989, "Computer-Aided Softwar Development Process Design", *IEEE Transactions on Software Engineering*, Vl. 15, No. 9 (Setembro), pp. 1025 – 1037

LIN, C.Y., ABDEL-HAMID, T., SHERIF, J.S., 1997, "Software-Engineering Process Simulation Model (SEPS)", *The Journal of Systems and Software*, Vol. 38, pp. 263 – 277

LIU, L., HOROWITZ, E., 1989, "A Formal Model for Software Project Management", *IEEE Transactions on Software Engineering*, Vol. 15, No. 10 (Outubro), pp. 1280-1293

JONES, C., 1994, *Assessment and Control of Software Risks*, Englewood Cliffs, NJ: Prentice-Hall, Inc.

JONES, C., 2000, *Software Assessments, Benchmarks, and Best Practices*, IN: Addison-Wesley Information Technology Series, Reading, MA: Addison-Wesley Publishing Co.

KÄNSÄLÄ, K., 1997, "Integrating Risk Assessment with Cost Estimation", *IEEE Software*, Vol. 14, No. 3 (Maio/Junho), pp. 61 – 68

KAROLAK, D.W., 1996, *Software Engineering Risk Management*, Los Alamitos, CA: IEEE Computer Society Press

KITCHENHAN, B., LINKMAN, S., 1997, "Estimates, Uncertainty, and Risk", *IEEE Software*, Vol. 14, No. 3 (Maio/Junho), pp. 69 - 74

KITCHENHAM, B.A., TRAVASSOS, G.H., von MAYRHAUSER, A. et al., 1999, "Towards an Ontology of Software Maintenance", *Journal Of Software Maintenance: Research And Practice*, No. 11, pp. 365–389

KLEIN, G., 1998, *Sources of Power*, Cambridge, MA: MIT Press

KONTIO, J., BASILI, V.R., 1996, "Risk Knowledge Capture in the Riskit Method", IN: *The Proceedings of the 21st Software Engineering Workshop*, Greenbelt, MD

KONTIO, J., 1997, *The RiskIt Method for Software Risk Management, version 1.00*, Relatório Técnico CS-TR-3782, UMIACS-TR-97-38, Instituto para Estudos Avançados em Computação, Departamento de Ciência da Computação, Universidade de Maryland, College Park, MD

MADACHY, R.J., 1997, "Heuristic Risk Assessment Using Cost Factors", *IEEE Software*, Vol. 14, No. 3 (Maio/Junho), pp. 51 – 60

MADACHY, R.J., BOEHM, B.W., 1999, *Software Process Dynamics*, Early Draft Version 4/99, disponível na URL http://www-rcf.usc.edu/~madachy/spd, Los Alamitos, CA: IEEE Computer Society Press

MAIER, F.H., STROHHECKER, J., 1996, "Do Management Flight Simulators Really Enhance Decision Effectiveness ?", IN: *The Proceedings of the 1996 International System Dynamics Conference*, Cambridge, MA (Julho)

MARTIN, L.A., 1997a, *The First Step*, Relatório Técnico D-4694, MIT System Dynamics Group, Cambridge, MA

MARTIN, L.A., 1997b, *An Introduction to Feedback*, Relatório Técnico D-4691, MIT System Dynamics Group, Cambridge, MA

MARTÍNEZ-GARCÍA, A.I., WARBOYS, B.C., 1998, "From RADs to DESs: a Mapping from Process Models to Discrete Event Simulation", IN: *Proceedings of the Software Process Simulation Modeling (ProSim) Workshop'98*, Portland, OH (Junho)

MCCABE, T., 1976, "A Complexity Measure", *IEEE Transactions on Software Engineering*, Vol. SE-2, No 6, (Dezembro)

MERRIL, D., 1995, *Training Software Development Project Managers with a Software Project Simulator*, Proposta de Tese de Mestrado, Arizona State University, Tempe, AZ, disponível na URL http://www.eas.asu.edu/~sdm

MOORE, J.E., 1998, "Risk Radar User's Guide: Version 1.1", *Software Program Managers Network - SPMN*, disponível na URL http://www.spmn.com/rsktrkr.html

MOORE, J.M., BAILIN, S.C., 1991, "Domain Analysis: Framework for Reuse", IN: *Domain Analysis and Software System Modeling*, 1st ed, Los Alamitos, CA: IEEE Computer Society Press, pp. 179 – 202

MOYNIHAM, T., 1997, "How Experienced Project Managers Assess Risk", *IEEE Software*, Vol. 14, No. 3 (Maio/Junho), pp. 35 – 42

MURTA, L.G.P., 1999, *Framedoc: um Framework para a Documentação de Componentes Reutilizáveis*, Projeto Final do Curso de Matemática Aplicada a Informática, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil

MYRTVEIT, M., 2000, "Object Oriented Extensions To System Dynamics", IN: *The Proceedings of the 18th International Conference of the System Dynamics Society*, Bergen, Noruega (Agosto), pp. 155 – 156

NIEGHBORS, J., 1981, *Software Construction Using Components*, Tese de Doutorado, Universidade da Califórnia, Irvine, Califórnia, Estados Unidos

OLIVEIRA, K.M., 1999, *Modelo para Construção de Ambientes de Desenvolvimento Orientados a Domínio*, Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil

PAULIK, M.C. et al., 1993, *The Capability Maturity Model for Software Version 1.1*, Relatório Técnico CMU/SEI-93-TR-024, Pittsburgh, PA

PFHAL, D., LEBSANFT, K., 1999, "Integration of System Dynamics Modelling with Descriptive Process Modelling and Goal-Oriented Measurement", *The Journal of Systems and Software,* Vol. 46, pp. 135-150

POWELL, A., CLARK, G.D., 1999, "Learning from Simulation? Mind your Language", Position Paper in: *Proceeding of the Software Engineering and Knowledge Engineering Conference, SEKE'99,* Kaiserslautern, Alemanha (Junho), pp. 251-252

PMI, 1996, "A Guide to the Project Management Body of Knowledge – PMBOK™ Guide", Project Management Institute (PMI) Standards Comitee, Pennsylvania, USA

PREECE, A., FLETT, A., SLEEMAN, D., CURRY, D., MEANY, N., PERRY, P., 2001, "Better Knowledge Management through Knowledge Engineering", *IEEE Intelligent Systems*, (Janeiro/Fevereiro)

PRESSMAN, R., 2000, *Software Engineering: A Practitioner's Approach*, 5ª. Edição, IN: McGraw-Hill Higher Education International Editions, London, UK: McGraw-Hill Co.

RAFFO, D., 1993, "Evaluating the Impact of Process Improvements Quantitatively using Process Modeling", *Proceedings of the IBM Centre for Advanced Studies and the National Research Council of Canada Conference, CASCON'93*, Toronto, Canada (Novembro), pp. 290 – 313

RAFFO, D.M., VANDEVILLE, J.V., MARTIN R.H., 1999, "Software Process Simulation to Achieve Higher CMM Levels", *The Journal of Systems and Software*, Vol. 46, pp. 163-172

REEL, J.S., 1999, "Critical Success Factors in Software Projects", *IEEE Software*, Vol. 16, No. 3 (Maio/Junho), pp. 18 – 23

RICHARDSON, G.P., 1982, "Problems with Causal-Loop Diagrams", *System Dynamics Review*, Vol. 2

RICHMOND, B., 1993, "Systems Thinking: Critical Thinking Skills for the 1990s and Beyond", *System Dynamics Review*, Vol. 9, No. 2

ROBERTS, N. et al., 1983, *Introduction to Computer Simulation,* Reading, MA: Addison-Wesley Publishing Co.

RODRIGUES, A. G., WILLIANS, T., 1996, "System Dynamics in Software Project Management: Towards the Development of a Formal Integrated Framework", IN: *The Proceedings of the 1996 International System Dynamics Conference*, Cambridge, MA (Julho)

ROSS, S.M., 1990, *A Course in Simulation*, New York, NY: Macmillan Publishing Co.

ROYCE, W., 1998, *Software Project Management: A Unified Framework*, Reading, MA: Addison-Wesley Publishing Co.

RUS, I., COLLOFELO, J., LAKEY, P., 1998, "Software Process Simulation for Reliability Strategy Assessment", IN: *The Proceedings of the International Workshop on Software Process Simulation Modeling - ProSim'98*, Portland, OH

SCHACCI, W., 1999, "Experience with Software Process Simulation and Modeling", *The Journal of Systems and Software*, Vol. 46, pp. 183-192

SENGE, P.M., 1990, *The Fifth Discipline*, New York, NY: Doubleday

SHULL, F., CARVER, J., TRAVASSOS, G.H., 2001, "An Empirical Methodology for Introducing Software Processes", IN: *Proceeding of the Joint 8th European Software Engineering Symposium and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Vienna, Austria (Setembro)

STANDISH GROUP, T., 1994, *The Chaos Report*, The Standish Group Report, disponível na URL http://www.pm2go.com/sample_research/chaos_1994_1.asp

STATZ, J., 1994, "Training Effective Project Managers", *American Programmer*, Vol. 7, No. 6 (Junho), pp. 44 – 48

STERMAN, J.D., 1988, *A Skeptic's Guide to Computer Models*, Relatório Técnico D-4101-1, MIT System Dynamics Group, Cambridge, MA

STERMAN, J.D., 1992, *System Dynamics Modeling for Project Management*, Relatório Técnico, MIT System Dynamics Group, Cambridge, MA

TRIVEDI, K. S., 1982, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, Englewood Cliffs, NJ: Prentice-Hall, Inc.

TVEDT, J.D., 1996, *An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time*, Tese de Doutorado, Arizona State University, Tempe, AZ, Estados Unidos

VENTANA SYSTEMS, 1999, *Vensim PLE Plus User's Guide Version 4*, Personal Learning Edition, Ventana Systems, disponível na URL http://www.vensim.com/ffiles/venple.pdf

VOSE, D., 1996, *Quantitative Risk Analysis: A Guide to Monte Carlo Simulation Modelling*, Nova York, NY: John Wiley & Sons Inc.

WARBOYS, B.C., 1998, *Business Information Systems: a Process Approach*, Relatório Técnico do Grupo de Processos de Informação, University of Manchester, UK

WEICK, K.E., 1979, *The Social Psychology of Organization*, 2 ed., Reading, MA: Addison-Wesley Publishing Co.

WHELAN, J.G., 1996, *Beginner Modeling Exercises Section 2: Mental Simulation of Simple Positive Feedback*, Relatório Técnico D-4487, MIT System Dynamics Group, Cambridge, MA

WIEST, J., LEVY, F., 1977, *A Management Guide to PERT/CPM*, Englewood Cliffs, NJ: Prentice-Hall, Inc.

WILLIANS, R.C., WALKER, J.A., DOROFEE, A.J. 1997, "Putting Risk Management into Practice", *IEEE Software*, Vol. 14, No. 3 (Maio/Junho), pp. 75 – 82

WOHLIN, C., RUNESON, P., HÖST, M. et al., 2000, *Experimentation in Software Engineering: an Introduction*, Norwell, MA: Kluver Academic Publishers

ZADEH, L.A., 1988, "Fuzzy Logic", *IEEE Computer*, Vol. 21, No. 4 (Abril), pp. 83 - 91

Apêndice A - A Dinâmica de Sistemas

A Dinâmica de Sistemas é uma disciplina de modelagem criada na década de 1960 por FORRESTER (1961), no Massachusetts Institute of Technology. As técnicas de dinâmica de sistemas podem ser aplicadas para entender e influenciar a forma com que os elementos de um sistema variam ao longo do tempo. As características da dinâmica de sistemas e a aplicação desta técnica de modelagem na engenharia de software foram exploradas no Capítulo 2. Neste apêndice, apresentamos os modelos construídos a partir das técnicas da Dinâmica de Sistemas e sua interpretação formal.

A.1 Representação de Modelos de Dinâmica de Sistemas

Os modelos desenvolvidos com as técnicas da Dinâmica de Sistemas podem ser representados através de diagramas de causa e efeito ou através de diagramas de repositórios e fluxos. Nesta seção, apresentamos as características de cada representação e as condições em que elas são aplicáveis.

A.1.1 Diagramas de Causa e Efeito

Os diagramas de causa e efeito representam o mecanismo mais simples para representação de modelos de dinâmica de sistemas. Posteriormente, estes diagramas devem ser refinados para diagramas de repositórios e fluxos, que são mais precisos na representação do modelo.

Os diagramas de causa e efeito apresentam os diversos componentes de um sistema e os efeitos que a acumulação ou redução no volume de um componente provoca sobre os demais componentes do sistema. Os componentes do sistema são normalmente representados como retângulos, conectados entre si por setas. As setas indicam o efeito que um componente provoca sobre o outro, assinalando se este efeito aumenta ou reduz a presença do componente destino.

O diagrama apresentado na Figura A.1 apresenta alguns aspectos do processo de desenvolvimento de software. A medida que a data de conclusão do projeto se aproxima, cresce a pressão de cronograma, que resulta em um aumento do número de horas de trabalho diário da equipe de desenvolvimento. A medida que os desenvolvedores trabalham mais horas por dia, sua produtividade aumenta, reduzindo o número de tarefas pendentes para a conclusão do projeto.

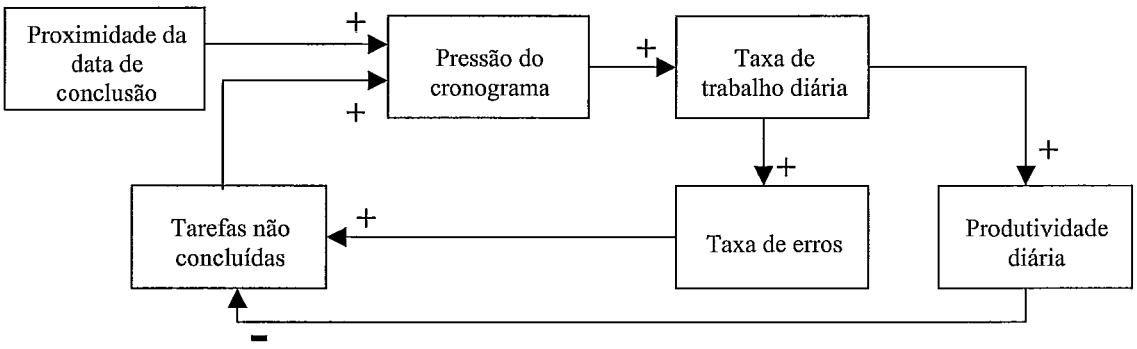

Figura A.1 – Um diagrama de causas e efeitos

Entretanto, conforme observado na literatura, a pressão de cronograma não faz com que a equipe trabalhe melhor: a equipe apenas trabalha mais (DEMARCO, 1982). Isto se reflete em um aumento na taxa de erros cometidos durante o desenvolvimento. A correção destes erros demanda esforço da equipe, aumentando o número de tarefas não concluídas. Estas, por sua vez, forçam o aumento da pressão do cronograma.

O exemplo da Figura A.1 apresenta dois ciclos de *feedback*. O primeiro, determinado pelos componentes "pressão de cronograma – taxa de trabalho diária – taxa de erros – tarefas não concluídas", é um exemplo de ciclo de *feedback* de reforço, também chamado de *feedback* positivo. Ciclos de feedback positivos se caracterizam por um aumento aplicado sobre uma variável provocar um aumento em outra variável do modelo (MARTIN, 1997b). Ciclos de *feedback* positivos reforçam o crescimento das variáveis de um modelo. A Figura A.2 ressalta o *feedback* positivo da Figura A.1.

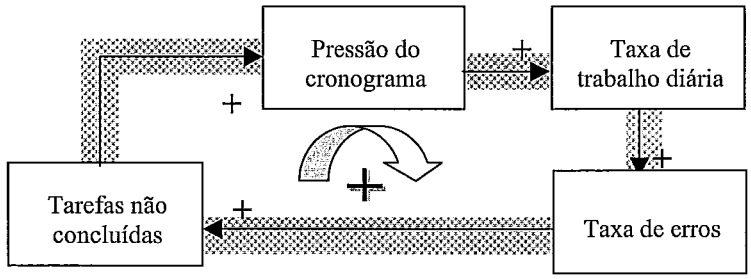

Figura A.2 - Feedback positivo no desenvolvimento de software

O segundo ciclo de *feedback*, determinado pelos componentes "pressão de cronograma – taxa de trabalho diária – produtividade diária – tarefas não concluídas", é um exemplo de ciclo de *feedback* negativo. Ciclos de *feedback* negativos se

caracterizam por um aumento aplicado sobre uma variável provocar uma redução em outra variável (MARTIN, 1997b). Ciclos de *feedback* negativos reforçam a estabilidade das variáveis de um modelo. A Figura A.3 ressalta o *feedback* negativo da Figura A.1.

Devido a sua simplicidade, diagramas de causa e efeito são normalmente utilizados para explicar o conhecimento extraído do modelo (ALBIN, 1997). Entretanto, esta mesma simplicidade impede que os diagramas de causa e efeito sejam utilizados para análise de regras e simulações. Assim, os diagramas de causa e efeito constituem uma ferramenta para facilitar o entendimento da dinâmica de um sistema, mas não para descrever sua estrutura (RICHARDSON, 1982) (RICHMOND, 1993).

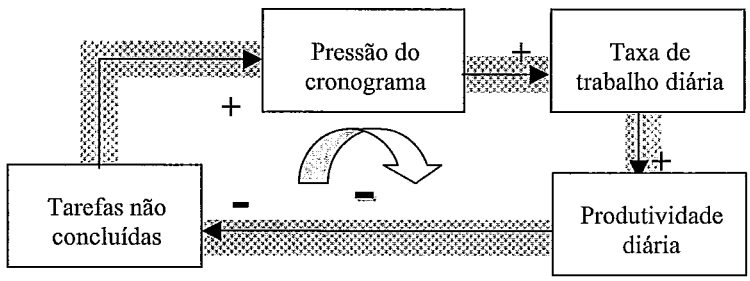

Figura A.3 - Feedback negativo no desenvolvimento de software

HOUSTON (1996) observa que a utilização de diagramas de causa e efeito nem sempre se faz necessária. Entretanto, em sistemas caracterizados por um alto grau de subjetividade, estes diagramas são úteis para a formulação de um entendimento comum sobre os componentes do sistema e as relações entre eles, antes de avançar para o formalismo dos diagramas de repositórios e fluxos.

A.1.2. Diagramas de Repositórios e Fluxos

Os diagramas de repositórios e fluxos apresentam um nível de detalhe maior que os diagramas de causa e efeito, forçando o responsável pela modelagem a refinar sua definição da estrutura do sistema (ALBIN, 1997). Estes diagramas são compostos por quatro blocos básicos – repositórios, fluxos, processos e conectores – e dois elementos complementares – os produtores e os consumidores infinitos. A Figura A.4 apresenta a representação diagramática destes elementos.

Um repositório representa um elemento que possa ser gradualmente acumulado ou consumido ao longo do tempo. Na modelagem de um projeto de desenvolvimento de software, o número de tarefas desenvolvidas é descrito como um repositório. Estas tarefas são produzidas gradualmente, durante as fases iniciais de desenvolvimento. O número de desenvolvedores inexperientes é outro repositório. Este repositório é

alimentado pelo processo de recrutamento de pessoal para a equipe do projeto. Por outro lado, o processo de treinamento da equipe consome os elementos representados neste repositório, transformando os desenvolvedores inexperientes em desenvolvedores experientes. Repositórios são representados por retângulos e são caracterizados por seu nível, que indica o número de elementos no repositório.

