Relatório Técnico

Modelagem de contexto sobre o domínio de processos de desenvolvimento de software

Vanessa Tavares Nunes
(vanunes@cos.ufrj.br)

Andréa Magalhães Magdaleno
(andrea@cos.ufrj.br)

Cláudia Maria Lima Werner
(werner@cos.ufrj.br)

COPPE/UFRJ

Rio de Janeiro, Maio de 2010

Modelagem de contexto sobre o domínio de processos de desenvolvimento de software

Vanessa Tavares Nunes

Andréa Magalhães Magdaleno

Cláudia Maria Lima Werner

[1]Programa de Engenharia de Sistemas e Computação (PESC) – COPPE/UFRJ
Caixa Postal 68.511 – 21945-970 – Rio de Janeiro – RJ – Brasil

{vanunes, andrea, werner}@cos.ufrj.br

RESUMO

Os modelos de desenvolvimento de software orientado ao planejamento, ágil e livre têm características em comum e diferenças significativas, mas nenhum deles é efetivo para todos os projetos. Apesar da maioria das propostas existentes para a conciliação de processos de desenvolvimento de software envolver uma combinação rígida das práticas dos diferentes modelos, este trabalho de pesquisa se diferencia por utilizar a gestão de contexto para obter o equilíbrio entre a colaboração e a disciplina dos processos de desenvolvimento. Para este balanceamento é necessário entender e caracterizar o contexto que envolve as pessoas e o ambiente onde o desenvolvimento de software ocorre. Assim, o objetivo deste trabalho é estudar formas de representação de informações de contexto referentes ao domínio de processos de desenvolvimento de software. Além disso, neste trabalho foram avaliadas também algumas ferramentas que sejam capazes de sistematizar regras de balanceamento baseadas em contexto no sentido de indicar combinações adequadas de colaboração e disciplina às necessidades e características específicas de cada projeto e organização de desenvolvimento de software.

SUMÁRIO

1. Introdução

Nas organizações atuais, o compartilhamento de conhecimento se tornou fundamental para o estímulo ao aprendizado das boas práticas nos processos de trabalho. Para que a aprendizagem organizacional seja viabilizada, é necessário obter e difundir informações provenientes não só dos processos de negócio, mas também dos conceitos que formam a base para construção destes (AGOSTINI ET AL., 1996, DAVENPORT E PRUSAK, 1998) e, desta forma, construir produtos com alto grau de qualidade. Em particular, as organizações de desenvolvimento de software são continuamente desafiadas pela necessidade de melhorar a qualidade dos produtos de software gerados e atender às necessidades de customização dos seus clientes.

Estas organizações se engajam em uma grande variedade de projetos de desenvolvimento de software, mas estes projetos possuem características distintas, onde os modelos de desenvolvimento orientado ao planejamento, ágil e livre se complementam, pois cada um funciona melhor ou enfrenta dificuldades em determinado aspecto (GLAZER ET AL., 2008).

Como a maioria dos projetos de desenvolvimento de software possui características diversificadas, nenhuma metodologia sozinha consegue oferecer todas as soluções necessárias. Assim, são necessárias abordagens que equilibrem os diferentes modelos de desenvolvimento (BOEHM E TURNER, 2003). Esta busca pelo equilíbrio pode ser observada pelo crescimento da adaptação dos processos de software por parte das organizações (HANSSON ET AL., 2006), enquanto decresce a quantidade de organizações que segue um modelo de qualidade de modo totalmente prescritivo (PATEL ET AL., 2006).

Porém, o entendimento atual de como as características do projeto, da organização e da equipe afetam as estratégias de adaptação ainda é limitado (PENG XU E RAMESH, 2008). Assim, propõem-se a adaptação de processos de desenvolvimento de software, através do balanceamento entre os fatores de colaboração e disciplina que predominantemente distinguem os três modelos de desenvolvimento em questão,

através da gestão de contexto, que utiliza as informações de contexto para capturar as características relevantes para esta abordagem (MAGDALENO, 2010).

Para ampliar a percepção sobre as informações de contexto que apóiam no balanceamento dos aspectos de colaboração e disciplina, é necessário explicitá-lo representando estas informações de maneira uniforme, para torná-las computacionalmente processáveis e acessíveis.

Tendo esta motivação em vista, o objetivo deste trabalho de pesquisa é realizar um estudo para identificar formas de representação das informações de contexto que caracterizam o domínio de processos de desenvolvimento de software. Além disso, também pretende-se explorar qual a ferramenta que oferece as funcionalidades necessárias para apoiar esta representação e oferecer o raciocínio lógico que pode auxiliar nesta adaptação.

Este trabalho está organizado da seguinte forma: a Seção 2 resume os principais modelos de desenvolvimento de software. A Seção 3 apresenta a proposta para o balanceamento entre colaboração e disciplina, nos processos de desenvolvimento de software. A Seção 4 apresenta os modelos de domínio para processos de desenvolvimento de software. A Seção 5 descreve a abordagem de gestão de contexto, exemplificando algumas informações e situações de contexto relevantes para processos de desenvolvimento de software e discutindo a necessidade de um modelo de representação de contexto. A Seção 6 descreve as linguagens de representação de conhecimento selecionadas para avaliação. Na Seção 7, é apresentado o estudo realizado utilizando duas ferramentas que implementam estas linguagens. Por fim, a Seção 8 discute alguns aspectos levantados durante a avaliação e conclui este estudo.

2. Modelos de desenvolvimento de software

Humphrey (1989) define processo de software como “o conjunto de tarefas de engenharia de software necessárias para transformar os requisitos dos usuários em software”. Um processo de software foi definido por Fuggetta (2000) como: “um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos necessários para conceber, desenvolver, implantar e manter um produto de software”.

Motivada pela preocupação com a qualidade do produto de software, a comunidade de Engenharia de Software, percebeu que esta qualidade depende do processo de desenvolvimento adotado para construí-lo e que muitos dos problemas de desenvolvimento podem ser resolvidos aperfeiçoando este processo (CUGOLA E GHEZZI, 1998, FEI DAI E TONG LI, 2007, FUGGETTA, 2000, OSTERWEIL, 1987, PRESSMAN, 2001, RAMAN, 2000). Em um processo de software são muitas as atividades que afetam diretamente a qualidade do produto. A execução ineficiente ou a não execução de algumas delas pode acarretar a obtenção de um produto de software que não esteja alinhado com as expectativas e requisitos dos usuários (MACHADO, 2000).

Pelo impacto que o desenvolvimento de software orientado ao planejamento, ágil e livre tiveram na indústria na última década (EBERT, 2007, THEUNISSEN ET AL., 2008) e por representarem os principais universos de desenvolvimento com características tão peculiares, neste trabalho optou-se por estudar especificamente estes três modelos de desenvolvimento.