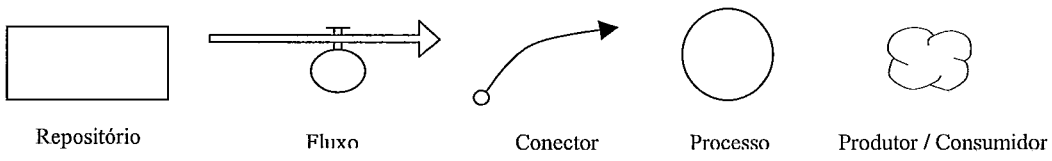

Figura A.4 – Blocos básicos dos modelos de dinâmica de sistemas

Um fluxo representa uma taxa de variação de um repositório em relação a um instante de tempo. Um fluxo indica o número de elementos que ingressam ou são retirados de um repositório a cada instante de tempo. Fluxos são representados por setas conectadas a repositórios. Fluxos são caracterizados pela equação que calcula a variação provocada pelo fluxo no repositório. Um fluxo pode estar ligado a um ou dois repositórios. Por exemplo, a taxa de desenvolvimento de software é um fluxo que indica o número de tarefas desenvolvidas a cada instante de tempo, alimentando o repositório que contém o número de tarefas concluídas. A taxa de recrutamento de profissionais é outro exemplo de fluxo ligado a um único repositório: o repositório que indica o número de desenvolvedores inexperientes.

Nos fluxos conectados a um único repositório, a ponta livre da seta é conectada a um produtor infinito ou a um consumidor infinito. O objetivo de um produtor infinito é fornecer quantas unidades sejam necessárias para um fluxo que alimente um repositório. Assim, o fluxo pode ser interpretado como uma válvula que permite a transferência de elementos do produtor infinito para o repositório. O objetivo de um consumidor infinito é absorver quantas unidades forem retiradas de um repositório por um fluxo. Novamente, o fluxo pode ser interpretado como uma válvula, desta vez permitindo a transferência de elementos do repositório para o consumidor infinito. Produtores e consumidores infinitos determinam os limites do sistema, sendo representados por nuvens.

Os fluxos conectados a dois repositórios operam como uma válvula que permite a transferência de elementos do primeiro repositório para o segundo. Assim, o fluxo consome elementos do primeiro repositório e alimenta o segundo repositório com estes

elementos. A taxa de treinamento da equipe de desenvolvimento é um exemplo de fluxo de transferência entre dois repositórios. A medida que os desenvolvedores inexperientes são treinados, eles adquirem experiência, passando a ser desenvolvedores experientes. Na notação de dinâmica de sistemas, a medida que os desenvolvedores são treinados, eles são transferidos do repositório de desenvolvedores inexperientes para o repositório de desenvolvedores experientes.

Um processo é utilizado para calcular informações a partir de um conjunto de parâmetros. Um parâmetro pode ser a taxa de variação indicada por um fluxo, o nível de um repositório ou o resultado de outro processo. A informação resultante de um processo pode ser utilizada em outros processos ou para determinar a intensidade de um ou mais fluxos. A data esperada para a conclusão de um projeto de desenvolvimento de software pode ser calculada por um processo, dadas as variáveis que caracterizam o estado atual do projeto e a data atual. Processos são representados por círculos e são caracterizados pela equação que calcula seu resultado a partir de seus parâmetros.

Um conector é uma via de transmissão de informações no modelo. Assim como os fluxos transferem elementos entre repositórios ou entre repositórios e os limites do sistema, um conector transfere informações entre dois processos, entre um repositório e um processo, entre um processo e um fluxo ou entre um fluxo e um processo. Um conector é representado por uma seta, indicando a origem e o destino da informação transmitida.

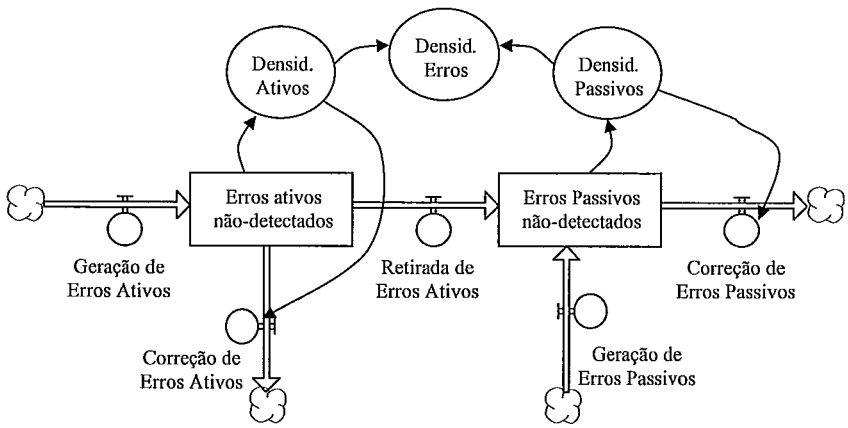

Figura A.5 – Exemplo de diagrama de repositórios e fluxos

A Figura A.5 apresenta um trecho do diagrama de repositórios e fluxos de um modelo do processo de desenvolvimento de software (ABDEL-HAMID, 1991). Este diagrama representa a geração e correção de erros ao longo do processo de desenvolvimento.

No diagrama, os erros são representados em dois repositórios distintos: erros ativos não-detectados e erros passivos não-detectados. Denominamos por erros ativos aqueles erros que, tendo sido gerados em um produto específico do processo de desenvolvimento, se propagam para novos produtos, construídos a partir do produto original do erro. Por exemplo, um erro de projeto se propaga, gerando novos erros na implementação e nos manuais do sistema. Erros passivos não se propagam além do produto em que foram gerados.

Os erros ativos e passivos são gerados pelas atividades de desenvolvimento de novos produtos. Esta geração se reflete nos fluxos de geração de erros ativos e passivos. Erros passivos também podem ser gerados a partir dos erros ativos: quando um produto que contém um erro ativo não será mais utilizado como base para construção de novos produtos no processo de desenvolvimento, o erro ativo contido no produto se transforma em um erro passivo. Esta transição é realizada pelo fluxo de retirada de erros ativos.

A medida que a atividade de teste encontra e corrige os erros, estes são retirados de seus repositórios. Esta retirada é realizada pelos fluxos de correção de erros ativos e correção de erros passivos. A densidade de erros, definida como o número de erros por tarefa ou produto, influencia a taxa de correção de erros. Esta relação é demonstrada através dos conectores que ligam os processos de cálculo da densidade de erros aos fluxos de correção de erros.

A.2 Formulação Matemática dos Modelos

Nesta seção, apresentamos a interpretação matemática dos modelos da Dinâmica de Sistemas. Somente os diagramas de repositórios e fluxos podem ser interpretados matematicamente, visto que os diagramas de causa e efeito não possuem informação suficiente para a definição das equações.

Os fluxos e processos de um diagrama de repositórios e fluxos são caracterizados por equações. Estas equações determinam a variação do nível de cada repositório do diagrama. Sendo assim, as equações fundamentais do diagrama estão nos fluxos. Somente quando estas equações são demasiadamente complexas, ou quando resultados parciais devem ser analisados durante a simulação, os processos são necessários. Se considerarmos o nível de um repositório como uma função do tempo, este nível pode ser definido por:

$$\text{Nível}(R, t) = \text{Nível}(R, t - \Delta t) + \sum F_d(t) - \sum F_o(t)$$

onde, R é o repositório
t é um instante de tempo
F_o = { F | origem (F) = R }
F_d = { F | destino (F) = R }

Na equação anterior, o nível de repositório em um determinado instante de tempo é igual ao nível do repositório no instante anterior, somado a variação de seus fluxos de entrada e subtraindo-se a variação de seus fluxos de saída. Reorganizando-se a equação, temos:

$$\text{Nível}(R, t) - \text{Nível}(R, t - \Delta t) = \sum F_d(t) - \sum F_o(t)$$

$$\Delta \text{ Nível}(R, t) = \sum F_d(t) - \sum F_o(t)$$

Considerando-se o intervalo de tempo infinitesimal, temos:

$$\frac{\partial nível(R)}{\partial t} = \sum Fd(t) - \sum Fo(t)$$

Assim, os fluxos descrevem derivadas parciais dos repositórios. Como um modelo pode ser composto por diversos repositórios, cada qual ligado a diversos fluxos, os modelos de dinâmica de sistemas são equivalentes a um conjunto de equações diferenciais parciais. Modelos complexos não podem ser resolvidos apenas por inspeção: somente a simulação pode revelar o comportamento implícito em sua estrutura, construída com o conhecimento das pessoas responsáveis por tomadas de decisão localizadas e pelo modo com que estas decisões se conectam (FORRESTER, 1991).

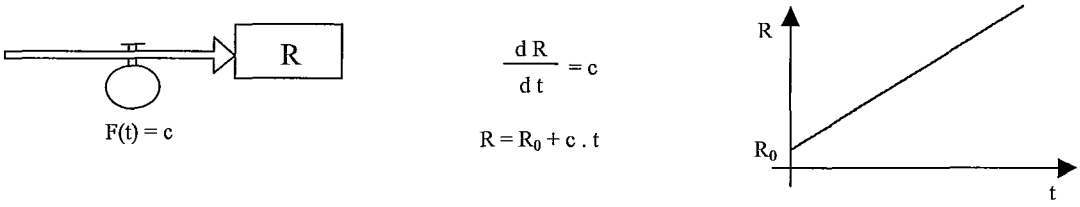

Figura A.6 – Evolução no tempo do nível de um repositório com fluxo constante

A partir da última equação, é possível definir o comportamento de diversas construções simples de dinâmica de sistemas. A Figura A.6 apresenta um repositório alimentado por um fluxo constante. A figura apresenta as equações que descrevem o sistema e um gráfico do comportamento do repositório. É importante observar que, como a taxa de crescimento do volume do repositório é constante ao longo do tempo, o volume é uma função linear desta taxa.

A Figura A.7 apresenta um ciclo de *feedback* positivo. É importante observar que a variação do fluxo no tempo depende do nível do repositório. Ciclos de *feedback* positivo apresentam comportamento de crescimento exponencial, o que indica que o aumento em uma variável provoca aumentos em outras variáveis do ciclo.

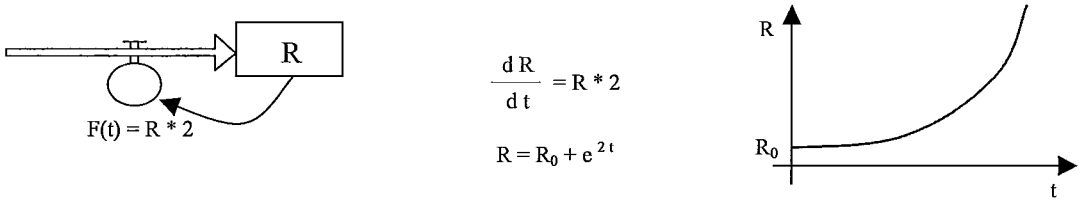

Figura A.7 – Evolução no tempo do nível de um ciclo de *feedback* positivo

A Figura A.8 apresenta um ciclo de *feedback* negativo. Novamente, a variação do fluxo no tempo depende do nível do repositório. Ciclos de *feedback* negativo apresentam comportamento de decaimento exponencial, o que indica que o aumento em uma variável provoca reduções em outras variáveis do ciclo.

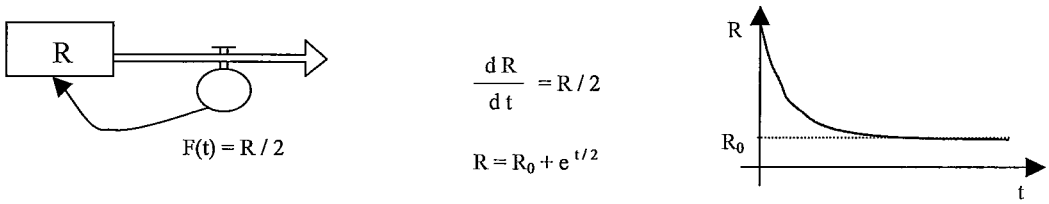

Figura A.8 – Evolução no tempo do nível de um ciclo de *feedback* negativo

A combinação de um grande número de construções simples torna um modelo de dinâmica de sistemas complexo. Em diversas ocasiões, a discretização das funções que determinam os fluxos e processos impossibilita a resolução analítica das derivadas. Assim, simulações são necessárias para a compreensão do modelo.

Apêndice B – Modelos Utilizados no Experimento

Abaixo apresentamos o modelo de domínio, o modelo de projeto e diversos modelos de cenários construídos com as técnicas propostas no Capítulo 4 e utilizados no estudo experimental do Capítulo 5. A dinâmica dos modelos é descrita no Apêndice C.

```
#
# Metamodel that represents a software development project
#
MODEL ProjectModel
{
   #
   # Class that represents a developer participating in a software project
   #
   CLASS Developer
   {
      PROPERTY ExpAnalysis     0;              # range [0, 1]
      PROPERTY ExpDesign       0;              # range [0, 1]
      PROPERTY ExpCoding       0;              # range [0, 1]
      PROPERTY ExpTesting      0;              # range [0, 1]
      PROPERTY ExpInspection   0;              # range [0, 1]
      PROPERTY HourlyCost      0;              # in $

      # Activity to which the developer is associated
      PROC AssociatedTask Groupmin(Bound([Activity], Team), DeveloperNeed);

      # Developer's Productivity for each activity
      PROC Cost HourlyCost * 8;

      # Developer's Productivity for each activity
      PROC  Productivity 1;
      PROC  ProdAnalysis Productivity;
      PROC  ProdDesign Productivity;
      PROC  ProdCoding Productivity;
      PROC  ProdTesting Productivity;
      PROC  ProdInspection Productivity;

      # Developer's error generation rate for each activity
      PROC  ErrorGenerationRate 1;
      PROC  ErrorRateAnalysis ErrorGenerationRate;
      PROC  ErrorRateDesign ErrorGenerationRate;
      PROC  ErrorRateCoding ErrorGenerationRate;
   };

   #
   # Class that represents an activity in a software project
   #
   CLASS Activity
   {
      PROPERTY AnalysisTask           0;          # 0 or 1
      PROPERTY ArchitecTask           0;          # 0 or 1
      PROPERTY DesignTask             0;          # 0 or 1
      PROPERTY CodingTask             0;          # 0 or 1
      PROPERTY TestingTask            0;          # 0 or 1
      PROPERTY InspectionTask         0;          # 0 or 1
      PROPERTY MinimumDuration        0;          # in days
      PROPERTY ExpectedDuration       0;          # in days
      PROPERTY MaximumDuration        0;          # in days
      PROPERTY Order                  0;          # 0 or +

      # Set the activity execution time
      PROC ExpectedTime ExpectedDuration;
      PROC MaximumTime MaximumDuration;
      PROC MinimumTime MinimumDuration;
      STOCK ExecutionTime ExpectedTime;
```

```
      # Determine if precedent activities are concluded
      PROC PrecConcluded AND (GroupMax (Precedences, PrecConcluded) >= 0, GroupMax
            (Precedences, RemainingTime) < 0.001);

      # Determine if the activity is concluded
      PROC Concluded RemainingTime < 0.001;

      # Determine if the activity is ready to run
      PROC Ready AND (PrecConcluded, NOT(Concluded));

      # Determine if there are resources available for the activity
      PROC DeveloperNeed IF (Ready, Order, 1000);
      PROC Executing AND (Ready, Team.AssociatedTask = Order);

      # Determine developer producticity
      PROC Productivity ((AnalysisTask * Team.ProdAnalysis)  + (ArchitecTask *
            Team.ProdDesign) + (DesignTask * Team.ProdDesign) + (CodingTask *
            Team.ProdCoding) + (TestingTask * Team.ProdTesting) + (InspectionTask *
            Team.ProdInspection)) * DT;

      # Determine activity executed time
      STOCK ExecutedTime 0;
      RATE (ExecutedTime) RTExecTime IF (Executing, if (OR(InspectionTask>0,
            TestingTask>0), MIN(RemainingTime, DT), MIN(RemainingTime, Productivity)),
            0) / DT;
      PROC RemainingTime ExecutionTime - ExecutedTime;

      # Calculates conclusion time for an activity
      STOCK ConclusionTime 0;
      RATE (ConclusionTime) RTConclusionTime if(AND(ConclusionTime < 0.01,
            RemainingTime-RTExecTime*DT < 0.01), TIME/DT+1, 0);

      # Accumulates activity cost
      STOCK Cost 0;
      RATE (Cost) RTCost if(Executing, Team.Cost, 0);

      # Errors latent in the activity
      STOCK Errors 0;
      RATE (Errors) RTErrors 0;
   };

   #
   # Class that represents a software project
   #
   CLASS Project
   {
      PROC Concluded GroupMin ([Activity], Concluded);

      STOCK ProjectTime 0;
      RATE (ProjectTime) RTProjectTime IF (Concluded, 0, 1);

      PROC ProjectCost GROUPSUM([Activity], Cost);
   };


   #
   # Relationships among classes
   #
   MULTIRELATION Precedences Activity, Activity (Successors);

   RELATION Team Activity, Developer;
};


#
# Scenario that reflects the variation in productivity due to experience
#
SCENARIO ProductivityDueExpertise ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      AFFECT ProdAnalysis (0.667 + ExpAnalysis * 0.666) * ProdAnalysis;
      AFFECT ProdDesign (0.667 + ExpDesign * 0.666) * ProdDesign;
      AFFECT ProdCoding (0.667 + ExpCoding * 0.666) * ProdCoding;
      AFFECT ProdTesting (0.667 + ExpTesting * 0.666) * ProdTesting;
      AFFECT ProdInspection (0.667 + ExpInspection * 0.666) * ProdInspection;
   };
```

```
};


#
# Scenario that reflects the variation in error generation rates due to experience
#
SCENARIO ErrorGenerationDueExpertise ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      AFFECT ErrorRateAnalysis (0.667 + (1 - ExpAnalysis) * 0.666) * ErrorRateAnalysis;
      AFFECT ErrorRateDesign (0.667 + (1 - ExpDesign) * 0.666) * ErrorRateDesign;
      AFFECT ErrorRateCoding (0.667 + (1 - ExpCoding) * 0.666) * ErrorRateCoding;
   };
};


#
# Scenario that reflects the gain in productivity due to learning
#
SCENARIO ProductivityDueLearning ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      STOCK DaysInProject 0;
      RATE  (DaysInProject) DaysInProjectCounter 1;

      PROC  DIPFactor MIN (DaysInProject / 20, 1.0);
      PROC  DIPModifier LOOKUP (LearningTable, DIPFactor, 0, 1);
      TABLE LearningTable 1.0, 1.0125, 1.0325, 1.055, 1.09, 1.15, 1.2, 1.22, 1.245,
            1.25, 1.25;

      AFFECT Productivity Productivity * DIPModifier;
   };
};


#
# Scenario that reflects the variation in productivity/error generation due to
            overworking
#
SCENARIO Overworking ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      PROPERTY WorkHours 8;           # 8 to 12 hours

      STOCK DailyWorkHours WorkHours;
      PROC  WHModifier 1 + (DailyWorkHours - 8) / (12 - 8);

      PROC  SEModifier LOOKUP (SchErrorsTable, WHModifier-1, 0, 1);
      TABLE SchErrorsTable 0.9, 0.94, 1, 1.05, 1.14, 1.24, 1.36, 1.5;

      AFFECT Cost Cost * DailyWorkHours / 8;
      AFFECT Productivity Productivity * WHModifier;
      AFFECT ErrorGenerationRate ErrorGenerationRate * SEModifier;
   };
};


#
# Scenario that reflects the willingness to overwork due to exhaustion
#
SCENARIO Exhaustion ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      STOCK Exhaustion 0;
      PROC  MaxExhaustion 50;
      RATE  (Exhaustion) ExhaustionRate if(Resting = 1, -MaxExhaustion / 20.0,
            EXModifier);

      PROC  EXModifier LOOKUP (ExaustionTable, DedicationFactor, 0, 1.5);
      PROC  DedicationFactor 1 - (1 - Dedication) / 0.4;
      PROC  Dedication 0.6 + (WHModifier - 1) * (1.2 - 0.6);
      TABLE ExaustionTable 0.0, 0.0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.15, 1.3,
            1.6, 1.9, 2.2, 2.5;
```

```
      STOCK Resting 0;
      RATE  (Resting) RestingRate1 IF (InitResting, 1 / DT, 0);
      RATE  (Resting) RestingRate2 IF (QuitResting, -1 / DT, 0);
      RATE  (DailyWorkHours) DWHRate IF (Resting = 1, (8 - DailyWorkHours) / DT, 0);

      PROC  InitResting AND(Resting = 0, Exhaustion > MaxExhaustion);
      PROC  QuitResting AND(Resting = 1, Exhaustion < 0.1);
   };

   CONSTRAINT TheDeveloper, Overworking.TheDeveloper;
};


#
# Scenario that allows measuring an activity in function points
#
SCENARIO FunctionPointMeasure ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROPERTY FunctionPoints 0;             # in FP
   };
};


#
# Scenario that estimates activity duration using function points
#
SCENARIO EstimationFunctionPoints ProjectModel
{
   CONNECTION TheActivity Activity
   {
      # Developer's productivity
      PROC AvgFPMonth 27.80;
      PROC MinFPMonth 15.41;
      PROC MaxFPMonth 40.19;

      # Evaluate the contribution of the activity type to the project
      PROC  ContrAnalysis AnalysisTask * 4.114;
      PROC  ContrArchitec ArchitecTask * 3.6958;
      PROC  ContrDesign DesignTask * 11.7052;
      PROC  ContrCoding CodingTask * 20.5459;
      PROC  ContrTesting TestingTask * 59.9416;
      PROC  Contribution ContrAnalysis + ContrArchitec + ContrDesign + ContrCoding +
            ContrTesting;

      # Evaluate the expected activity execution time
      AFFECT ExpectedTime (ExpectedTime * 0.0) + FunctionPoints * (30.0 / AvgFPMonth) *
            (Contribution / 100);
      AFFECT MaximumTime (MaximumTime * 0.0) + FunctionPoints * (30.0 / MinFPMonth) *
            (Contribution / 100);
      AFFECT MinimumTime (MinimumTime * 0.0) + FunctionPoints * (30.0 / MaxFPMonth) *
            (Contribution / 100);
   };

   CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};


#
# Scenario that represents error generation in an activity
#
SCENARIO ErrorGeneration ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC NewErrors (AnalysisTask + ArchitecTask + DesignTask + CodingTask) *
            ErrorGeneration;
      PROC ErrorGen ((AnalysisTask * Team.ErrorRateAnalysis)  + (ArchitecTask *
            Team.ErrorRateDesign) + (DesignTask * Team.ErrorRateDesign) + (CodingTask *
            Team.ErrorRateCoding));
      PROC ErrorGeneration if (ExecutionTime > 0, ErrorsFP * FunctionPoints * ErrorGen *
            RTExecTime / ExecutionTime, 0);
      PROC ErrorsFP (AnalysisTask * 0.6)  + (ArchitecTask * 0.18) + (DesignTask * 0.57)
            + (CodingTask * 1.05);
      AFFECT RTErrors RTErrors + NewErrors;
```

```
   };

   CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};


#
# Scenario that represents error propagation between two sequential activities
#
SCENARIO ErrorPropagation ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC SuccessorCount COUNT(Successors);
      PROC ErrorsToGo if (SuccessorCount > 0, (Errors + RTErrors*DT) /  SuccessorCount,
            0);

      PROC Initializing AND (ExecutionTime > 0.0, ExecutedTime < 0.001, Executing);
      PROC ReadyToInherit OR (AND(ExecutionTime < 0.001, PrecConcluded, Errors < 0.001),
            Initializing);
      PROC InheritedErrors if (ReadyToInherit, GROUPSUM (Precedences, ErrorsToGo), 0);

      AFFECT RTErrors RTErrors + InheritedErrors / DT;
   };

   CONSTRAINT TheActivity.Precedences, ErrorPropagation.TheActivity;
};


#
# Scenario that represents error regeneration on sequential activities
#
SCENARIO ErrorRegeneration ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC InheritedDensity InheritedErrors / FunctionPoints;
      PROC RegenErrors (AnalysisTask + ArchitecTask + DesignTask + CodingTask) *
            InheritedErrors * 0.24 * RegenFactor;
      PROC RegenFactor Max (1, LOOKUP (ActiveErrosDens, InheritedDensity , 0, 10));
      TABLE ActiveErrosDens 1, 1.1, 1.2, 1.325, 1.45, 1.6, 2.0, 2.5, 3.25, 4.35, 6.0;
      AFFECT RTErrors RTErrors + RegenErrors / DT;
   };

   CONSTRAINT TheActivity, ErrorPropagation.TheActivity;

   CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};


#
# Scenario that represents error correction in an activity
#
SCENARIO ErrorCorrection ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROPERTY Target 95.0;          # in %

      PROC AvgErrorFP 2.4;           # Average errors per function point

      # Errors corrected acummulator for the activity
      STOCK ErrorsCorrected 0;
      RATE (ErrorsCorrected) RTCorrection CorrErrors;

      # Error correction in the activity
      RATE (Errors) RTCorrErrors -CorrErrors;
      PROC CorrErrors (InspectionTask + TestingTask) * RTExecTime * Productivity /
            (DetectionCost * DT);
      PROC DetectionCost 0.28;

      # Adjustment of time for testng activities
      RATE (ExecutionTime) RTTesting if (AND(Executing, TestingTask > 0), -
            ExecutionTime+ExecutedTime+TestingEffort, 0) / DT;
      PROC TestingEffort TestingDifference * DetectionCost;
      PROC TestingDifference Max(Errors + InheritedErrors - TestingTarget, 0);
      PROC TestingTarget FunctionPoints * AvgErrorFP * (1 - Target / 100.0);
```

```
   };

   CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};


#
# Scenario that represents the density effect over error correction
#
SCENARIO ErrorCorrectionDensity ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC ErrorDensityMultiplier Max (1, LOOKUP (TableErrorDensityMultiplier,
            ErrorDensity, 0, 1));
      TABLE TableErrorDensityMultiplier 8, 6.75, 5.25, 4, 3, 2, 1.8, 1.6, 1.2, 1.1, 1;
      PROC ErrorDensity (Errors + RTErrors*DT) / FunctionPoints;
      AFFECT DetectionCost DetectionCost * ErrorDensityMultiplier;
   };

   CONSTRAINT TheActivity, ErrorCorrection.TheActivity;
};


#
# A software project model
#
DEFINE ControlPESC ProjectModel
{
   SPEC DT 0.1;

   CtrlPESC = NEW Project

   Grady = NEW Developer
      SET ExpAnalysis = 0.5;
      SET ExpDesign = 0.5;
      SET ExpCoding = 0.5;
      SET ExpTesting = 0.5;
      SET ExpInspection = 0.5;
      SET HourlyCost = 30;

   Johnny = NEW Developer
      SET ExpAnalysis = 0.9;
      SET ExpDesign = 0.7;
      SET ExpCoding = 0.3;
      SET ExpTesting = 0.5;
      SET ExpInspection = 0.5;
      SET HourlyCost = 40;

   Martha = NEW Developer
      SET ExpAnalysis = 0.1;
      SET ExpDesign = 0.8;
      SET ExpCoding = 0.9;
      SET ExpTesting = 0.4;
      SET ExpInspection = 0.1;
      SET HourlyCost = 15;

   Lucille = NEW Developer
      SET ExpAnalysis = 0.9;
      SET ExpDesign = 1.0;
      SET ExpCoding = 0.6;
      SET ExpTesting = 0.2;
      SET ExpInspection = 0.2;
      SET HourlyCost = 25;

   Elonor = NEW Developer
      SET ExpAnalysis = 0.2;
      SET ExpDesign = 0.2;
      SET ExpCoding = 0.4;
      SET ExpTesting = 0.2;
      SET ExpInspection = 0.1;
      SET HourlyCost = 10;

   Jimmy = NEW Developer
      SET ExpAnalysis = 0.1;
      SET ExpDesign = 0.3;
      SET ExpCoding = 0.5;
```

```
   SET ExpTesting = 0.3;
   SET ExpInspection = 0.1;
   SET HourlyCost = 20;

Phill = NEW Developer
   SET ExpAnalysis = 0.2;
   SET ExpDesign = 0.7;
   SET ExpCoding = 0.6;
   SET ExpTesting = 0.5;
   SET ExpInspection = 0.5;
   SET HourlyCost = 30;

Bobby = NEW Developer
   SET ExpAnalysis = 0.2;
   SET ExpDesign = 0.4;
   SET ExpCoding = 0.9;
   SET ExpTesting = 0.9;
   SET ExpInspection = 0.9;
   SET HourlyCost = 30;

Louis = NEW Developer
   SET ExpAnalysis = 0.3;
   SET ExpDesign = 0.3;
   SET ExpCoding = 0.5;
   SET ExpTesting = 0.3;
   SET ExpInspection = 0.9;
   SET HourlyCost = 40;

UserProf_UseCases = NEW Activity
   SET AnalysisTask = 1;
   SET FunctionPoints = 19.71;
   SET Order = 0;
   LINK Team Grady;

AreaDisc_UseCases = NEW Activity
   SET AnalysisTask = 1;
   SET FunctionPoints = 20.44;
   SET Order = 1;
   LINK Team Grady;

StudRegs_UseCases = NEW Activity
   SET AnalysisTask = 1;
   SET FunctionPoints = 23.36;
   SET Order = 2;
   LINK Team Grady;

System_Architecture = NEW Activity
   SET ArchitecTask = 1;
   SET FunctionPoints = 63.51;
   SET Order = 3;
   LINK Precedences StudRegs_UseCases, AreaDisc_UseCases, UserProf_UseCases;
   LINK Team Grady;

UserProf_Design = NEW Activity
   SET DesignTask = 1;
   SET FunctionPoints = 19.71;
   SET Order = 4;
   LINK Precedences System_Architecture;
   LINK Team Grady;

AreaDisc_Design = NEW Activity
   SET DesignTask = 1;
   SET FunctionPoints = 20.44;
   SET Order = 5;
   LINK Precedences System_Architecture;
   LINK Team Grady;

StudRegs_Design = NEW Activity
   SET DesignTask = 1;
   SET FunctionPoints = 23.36;
   SET Order = 6;
   LINK Precedences System_Architecture;
   LINK Team Grady;

UserProf_Inspection = NEW Activity
   SET InspectionTask = 1;
   SET FunctionPoints = 19.71;
```

```
   SET Order = 7;
   LINK Precedences UserProf_Design;
   LINK Team Grady;

AreaDisc_Inspection = NEW Activity
   SET InspectionTask = 1;
   SET FunctionPoints = 20.44;
   SET Order = 8;
   LINK Precedences AreaDisc_Design;
   LINK Team Grady;

StudRegs_Inspection = NEW Activity
   SET InspectionTask = 1;
   SET FunctionPoints = 23.36;
   SET Order = 9;
   LINK Precedences StudRegs_Design;
   LINK Team Grady;

UserProf_Coding = NEW Activity
   SET CodingTask = 1;
   SET FunctionPoints = 19.71;
   SET Order = 10;
   LINK Precedences UserProf_Inspection;
   LINK Team Grady;

AreaDisc_Coding = NEW Activity
   SET CodingTask = 1;
   SET FunctionPoints = 20.44;
   SET Order = 11;
   LINK Precedences AreaDisc_Inspection;
   LINK Team Grady;

StudRegs_Coding = NEW Activity
   SET CodingTask = 1;
   SET FunctionPoints = 23.36;
   SET Order = 12;
   LINK Precedences StudRegs_Inspection;
   LINK Team Grady;

UserProf_Testing = NEW Activity
   SET TestingTask = 1;
   SET FunctionPoints = 19.71;
   SET Order = 13;
   LINK Precedences UserProf_Coding;
   LINK Team Grady;

AreaDisc_Testing = NEW Activity
   SET TestingTask = 1;
   SET FunctionPoints = 20.44;
   SET Order = 14;
   LINK Precedences AreaDisc_Coding;
   LINK Team Grady;

StudRegs_Testing = NEW Activity
   SET TestingTask = 1;
   SET FunctionPoints = 23.36;
   SET Order = 15;
   LINK Precedences StudRegs_Coding;
   LINK Team Grady;

ACTIVATE ProductivityDueExpertise
   CONNECT TheDeveloper Grady;

ACTIVATE ErrorGenerationDueExpertise
   CONNECT TheDeveloper Grady;

ACTIVATE ProductivityDueLearning
   CONNECT TheDeveloper Grady;

ACTIVATE Overworking
   CONNECT TheDeveloper Grady;

ACTIVATE Exhaustion
   CONNECT TheDeveloper Grady;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserProf_UseCases;
```

```
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaDisc_UseCases;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity StudRegs_UseCases;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity System_Architecture;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserProf_Design;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaDisc_Design;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity StudRegs_Design;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserProf_Inspection;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaDisc_Inspection;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity StudRegs_Inspection;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserProf_Coding;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaDisc_Coding;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity StudRegs_Coding;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserProf_Testing;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaDisc_Testing;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity StudRegs_Testing;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity UserProf_UseCases;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity AreaDisc_UseCases;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity StudRegs_UseCases;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity System_Architecture;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity UserProf_Design;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity AreaDisc_Design;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity StudRegs_Design;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity UserProf_Coding;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity AreaDisc_Coding;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity StudRegs_Coding;
```

```
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity UserProf_Testing;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity AreaDisc_Testing;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity StudRegs_Testing;

ACTIVATE ErrorPropagation
   CONNECT TheActivity UserProf_UseCases;

ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaDisc_UseCases;

ACTIVATE ErrorPropagation
   CONNECT TheActivity StudRegs_UseCases;

ACTIVATE ErrorPropagation
   CONNECT TheActivity System_Architecture;

ACTIVATE ErrorPropagation
   CONNECT TheActivity UserProf_Design;

ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaDisc_Design;

ACTIVATE ErrorPropagation
   CONNECT TheActivity StudRegs_Design;

ACTIVATE ErrorPropagation
   CONNECT TheActivity UserProf_Inspection;

ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaDisc_Inspection;

ACTIVATE ErrorPropagation
   CONNECT TheActivity StudRegs_Inspection;

ACTIVATE ErrorPropagation
   CONNECT TheActivity UserProf_Coding;

ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaDisc_Coding;

ACTIVATE ErrorPropagation
   CONNECT TheActivity StudRegs_Coding;

ACTIVATE ErrorPropagation
   CONNECT TheActivity UserProf_Testing;

ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaDisc_Testing;

ACTIVATE ErrorPropagation
   CONNECT TheActivity StudRegs_Testing;

ACTIVATE ErrorRegeneration
   CONNECT TheActivity UserProf_UseCases;

ACTIVATE ErrorRegeneration
   CONNECT TheActivity AreaDisc_UseCases;

ACTIVATE ErrorRegeneration
   CONNECT TheActivity StudRegs_UseCases;

ACTIVATE ErrorRegeneration
   CONNECT TheActivity System_Architecture;

ACTIVATE ErrorRegeneration
   CONNECT TheActivity UserProf_Design;

ACTIVATE ErrorRegeneration
   CONNECT TheActivity AreaDisc_Design;

ACTIVATE ErrorRegeneration
   CONNECT TheActivity StudRegs_Design;
```

```
   ACTIVATE ErrorRegeneration
      CONNECT TheActivity UserProf_Coding;

   ACTIVATE ErrorRegeneration
      CONNECT TheActivity AreaDisc_Coding;

   ACTIVATE ErrorRegeneration
      CONNECT TheActivity StudRegs_Coding;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity UserProf_UseCases;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity AreaDisc_UseCases;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity StudRegs_UseCases;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity System_Architecture;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity UserProf_Design;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity AreaDisc_Design;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity StudRegs_Design;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity UserProf_Coding;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity AreaDisc_Coding;

   ACTIVATE ErrorGeneration
      CONNECT TheActivity StudRegs_Coding;

   ACTIVATE ErrorCorrection
      CONNECT TheActivity UserProf_Inspection;

   ACTIVATE ErrorCorrection
      CONNECT TheActivity AreaDisc_Inspection;

   ACTIVATE ErrorCorrection
      CONNECT TheActivity StudRegs_Inspection;

   ACTIVATE ErrorCorrection
      CONNECT TheActivity UserProf_Testing;

   ACTIVATE ErrorCorrection
      CONNECT TheActivity AreaDisc_Testing;

   ACTIVATE ErrorCorrection
      CONNECT TheActivity StudRegs_Testing;

   ACTIVATE ErrorCorrectionDensity
      CONNECT TheActivity UserProf_Inspection;

   ACTIVATE ErrorCorrectionDensity
      CONNECT TheActivity AreaDisc_Inspection;

   ACTIVATE ErrorCorrectionDensity
      CONNECT TheActivity StudRegs_Inspection;

   ACTIVATE ErrorCorrectionDensity
      CONNECT TheActivity UserProf_Testing;

   ACTIVATE ErrorCorrectionDensity
      CONNECT TheActivity AreaDisc_Testing;

   ACTIVATE ErrorCorrectionDensity
      CONNECT TheActivity StudRegs_Testing;
};
```

Apêndice C – A Dinâmica Utilizada no Estudo Experimental

C.1 Introdução

Este documento descreve a dinâmica do emulador de projetos de software desenvolvido para o estudo experimental das técnicas de integração e simulação de modelos de projeto e cenários no contexto do gerenciamento de um projeto de desenvolvimento de software. A mesma dinâmica é replicada nos cenários apresentados no Apêndice B. Para apresentar a dinâmica, vamos abordar o processo de desenvolvimento adotado para a construção do sistema CtrlPESC, apresentado na seção 5.3, e as equações que descrevem a dinâmica deste processo.

Este apêndice está escrito em três seções. A primeira contém esta introdução. Em seguida, a seção C.2 descreve o processo de desenvolvimento adotado para o sistema CtrlPESC. Finalmente, a seção C.3 descreve as equações que determinam a dinâmica do processo de desenvolvimento.

C.2 O Processo de Desenvolvimento

Nesta seção, descrevemos o processo de desenvolvimento de software definido para o sistema CtrlPESC e apresentado no emulador de projetos de software utilizado no estudo experimental. Antes de tratarmos do processo de desenvolvimento propriamente dito, cabe uma observação sobre a escolha do sistema para o estudo experimental. Esta escolha foi norteada pelas características do processo de desenvolvimento, conforme descrito a seguir.

C.2.1 Sobre o Sistema Escolhido para o Estudo Experimental

No início do planejamento do estudo experimental, pensávamos em utilizar outro sistema como base do emulador de projetos. Este seria um sistema de controle de estacionamentos, desenvolvido por quatorze grupos de alunos do curso de Engenharia de Software oferecido na graduação em Informática da Universidade de Maryland. O sistema foi desenvolvido sob a coordenação do prof. Guilherme Travassos, que definiu seu processo de desenvolvimento e coletou informações detalhadas sobre este processo, entre elas o tempo consumido pelos alunos em cada atividade do projeto.

Entretanto, com o desenrolar do planejamento do estudo experimental, percebemos que o processo de desenvolvimento do sistema de estacionamento não se adequava perfeitamente à simulação pretendida. O principal problema está relacionado com o nível de desagregação desejado para este processo de desenvolvimento.

O processo de desenvolvimento do sistema de controle de estacionamento foi construído com base nas atividades do processo de desenvolvimento de software, sendo apresentado aos alunos em um nível de agregação tal que uma atividade poderia ser desenvolvida por mais de um aluno. Tal ocorreu porque o professor deixou a cargo dos grupos de alunos a divisão das responsabilidades dentro de cada atividade. Entretanto, as medidas realizadas pelos grupos contemplam as atividades no nível em que foram definidas no processo, não apresentando o tempo consumido por cada desenvolvedor em cada divisão destas atividades.