2.1. Modelo de desenvolvimento de software orientado ao planejamento

O modelo de desenvolvimento orientado ao planejamento (também denominado na literatura como tradicional) é tipicamente exemplificado pelos modelos de qualidade, tais como o CMMI (CHRISSIS ET AL., 2006), a ISO 12207 (ISO/IEC, 2007) e o MPS-BR (SOFTEX, 2009). Historicamente, estes modelos de qualidade foram concebidos para lidar com ambientes de alto risco e projetos grandes e complexos com altos custos envolvidos, onde os relacionamentos entre cliente e equipe de desenvolvimento se caracterizam pelo baixo nível de confiança e são governados por definições contratuais. A rigidez de um contrato torna mais difícil as adaptações às mudanças. Além disso, devido ao baixo nível de confiança, são necessárias aprovações frequentes e, ao longo de cada fase do ciclo de vida de desenvolvimento, existe uma preocupação com a completude da documentação e com a rastreabilidade entre a documentação gerada em cada uma das fases (GINSBERG E QUINN, 1995, GLAZER ET AL., 2008).

Assim, à luz destes modelos de qualidade, e visando alcançar os objetivos de previsibilidade, estabilidade e confiabilidade, este modelo de desenvolvimento é

fortemente orientado a planejamento e baseado em processos bem definidos e melhorados continuamente pela organização (BOEHM E TURNER, 2003). Estes processos estabelecem uma abordagem sistemática que guia o desenvolvimento do software desde a definição dos requisitos até a implantação do software.

De modo geral, este modelo atinge bem os seus objetivos em ambientes relativamente estáveis, mas quando confrontado com projetos inovadores, a previsibilidade e a estabilidade degradam e o projeto incorre em gastos significativos para tentar manter a aderência ao processo e o planejamento atualizado (BOEHM E TURNER, 2003).

Este modelo de desenvolvimento utiliza uma estrutura de coordenação baseada em comando e controle e se baseia no registro do conhecimento explícito sobre o produto construído. A comunicação entre os membros do projeto também é formalizada através dessa documentação. O cliente desempenha um papel importante neste modelo, mas só durante as fases de especificação de requisitos e homologação (NERUR ET AL., 2005).

2.2. Modelo de desenvolvimento de software ágil

As características (ABRANTES E TRAVASSOS, 2007, QUMER E HENDERSON-SELLERS, 2008) e premissas do desenvolvimento ágil, observadas em métodos tais como XP (*Extreme Programming*) (BECK, 1999), Scrum (SCHWABER, 2004), FDD (*Feature Driven Development*) (PALMER E FELSING, 2002), DSDM (*Dynamic Systems Development Methodology*) (STAPLETON E CONSTABLE, 1997) e *Crystal* (COCKBURN, 2004), podem ser resumidas pelos quatro valores fundamentais do *Manifesto for Agile Software Development* (BECK ET AL., 2001): **indivíduos e interações** entre eles mais que processos e ferramentas; **software em funcionamento** mais que documentação abrangente; **colaboração com o cliente** mais que negociação de contratos; **responder a mudanças** mais que seguir um plano.

De modo geral, os objetivos principais dos métodos ágeis são fornecer valor rapidamente e responder às mudanças de mercado, tecnologia e ambiente (COCKBURN, 2001, HIGHSMITH E COCKBURN, 2001). Graças a esta orientação à adaptação, ao invés de predição, o projeto é executado de forma incremental, com ciclos de desenvolvimento curtos (MILLER, 2001).

O desenvolvimento ágil é mais aplicável em ambientes turbulentos, que sofrem muitas mudanças e onde existe certo nível de incerteza técnica (BOEHM, 2002, HIGHSMITH E COCKBURN, 2001, LINDVALL ET AL., 2002, PATEL ET AL., 2006) que requer a combinação de habilidades através do trabalho colaborativo dos membros da equipe. Ele também opera melhor em uma cultura organizacional centrada em pessoas e colaborativa (COCKBURN E HIGHSMITH, 2001).

Durante o projeto existe um grande envolvimento e participação do cliente no levantamento, priorização e validação dos requisitos. Os especialistas do negócio ficam disponíveis para a equipe de desenvolvimento e chegam a fazer parte da equipe. Esta disponibilidade do especialista reduz o tempo de *feedback* da solução proposta e agiliza o entendimento da equipe sobre as necessidades dos usuários (ABRAHAMSSON ET AL., 2003, COCKBURN, 2001, HIGHSMITH E COCKBURN, 2001, LINDVALL ET AL., 2002, TURK ET AL., 2002).

Como resultado, o desenvolvimento ágil produz software com parcimônia na documentação (NERUR ET AL., 2005, PATEL ET AL., 2006), desenvolve apenas o mínimo realmente necessário para atender ao conjunto de requisitos da iteração atual, integra o código constantemente e usa os testes de regressão. Concentrando-se completamente na iteração atual, é possível entregar um pequeno produto no prazo e satisfazer ao cliente (BECK ET AL., 2001, COCKBURN E HIGHSMITH, 2001, COCKBURN, 2001).

Segundo Kent Beck (1999), os métodos ágeis parecem funcionar melhor com equipes pequenas, geograficamente centralizadas, trabalhando em aplicações pequenas, pois este modelo de desenvolvimento se baseia fortemente no conhecimento tácito (NERUR ET AL., 2005, PATEL ET AL., 2006) e na comunicação face-a-face, o que requer que os desenvolvedores estejam fisicamente próximos (TURK ET AL., 2002).

No desenvolvimento ágil, o papel convencional do gerente de projeto como planejador, organizador e controlador é substituído por um papel de facilitador do trabalho da equipe que se auto-direciona e se auto-organiza para lidar com o trabalho (BECK, 1999, NERUR ET AL., 2005). Além disso, o gerente de projeto vai se dedicar a estabelecer uma relação de parceria com os clientes (COCKBURN E HIGHSMITH, 2001).

2.3. Modelo de desenvolvimento de software livre

Apesar de terem algumas características em comum, existem muitas diferenças entres os projetos de software livre (GACEK E ARIEF, 2004) e estas diferenças influenciam também no processo de desenvolvimento adotado. Desta forma, nesta seção, serão abordadas algumas características gerais, presentes na maioria dos projetos de software livre e que podem oferecer uma visão geral deste modelo de desenvolvimento.

O modelo de desenvolvimento de software livre pode ser entendido pela metáfora do bazar (RAYMOND, 2001), onde os projetos são desenvolvidos de forma colaborativa e transparente. Em geral, os projetos de software livre se caracterizam pelo trabalho voluntário e colaborativo de desenvolvedores, com habilidades e disponibilidades distintas, geograficamente distribuídos, organizados em uma comunidade virtual através da Internet, e que contribuem espontaneamente com o rápido desenvolvimento do software. O software produzido é distribuído também pela Internet para ser utilizado e modificado por quem se interessar. O engajamento voluntário confere às comunidades um caráter de alto compromisso com seu sucesso, o que pode explicar sua capacidade de produção que sobrepõe às dificuldades técnicas naturais do trabalho remoto e distribuído (CAPILUPPI ET AL., 2003, CUBRANIC E BOOTH, 1999, FELLER E FITZGERALD, 2001, GACEK E ARIEF, 2004, HEALY E SCHUSSMAN, 2003, REIS, 2003, WARSTA E ABRAHAMSSON, 2003, YAMAUCHI ET AL., 2000).