No estudo experimental atual, precisamos de um nível maior de desagregação. Como o estado da arte da Engenharia de Software não dispõe de dados precisos sobre a interação de diversos desenvolvedores realizando uma mesma atividade, decidimos que as atividades deveriam ser decompostas até que cada atividade fosse realizada por um único desenvolvedor. A inexistência deste nível de detalhamento no sistema de estacionamento dificulta sua utilização no estudo experimental.

Assim, decidimos adotar um sistema de informação como base do estudo experimental. O sistema CtrlPESC foi selecionado considerando os seguintes critérios:

- **Porte**: conforme descrito na seção C.3.1 que calcula a complexidade do sistema, o CtrlPESC é um sistema de pequeno porte, o que permite seu gerenciamento durante o estudo experimental;
- **Disponibilidade do Documentação**: o modelo de dados do sistema e outros documentos de apoio estão disponíveis para avaliação da complexidade do sistema;
- **Disponibilidade do Desenvolvedor**: o responsável pelo desenvolvimento e implantação do sistema está disponível para resolver qualquer dúvida em relação ao sistema;

Atendidas as características acima, o sistema CtrlPESC foi medido utilizando-se a métrica de pontos por função e sua dinâmica foi definida combinando-se equações e dados disponíveis na literatura. Estes dados são detalhados na seção C.3.

C.2.2 As Etapas e Atividades do Processo de Desenvolvimento

O processo de desenvolvimento definido para o sistema CtrlPESC leva em consideração tanto uma divisão vertical quanto uma divisão horizontal das atividades relacionadas com o desenvolvimento de software. A divisão vertical considera as etapas do processo de desenvolvimento. Foram consideradas as seguintes etapas:

- Análise de Requisitos
- Projeto de Arquitetura
- Projeto Detalhado
- Inspeções de Projeto
- Codificação
- Testes

O processo foi definido utilizando-se um modelo em cascata, segundo o qual uma etapa do processo de desenvolvimento não começa antes que as anteriores tenham sido cumpridas. A decomposição horizontal considera as divisões realizadas em cada uma desta etapas. Nesta divisão, consideramos os diversos componentes (produtos ou artefatos) desenvolvidos pelo projeto. Esta divisão orienta a medição do projeto através de pontos por função e a distribuição do esforço de desenvolvimento ao longo do tempo. Cada divisão é considerada uma atividade, sendo cumprida por um desenvolvedor.

- Cadastro de Usuários
- Cadastro de Professores
- Cadastro de Linhas de Pesquisa
- Cadastro de Disciplinas
- Cadastro de Alunos
- Inscrições em Disciplinas

O processo de desenvolvimento inicial do sistema CtrlPESC está apresentado abaixo. As inspeções de projeto são opcionais, podendo ser ativadas pelo gerente se este as julgar necessárias e vantajosas.

- Análise de Requisitos
  - Casos de Uso de Usuários
  - Casos de Uso de Professores
  - Casos de Uso de Linhas de Pesquisa
  - Casos de Uso de Disciplinas
  - Casos de Uso de Alunos
  - Casos de Uso de Inscrições em Disciplinas
- Projeto de Arquitetura

- Projeto Detalhado
  - Cadastro de Usuários
  - Cadastro de Professores
  - Cadastro de Linhas de Pesquisa
  - Cadastro de Disciplinas
  - Cadastro de Alunos
  - Inscrições em Disciplinas
- Inspeções de Projeto
  - Revisão do Cadastro de Usuários
  - Revisão do Cadastro de Professores
  - Revisão do Cadastro de Linhas de Pesquisa
  - Revisão do Cadastro de Disciplinas
  - Revisão do Cadastro de Alunos
  - Revisão das Inscrições em Disciplinas
- Codificação
  - Código do Cadastro de Usuários
  - Código do Cadastro de Professores
  - Código do Cadastro de Linhas de Pesquisa
  - Código do Cadastro de Disciplinas
  - Código do Cadastro de Alunos
  - Código das Inscrições em Disciplinas
- Testes
  - Testes do Cadastro de Usuários
  - Testes do Cadastro de Professores
  - Testes do Cadastro de Linhas de Pesquisa
  - Testes do Cadastro de Disciplinas
  - Testes do Cadastro de Alunos
  - Testes das Inscrições em Disciplinas

Para reduzir o número de atividades do projeto e facilitar seu gerenciamento durante o estudo experimental, algumas atividades foram agregadas, constando do desenvolvimento de dois artefatos em uma mesma atividade. Assim, a descrição do projeto foi modificada, sendo descrita a seguir:

- Análise de Requisitos
  - Casos de Uso de Usuários e Professores
  - Casos de Uso de Linhas e Disciplinas
  - Casos de Uso de Alunos e Inscrições
- Projeto de Arquitetura
- Projeto Detalhado
  - Cadastro de Usuários e Professores
  - Cadastro de Linhas e Disciplinas
  - Cadastro de Alunos e Inscrições
- Inspeções de Projeto
  - Revisão do Cadastro de Usuários e Professores
  - Revisão do Cadastro de Linhas e Disciplinas
  - Revisão do Cadastro de Alunos e Inscrições

- Codificação
  - Código do Cadastro de Usuários e Professores
  - Código do Cadastro de Linhas e Disciplinas
  - Código do Cadastro de Alunos e Inscrições
- Testes
  - Testes do Cadastro de Usuários e Professores
  - Testes do Cadastro de Linhas e Disciplinas
  - Testes do Cadastro de Alunos e Inscrições

C.3 A Dinâmica do Processo de Desenvolvimento

Nesta seção, apresentamos e justificamos os números, as equações e as teorias utilizadas no emulador de projetos de desenvolvimento de software que foi construído para o estudo experimental. Estes dados foram derivados ou extraídos de três fontes básicas:

- O modelo de projetos de desenvolvimento de software de ABDEL-HAMID e MADNICK (1991), documentado no livro dos mesmos autores. Este modelo foi desenvolvido ao longo da década de 1980 e seus dados acerca de produtividade, taxas de geração de erros e outras medidas podem estar desatualizados. Assim, utilizaremos apenas as teorias aqui publicadas, utilizando fontes mais recentes para determinação dos valores propriamente ditos;
- O manual de contagem de pontos por função do IFPUG (1999) - *International Function Point Users Group*. Este manual foi utilizado para avaliar a complexidade do sistema CtrlPESC. A medida de pontos por função foi selecionada devido a disponibilidade de dados recentes sobre produtividade e qualidade de projetos de software, apresentados em nossa terceira referência, descrita a seguir;
- O livro "Software Assessments, Benchmarks, and Best Practices" (JONES, 2000), que classifica os produtos de software em cinco categorias (as mesmas previamente utilizadas pelo autor em seu livro sobre análise de riscos), apresentando dados sobre produtividade e qualidade (número de erros) em projetos de software em cada uma destas categorias. A métrica de pontos por função é utilizada como base destas medidas de produtividade e qualidade.

C.3.1 Medida de Complexidade

Como os dados sobre produtividade foram extraídos de JONES (2000), temos que medir a complexidade do sistema CtrlPESC na mesma unidade utilizada neste livro:

pontos por função. A melhor referência para esta medida é o manual de contagem de pontos por função do IFPUG (1999), grupo internacional que busca padronizar a medida de projetos de software por pontos de função. Na página 67 de seu livro, JONES (2000) determina que *"Os dados utilizados ... se baseiam na versão 4.1 das regras de contagem de pontos por função do IFPUG"*. O mesmo manual foi utilizado na contagem do número de pontos por função do sistema CtrlPESC.

Na Tabela C.1, apresentamos um resumo do número de campos (DETS) dos arquivos internos (ILFS) utilizados pelo sistema CtrlPESC. Ao lado, figura o número de pontos por função não-ajustados (UFP) decorrentes destes arquivos.

Arquivos Internos (ILF)	DETS	UFP
Usuários	4	7
Linhas	2	7
Professores	5	7
Disciplinas	7	7
Alunos	13	7
Inscrições	4	7
Total		42

Tabela C.1 – Pontos de função decorrentes dos arquivos internos do sistema CtrlPESC

O sistema CtrlPESC não utiliza arquivos externos nem saídas externas na forma de relatórios. Todas as suas páginas de saída foram consideradas como consultas, sendo contabilizadas desta forma (EQ). As páginas de entrada de dados são consideradas como operações de entrada de informações (EI). Abaixo, apresentamos uma tabela com as informações referentes a estas entradas.

Operações	EI						EQ			Total
	FTR	FLD	MSG	OPR	DET	UFP	FTR	FLD	UFP	UFP
Cadastro de Usuários	1	4	1	3	8	3	1	4	3	6
Cadastro de Linhas	1	2	1	3	6	3	1	2	3	6
Cadastro de Professores	2	5	1	3	9	4	2	5	3	7
Cadastro de Disciplinas	2	7	1	3	11	4	2	7	4	8
Cadastro de Alunos	3	11	1	3	15	6	0	0	0	6
Inscrição em Disciplinas	4	5	1	2	8	6	0	0	0	6
Consulta de Disciplinas	0	0	0	0	0	0	4	14	6	6
Total										45

Tabela C.2 – Pontos de função decorrentes de consultas e operações de cadastramento

Na Tabela C.2, o campo FTR indica o número de arquivos afetados por cada operação do sistema CtrlPESC. Por exemplo, o cadastro de professores precisa acessar o arquivo de professores e o arquivo de linhas para completar a referência entre os professores e as linhas de pesquisa do departamento.

O campo FLD indica o número de campos de dados atômicos que são utilizados em cada operação do sistema. No caso do cadastro de linhas, por exemplo, temos dois campos nas páginas de inclusão, alteração e remoção (EI) e dois campos na operação de consulta (EQ).

O campo MSG indica se a operação de entrada gera mensagens de erro ou críticas para o usuário. Este campo não é considerado nas operações de consulta.

O campo OPR indica o número de operações associadas a uma página de entrada. No cadastro de usuários, por exemplo, temos as operações de inclusão, alteração e remoção. As três operações estão associadas a mesma página de entrada de dados. O mesmo ocorre em todos os cadastros, exceto no de disciplinas, onde temos apenas as operações de inclusão e remoção. Assim como o campo MSG, este campo também não é considerado nas operações de consulta.

O campo DET representa o somatório dos campos FLD, MSG e OPR, sendo considerado apenas nas operações de entrada. Este campo é utilizado em conjunto com o campo FTR no cálculo do número de pontos por função não ajustados (UFP) decorrente das páginas de operações de entrada. Nas operações de consulta, os campos FTR e FLD são utilizados com o mesmo fim.

O número total de pontos por função não-ajustados do sistema CtrlPESC é então calculado como 45 UFP's. Devemos agora ajustar este número segundo os fatores de ajuste do IFPUG. Estes fatores foram avaliados conforme a tabela Tabela C.3.

Fator	Ajuste
Comunicação de Dados	0
Processamento Distribuído	0
Desempenho	1
Capacidade de Configuração	0
Taxa de Transações	0
Entrada de Dados Online	2
Eficiência no Usuário Final	2
Atualização Online	0
Processamento Complexo	0
Reusabilidade	0
Facilidade de Instalação	2
Facilidade de Operação	1
Diversas Instalações	0
Facilidade de Alteração	0

Tabela C.3 –Fatores de ajuste do números de pontos de função do sistema CtrlPESC

Com tais considerações, o número de pontos por função ajustado foi calculado conforme somando-se os fatores de ajuste, multiplicando o somatório por 0,01 e somando o resultado a 0,65. O resultado desta operação é multiplicado pelo número de pontos de função não-ajustados, gerando o número de pontos de função ajustado.

```
AFP = (0.65 + SUM(Ajustes) * 0.01) * UFP
AFP = 63.5 FP ou 64 FP
```

C.3.2 Complexidade pelo Processo de Desenvolvimento

Com base no número total de pontos por função da aplicação, vamos distribuir a complexidade pelos artefatos componentes do sistema. Nesta divisão, apresentada na Tabela C.4, consideramos a desagregação proposta no processo de desenvolvimento.

FP por Atividade	UFP	AFP	% Projeto
Usuários e Professores	27,0	19,7	31%
Linhas e Disciplinas	28,0	20,4	32%
Alunos e Inscrições	32,0	23,4	37%
Total	87,0	63,5	100%

Tabela C.4 – Divisão da complexidade pelos artefatos componentes do sistema CtrlPESC

A seguir, precisamos calcular o número de pontos por função por atividade do processo de desenvolvimento. Isto é possível porque JONES (2000) oferece dados sobre a produtividade e taxa de geração de erros em pontos por função. Assim, se tivermos o número de pontos dividido pelas atividades do processo de desenvolvimento, poderemos estabelecer o tempo necessário para o cumprimento de cada uma destas atividades, assim como o número de erros gerados por seus desenvolvedores.

Atividade	% Tempo
Requisitos	3,66
Projeto Inicial	3,29
Projeto Detalhado	4,39
Codificação	18,29
Reutilização	0,32
Gerência de Configuração	1,32
Documentação	4,39
Testes de Unidade	16,46
Testes de Função	14,32
Testes de Sistema	13,17
Testes de Aceitação	9,41
Gerência de Projeto	10,98
Total	100,00

Tabela C.5 –Percentual estimado do tempo de projeto dedicado por atividade

Para dividir o número de pontos por função por atividade utilizamos a Tabela C.5, apresentada nas páginas 188 e 189 de JONES (2000). Esta tabela apresenta o percentual médio do tempo consumido em cada uma das atividades do processo de desenvolvimento de software.

As atividades utilizadas por CI não podem ser diretamente mapeadas sobre as atividades apresentadas no processo de desenvolvimento selecionado para o sistema CtrlPESC. Assim, realizamos um mapeamento entre as atividades apresentadas por JONES (2000) e as atividades definidas para o processo. Este mapeamento é apresentado na Tabela C.6.

Atividade	% Tempo
Requisitos	4,11
Projeto de Arquitetura	3,70
Projeto Detalhado	11,71
Codificação	20,54
Testes	59,94
Total	100,00

Tabela C.6 – Percentual estimado do tempo de projeto dedicado por atividade (após o mapeamento)

No mapeamento, assumimos as seguintes condições:

- Consideramos que o projeto inicial de JONES (2000) é o projeto arquitetônico, onde os principais componentes do sistema são definidos e modelados;
- Consideramos que as atividades de reutilização, gerência de configuração e de documentação fazem parte do projeto detalhado do sistema. O tempo consumido por estas atividades é distribuído uniformemente ao longo do tempo dedicado ao projeto detalhado;
- Consideramos que as atividades de teste de unidade, de sistema, função e aceitação são unificadas em uma única atividade, denominada apenas por teste no modelo;
- O tempo consumido na gerência de projetos é distribuído linearmente ao longo do processo de desenvolvimento, de acordo com o percentual de tempo calculado para cada atividade. Esta distribuição deve ser feita após as agregações apresentadas acima, visto que somente então teremos calculado o tempo dedicado a cada atividade do processo de desenvolvimento.

Com base na divisão do tempo de projeto apresentada acima, podemos definir o número de pontos por função associado a cada atividade do processo de desenvolvimento do sistema CtrlPESC. Estes dados são apresentados na Tabela C.7. As

inspeções não foram consideradas, sendo tratadas de forma conveniente, conforme apresentado na seção C.3.5.2.

Atividade	% Projeto	AFP
Requisitos	4,11	2,61
Use Cases de Usuários e Professores	1,27	0,81
Use Cases de Linhas e Disciplinas	1,31	0,84
Use Cases de Alunos e Inscrições	1,52	0,96
Projeto de Arquitetura	3,70	2,53
Projeto Detalhado	11,71	7,43
Projeto de Usuários e Professores	3,63	2,31
Projeto de Linhas e Disciplinas	3,75	2,39
Projeto de Alunos e Inscrições	4,33	2,73
Codificação	20,54	13,05
Código de Usuários e Professores	6,38	4,05
Código de Linhas e Disciplinas	6,61	4,20
Código de Alunos e Inscrições	7,56	4,80
Testes	59,94	38,06
Testes de Usuários e Professores	18,60	11,81
Testes de Linhas e Disciplinas	19,29	12,25
Testes de Alunos e Inscrições	22,05	14,00

Tabela C.7 – Percentual estimado do tempo de projeto dedicado a cada atividade do sistema CtrlPESC

C.3.3 Produtividade

JONES (2000) apresenta dados sobre produtividade nas diversas categorias em que classifica os produtos de software. Dentro destas categorias, os sistemas são agrupados segundo sua complexidade, medida em pontos por função. Dados sobre produtividade estão disponíveis para cada grupo de cada categoria.

Entre as categorias definidas por JONES (2000), encontram-se os sistemas de informação, ou MIS, onde melhor se enquadra o sistema CtrlPESC. Nesta categoria, o autor caracteriza três dimensões de projetos de software: projetos de até 100 pontos por função, projetos de até 1000 pontos por função e projetos de até 10000 pontos por função. O sistema CtrlPESC se enquadra na primeira categoria, para a qual o autor apresenta os dados sobre produtividade (na página 191) indicados na Tabela C.8.

Número médio de pontos por função por mês	27.80
Número máximo de pontos por função por mês	40.19

Tabela C.8 - Produtividade de desenvolvedores em projetos com até 100 pontos de função

Estes dados nos levam a derivar uma distribuição de probabilidade. A distribuição mais natural é uma distribuição beta PERT, utilizada quando podemos indicar o número

mais provável, o máximo e o mínimo que pode ser assumido pelo elemento representado pela distribuição. Como dispomos apenas da média e do máximo, precisamos definir um número mínimo de pontos por função por mês, completando a distribuição de probabilidade da produtividade dos desenvolvedores de software.

Para definir este número mínimo, assumimos simetria em relação a média, rebatendo a diferença entre o máximo e a média em relação a esta última. Assim, temos

```
Mínimo = Média - (Máximo - Média)
Mínimo = 27.80 - (40.19 - 27.80)
Mínimo = 27.80 - 12.39 = 15.41
```

De posse destes dados, temos a função de distribuição de probabilidade para a produtividade de desenvolvedores de software apresentada na Tabela C.9.

Produtividade = beta-PERT (mínimo, média, máximo)	
Mínimo de PF/mês	15.41
Média de PF/mês	27.80
Máximo de PF/mês	40.19

Tabela C.9 – Função de distribuição de probabilidade que modela a produtividade dos desenvolvedores

Com base nesta distribuição e na complexidade de cada atividade do processo de desenvolvimento definido para o sistema CtrlPESC, podemos determinar o tempo mínimo, médio e máximo necessário para a realização de cada atividade. O tempo médio para a realização de uma atividade é calculado pela seguinte equação:

```
Tempo = PontosFuncao . (30/MédiaPontosMes) . (Contribuicao/100)
```

A equação acima pode ser dividida em duas partes. A primeira calcula o número de pontos por função produzidos por dia. Este cálculo é realizado dividindo-se o número de dias por mês, determinado em dias corridos, pela produtividade média, medida em pontos por função por mês.

A segunda parte da equação calcula a contribuição da atividade para o processo de desenvolvimento de software. Para tanto, utilizamos o número de pontos por função calculados para a atividade. Este número é multiplicado pela contribuição do tipo de atividade dentro do processo de desenvolvimento, calculando-se assim o número de pontos por função ajustado pelo tipo de atividade.

A equação acima é utilizada para calcular a duração média de uma atividade do processo de desenvolvimento. Neste cálculo, utiliza-se a produtividade média dos desenvolvedores, como dividendo do primeiro termo da equação. Para calcularmos a

duração mínima de cada atividade, utilizamos a produtividade máxima dos desenvolvedores. De forma similar, utilizamos a produtividade mínima dos desenvolvedores para calcular a duração máxima de cada atividade.