O modelo de desenvolvimento de software livre se destaca pelo seu alto grau de colaboração e interações. Com uma equipe geograficamente dispersa, raramente seus membros se encontram pessoalmente. Devido a essa ausência de comunicação face-a-face, a comunicação acontece predominantemente de forma assíncrona e escrita, utilizando a Internet como canal de comunicação a distância. Além disso, as comunidades contam também com a participação atuante dos usuários finais que se comunicam com os desenvolvedores e entre si, comunicando problemas e trocando experiências do uso do software (CUBRANIC E BOOTH, 1999, FELLER E FITZGERALD, 2001, WARSTA E ABRAHAMSSON, 2003, YAMAUCHI ET AL., 2000).

O prestígio na comunidade é a base para o modelo de liderança meritocrático, onde um ou mais indivíduos contam com a confiança e respeito do grupo para coordenar

informalmente o trabalho (CUBRANIC E BOOTH, 1999, YAMAUCHI ET AL., 2000). Devido ao caráter de trabalho voluntário, os desenvolvedores possuem liberdade para a escolha das tarefas que serão realizadas, com base em suas preferências e habilidades (HAEFLIGER ET AL., 2007, SCACCHI, 2007).

O desenvolvimento de software livre se adapta melhor em ambientes onde predomina a capacidade criativa e de inovação (EBERT, 2007) e existe pouca sistematização do trabalho. Outras características do desenvolvimento de software livre, especificamente voltadas à colaboração e reutilização, foram detalhadas em Magdaleno e Werner (2008).

3. Balanceamento de colaboração e disciplina nos processos de desenvolvimento de software

Os modelos de desenvolvimento de software orientado ao planejamento, livre e ágil têm o mesmo objetivo: melhorar o desenvolvimento de software; mas adotam enfoques distintos. Enquanto no desenvolvimento orientado ao planejamento busca-se previsibilidade, estabilidade e confiabilidade (CHRISSIS ET AL., 2006), o desenvolvimento ágil tenta agregar valor ao negócio rapidamente e se adaptar às mudanças de mercado, tecnologia e ambiente (COCKBURN, 2001). Por outro lado, no desenvolvimento de software livre, o objetivo principal é garantir as liberdades básicas dos usuários para executar, estudar, adaptar, melhorar e distribuir o código do programa (FSF, 2008).

Cada um com as suas peculiaridades, seus casos de sucesso e seus desafios, os modelos de desenvolvimento orientado ao planejamento, ágil e livre seguiram caminhos distintos. Devido a problemas de diferenças de vocabulário, más interpretações e talvez mau uso das abordagens, os três modelos costumam ser percebidos como opositores (GLAZER ET AL., 2008). Entretanto, a conciliação de processos de desenvolvimento de software (BARNETT, 2004, BOEHM E TURNER, 2003, GLASS, 2001, GLAZER ET AL., 2008, LINDVALL ET AL., 2002, PATEL ET AL., 2006, PAULK, 2001, TURK ET AL., 2002, VINEKAR ET AL., 2006, WARSTA E ABRAHAMSSON, 2003) tenta entender as similaridades e distinções entre estes modelos, identificar sob que condições cada um funciona melhor e compreender como o melhor de cada um pode ser combinado, trazendo ganhos de produtividade e qualidade para as organizações.

De acordo com os resultados de uma revisão *quasi*-sistemática (BIOLCHINI ET AL., 2005) sobre conciliação de processos de desenvolvimento de software realizada (MAGDALENO ET AL., 2009), foi possível observar que, em geral, as propostas de conciliação existentes caminham no sentido da comparação e combinação das práticas sugeridas pelos diferentes modelos, visando à obtenção de um novo modelo de processos híbrido (FRITZSCHE E KEIL, 2007, SANTANA ET AL., 2006). Contudo, a natureza complexa da atividade de desenvolvimento de software e a grande variedade de métodos existentes tornam a tarefa de comparação dos modelos de desenvolvimento, uma tarefa árdua e imprecisa (BOEHM E TURNER, 2003). Além disso, neste tipo de proposta não se consegue garantir que o processo resultante realmente tenha as características desejadas.

Assim, defende-se que a conciliação significa mais do que a combinação de práticas dos diferentes modelos de desenvolvimento. O enfoque de solução deste trabalho envolve a adaptação dos processos de desenvolvimento de software através do balanceamento dos aspectos de colaboração e disciplina (MAGDALENO, 2010).

A abordagem, utilizada como base para este trabalho, investiga a possibilidade de balancear os aspectos de colaboração e disciplina presentes, ainda que com diferentes ênfases, em cada um dos modelos de desenvolvimento, através da gestão de contexto (MAGDALENO, 2010) (Figura 1).

A ideia é que a **colaboração** entre as pessoas envolvidas no projeto de desenvolvimento possa ser compreendida através da análise de **redes sociais** (BARABASI, 2003). Uma rede social consiste em um conjunto finito de atores e as relações definidas entre eles (WASSERMAN E FAUST, 1994). Existem diversos trabalhos (GAO ET AL., 2003, GOTO ET AL., 2008, JIN XU ET AL., 2005, LOPEZ-FERNANDEZ ET AL., 2004, MADEY ET AL., 2002) que apontam para o potencial das redes sociais em explicitar como a colaboração acontece dentro de um grupo.

Araujo e Borges (2007) defendem que ao explicitar a colaboração, aumenta-se a sua visibilidade, de forma que os membros da organização atinjam maior compreensão e se motivem. Desta forma, o entendimento das redes sociais envolvidas nos projetos de desenvolvimento pode ajudar a calibrar o nível de colaboração desejado.

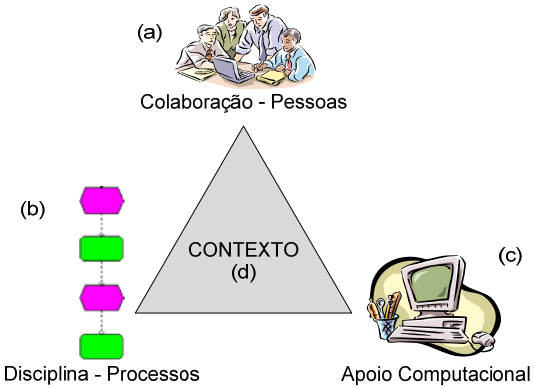

Figura 1 – Uso da gestão de contexto para o balanceamento entre colaboração e disciplina

Por outro lado, a representação dos processos de desenvolvimento de software pode ser mais formal, através de notações como BPMN (*Business Process Management Notation*) (OMG, 2009) ou SPEM (*Software & Systems Process Engineering Meta-Model*) (OMG, 2008), ou mais flexível, adotando a ideia de linha de processos (BARRETO ET AL., 2009, WASHIZAKI, 2006), o que facilita a modelagem do processo e a escolha de componentes de processo com o devido grau de formalismo.

Uma **linha de processos** é um conjunto de componentes de processos, organizados de forma a representar partes (conjunto de atividades) comuns e variantes dentro de um domínio específico, que podem ser reutilizados e combinados entre si, segundo regras de composição e recorte, para compor e adaptar processos dinamicamente (NUNES ET AL., 2010).

As linhas de processos são um importante instrumento para explicitar a **disciplina**, devido a sua flexibilidade de representação que permite regular o nível de formalismo introduzido no processo definido. Além disso, a sua capacidade de adaptação dinâmica também permite que seja dosado o controle sobre o processo executado.

Como um projeto de desenvolvimento de software é baseado em três pilares: pessoas, processos e tecnologia (Figura 1), o enfoque de solução desta abordagem também se baseia na integração entre estes três aspectos para balancear colaboração e disciplina, pois todos eles estão presentes nos processos de desenvolvimento de software.

4. Modelo de domínio de processos de desenvolvimento de software

Este trabalho foca na representação de um domínio específico, o domínio de processos de desenvolvimento de software. Existem alguns exemplos de modelos para este domínio que lidam com o problema de estabelecer um entendimento comum sobre o software entre os participantes, através de diferentes técnicas. Diante das opções, não será proposto nenhum novo modelo de domínio. Apenas serão apresentadas as duas opções mais relevantes para este trabalho.

4.1. Exemplos de modelos de domínio para processos de desenvolvimento de software

O modelo de domínio mais importante atualmente é o SPEM (OMG, 2008) que descreve processos de desenvolvimento de software e seus componentes. O escopo do SPEM é limitado aos elementos mínimos necessários para definir um processo de desenvolvimento de software sem adicionar características específicas para determinados sub-domínios ou disciplinas (Figura 2).

O SPEM é um meta-modelo de engenharia de processos assim como um *framework* conceitual que pode fornecer os conceitos necessários para modelar, documentar, apresentar, gerenciar e executar processos de desenvolvimento. Além disso, ele descreve uma linguagem e um esquema de representação. O SPEM segue uma abordagem orientada a objetos para modelar famílias de processos relacionados e a sua especificação é estruturada como um perfil (do inglês *profile*) UML 2 (*Unified Modeling Language*).

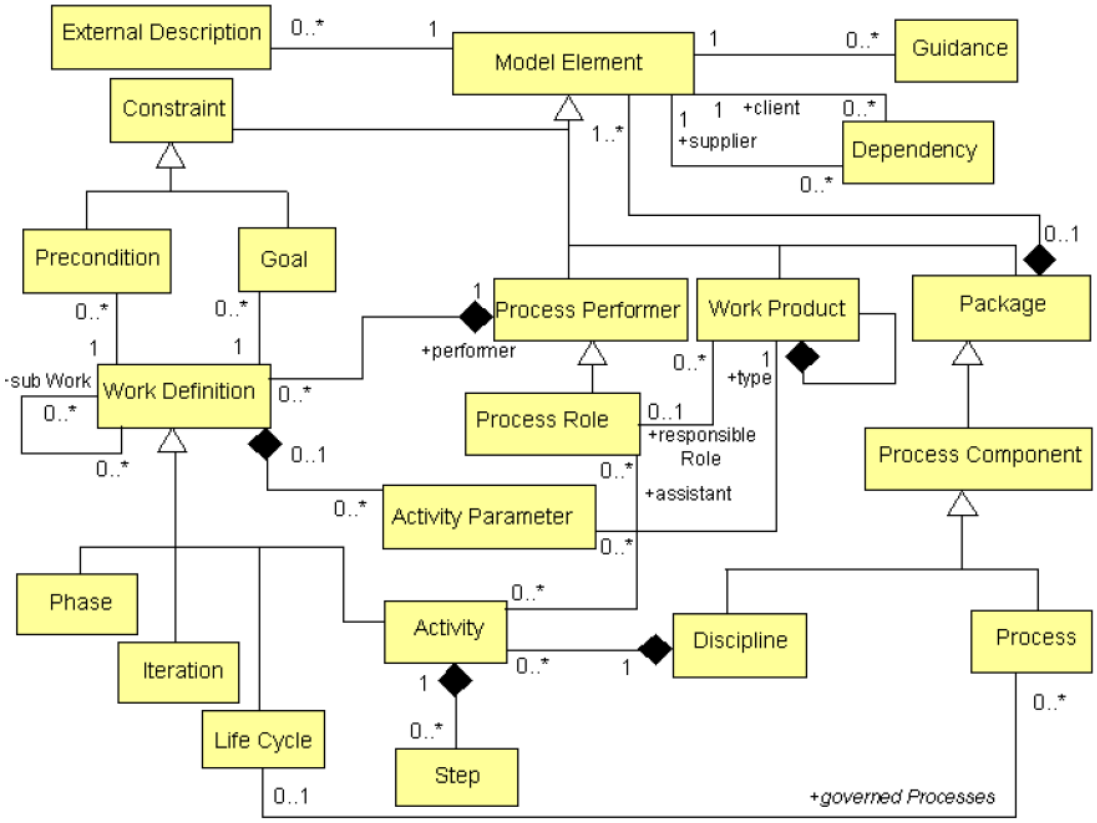

Figura 2 – Principais conceitos do SPEM (OMG, 2008)

Outro modelo que se destaca e que foi utilizado como ponto de partida para definir as informações que compõem este domínio, é a ontologia[1] para o domínio de processos de desenvolvimento de software construída por Falbo (1998) e evoluída mais recentemente (FALBO E BERTOLLO, 2005). A ontologia proposta também já indicou ter correspondência com os elementos propostos pelo SPEM (OMG, 2008). Um modelo parcial dessa ontologia é apresentado pela Figura 3**Erro! Fonte de referência não encontrada.**, onde os conceitos em cinza foram criados nesta evolução recente do trabalho.

Dois conceitos são fundamentais no contexto de processos de desenvolvimento de software: processo e atividade. Um processo é uma infraestrutura contendo as atividades das diversas naturezas envolvidas no desenvolvimento de software. Processos podem ser decompostos em outros processos ou em atividades. Além disso, processos podem ter interações com outros processos de mesmo nível. Processos, de

[1] Uma ontologia é uma especificação formal de conceitos e termos do domínio com a definição de regras que regulam a combinação entre termos (GRUBER, 1995).

maneira geral, são classificados em Categorias de Processos, podem ou não ser aderentes a Normas e podem ser definidos levando em consideração um Paradigma, Tipo de Software ou Domínio de Aplicação específico (FALBO, 1998).