C.3.4 Desenvolvedores

Tendo determinado a distribuição de probabilidade da duração de cada atividade, passamos a caracterizar os desenvolvedores de software que serão os agentes disponíveis no emulador de projetos para a realização das atividades. No emulador de projetos, um gerente pode realizar as seguintes operações relacionadas com desenvolvedores:

- **Definição da Equipe:** existe um conjunto de desenvolvedores candidatos a participarem do projeto, dentre os quais o gerente deve selecionar aqueles que efetivamente irão participar do projeto. Cada desenvolvedor possui um custo por hora e o custo por hora do projeto será determinado pelo somatório dos custos por hora de cada desenvolvedor realizando alguma atividade no projeto;
- **Realizador de Atividade:** para cada atividade do processo de desenvolvimento, o gerente deve selecionar o desenvolvedor responsável pela atividade. Conforme dito anteriormente, uma atividade deve ser realizada por um único desenvolvedor;
- **Dedicação Diária:** o gerente pode solicitar a um ou mais desenvolvedores que trabalhem um determinado número de horas extras durante um período. A aceitação do trabalho extra depende de regras indicadas na seção C.3.4.3.

Cada desenvolvedor é caracterizado pelas seguintes informações:

- **Nome**: nome do desenvolvedor, que o identifica no sistema;
- **Custo por Hora**: custo da hora de trabalho do desenvolvedor;
- **Tempo no Projeto**: tempo de participação do desenvolvedor no projeto, em dias;
- **Experiência**: nível de experiência do desenvolvedor em cada tipo de atividade.

O custo por hora do desenvolvedor é utilizado para calcular o custo do projeto ao longo do tempo. No emulador de projetos, o gerente seleciona um conjunto de desenvolvedores para comporem sua equipe. A cada passo de simulação realizado, o emulador contabiliza o custo do tempo passado para cada desenvolvedor participante da equipe, somando este custo ao custo atual do projeto.

Nas próximas subseções, discutimos como as características dos desenvolvedores afetam seus principais fatores dinâmicos, determinantes na dinâmica dos projetos em que participam: sua produtividade e sua taxa de geração de erros.

C.3.4.1 Produtividade

A experiência do desenvolvedor em um determinado tipo de atividade influencia sua produtividade na realização das atividades do mesmo tipo, componentes do processo do sistema CtrlPESC. No modelo de desenvolvedor adotado pelo emulador de projetos, a experiência varia de zero a um, indicando zero para os desenvolvedores sem experiência e um para desenvolvedores experientes.

A influência da experiência na produtividade de um desenvolvedor é determinada por um mecanismo similar ao utilizado no modelo de ABDEL-HAMID e MADNICK (1991). Este modelo determina que um desenvolvedor experiente produz o dobro que um desenvolvedor inexperiente. Tal constatação é comprovada pelo resultado de entrevistas e por uma pesquisa na literatura de Engenharia de Software, conforme apresentado na página 83 do livro que descreve o modelo.

Como os dados de produtividade de JONES (2000) são determinados para um desenvolvedor de experiência mediana, assumimos que sua produtividade é equivalente a um. Devemos então, calcular um modificador de produtividade linear com a experiência do desenvolvedor na determinada atividade. Temos então as seguintes condições:

```
Produtividade Máxima: 1 + X
Produtividade Mínima: 1 - X
```

Pelo modelo de ABDEL-HAMID e MADNICK (1991), temos:

```
(1 - X) . 2 = 1 + X
2 - 2X = 1 + X
X = 1 / 3 = 0.333
```

Assim:

```
Produtividade Máxima: 1 + X = 1.333
Produtividade Mínima: 1 - X = 0.667
Modificador de Experiência = (1 - 0.337) + 2*0.337*Experiência
```

Outro fator que influencia a produtividade de um desenvolvedor é o tempo de participação no projeto. Segundo ABDEL-HAMID e MADNICK (1991), "a medida que

um projeto avança, seus desenvolvedores aprendem melhor seu trabalho". Uma curva de aprendizado reflete este conhecimento. A medida que aumenta o tempo de participação de um desenvolvedor em um projeto, sua produtividade aumenta de acordo com a curva de aprendizado.

O modelo de ABDEL-HAMID e MADNICK (1991) fornece uma curva de aprendizado, indicando o impacto na produtividade decorrente do tempo de participação de um desenvolvedor em um projeto. Este impacto é modelado como um modificador multiplicativo, devendo ser multiplicado pela produtividade original para determinar a produtividade decorrente do aprendizado. Ele é apresentado na Tabela C.10.

No modelo de ABDEL-HAMID e MADNICK (1991), o tempo de participação no projeto é medido como um percentual desde o início do projeto até a data projetada para sua conclusão. O modelo supõe que o desenvolvedor participa do projeto desde o seu início. Tal decorre, provavelmente, das limitações da Dinâmica de Sistemas na representação dos atributos particulares de cada elemento. Isto leva diversos autores de modelos a utilizarem médias ou grandes números que representem uma população supostamente homogênea.

0%	1.0
10%	1.0125
20%	1.0325
30%	1.055
40%	1.09
50%	1.15
60%	1.2
70%	1.22
80%	1.245
90%	1.25
100%	1.25

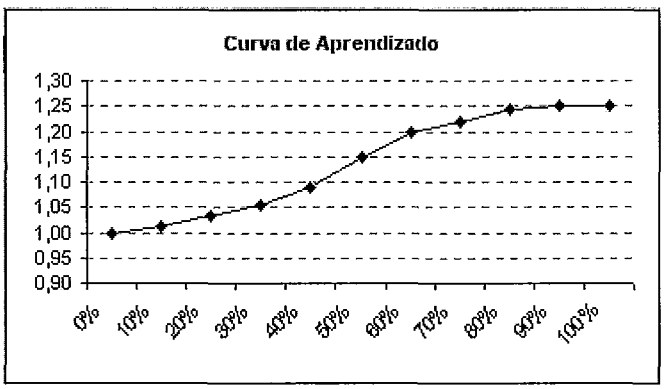

Tabela C.10 – Curva de aprendizado ao longo da execução do projeto

No modelo original, o pleno aprendizado (ponto mais alto da curva) somente é alcançado no final do projeto. Acreditamos que esta afirmação não é razoável para qualquer tipo de projeto. Em projetos mais simples, acreditamos que o conhecimento pleno do domínio da aplicação, pelo menos no contexto do projeto, pode ser atingido muito antes de sua conclusão. Assim, estipulamos um tempo para que um desenvolvedor alcance o nível de pleno conhecimento do domínio da aplicação. Este tempo, medido em dias, permite o cálculo do modificador de produtividade decorrente

do aprendizado. O percentual de tempo de projeto, primeira coluna da tabela acima, é calculado com base neste tempo.

A cada passo de simulação em que um desenvolvedor permanece no projeto, o emulador de projetos incrementa seu tempo de permanência no mesmo, aumentando seu modificador de produtividade. Para o sistema CtrlPESC, estipulamos que 20 dias são necessários para que um desenvolvedor atinja o pleno conhecimento.

Finalmente, o último fator que influencia a produtividade de um desenvolvedor é o tempo de trabalho diário . Quando um desenvolvedor é escalado para o projeto, este começa a trabalhar em ritmo normal, que o modelo considera como 8 horas de trabalho por dia. Entretanto, devido a atraso ou a inclusão de novas funcionalidades sem o devido ajuste no cronograma, o gerente pode requisitar ao desenvolvedor que trabalhe um número maior de horas por dia. O modelo assume um limite de 12 horas de trabalho por dia, ou seja, 4 horas extras.

Para determinar o modificador de produtividade decorrente do número de horas de trabalho diário, utilizamos a teoria expressa no modelo de ABDEL-HAMID e MADNICK (1991). Esta teoria determina que um desenvolvedor trabalhando 8 horas por dia utiliza apenas 60% deste tempo em trabalho útil. O restante do tempo é despendido em outras atividades não relacionadas ou indiretamente relacionadas com o desenvolvimento de software, tais como, como reuniões, conversas, telefone e resolução de problemas pessoais[1]. Entretanto, em determinadas situações, especialmente quando o projeto está atrasado, os desenvolvedores têm que trabalhar mais horas por dia. Neste caso, sua produtividade aumenta, chegando a dobrar o tempo útil diário (que atinge 120% do número de horas de trabalho do período normal) com um acréscimo de apenas 50% de tempo trabalho diário (12 horas). O modelo assume que a produtividade aumenta linearmente com a pressão de cronograma.

```
Dedicação = 0.6 + (1.2 - 0.6) * (NumeroHoras - 8) / (12 - 8)
```

Assim, se assumirmos que a produtividade de um desenvolvedor com experiência mediana equivale ao valor unitário, a capacidade de produção de um desenvolvedor é determinada pelo produto dos três modificadores de produtividade: experiência, aprendizado e dedicação ao projeto.

---

[1] No original, este tempo restante é denominado *slack-time.*

C.3.4.2 Geração de Erros

Alguns dos modificadores que determinam a produtividade de um desenvolvedor também influenciam sua taxa de geração de erros, ou seja, a qualidade dos artefatos de software construídos por eles. A experiência do desenvolvedor afeta a taxa de geração de erros. JONES (2000) apresenta o número médio de erros produzidos por pontos por função em cada uma das atividades do processo de desenvolvimento de software. ABDEL-HAMID e MADNICK (1991), novamente apoiados em uma revisão literária e em um conjunto de entrevistas com especialistas, indicam que um desenvolvedor inexperiente produz duas vezes mais erros que um desenvolvedor experiente. Aplicando uma fórmula similar a que foi utilizada acima no cálculo da produtividade, temos a equação que determina o modificador de geração de erros de um desenvolvedor, de acordo com sua experiência em uma determinada atividade do processo de desenvolvimento.

```
Modificador = (1 - 0.337) + 2 * 0.337 * (1 - Experiência)
```

Outro fator que influencia a taxa de geração de erros de um desenvolvedor é o número de horas de trabalho por dia. Segundo DEMARCO (1982), "desenvolvedores trabalhando um número maior de horas, trabalham mais, não melhor". ABDEL-HAMID e MADNICK (1991) utilizaram esta premissa, junto com um conjunto de entrevistas, para determinar o efeito do tempo de trabalho diário sobre a taxa de geração de erros. Os resultados encontrados são apresentados na Tabela C.11.

Horas/Dia	Modificador de Erros
8 horas	0.90
8 ½ horas	0.94
9 horas	1.00
9 ½ horas	1.05
10 horas	1.14
10 ½ horas	1.24
11 horas	1.36
12 horas	1.50

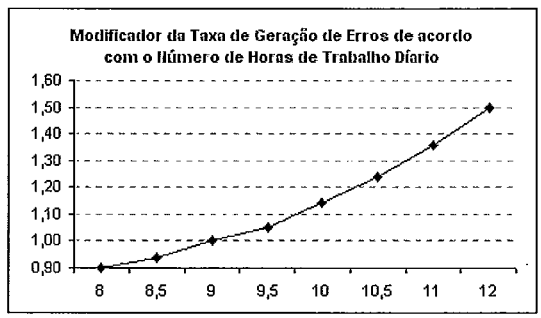

Tabela C.11 – Modificador da taxa de geração de erros pelo número horas de trabalho diário

A tabela e o gráfico acima expressam o modificador da taxa de geração de erros decorrente do número de horas de trabalho diárias de um desenvolvedor. Pela tabela, percebe-se que, em ritmo normal de trabalho, a taxa de geração de erros de um desenvolvedor é muito inferior da mesma taxa em ritmo acelerado de trabalho. A curva

expressa a afirmativa de DEMARCO (1982) e as dificuldades relacionadas com o trabalho em hora extra no desenvolvimento de software.

Com os dois modificadores acima, podemos calcular a taxa de geração de erros de um desenvolvedor. Considerando-se que a taxa de geração de erros de um desenvolvedor com experiência mediana é o valor unitário, a taxa de geração de erros de um desenvolvedor é determinada pelo produto dos dois modificadores supracitados: sua experiência e sua dedicação diária ao projeto.

C.3.4.3 Exaustão

ABDEL-HAMID e MADNICK (1991) apresentam uma teoria sobre o trabalho de desenvolvedores em horas extras durante um período de tempo prolongado. Segundo esta teoria, a medida que os desenvolvedores trabalham em ritmo acelerado, além dos efeitos provocados sobre a produtividade e a taxa de geração de erros, existe um efeito de exaustão que afeta a capacidade dos desenvolvedores em manterem o ritmo de trabalho acelerado.

Segundo os autores, a medida em que trabalham horas extras, os desenvolvedores ficam progressivamente mais cansados e menos propensos a trabalhar novamente em horas extras. Se o período de trabalho extra se estender, ele pode atingir um limite onde os desenvolvedores se recusam a trabalhar mais horas extras, retornando a um ritmo normal de trabalho (8 horas/dia) até que estejam descansados, recuperados do período de estresse.

O modelo de projeto utilizado no emulador de projetos segue a teoria de exaustão de ABDEL-HAMID e MADNICK (1991). Por este modelo, inicialmente calcula-se um modificador (α), que varia linearmente entre 0 a 1, de acordo com o número de horas trabalhadas diariamente no projeto.

```
α = (NumeroHoras - 8) / (12 - 8)
```

A partir deste modificador, calcula-se a dedicação diária do desenvolvedor ao projeto, que varia entre 60% e 120% (ver seção C.3.4.1).

```
Dedicacao = 0.6 + α * (1.2 - 0.6)
```

Finalmente, a dedicação ao projeto é utilizada para o cálculo do impacto de exaustão devido a um passo de simulação. Para tanto, utiliza-se a equação abaixo para determinar um índice entre -0.5 e 1, que é utilizado para interpolar linearmente a tabela

de exaustão (Tabela C.12), apresentada abaixo e no modelo de ABDEL-HAMID e MADNICK (1991).

```
Indice = 2,5 * Dedicacao - 2
```

Índice	Exaustão
-0.5	0.0
-0.4	0.0
-0.3	0.2
-0.2	0.3
-0.1	0.4
0.0	0.5
0.1	0.6
0.2	0.7
0.3	0.8
0.4	0.9
0.5	1.15
0.6	1.3
0.7	1.6
0.8	1.9
0.9	2.2
1.0	2.5

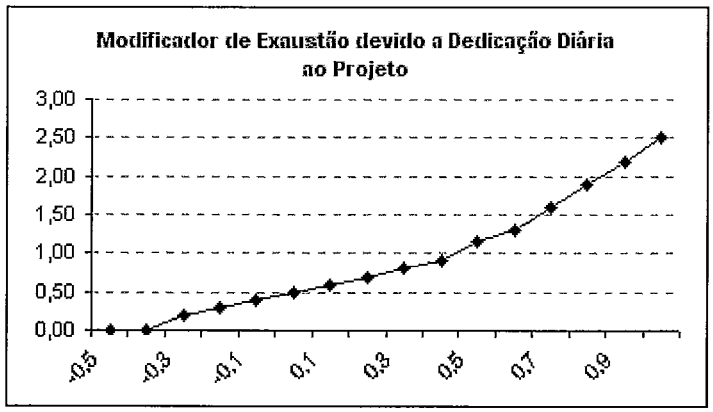

Tabela C.12 – Modificador de exaustão devido ao tempo de trabalho diário

A tabela de exaustão mostra a exaustão acumulada por passo de simulação, de acordo com o número de horas de trabalho diárias no projeto. A exaustão é medida em "pontos de exaustão", uma unidade definida no modelo de ABDEL-HAMID e MADNICK (1991), que também é utilizada para determinar o ponto de máxima exaustão que um desenvolvedor pode atingir.

Pela tabela, percebe-se que pouca exaustão é acumulada quando o desenvolvedor trabalha em ritmo normal, ou seja, 8 horas por dia. Nestas condições, o índice usado na primeira coluna da tabela vale aproximadamente -0,5. A medida que o número de horas de trabalho diário aumenta, mais exaustão é acumulada pelo desenvolvedor. Por exemplo, quando o número de horas trabalhadas por dia se aproxima de 10 horas, o índice encontra-se por volta de 0.0. O acúmulo de exaustão aumenta muito quando o desenvolvedor trabalha por volta de 12 horas por dias. Neste ritmo de trabalho, o índice se aproxima de 1.0.

Quando o acúmulo de exaustão ultrapassa um determinado limite, expresso no modelo de ABDEL-HAMID e MADNICK (1991) como 50 "pontos de exaustão", o desenvolvedor retorna ao ritmo normal de trabalho, 8 horas por dia, independente de

qualquer pressão no sentido de aumentar seu ritmo de trabalho. O limite de 50 "pontos de exaustão" foi determinado com base em um conjunto de entrevistas envolvendo profissionais do MIT e da DEC.

Durante o período em que não trabalha horas extras, o desenvolvedor está descansando. O mesmo ocorre se o desenvolvedor não estiver associado a nenhuma atividade do processo de desenvolvimento que possa ser realizada no período (atividades podem estar bloqueadas devido a dependência de conclusão de outras atividades). O período de descanso para que o desenvolvedor aceite trabalhar horas extras novamente também é determinado pelo modelo de ABDEL-HAMID e MADNICK (1991). Este tempo foi determinado como 20 dias, de acordo com resultado de um conjunto de entrevistas realizadas na DEC.

C.3.5 Atividades do Processo

As atividades do processo de desenvolvimento de software definem o tempo e o custo para a conclusão do projeto. O tempo é contabilizado de acordo com o número de passos de simulação decorridos entre o início da execução da primeira atividade e a conclusão da última atividade do projeto. O custo do projeto é calculado através da soma do tempo de permanência de cada integrante na equipe multiplicado pelo custo da hora de trabalho de cada integrante. As atividades do processo são representadas pelas seguintes características:

- Um conjunto de relações de precedência: uma relação de precedência RP(A1, A2) é uma ligação direcionada entre duas atividades que indica que uma atividade (A1) deve ser concluída antes que a execução de outra atividade (A2) tenha início;
- Um desenvolvedor, que será o responsável pela execução da atividade. A atividade somente entrará em execução quando houver um desenvolvedor associado a ela. O comportamento dos desenvolvedores, explicado na seção C.3.4, afeta a execução das diversas atividades do processo de desenvolvimento de software.

No modelo atual, as atividades do processo são classificadas em atividades de desenvolvimento e atividades de depuração. As atividades de desenvolvimento compreendem a análise de requisitos, o projeto arquitetônico, o projeto detalhado e a codificação. As atividades de depuração compreendem as inspeções, realizadas para a detecção e correção de erros de projeto, e os testes, realizados para detecção e correção

de erros a partir da análise do software executável. Nas próximas subseções, focalizamos, respectivamente, o comportamento das atividades de desenvolvimento, das atividades de inspeção e de testes.

C.3.5.1 Atividades de Desenvolvimento

Dois fatores caracterizam as atividades de desenvolvimento em relação às atividades de depuração: a produção de artefatos de software e a geração de erros.

As atividades de desenvolvimento efetivamente constróem os artefatos de software. Embora as atividades de depuração possam produzir código ou outros artefatos em decorrência da correção de erros, assumimos que estes não são acréscimos consideráveis ao projeto, sendo tratados dentro da própria atividade de depuração.

Durante a construção dos artefatos, as atividades de desenvolvimento produzem erros. Estes erros serão posteriormente detectados e corrigidos pelas atividades de depuração. Embora as próprias atividades de depuração possam produzir novos erros, por razões de simplificação, o modelo atual não contempla estes erros.

Estes dois fatores são determinantes para a definição da dinâmica das atividades de desenvolvimento de software. Para descrever esta dinâmica, devemos considerar a duração esperada das atividades de desenvolvimento se estas fossem realizadas por um desenvolvedor com experiência mediana. Este tempo foi definido como uma distribuição de probabilidade beta-PERT, que foi calculada na seção C.3.3.

Com base na duração esperada de uma atividade, devemos calcular a contribuição de cada passo de simulação ao tempo de execução desta atividade. A contribuição de cada passo é definida de forma que a atividade tenha uma duração fixa, independente do desenvolvedor encarregado por sua realização. A duração das atividades é calculada no início da simulação através de uma amostra da distribuição de probabilidade que a modela.