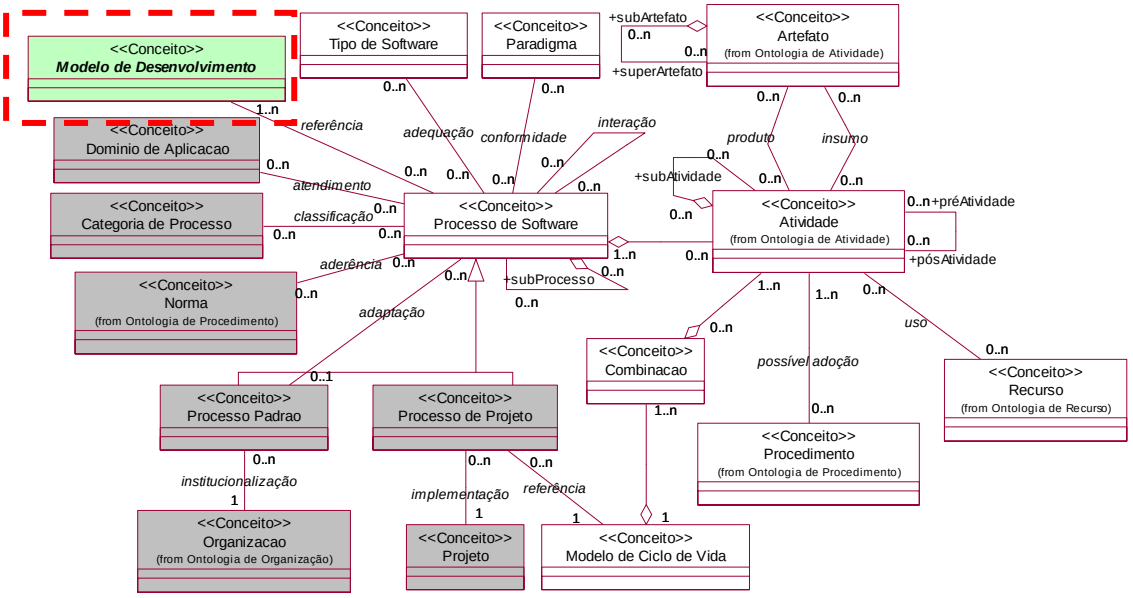

Figura 3 – Modelo parcial da ontologia de processos de software. Adaptado de: (BERTOLLO ET AL., 2006)

Atividades podem ocorrer em vários níveis, desde uma tarefa elementar até uma etapa do processo de desenvolvimento. Assim, pequenas atividades podem compor uma atividade maior. Por isso, foram introduzidos os conceitos de super-atividade e sub-atividade. Além disso, uma atividade utiliza artefatos de entrada para gerar artefatos de saída, apoiada por recursos. Por fim, atividades são realizadas seguindo uma sequência específica, por isso a ideia de pré-atividade e pós-atividade (FALBO, 1998).

Em relação à adaptação de processos, os conceitos de Processo Padrão e Processo de Projeto também são significativos. Um Processo Padrão está sempre associado a uma Organização, enquanto um Processo de Projeto é definido para um Projeto específico. Um Processo de Projeto é uma adaptação de um Processo Padrão. Além disso, um Processo de Projeto define sempre um Modelo de Ciclo de Vida como referência (FALBO, 1998).

4.2. Exemplos de modelos e práticas de desenvolvimento de software

Apesar de contemplar os principais conceitos relevantes para o domínio de processos de desenvolvimento de software, esta ontologia não contempla os diferentes modelos de

desenvolvimento atualmente existentes, tais como o desenvolvimento ágil e livre, pois ela é direcionada apenas para o desenvolvimento orientado ao planejamento. Desta forma, foi necessário introduzir um novo conceito chamado “Modelo de Desenvolvimento”, que se relaciona com o conceito “Processo de Software” e está destacada na Figura 3**Erro! Fonte de referência não encontrada.**.

A partir do conceito “Modelo de Desenvolvimento” foram representados os três modelos de desenvolvimento contemplados neste trabalho: orientado ao planejamento, ágil e livre. Em seguida, foram identificados alguns exemplos para cada um destes modelos de desenvolvimento. Para o modelo orientado ao planejamento, foram considerados o CMMI (CHRISSIS ET AL., 2006), a ISO 12207 (ISO/IEC, 2007) e o MPS-BR (SOFTEX, 2009). O modelo de desenvolvimento de software livre foi caracterizado em caverna e comunidade seguindo o trabalho de Krishnamurthy (2002). Por fim, no caso do desenvolvimento ágil foram elencados os seguintes representantes: XP (BECK, 1999), Scrum (SCHWABER, 2004) e Crystal (COCKBURN, 2001).

Uma vez identificados os modelos de desenvolvimento, é possível identificar as suas principais práticas. Neste sentido, um exemplo parcial de práticas para projetos de software livre, focando nas disciplinas de gerência de configuração e desenvolvimento, foi modelado no diagrama apresentado na Figura 4.

O objetivo da gerência de configuração é permitir que o grupo de desenvolvedores se torne o mais eficiente possível. Para facilitar o desenvolvimento rápido e paralelo e ao mesmo tempo manter uma evolução controlada do software, a maioria dos projetos de software livre adota práticas e ferramentas de gerência de configuração (ASKLUND E BENDIX, 2002).

Segundo Asklund e Bendix (2002), as práticas de gerência de configuração importantes no desenvolvimento de software livre são: controle de versões, gerenciamento de *builds*, seleção de configuração, gerenciamento do espaço de trabalho, controle de concorrência, gestão de mudanças e gestão de *releases*. Dentre essas, foram selecionadas as práticas de controle de versões e controle de concorrência para ilustrar parcialmente a ideia deste trabalho.

O controle de concorrência gerencia ou bloqueia os acessos simultâneos de diversos desenvolvedores. Assim, os projetos podem adotar uma política otimista ou pessimista, mas uma e somente uma delas (Figura 4). A política otimista significa que os arquivos não são bloqueados ao serem copiados do repositório. Na política pessimista, os arquivos são bloqueados, evitando que alguém os modifique em paralelo (ASKLUND E BENDIX, 2002). O controle de versões se refere à possibilidade de armazenar diferentes versões de um artefato e, posteriormente, ser capaz de recuperá-lo e compará-lo com outras versões (ASKLUND E BENDIX, 2002), através de um repositório centralizado ou distribuído (Figura 4).