Se a atividade for realizada por um desenvolvedor de experiência mediana, o trabalho produzido em um passo de simulação será equivalente ao trabalho projetado para ser realizado no mesmo tempo. Entretanto, se a atividade for realizada por um desenvolvedor mais experiente, sua maior produtividade permitirá realizar mais trabalho no mesmo tempo. Assim, o trabalho produzido em um passo de simulação por um desenvolvedor experiente é superior ao produzido por um desenvolvedor mediano no mesmo período. Descrevemos, assim, tanto a produtividade quanto o trabalho relacionado com uma atividade em uma mesma unidade: o tempo.

Assim, a contribuição de um desenvolvedor para o tempo de execução de uma atividade depende de sua produtividade, determinada na seção C.3.4.1. Em cada passo de simulação, reduz-se a contribuição do desenvolvedor do tempo de execução da atividade a que este se encontra associado. Esta redução somente ocorre se a atividade puder ser executada, ou seja, quando suas atividades precedentes estiverem concluídas.

A geração de erros em uma atividade de desenvolvimento pode ser dividida em três etapas: a propagação de erros de atividades anteriores, a regeneração destes erros e a taxa de geração de erros da própria atividade.

A propagação de erros entre atividades do processo de desenvolvimento ocorre quando uma atividade é iniciada. Neste momento, os erros das atividades precedentes são transferidos para a atividade que está se iniciando. Se a conclusão de uma atividade precedente provoca o início de diversas atividades, os erros da atividade precedente são distribuídos por estas atividades linearmente ao tempo de execução de cada uma.

ABDEL-HAMID e MADNICK (1991) classificam os erros gerados durante o desenvolvimento de software em erros ativos e erros passivos. Erros ativos são erros herdados de atividades anteriores que, além de estarem presentes na atividade corrente, provocam novos erros nesta atividade. Por exemplo, um erro na interpretação de um requisito pode gerar diversos erros no projeto do sistema. Este efeito é chamado regeneração de erros. Erros passivos são erros que não se regeneram.

Para calcularmos o número de erros provocados pela regeneração de erros ativos, devemos determinar o percentual de erros ativos dentre os erros recebidos de atividades precedentes. ABDEL-HAMID e MADNICK (1991) apresentam uma formulação para este percentual, indicada na Tabela C.13.

% Atividade	% Erros Ativos
0	0.85
20	0.5
40	0.2
60	0.075
80	0
100	0

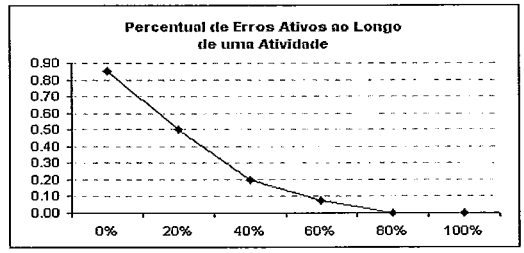

Tabela C.13 – Percentual de erros ativos ao longo de uma atividade

Como pode-se perceber pela tabela e pelo gráfico, o número de erros ativos começa muito alto no início da atividade. Assim, um erro de projeto influencia muitas das primeiras decisões de programação. Um erro de análise influencia as decisões de projeto. A medida que a atividade vai atingindo sua conclusão, os erros herdados

passam a ter menor efeito, uma vez que a atividade está refinando o projeto de acordo com as decisões previamente tomadas.

Determinado o número de erros ativos, passamos a uma análise do número de erros provocados pela regeneração de um erro ativo. Novamente, ABDEL-HAMID e MADNICK (1991) oferecem uma teoria que descreve esta regeneração. Segundo os autores, o número de erros gerados por cada erro ativo depende da densidade de erros ativos por milhar de linha de código (KDSI). A teoria se baseia na constatação de que quanto maior o número de erros presentes em um artefato de software, maior a chance de geração de novos erros.

A Tabela C.14 apresenta a relação entre a densidade de erros ativos, medida tanto em erros/KDSI quanto em erros/FP, e a geração de novos erros. A conversão de linhas de código para pontos por função assume que 1 FP = 100 DSI, conforme média apresentada por JONES (2000).

Erros/KDSI	Erros/FP	Regen./Ativo
0	0	1.0
10	1	1.1
20	2	1.2
30	3	1.325
40	4	1.450
50	5	1.6
60	6	2.0
70	7	2.5
80	8	3.25
90	9	4.35
100	10	6.0

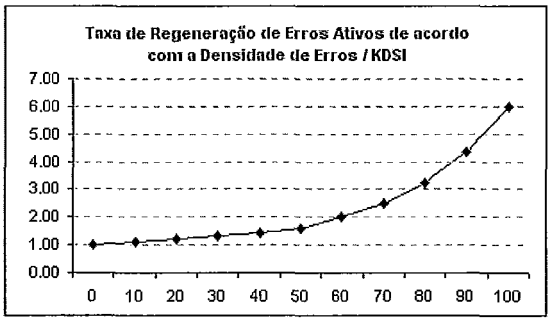

Tabela C.14 – Taxa de regeneração de erros ativos de acordo com a densidade de erros

Conhecendo então a densidade de erros por pontos por função e o número de erros recebidos de atividades precedentes, podemos calcular o número de erros decorrentes da regeneração dos erros ativos herdados. Este número é calculado pela seguinte expressão:

```
Erros = ErrosRecebidos . ErrosFP(d) . Integral(PercAtivos(p), 0, 100)
Erros = ErrosRecebidos . ErrosFP(d) . 0,24
```

A taxa de geração de erros de uma atividade de desenvolvimento depende do desenvolvedor que foi indicado para a realização da atividade. Conforme indicado na seção C.3.4.2, um desenvolvedor possui uma taxa de geração de erros, que depende de sua produtividade e do número de horas de trabalho diário no projeto. O número de

erros produzidos por um desenvolvedor em um passo de simulação é calculado pela seguinte equação.

```
NumeroErros = TaxaErros . ErrosPF . PontosFuncao . Prod / TempoExecucao
```

Na equação acima, **TaxaErros** indica a taxa de geração de erros do desenvolvedor responsável pela atividade, conforme calculada na seção C.3.4.2. **ErrosPF** indica o número médio de erros produzidos por ponto por função de acordo com o tipo da atividade corrente. Este número foi extraído da Tabela C.15, apresentada na página 106 de JONES (2000).

Atividade	Média Erros/FP
Análise	1.00
Projeto	1.25
Codificação	1.75
Testes	0.4
Documentação	0.6

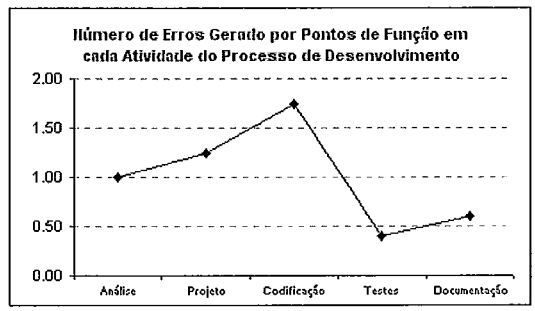

Tabela C.15 – Número de erros gerado por pontos de função por atividade

A Tabela C.15 apresenta o número médio de erros gerados por ponto por função independente da categoria e tamanho dos projetos de software. Refinando suas informações, JONES (2000) apresenta que o número médio de erros gerados por ponto por função nos sistemas de informação com até 100 pontos por função. Este número é estabelecido como 3 erros/PF.

Como o autor não apresenta a distribuição dos erros pelas atividades do processo, assumimos que esta é a mesma da Tabela C.15. Assim, corrigimos linearmente o número de erros gerados em cada atividade (Tabela C.16), obtendo o número de erros de análise, projeto e codificação em sistemas de até 100 pontos por função. Desprezamos os erros gerados nas atividades de teste e documentação devido aos limites do modelo.

Atividade	Média Erros/FP
Análise	0.60
Projeto	0.75
Codificação	1.05

Tabela C.16 – Número de erros por atividade do processo do sistema CtrlPESC

Retornando à equação que descreve a taxa de geração de erros, **PontosFuncao** indica o número de pontos por função da atividade de desenvolvimento. **Prod** indica a produtividade do desenvolvedor responsável pela realização da atividade. **TempoExecucao** indica o tempo de execução projetado para a atividade, originalmente calculado a partir da distribuição beta-PERT.

C.3.5.2 Atividades de Depuração

As atividades de depuração envolvem as atividades de inspeção e de testes. Estas atividades compartilham diversas características comuns, mas diferem em pontos fundamentais para o gerenciamento do projeto.

Conforme apontado por ABDEL-HAMID e MADNICK (1991), as atividades de inspeção são executadas para a retirada de erros cometidos durante as atividades de análise e de projeto. Entretanto, como o efeito destes erros não é imediato nos artefatos de software produzidos, muitos gerentes decidem não realizar atividades de inspeção. Esta decisão geralmente é incorreta, pois conforme visto na seção anterior, os erros cometidos durante a análise e o projeto vão se regenerar nas próximas atividades, aumentando o número de erros a serem removidos durante a atividades de testes.

As atividades de inspeção estão originalmente desativadas no processo de desenvolvimento selecionado para o sistema CtrlPESC. O emulador de projetos assume que o gerente não deseja executar estas atividades. Entretanto, se o gerente decidir ao contrário, existe um comando para ativar uma atividade de inspeção. Existem seis atividades de inspeção previstas, cada qual podendo ser ativada ou desativada independentemente.

Tendo decidido por ativar uma atividade de inspeção, o gerente deve decidir ainda quanto tempo será investido nesta atividade. O emulador de projetos prevê uma duração original de 4 horas para cada atividade de inspeção. Este tempo determina o número de erros que podem ser encontrados pela atividade, conforme especificado em uma equação apresentada mais adiante nesta seção. O gerente pode decidir por aumentar o tempo investido em inspeções, encontrando desta forma um maior número de erros antes de seguir para as próximas atividades de desenvolvimento.

As atividades de testes não são direcionadas pelo tempo de projeto, mas pelo número de erros presentes nos artefatos. Conforme determinado por JONES (2000), as empresas geralmente estabelecem um nível de qualidade a ser atingido, executando testes nos artefatos até que o número de erros encontrados seja compatível com o nível

de qualidade. JONES (2000) indica que o nível médio de qualidade esperada na categoria de sistemas de informação até 100 pontos por função é de aproximadamente 95%. Assim, 95% dos erros presentes nos artefatos construídos devem ser removidos antes do término das atividades de testes. Em seu modelo, ABDEL-HAMID e MADNICK (1991) utilizam uma hipótese mais forte, determinando que a atividade de testes somente será concluída quando todos os erros forem removidos, o que dificilmente ocorre na indústria de software.

O número de erros que atingem cada atividade de testes foi calculado de acordo com as taxas de geração de erros das atividades precedentes, ou seja, das atividades de desenvolvimento. Este número de erros foi calculado segundo as equações apresentadas na seção C.3.5.1.

A qualidade esperada no produto final do projeto determina o número de erros que serão corrigidos durante a atividade de testes. Se a atividade de testes começa com um produto de software contendo X erros e espera-se uma qualidade de 90%, a atividade de testes será executada até que 0,9X erros sejam detectados e corrigidos. O nível médio de qualidade para sistemas de informação de pequeno porte determinado por JONES (2000) como 95%, ou seja, apenas 5% dos erros gerados durante o processo de desenvolvimento permanecem no produto final, será utilizado como objetivo de qualidade do sistema CtrlPESC no emulador de projetos.

Tendo-se determinado as condições de início e término das atividades de depuração, devemos agora calcular o tempo necessário para a detecção e correção de um erro. Este tempo é determinado pelos seguintes fatores, segundo o modelo de ABDEL-HAMID e MADNICK (1991).

```
TempoDeteccao = CustoDeteccaoMedio * ModificadorDensidade
```

Na equação acima, o custo de detecção médio é estabelecido como uma constante. Conforme indicado no modelo de ABDEL-HAMID e MADNICK (1991), este custo é afetado pela densidade de erros do projeto: é mais fácil encontrar um erro em um projeto com muitos erros do que encontrar um erro em um projeto com poucos erros. O efeito provocado pela densidade de erros explica a natureza não linear da atividade de testes, onde geralmente encontram-se muitos erros no início e, a medida que o número de erros se reduz, um menor número de erros é encontrado por unidade de tempo.

A Tabela C.17 descreve a função do modificador do custo de detecção e correção de erros de acordo com a densidade de erros na atividade, determinada em erros por pontos por função.

Erros/FP	Custo
> 1.0	1.0
> 0.7	1.2
> 0.5	2.0
> 0.3	4.0
< 0.3	8.0

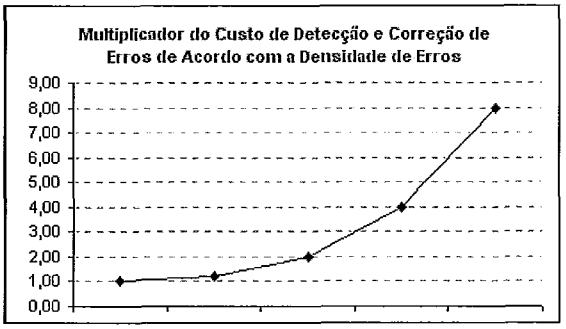

Tabela C.17 – Número de erros gerados por pontos de função em cada atividade

O custo de detecção de um erro foi determinado através de uma conta inversa com os dados apresentados por JONES (2000). Segundo o autor, o tempo médio investido em testes soma em média 59.51% do tempo de projeto, no desenvolvimento de sistemas da mesma categoria do sistema CtrlPESC. Conforme visto anteriormente, estimamos este tempo em cerca de 41 dias para um desenvolvedor com experiência mediana.

Por outro lado, o autor indica a incidência de em média 2.4 erros por ponto por função em sistemas da mesma categoria e a qualidade média dos artefatos após a atividade de testes (95%). Com estes dados podemos estimar o custo médio de detecção e correção de um erro como

```
41,08 = 2,4 . 63,5 . 0,95 . CC
CC = 41,08 / (2,4 . 63,5 . 0,95)
CC = 0,283 dias / erro
```

C.3.6 Pontos de Discussão

Nas seções deste apêndice, apresentamos a dinâmica utilizada no emulador de projetos que acompanha a execução do processo de desenvolvimento do sistema CtrlPESC. Procuramos justificar os números e as equações utilizadas, porém as justificativas nem sempre são irrefutáveis. Na falta de um modelo melhor, procuramos construir o modelo que estava ao nosso alcance. O sucesso no estudo experimental e, talvez, em futuras aplicações do modelo trarão as pesquisas necessárias para sua melhoria.

Dentre as simplificações adotadas, talvez a mais notável esteja relacionada com a regeneração de erros e os números apresentados por JONES (2000). Estes últimos classificam os erros encontrados durante o processo de desenvolvimento de software, mas não informam sobre o momento em que os erros foram determinados. Teriam sido eles detectados por inspeções e corrigidos? Os erros de projeto contemplam os erros de análise regenerados ou apenas os erros de projeto? Existem erros de projeto ativos nos erros contabilizados como programação? Ou teriam os erros sido detectados durante os testes e suas origens traçadas até erros nos modelos de requisitos? Na falta de respostas, adotamos o modelo que nos pareceu mais consistente.

Da mesma incerteza deriva o erro no tempo de detecção e correção de erros: no cálculo deste tempo, não consideramos o efeito da densidade de erros. Para tanto, o lado direito da primeira equação deveria ser multiplicado pela integral entre 2,4 e 0,12 (2,4 . 0,95 = 0,12) da curva do multiplicador do custo de detecção e correção de erros pela densidade.

Outras simplificações foram exercidas. Entretanto, é importante frisar que mais importante que os resultados numéricos expressos pelas teorias é o seu "modus operandi", ou seja, sua capacidade de replicar eventos perceptíveis no mundo real, como a dificuldade de detectar e corrigir os últimos erros ou a multiplicação de erros provenientes das primeiras atividades do processo de desenvolvimento de software.

A dinâmica apresentada também não é completa. Diversas características dinâmicas do processo de desenvolvimento de software não foram consideradas por razões de simplicidade ou por falta de informações confiáveis para a determinação das equações que as definem. Entre estas características, podemos citar os custos de comunicação dentro da equipe, a produção de erros pelas atividades de depuração, entre outros.

Apêndice D - Sintaxe BNF do Metamodelo da Dinâmica de Sistemas

Neste apêndice apresentamos a sintaxe BNF para a definição de modelos de domínio, para a instanciação de modelos a partir do modelo de domínios, para a definição e integração de cenários. Na apresentação da sintaxe BNF, as palavras reservadas da linguagem de modelagem são apresentadas entre aspas, enquanto os símbolos não terminais são apresentados em fonte itálica.

Primeiro, a seção D.1 apresenta a sintaxe BNF para a construção de modelos de domínio. Em seguida, a seção D.2 apresenta a sintaxe para a construção de modelos de cenário. Finalmente, a seção D.3 apresenta a sintaxe para a construção de modelos a partir de modelos de domínio e para integração de cenários nestes modelos.