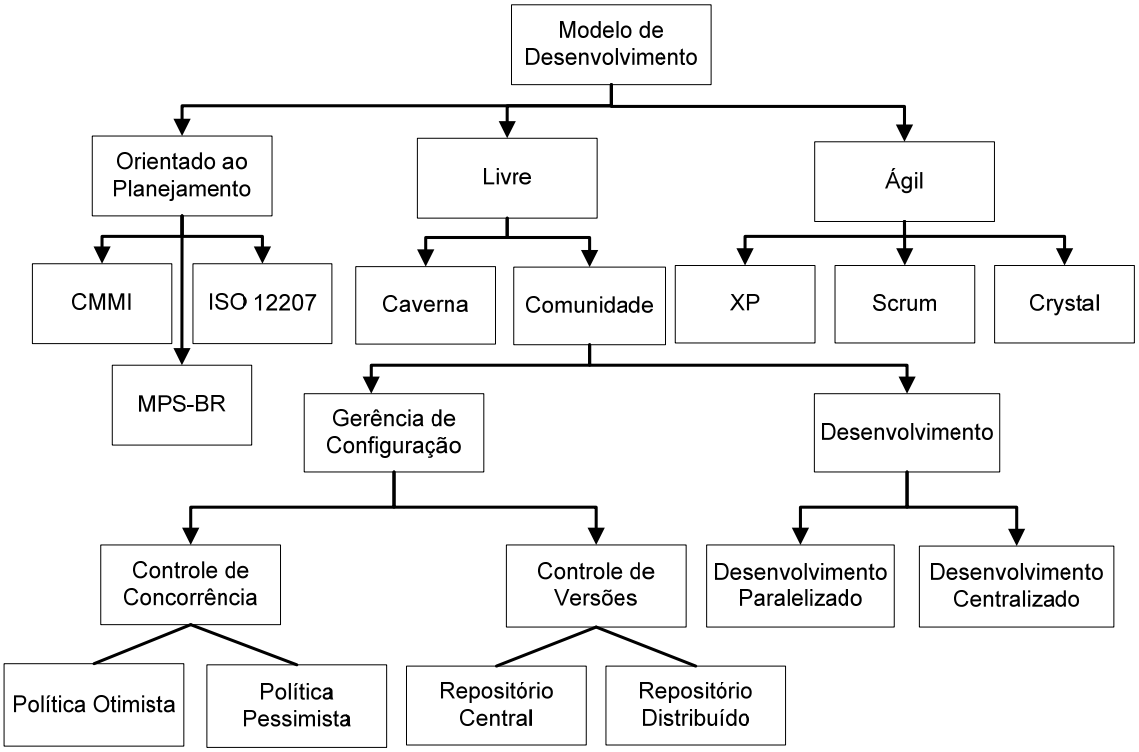

Figura 4 – Exemplo parcial de práticas de desenvolvimento de software

A representação do domínio do problema que está sendo tratado é o ponto de partida para entendimento do domínio e das informações de contexto que se deseja tratar neste meio. Na próxima seção é apresentada a abordagem para gerir o contexto com foco no domínio de processos de desenvolvimento de software.

5. Gestão de contexto no domínio de processos de desenvolvimento de software

5.1. Contexto

Vários pesquisadores tentam formalizar a definição de contexto (BREZILLON E POMEROL, 1999, BREZILLON, 1999, 2003, CHEN E KOTZ, 2000, SCHMIDT ET AL., 1999), mas com visões diferentes. Bazire e Brézillon (2005) catalogaram mais de 150 definições de contexto e observaram que elas várias de acordo com os diferentes domínios.

Uma definição largamente utilizada declara que "contexto é qualquer informação que possa ser utilizada para caracterizar a situação de uma entidade, onde uma entidade pode ser uma pessoa, lugar, ação ou objeto que seja considerado relevante para a situação" (DEY ET AL., 2001). Neste trabalho, a definição adotada é a de Brezillon (1999), segundo a qual contexto é "uma descrição complexa do conhecimento compartilhado sobre circunstâncias físicas, sociais, históricas e outras dentro das quais ações ou eventos ocorrem".

Brezillon e Pomerol (1999) propuseram um modelo que classifica o contexto de acordo com o *foco de atenção* de uma situação em particular. O foco pode representar uma tarefa (NUNES, 2007), um passo na solução de um problema ou uma tomada de decisão. Eles argumentam que o contexto não pode ser considerado de forma isolada ou ampla demais. Ao invés disso, é o foco de atenção que vai determinar o que é relevante para um determinado contexto.

De acordo com o foco de atenção, os autores classificam o contexto em três partes (Figura 5). O *conhecimento externo* representa a parte do conhecimento que não tem relevância para o foco definido. Por exemplo, suponha que o foco de um usuário é encontrar especialistas para ajudá-lo em uma tarefa de desenvolvimento de software. Neste caso, o conhecimento externo pode incluir informações dos especialistas tais como: peso e estado civil. Ainda que estas informações façam parte do conhecimento sobre os especialistas, elas não são úteis para o foco de atenção atual.

O *conhecimento contextual* representa o conhecimento que é imediatamente relevante para a tarefa e que possui uma forte relação com o foco de atenção em questão. No caso

do exemplo anterior, o conhecimento contextual inclui informações sobre a localização, presença, disponibilidade, habilidade, reputação, experiência e linguagem de desenvolvimento dominada pelos especialistas.

Finalmente, o *contexto proceduralizado* é a parte do conhecimento contextual que é demandado, organizado e estruturado de acordo com o foco em questão. Esta é a parte do contexto que realmente é necessária para a situação atual. Assim, no caso do exemplo anterior isto significa saber que dois especialistas estão presentes, sendo que João está disponível e é especialista em Java, enquanto Carlos está ocupado e domina a linguagem Delphi.

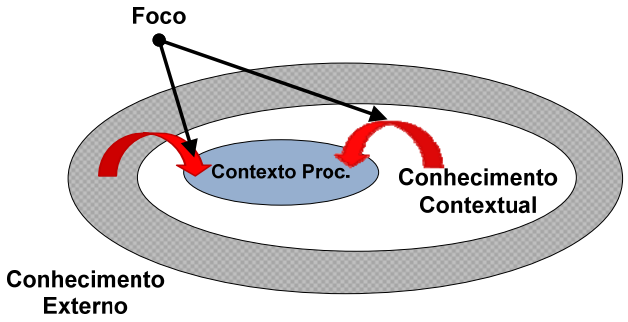

Figura 5 – Dinâmica do Contexto. Adaptado de: (BREZILLON E POMEROL, 1999)

5.2. Gestão de contexto

A importância da informação contextualizada se deve ao fato desta possuir a capacidade de prover maior significado as atividades, fatos, artefatos gerados e decisões tomadas. Na filosofia afirma-se que não existe informação desprovida de contexto (HEIDEGGER, 1978). Além disso, o conhecimento contextual, quando tratado, atua: propiciando práticas mais efetivas e eficientes nos negócios; a aderência do formato de trabalho às necessidades e cultura da organização; e promovendo a inovação e interação dos grupos de trabalho e da própria organização.

Se a percepção sobre o conhecimento contextual for ampliada, maior será o apoio à execução das atividades em um processo de trabalho. Todo este conhecimento não é uma parte das ações que serão executadas ou dos eventos que ocorrem, mas irá subsidiar de forma mais eficaz e eficiente a execução de uma ação ou de uma interpretação do evento sem intervir nele explicitamente (BREZILLON, 1999). Contudo, para ampliar essa percepção é necessário explicitar o conhecimento

contextual, representá-lo de maneira uniforme, organizá-lo e torná-lo utilizável pelas pessoas.

A partir da abordagem proposta por Nunes (2007), quatro questões referentes à gestão de contexto, apresentadas no ciclo da Figura 6, são examinadas:

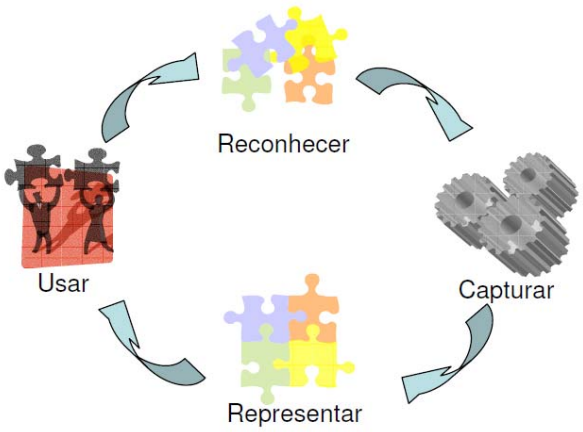

Figura 6 – Ciclo de transformação do contexto (NUNES, 2007)

- Como reconhecer quais informações de contexto são necessárias?
- Como estabelecer os mecanismos de captura de contexto?
- Como representar o contexto?
- Como usar as informações de contexto?

Consequentemente, o contexto é estudado através de quatro diferentes aspectos (NUNES, 2007):

• **Definição de tipos de contexto, classificações e relacionamentos entre eles:** As informações sobre o domínio necessitam ser formalizadas para que o contexto possa ser gerenciado;

• **Mecanismos que reconheçam informação contextual:** Como não existe uma forma única de reconhecimento e captura de informação, é necessário formalizar os mecanismos de captura que melhor se adéquem à situação e informação contextual;

• **Aplicação de regras que identifiquem contextos relacionados ao atual:** Para utilizar informações contextuais de atividades passadas e do ambiente de uma maneira geral, regras de inferência, construídas e evoluídas a partir da execução das atividades em diversas situações, devem ser aplicadas de forma a apoiar na identificação de novas informações, ações e funções que possam ser úteis ao contexto atual;

• **Definição de modelos de apresentação de contexto que o tornem reutilizável:** Uma questão muito relevante é a visualização de informações contextuais para um indivíduo ou grupo. As decisões sobre como será a explicitação dos contextos recuperados é um dos fatores de sucesso ou fracasso na reutilização destas informações.

Para entender as questões e aspectos levantados, na próxima seção é proposta uma abordagem para a gestão de contexto num ambiente de trabalho qualquer.

5.3. Abordagem de gestão de contexto

Para concretizar a gestão de contexto, podemos conceber uma abordagem onde as informações de contexto poderiam ser capturadas (Figura 7b), a partir de diferentes fontes de informações, por mecanismos manuais ou automatizados. Como resultado, as informações de contexto seriam combinadas de forma a apoiar na definição de um processo de desenvolvimento de software que equilibrasse os aspectos de colaboração e disciplina nas medidas necessárias ao projeto de desenvolvimento. Durante a execução do projeto, outras informações de contexto seriam geradas e o processo adaptado ao projeto poderia ser dinamicamente revisto. A visão geral da abordagem de gestão de contexto para o balanceamento entre colaboração e disciplina é apresentada pela Figura 7. Esta abordagem é composta por seis partes importantes, que englobam toda a gestão de contexto, detalhadas nas próximas seções, mas antes de detalhá-las individualmente é preciso entender como se daria o uso desta abordagem.

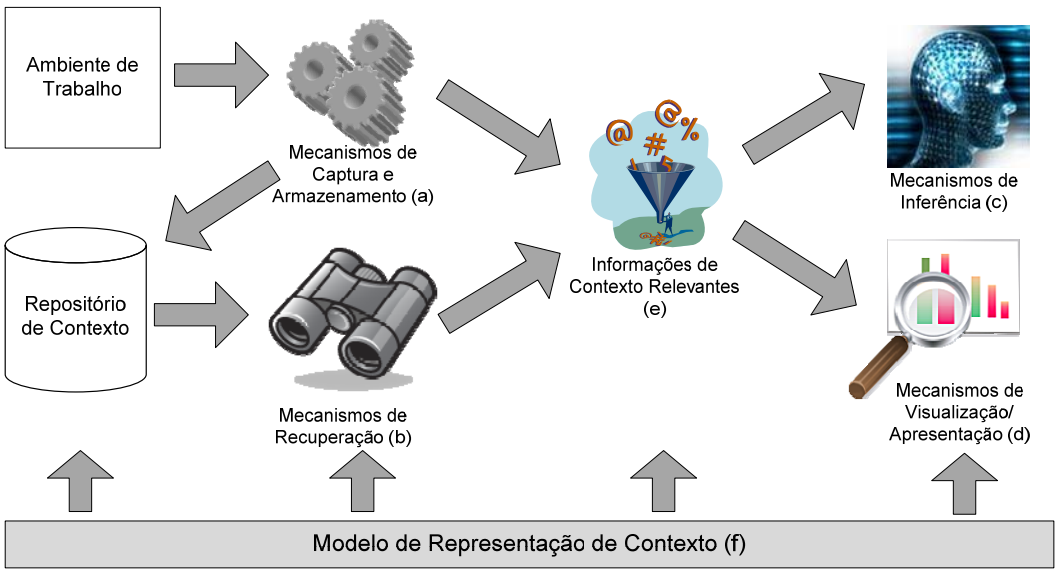

Figura 7 – Visão geral da abordagem de gestão de contexto

Neste trabalho são propostos dois cenários de uso das informações de contexto para aplicação da proposta. Estes cenários foram escolhidos, com base na literatura (PEDREIRA ET AL., 2007) e na experiência dos autores, por representarem momentos importantes durante um projeto de desenvolvimento de software, onde o gerente do projeto e a equipe de desenvolvimento precisam estar munidos de informações para tomar decisões e ações sobre o futuro do projeto.

O primeiro cenário é mais simples e consiste na atividade de definição de um processo para um novo projeto, realizada pelo Gerente de projeto. Neste cenário, são utilizadas como entrada as informações de contexto organizacionais e informações de contexto de outros projetos similares, recuperadas do repositório de contexto. Além disso, serão capturadas, através de interação com o agente humano, as informações do contexto atual. Estas informações também serão alimentadas no repositório de contexto (Figura 8).

Com base em todas estas informações de contexto são geradas sugestões para adaptação do processo do projeto, levando em consideração o balanceamento necessário entre colaboração e disciplina (Figura 8). Além disso, é oferecida ao gerente de projeto uma visualização do contexto organizacional, do contexto de projetos similares e do contexto atual (Figura 8).