D.1. Sintaxe BNF para a Construção de Modelos de Domínio

model = "MODEL" *model_name* "{" {*model_item*} "}" ";"
model_name = *identifier*
model_item = (*class* | *relation*) ";"
class = "CLASS" *class_name* "{" {*class_item*} "}"
class_name = *identifier*
class_item = (*property* | *behavior* | "PUBLIC" | "PRIVATE") ";"
property = "PROPERTY" *property_name default_value*
property_name = *identifier*
default_value = *constant*
behavior = *stock* | *rate* | *proc* | *table*
stock = "STOCK" *stock_name expression*
stock_name = *identifier*
rate = "RATE" "(" *affected_stock* ")" *rate_name expression*
affected_stock = *stock_name* | *relation_name* "."*stock_name*
rate_name = *identifier*
proc = "PROC" *proc_name expression*
proc_name = *identifier*
table = "TABLE" *table_name value_list*
table_name = *identifier*
value_list = *constant* { "," *constant* }
relation = *multi_relation* | single_*relation*
multi_relation = "MULTIRELATION" *relation_name source_class* "," *target_ class* [*target_role*]
relation_name = *identifier*
source_class = *identifier*
target_class = *identifier*
target_role = "(" *role_name* ")"
role_name = *identifier*
single_relation = "RELATION" *relation_name source_class* "," *target_class* [*target_role*]
expression = *constant* | *reference* | *expression artitop expression* | *expression relop expression* | *parent_expr* | *function_expr* | *group_expr* | "TIME" | "DT"

reference = *variable_name* | *relation_name* "." *variable_name*
variable_name = *proc_name* | *stock_name* | *rate_name*
artitop = "+" | "-" | "*" | "/" | "^"
relop = "=" | "<>" | ">" | "<" | ">=" | "<="
parent_expr = "(" expression ")"
function_expr = *singfn_expr* | *twofn_expr* | *threefn_expr* | *fourfn_expr* | *multifn_expr*
singfn_expr = *singopfn* "(" expression ")"
singopfn = "EXP" | "LN" | "ROUND" | "NOT"
twofn_expr = *twoopfn* "(" expression "," expression ")"
twoopfn = "NORMAL" | "UNIFORM" | "SMOOTH" | "DELAY3"
threefn_expr = *threeopfn* "(" expression "," expression "," expression ")"
threeopfn = "BETA" | "IF"
fourfn_expr = "LOOKUP" "(" *table_name* "," expression "," expression "," expression ")"
multifn_expr = *multiopfn* "(" params ")"
multiop = "MAX" | "MIN" | "AND" | "OR"
params = *param* { "," | *param* }
param = *expression*
group_expr = *groupop* "(" relation_name "," *variable_name* ")"
groupop = "GROUPSUM" | "GROUPMAX" | "GROUPMIN"
identifier = *alpha* { *alpha* | *digit* }
constant = [*signal*] { *digit* } ["."] { digit }
signal = "-" | "+"
alpha = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "_"
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

D.2. Sintaxe BNF para a Construção de Modelos de Cenário

scenario = "SCENARIO" *scenario_name model_name* "{" { *scenario_item*} "}" ";"
scenario_name = *identifier*
scenario_item = (*connection* | *constraint*) ";"
connection = "CONNECTION" *conn_name class_name* "{" {*conn_item*} "}"
conn_name = *identifier*
conn _item = (*property* | *behavior* | *affect* | "PUBLIC" | "PRIVATE") ";"
property = "PROPERTY" *property_name default_value*
property_name = *identifier*
default_value = *constant*
behavior = *stock* | *rate* | *proc* | *table*
stock = "STOCK" *stock_name expression*
stock_name = *identifier*
rate = "RATE" "(" *affected_stock* ")" *rate_name expression*
affected_stock = *stock_name* | *relation_name* "."*stock_name*
rate_name = *identifier*
proc = "PROC" *proc_name expression*
proc_name = *identifier*
table = "TABLE" *table_name value_list*
table_name = *identifier*
value_list = *constant* { "," *constant* }
affect = "AFFECT" *variable_name* expression

affect_name = *identifier*
constraint = ""CONSTRAINT" *const_origin* "," *const_target*
const_origin = *conn_name* ["." *relation_name*]
const_target = *scenario_name* "." *conn_name*
relation_name = *identifier*
expression = *constant* | *reference* | *expression artitop expression* | *expression relop expression* | *parent_expr* | *function_expr* | *group_expr* | "TIME" | "DT"
reference = *variable_name* | *relation_name* "." *variable_name*
variable_name = *proc_name* | *stock_name* | *rate_name*
artitop = "+" | "-" | "*" | "/" | "^"
relop = "=" | "<>" | ">" | "<" | ">=" | "<="
parent_expr = "(" expression ")"
function_expr = *singfn_expr* | *twofn_expr* | *threefn_expr* | *fourfn_expr* | *multifn_expr*
singfn_expr = *singopfn* "(" expression ")"
singopfn = "EXP" | "LN" | "ROUND" | "NOT"
twofn_expr = *twoopfn* "(" expression "," expression ")"
twoopfn = "NORMAL" | "UNIFORM" | "SMOOTH" | "DELAY3"
threefn_expr = *threeopfn* "(" expression "," expression "," expression ")"
threeopfn = "BETA" | "IF"
fourfn_expr = "LOOKUP" "(" *table_name* "," expression "," expression "," expression ")"
multifn_expr = *multiopfn* "(" params ")"
multiop = "MAX" | "MIN" | "AND" | "OR"
params = *param* { "," | *param* }
param = *expression*
group_expr = *groupop* "(" relation_name "," *variable_name* ")"
groupop = "GROUPSUM" | "GROUPMAX" | "GROUPMIN"
identifier = *alpha* { *alpha* | *digit* }
constant = [*signal*] { *digit* } ["."] { digit }
signal = "-" | "+"
alpha = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "_"
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

D.3. Sintaxe BNF para a Construção de Modelos e Integração de Cenários

instmodel = "DEFINE" *instmod_name model_name* "{" { *instmod_item*} "}" ";"
instmod_name = *identifier*
instmod _item = (*instance* | *activation*) ";"
instance = *instance_name* "=" *class_name* { *instance_item* ";" }
instance_name = *identifier*
instance_item = *instance_prop* | *instance_link*
instance_prop = "SET" *property_name* "=" *property_value*
property_value = *constant*
instance_link = "LINK" *relation_name links*
links = *instance_name* { "," *instance_name* }
activation = "ACTIVATE" *scenario_name* { *instance_conn* ";" }
instance_conn = "CONNECT" *connection_name instance_name*

Apêndice E - As Ferramentas de Simulação

Três ferramentas de simulação foram desenvolvidas para oferecer suporte ao gerenciamento de projetos baseado em cenários e permitir a realização do estudo experimental descrito no Capítulo 5. A ferramenta *Illium* é um simulador de modelos escritos com construtores tradicionais da Dinâmica de Sistemas. A ferramenta *Hector* é um simulador que recebe modelos escritos segundo o metamodelo da Dinâmica de Sistemas e traduz estes modelos para construtores tradicionais. O emulador *Manager Master* simula o comportamento de um projeto de acordo com as decisões tomadas por seu gerente.

Neste apêndice, descrevemos brevemente as três ferramentas. A seção E.1 apresenta a ferramenta *Illium*, assim como sua linguagem de modelagem, baseada nos construtores da Dinâmica de Sistemas. A seção E.2 apresenta a ferramenta *Hector*, cujos modelos seguem a sintaxe da Dinâmica de Sistemas, apresentada no Capítulo 4. A seção E.3 apresenta o emulador de projetos *Manager Master*.

E.1 O Simulador de Modelos da Dinâmica de Sistemas

Illium é uma ferramenta que permite a construção e avaliação de modelos dinâmicos de projetos de software, utilizando a técnica de Dinâmica de Sistemas. *Illium* define uma linguagem para a representação destes modelos e possui diversos mecanismos de simulação para a avaliação de seu comportamento. *Illium* tem como principal objetivo oferecer diversos mecanismos de simulação distintos, não focalizando a construção e a representação gráfica de modelos de projeto, funções contempladas por outras ferramentas, como Cycles (GARCIA e MASIERO, 1999).

Para permitir a criação de modelos utilizando a Dinâmica de Sistemas, a representação da incerteza nestes modelos e a extração de resultados gerados por eles, definimos uma linguagem e um simulador para construção de modelos de Dinâmica de Sistemas. O protótipo implementado do simulador é capaz de carregar modelos baseados nos construtores da Dinâmica de Sistemas, simulando e apresentando o comportamento destes modelos. Ele é utilizado para a simulação dos modelos compilados, gerados a partir de modelos instanciados pelo metamodelo da Dinâmica de Sistemas apresentado no Capítulo 4.

A interface com o usuário do simulador foi implementada em Delphi. O código de interpretação de modelos e simulação foi desenvolvido em C, tendo sido compilado na forma de uma biblioteca de linquedição dinâmica, que é utilizada pela interface com o usuário do simulador.

E.1.1 A Linguagem de Definição de Modelos

Na ferramenta *Illium*, adotamos uma descrição textual dos modelos dinâmicos, utilizando uma linguagem desenvolvida especificamente para esta tarefa. Optamos por uma descrição textual, em detrimento de uma notação gráfica, porque acreditamos que a notação gráfica da dinâmica de sistemas não é adequada para a descrição de grandes modelos, como o modelo de ABDEL-HAMID e MADNICK (1991), devido a sua baixa granularidade semântica. Além disso, a ferramenta *Illium* faz parte de um projeto onde esperamos gerar modelos dinâmicos de projetos de software automaticamente, a partir de uma notação de mais alto nível, que será convertida para a linguagem de representação de modelos da ferramenta.

model	= { *constructor* ";" }
constructor	= (*stock* | *rate* | *process* | *table*)
stock	= "STOCK" *stock_name expression*
rate	= "RATE" "(" *source* "," *target* ")" *rate_name expression*
source	= (*stock_name* | "SOURCE")
target	= (*stock_name* | "SINK")
process	= "PROC" *process_name expression*
table	= "TABLE" *table_name values*

Tabela E.1 – Sintaxe da linguagem de descrição de modelos dinâmicos

A Tabela E.1 apresenta a sintaxe BNF simplificada da linguagem de descrição de modelos dinâmicos definida na ferramenta *Illium*. As palavras reservadas da linguagem de modelagem são apresentadas entre aspas, enquanto os símbolos não terminais são apresentados em fonte itálica.

O comando **STOCK** permite a criação de um repositório, indicando seu nome e uma expressão, que determina o nível inicial do repositório. O nome de um elemento - repositório, taxa, processo ou tabela - deve ser único dentre os elementos do modelo, sendo utilizado como referência para seu valor nas expressões.

O comando **RATE** permite a criação de uma taxa, indicando seu repositório de origem, seu repositório destino, seu nome e sua expressão. Esta expressão indica a variação instantânea dos níveis dos repositórios associados à taxa. Em cada passo de

simulação, o nível do repositório de origem será decrescido do valor resultante da expressão da taxa. De forma similar, o nível do repositório destino será acrescido deste montante.

O repositório origem de uma taxa pode ser um provedor universal, indicado pela palavra reservada **SOURCE**. Quando uma taxa está associada a um provedor universal, nenhum repositório do modelo terá seu nível debitado do valor da taxa. Assume-se que existe um repositório infinito, exógeno ao modelo, que fornece o montante transferido para o repositório destino. Da mesma forma, o repositório destino de uma taxa pode ser um receptor universal, indicado pela palavra reservada **SINK**. Neste caso, o receptor universal indica que nenhum repositório terá seu nível elevado do montante transferido pela taxa. Assume-se a existência de um repositório de capacidade infinita, também externo ao modelo, que recebe o montante transferido pela taxa.

O comando **PROC** permite a definição de um processo, indicando seu nome e sua expressão. A expressão de um processo calcula um valor intermediário, que pode ser utilizado em expressões de outros processos ou taxas do modelo.

As expressões, utilizadas na especificação dos repositórios, taxas e processos, podem conter operadores algébricos (soma, subtração, produto, divisão e potência), operadores lógicos (e, ou, negação, teste de condição), operadores relacionais (maior, menor, maior ou igual, menor ou igual, igual e diferente) e funções (mínimo, máximo, logaritmo e exponencial). As palavras reservadas **TIME** e **DT**, também podem ser utilizadas nas expressões, indicando, respectivamente, o tempo decorrido desde o início da simulação e a variação deste tempo em cada passo de simulação.

As expressões também podem utilizar valores definidos em outros repositórios, taxas ou processos, além de definir os parâmetros do modelo como funções de distribuição de probabilidade normais, beta ou uniformes não correlacionadas. Esta definição permite que o simulador realize simulações de Monte Carlo sobre o modelo, apresentando graficamente as distribuições de probabilidade de seus resultados.

O comando **TABLE** permite a definição de uma tabela, indicando seu nome e seus valores, que são separados por vírgulas. A função **LOOKUP**, disponível para as expressões, realiza uma interpolação linear entre os elementos de uma tabela, tratando esta como uma função discreta. Os limites da abscissa dos valores da tabela e o valor da posição que se deseja interpolar são fornecidos na função **LOOKUP**.

Finalmente, O comando de controle **SPEC DT** especifica o tempo decorrido entre dois intervalos de simulação. Esta construção somente pode figurar uma vez em

cada modelo. A Tabela E.2 apresenta um trecho do modelo de ABDEL-HAMID e MADNICK (1991) escrito com a linguagem de modelagem da ferramenta *Illium*.

```
# New Workforce
STOCK WFNEW 0;

# Hiring Rate (People / Day)
RATE (SOURCE, WFNEW) HIRERT MAX (0, WFGAP / HIREDY);

# Hiring Delay (Day)
PROC HIREDY 40;

# Workforce Gap (People)
PROC WFGAP WFS - TOTWF;

# New Employees Transfer Rate Out (People / Day)
RATE (WFNEW, SINK) NEWTRR MIN(TRNFRT, WFNEW / DT);

# Transfer Rate of People Out of Project (People / Day)
PROC TRNFRT MAX (0, - WFGAP / TRNSDY);

# Time Delay to Transfer People Out (Days)
PROC TRNSDY 10;

# Assimilation Rate for New Employees (People / Day)
RATE (WFNEW, WFEXP) ASIMRT if (time < 860, WFNEW / ASIMDY, 0);
```

Tabela E.2 - Trecho do modelo de Abdel-Hamid e Madnick descrito na linguagem do simulador

E.1.2 O Simulador

A Figura E.1 apresenta a janela principal da ferramenta *Illium*. Nesta janela, o usuário pode carregar, editar e salvar o modelo na linguagem descrita na seção E.1.1. Os botões na barra de ferramentas e os comandos do menu principal da janela permitem o acesso aos diversos mecanismos de análise disponíveis no simulador.

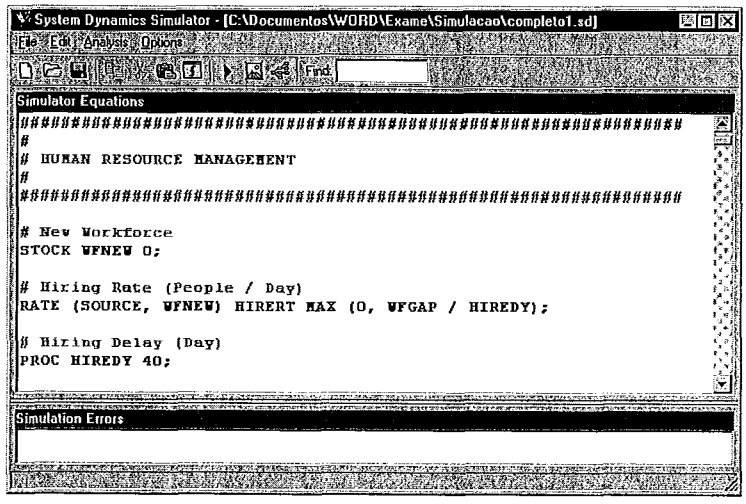

Figura E.1 - Tela principal do simulador com um trecho do modelo de Abdel-Hamid e Madnick

O primeiro mecanismo de análise disponível no simulador é o gráfico de análise no tempo. Este gráfico permite que o usuário acompanhe a evolução de uma ou mais variáveis ao longo do tempo de simulação. Ele é apresentado na Figura E.2.

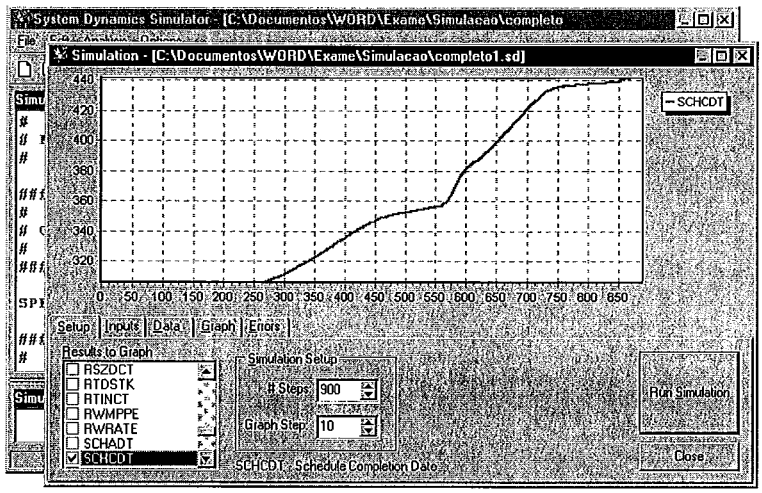

Figura E.2 – Gráfico de análise do valor de variáveis no tempo

O gráfico de análise no tempo permite que o usuário selecione um subconjunto das variáveis componentes do modelo para apresentação da evolução de seu valor no tempo. As variáveis são apresentadas na lista à esquerda da janela. O comentário associado à variável, extraído automaticamente pelo simulador, é apresentado ao lado da lista de variáveis, facilitando a seleção do usuário.

Também ao lado da lista de variáveis do modelo, encontram-se os controles de configuração da simulação. Estes controles permitem que o usuário indique o horizonte de simulação, ou seja, o número de intervalos que serão simulados, e o intervalo de apresentação do gráfico, expresso em número de pontos calculados.

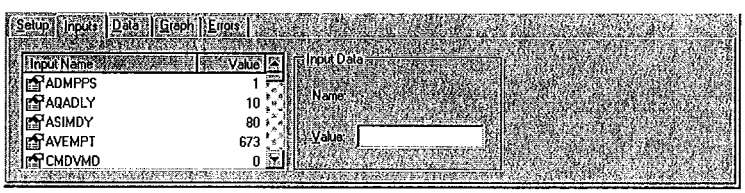

Figura E.3 – O painel de configuração de cenários do gráfico de análise no tempo

A análise no tempo também pode ser realizada mediante um cenário. Em um cenário, o usuário determina explicitamente o valor de uma ou mais variáveis do modelo. Este mecanismo de análise oferece ao usuário um panorama da evolução do

modelo em cenários distintos do planejado originalmente. A Figura E.3 apresenta o painel de configuração de cenários

Além do gráfico de evolução no tempo, a ferramenta *Illium* pode apresentar ao usuário os valores das variáveis selecionadas. O sistema exporta estes valores para a área de transferência (*clipboard*) do Windows, permitindo que eles sejam analisados em outros programas. A Figura E.4 apresenta o painel de análise e exportação de dados.

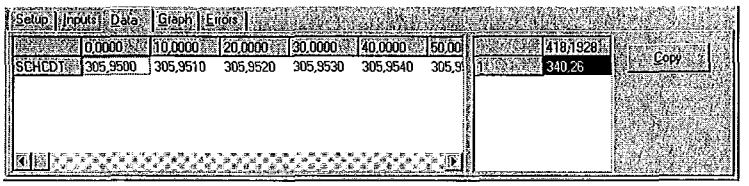

Figura E.4 – O painel de análise e exportação de dados

Os painéis de cenários e de exportação de dados apresentados para o gráfico de evolução no tempo também estão disponíveis para um segundo mecanismo de análise oferecido pelo simulador: o gráfico de simulação de simulação de Monte Carlo. Este mecanismo de análise apresenta graficamente a distribuição de um resultado do modelo ao longo de um determinado número de simulações.

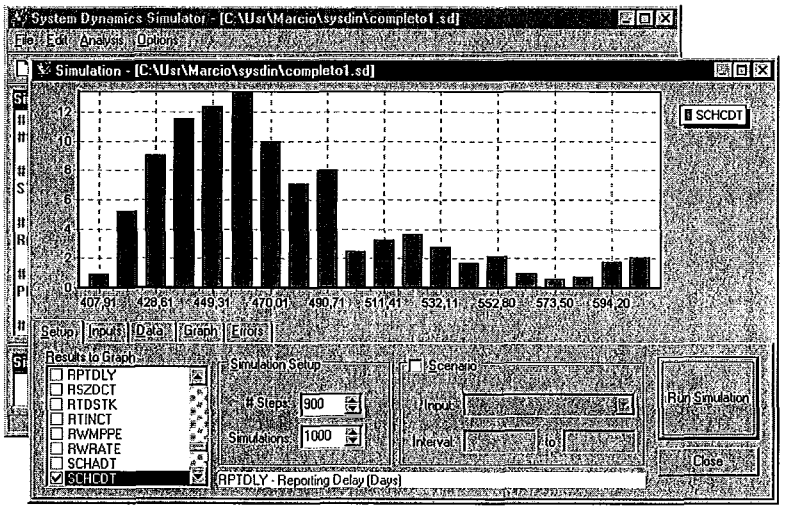

Figura E.5 - Distribuição de probabilidade da data de fim de projeto

A Figura E.5 apresenta o resultado de 1000 simulações de Monte Carlo sobre o modelo de ABDEL-HAMID e MADNICK (1991). O gráfico apresenta os diversos valores assumidos pela data de conclusão do projeto, expressa em dias, ao longo das 1000 simulações. O modelo original de Abdel-Hamid e Madnick foi modificado para expressar o grau de subestimação do tamanho do projeto como uma distribuição normal com média 33% e desvio padrão de 10%. Esta representação implica em 68% de chance

do projeto estar subestimado entre 43% e 23%. Os valores apresentados no gráfico estão organizados em um histograma, que, após sua normalização, representa a distribuição de probabilidade da data de conclusão do projeto, dadas as premissas do modelo. Os valores da distribuição de probabilidade também podem ser apresentados na forma de um histograma cumulativo.

A ferramenta *Illium*, assim como sua documentação, está disponível na URL http://www.cos.ufrj.br/~marcio/Illium.html.