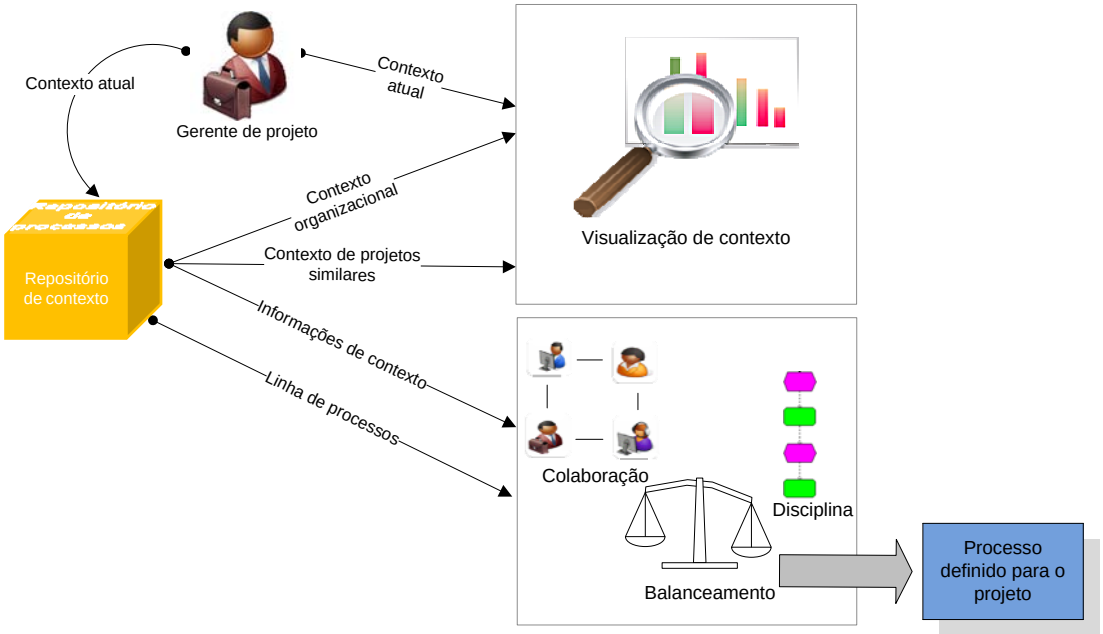

Figura 8 – Cenário 1 de balanceamento entre colaboração e disciplina

O cenário 2 ocorre nos marcos definidos para o acompanhamento do projeto. Neste momento, a equipe de desenvolvimento precisa ser munida de informações sobre o projeto para tomar decisões e possíveis ações corretivas. Então, neste cenário são utilizadas como entrada as seguintes informações de contexto: organizacionais; de outros projetos similares; do contexto anterior do projeto; e do contexto atual do projeto (Figura 9).

As informações de contexto do projeto atual são recuperadas com base nas informações de contexto capturadas, através do gerente de projeto, no cenário anterior, mas também podem ser atualizadas neste cenário, caso tenham se modificado ao longo da execução do projeto.

Além da visualização de contexto, durante a realização desta atividade pretende-se oferecer a visualização do processo executado pelo projeto e a visualização das redes sociais existentes no projeto (Figura 9). Com base em todas estas informações, o balanceamento entre colaboração e disciplina é atualizado com novas sugestões.

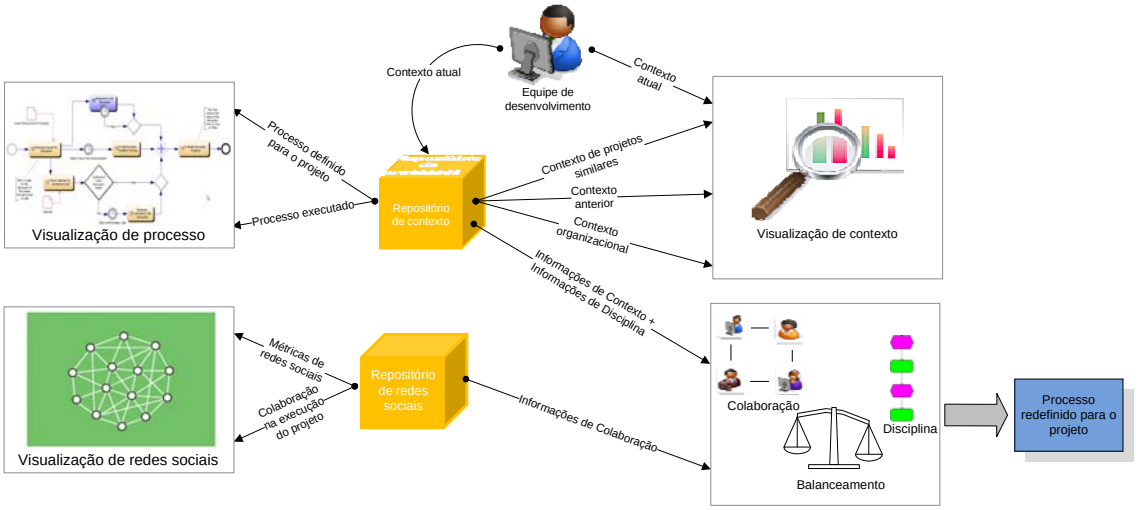

Figura 9 – Cenário 2 de balanceamento

Durante a execução destes cenários, pretende-se oferecer uma visualização integrada de contextos similares, do processo adaptado para o projeto, do processo realmente executado e das redes sociais do projeto. Estas visualizações irão ajudar a aumentar a percepção da equipe de desenvolvimento sobre o estado atual da colaboração de disciplina no projeto.

a) Mecanismos de captura e armazenamento de contexto

Os mecanismos para capturar o contexto (Figura 7a) existente devem procurar a informação que está disponível durante a execução das atividades dos processos. Os mecanismos de captura podem ser manuais (informações fornecidas pelo próprio agente humano), automatizados (agentes ou sensores) ou semi-automatizados (quando existe interação entre o profissional e o sistema).

Existe um grande esforço neste sentido, uma vez que a disponibilização do contexto pode ser feita de várias formas. As informações de contexto podem ser adquiridas de dispositivos não tradicionais, de fontes múltiplas e heterogêneas. Além disso, podem ter uma baixa granularidade e precisar de abstração para que possam ser compreendidas, podem mudar rapidamente e devem levar em consideração o histórico (SANTOS, 2008).

Portanto, os mecanismos para capturar o conhecimento contextual devem se adaptar ao ambiente. Para algumas informações, a captura pode ser puramente humana onde o próprio profissional que participa da atividade registra o seu contexto. Outras informações se encontram disponíveis no próprio ambiente ou fazem parte do resultado da execução de uma atividade, o que requer um mecanismo que identifique-as e capture-as de acordo com o modelo de representação (NUNES, 2007).

O armazenamento das informações de contexto capturadas realimenta a base de gestão de contexto, introduzindo contextos novos, que poderão ser consultados posteriormente quando da execução de atividades semelhantes (NUNES, 2007).

b) Mecanismos de recuperação de contexto

Os mecanismos de recuperação (Figura 7b) são capazes de recuperar conhecimentos, novos ou não, existentes na organização e que poderão ser consultados quando da execução de atividades com contextos similares. Esta recuperação permite que estes conhecimentos prévios possam auxiliar na execução da atividade atual.

A recuperação de informações de contextos de atividades passadas pode ser manual, permitindo com o que o próprio profissional possa recuperar o conhecimento desejado. Por outro lado, os mecanismos de inferência podem ajudam a determinar o contexto da atividade atual, com base nas suas características, para que os mecanismos de

recuperação automáticos sejam capazes de recuperar contextos existentes relacionados à