E.2 O Simulador Baseado no Metamodelo da Dinâmica de Sistemas

A ferramenta *Illium*, apresentada na seção anterior, simula modelos baseados nos construtores da Dinâmica de Sistemas. Conforme visto no Capítulo 4, o metamodelo da Dinâmica de Sistemas permite a definição de modelos em um grau de abstração mais alto, traduzindo estes modelos para os construtores tradicionais da Dinâmica de Sistemas posteriormente.

A ferramenta *Hector* é responsável por esta tradução. Esta ferramenta carrega um modelo construído segundo o processo apresentado no Capítulo 4, gerando sua versão compilada. Esta versão pode, por sua vez, ser carregada e simulada na ferramenta *Illium*. *Hector* foi implementada em Delphi, utilizando uma biblioteca de linquedição dinâmica escrita em C. A biblioteca é responsável pela compilação dos modelos representados através do metamodelo da Dinâmica de Sistemas, enquanto o código Delphi atua na interface com o usuário.

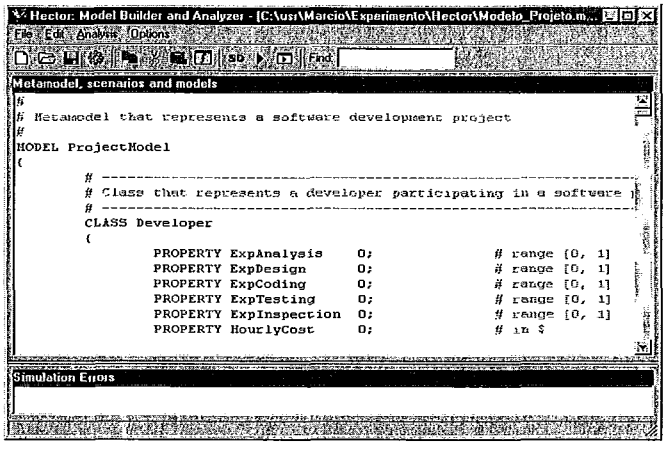

Figura E.6 – Tela principal da ferramenta *Hector*

A Figura E.6 apresenta a tela principal da ferramenta *Hector*. Ela permite a leitura, edição e salvamento de um modelo, a configuração do compilador (que indica o

número máximo de iterações de otimização realizadas sobre um modelo compilado), a compilação de um modelo e um mecanismo simplificado de simulação, equivalente ao gráfico de variação no tempo da ferramenta *Illium*.

A ferramenta *Hector*, assim como sua documentação, está disponível na URL http://www.cos.ufrj.br/~marcio/Hector.html.

E.3 O Emulador de Projetos Manager Master

O emulador de projetos *Manager Master* foi desenvolvido para permitir a realização do estudo experimental para análise de viabilidade das técnicas de modelagem e simulação. O emulador contém o modelo do projeto CtrlPESC, apresentado na seção 5.3, e permite que seu usuário atue como gerente do projeto. A Figura E.7 apresenta a tela principal do emulador de projetos.

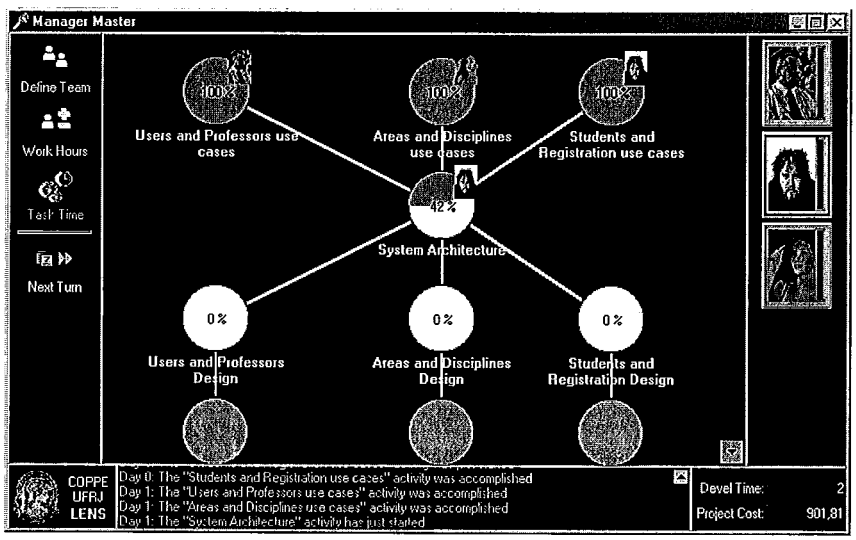

Figura E.7 – Tela principal da ferramenta *Manager Master*

A tela principal é dividida em cinco painéis. O painel de comandos, apresentado na lateral esquerda da janela, indica as decisões que o usuário, enquanto gerente, pode tomar durante o desenvolvimento do projeto. Estas decisões incluem a definição da equipe do projeto, selecionando os seus desenvolvedores dentre um conjunto de candidatos, a determinação do número de horas de trabalho diário dos desenvolvedores e do tempo investido em atividades de controle de qualidade, como inspeções. Uma última decisão consiste na determinação de que desenvolvedor realizará cada atividade do projeto. Esta decisão é realizada arrastando-se a figura do desenvolvedor do painel da equipe (na lateral direita da janela) para o painel de atividades (no centro da janela).

O painel da equipe apresenta as características dos desenvolvedores selecionados pelo gerente para participarem do projeto. O painel de atividades apresenta as atividades que devem ser realizadas para a conclusão do projeto, suas relações de dependência e os desenvolvedores responsáveis por cada atividade.

Finalmente, o rodapé da janela contém os dois últimos painéis do emulador. No centro do rodapé, o painel de mensagens apresenta informações sobre fatos relevantes sobre o projeto, tais como a conclusão de uma atividade ou a detecção e posterior correção de erros. À direita do painel de mensagens, o painel de estado apresenta o tempo de desenvolvimento do projeto e seu custo até o momento.

O emulador *Manager Master*, assim como sua documentação, está disponível na URL http://www.cos.ufrj.br/~marcio/mmaster.

Apêndice F - Questionários Utilizados no Estudo Experimental

Neste apêndice apresentamos os dois questionários utilizados no estudo experimental descrito no Capítulo 5. O questionário para caracterização dos participantes (Q1) foi preenchido por todos os participantes, enquanto o questionário para avaliação das técnicas propostas (Q2) foi preenchido apenas pelos participantes que aplicaram as técnicas de integração e simulação de modelos de projeto e cenário.

Q1	Questionário para Estudo sobre as Técnicas de Integração e Simulação de Modelos de Projeto e Cenário	ID

Nas perguntas abaixo, quando duas ou mais alternativas forem válidas, marque a alternativa que mais se aplica a seu caso.

Determine seu nível de formação.

[ ] Graduando
[ ] Mestrando
[ ] Doutorando
[ ] Graduado
[ ] Mestre
[ ] Doutor

Determine a sua experiência no desenvolvimento de software ?

[ ] Eu nunca desenvolvi software
[ ] Eu desenvolvi software para uso pessoal
[ ] Eu desenvolvi software como parte de trabalhos de curso (____ projetos)
[ ] Eu desenvolvi software em ambiente industrial (____ anos)

Determine a sua experiência na gerência de projetos de desenvolvimento de software ?

[ ] Eu nunca fui gerente de projetos de software
[ ] Eu fui líder de grupos de trabalho em projetos de curso
[ ] Eu fui líder em poucos (≤ 3) projetos na indústria (___ anos)
[ ] Eu fui líder em muitos (> 3) projetos na indústria (___ anos)

Por favor, indique sua experiência nas questões abaixo, assinalando uma das alternativas da escala de 1 a 5, segundo a tabela abaixo:

1 – nenhuma
2 – estudo em cursos ou livros
3 – praticado em projetos de curso
4 – praticado na indústria
5 – praticado em diversos (> 3) projetos da indústria

Experiência em desenvolvimento de software orientado a objetos	1	2	3	4	5
Experiência na definição de redes de tarefas (redes de PERT)	1	2	3	4	5
Experiência em estimativas de tempo e custo de software	1	2	3	4	5
Experiência na formação e gerenciamento de grupos de trabalho	1	2	3	4	5

Q2	Dificuldades na Utilização das Técnicas de Integração e Simulação de Modelos de Projeto e Cenário	ID

Você considera que as técnicas de integração e simulação de modelos de projeto e cenário tiveram alguma utilidade no gerenciamento do projeto proposto? Você utilizaria estas técnicas novamente? Justifique sua resposta.

Você considera que os cenários apresentados tiveram utilidade para o projeto? Justifique.

Você considera que o treinamento aplicado para o uso das técnicas foi suficiente? Sugestões?

Você considera que existiu alguma dificuldade na interpretação dos resultados e gráficos apresentados pelas ferramentas que suportam as técnicas? Justifique sua resposta.

As informações disponibilizadas sobre o sistema a ser gerenciado, as técnicas propostas e sua aplicação foram de fácil compreensão? Você sentiu falta de alguma informação nesta descrição?

Apêndice G - Exemplos de Arquétipos de Risco

Neste apêndice apresentamos alguns arquétipos de risco identificados no projeto de implantação de um pacote de gerenciamento de riscos para o mercado financeiro em uma empresa brasileira do setor petrolífero. Os arquétipos apresentados a seguir atendem ao modelo definido no Capítulo 6, com exceção da presença de modelos de cenários, que não foram desenvolvidos durante o projeto.

IDENTIFICAÇÃO

1. **Nome**: Falha de proteção de fluxo

2. **Sinônimos**: *Hedge* incorreto; dicotomia do *hedge*

3. **Problema potencial**: o mapeamento de instrumentos projetados para a proteção dos fluxos operacionais ou de dívida futuros não se mostra eficaz nesta proteção

4. **Efeitos**: invalidação da utilização do sistema com relação a estes fluxos (pois os instrumentos não medem corretamente o risco) ou aumento no tempo de desenvolvimento (devido a redefinição dos instrumentos)

MECANISMOS DE IDENTIFICAÇÃO

1. **Contexto**: projetos com instrumentos inovadores, projetos com instrumentos não negociados em mercado aberto, projetos com instrumentos desenvolvidos particularmente para o cliente, fluxos de *commodities* que obedecem a processos estocásticos não convencionais.

2. **Questionário**:
   - Avalie o conhecimento da equipe em relação aos instrumentos
   - Avalie se os *commodities* referentes aos instrumentos são amplamente negociados em mercado
   - Avalie se o processo estocástico do instrumento é compatível com o mapeamento do *hedge*

3. **Casos conhecidos**:
   - Projeto Petrobras: processo estocástico de evolução do preço do petróleo e dos derivados

ESTRATÉGIAS DE CONTENÇÃO

1. **Plano**: Desenvolvimento de protótipos
2. **Condição**: Instrumentos não são amplamente conhecidos, mapeamento não claramente definido
3. **Efeitos**: Aumento de cronograma (devido à construção dos protótipos)

1. **Plano**: Análise de fórmulas de mapeamento
2. **Condição**: Mapeamento não claramente definido, *commodities* não convencionais
3. **Efeitos**: Aumento de cronograma (tempo de estudo)

1. **Plano**: Adaptação dos processos estocásticos
2. **Condição**: Commodities não convencionais
3. **Efeitos**: Aumento de cronograma (tempo de estudo), redução de qualidade (processos não realistas)

ESTRATÉGIAS DE CONTINGÊNCIA

1. **Plano**: Aumento do cronograma
2. **Efeitos**: Permitir a construção de protótipos e o estudo de modelos

1. **Plano**: Remoção dos instrumentos
2. **Efeitos**: Redução da usabilidade do sistema

1. **Plano**: Utilizar simulações de Monte Carlo
2. **Efeitos**: Torna desnecessários os modelos analíticos de mapeamento, aumento do tempo de cálculo

ARQUÉTIPOS RELACIONADOS

1. **Riscos Induzidos**: Necessidade de Simulação, Atraso de Cronograma, Aumento de Custo, Reputação
2. **Riscos Similares**:

IDENTIFICAÇÃO

1. **Nome**: Necessidade de simulação

2. **Sinônimos**: Mapeamento por Monte Carlo

3. **Problema potencial**: a inexistência de fórmula analítica para o mapeamento de instrumentos complexos pode levar a necessidade de mapeamento por simulações de Monte Carlo

4. **Efeitos**: redução de qualidade do sistema (aumento do tempo de execução - no processo de mapeamento) e aumento no tempo de desenvolvimento (desenvolvimento da máquina de mapeamento por SMC)

MECANISMOS DE IDENTIFICAÇÃO

1. **Contexto**: projetos com instrumentos inovadores, projetos com instrumentos não negociados em mercado aberto, projetos com instrumentos desenvolvidos particularmente para o cliente, projetos com instrumentos asiáticos, fluxos de *commodities* que obedecem a processos estocásticos não convencionais.

2. **Questionário**:
   - Avalie o conhecimento da equipe em relação aos instrumentos
   - Avalie a existência de fórmula analítica para todos os instrumentos

3. **Casos conhecidos**:
   - Projeto Petrobras: opções e *swaps* asiáticos de dólar médio

ESTRATÉGIAS DE CONTENÇÃO

1. **Plano**: Análise de fórmulas de mapeamento
2. **Condição**: Mapeamento não claramente definido, *commodities* não convencionais
3. **Efeitos**: Aumento de cronograma (tempo de estudo), reduzir especificação de performance do mapeamento

1. **Plano**: Adaptação dos processos estocásticos
2. **Condição**: Fórmulas de mapeamento desconhecidas, commodities não convencionais
3. **Efeitos**: Aumento de cronograma (tempo de estudo), redução de qualidade (processos não realistas)

ESTRATÉGIAS DE CONTINGÊNCIA

1. **Plano**: Aumento do cronograma
2. **Efeitos**: Permitir a construção de protótipos e o estudo de modelos

1. **Plano**: Desenvolvimento de uma máquina de SMC com múltiplos processos estocásticos
2. **Efeitos**: Aumento do tempo de mapeamento, aumento da flexibilidade (qualidade) do sistema

1. **Plano**: Remover determinados instrumentos
2. **Efeitos**: Redução da qualidade do projeto, incapacitando-o a tratar determinados instrumentos

ARQUÉTIPOS RELACIONADOS

1. **Riscos Induzidos**: Aumento de Cronograma, Aumento de Custo, Reputação
2. **Riscos Similares**:

Identificação

1. **Nome**: Características de mercado incorporadas

2. **Sinônimos**: Instrumentos com características de mercado

3. **Problema potencial**: a inclusão de dados de mercado junto às características de instrumentos de aplicação pode acarretar a replicação destes dados e possíveis inconsistências no cálculo de risco;

4. **Efeitos**: redução da qualidade do sistema (transferência para os usuários da manutenção da consistência entre os dados de mercado)

Mecanismos de Identificação

1. **Contexto**: praticamente qualquer projeto, mas especialmente projetos com instrumentos inovadores, projetos com instrumentos não negociados em mercado aberto, projetos com instrumentos desenvolvidos particularmente para o cliente.

2. **Questionário**:
   - Avalie o conhecimento da equipe em relação aos instrumentos
   - Avalie a existência de dados de mercado entre os instrumentos

3. **Casos conhecidos**:
   - Projeto CSN: PU negociado em contratos de DI futuro, swaps DI pré e CDI acumulado
   - Projeto Petrobras: os mesmos instrumentos

Estratégias de Contenção

1. **Plano**: Criação de instrumentos de mercado para captura dos dados de mercado dos instrumentos
2. **Condição**: qualquer condição
3. **Efeitos**: Aumento de cronograma (mais código), reduzir performance do mapeamento (interpolações)

1. **Plano**: Acesso do mapeamento às séries históricas
2. **Condição**: qualquer condição
3. **Efeitos**: Aumento de cronograma (mais código e grande alteração no sistema)

Estratégias de Contingência

1. **Plano**: Documentação extensa no manual do usuário do sistema
2. **Efeitos**: Aumento do cronograma (aprimoramento do manual) e redução da qualidade (pela exigência de memória do usuário)

Arquétipos Relacionados

1. **Riscos Induzidos**: Aumento de Cronograma, Aumento de Custo, Aumento do Tempo de Teste
2. **Riscos Similares**:

Identificação

1. **Nome**: Aparecimento de novos instrumentos

2. **Sinônimos**: Novos mapeamentos

3. **Problema potencial**: a inclusão de novos instrumentos em fases tardias do processo de desenvolvimento acarreta alterações em diversos artefatos, como o arquivo de instrumentos, o código de mapeamento, os manuais e, eventualmente, a lista de fatores, grupos e séries históricas.

4. **Efeitos**: aumento do tempo de desenvolvimento do projeto

Mecanismos de Identificação

1. **Contexto**: praticamente qualquer projeto, mas especialmente empresas em que não se conhecem os mercados que negociam instrumentos de hedge. O problema parece menor em empresas governamentais, cujo potencial para negociação é limitado por regras do governo.

2. **Questionário**:
   - Avalie a capacidade da empresa em negociar instrumentos em mercados menos conhecidos
   - Avalie o conhecimento da equipe nos mercados de derivativos do setor da empresa
   - Avalie se os instrumentos podem ser decompostos em instrumentos de mercado conhecidos

3. **Casos conhecidos**:
   - Projeto Petrobras: *swaps* de dólar médio
   - Projeto CVRD: opções com barreiras

Estratégias de Contenção

1. **Plano**: Análise dos novos instrumentos
2. **Condição**: a equipe não conhece os mercados ou derivativos de *hedge*
3. **Efeitos**: Aumento de cronograma (estudo, implementação e documentação)

1. **Plano**: Decomposição dos instrumentos
2. **Condição**: instrumentos podem ser decompostos
3. **Efeitos**: Aumento de cronograma (estudo, implementação e documentação)

Estratégias de Contingência

1. **Plano**: Remoção dos instrumentos
2. **Efeitos**: Redução da qualidade (o sistema não captura o risco associado aos instrumentos)

Arquétipos Relacionados

1. **Riscos Induzidos**: Aumento de Cronograma, Aumento de Custo, Reputação
2. **Riscos Similares**:

Identificação

1. **Nome**: Demora no processo de validação

2. **Sinônimos**: Indisponibilidade para validação

3. **Problema potencial**: os usuários do sistema não são capazes ou demoram a validá-lo, por pouco conhecimento do domínio do sistema, por falta de dados reais de validação ou por falta de motivação.

4. **Efeitos**: aumento do tempo de treinamento, demora na homologação do projeto

Mecanismos de Identificação

1. **Contexto**: praticamente qualquer projeto, mas especialmente empresas sem profissionais especializados em engenharia financeira ou sem mesas de operação ativas.

2. **Questionário**:
   - Avalie o conhecimento dos usuários em engenharia financeira
   - Avalie a disponibilidade de dados reais para avaliação
   - Avalie a motivação dos usuários na utilização do sistema

3. **Casos conhecidos**:
   - Projeto Petrobras: demora na aquisição de dados reais

Estratégias de Contenção

1. **Plano**: Requisição dos dados reais desde o início do projeto
2. **Condição**: dados reais dispersos por diversos departamentos ou indisponíveis
3. **Efeitos**: Aumento de cronograma (captura, validação e agregação dos dados reais) ou aumento do custo (outros profissionais realizando o trabalho em paralelo)

1. **Plano**: Treinamento dos usuários
2. **Condição**: falta de conhecimento e motivação
3. **Efeitos**: Aumento de cronograma (tempo de treinamento), aumento de custo (treinamento externo)

1. **Plano**: Palestras sobre a importância do sistema
2. **Condição**: falta de motivação
3. **Efeitos**: Aumento de cronograma (tempo de preparação das palestras)

Estratégias de Contingência

1. **Plano**: Espera pela disponibilidade dos dados
2. **Efeitos**: Aumento do tempo de projeto

1. **Plano**: Cobrança pelo tempo de espera por validação
2. **Efeitos**: Redução da qualidade do projeto (relações possivelmente abaladas com o cliente)

Arquétipos Relacionados

1. **Riscos Induzidos**: Aumento de Cronograma, Aumento de Custo, Reputação
2. **Riscos Similares**: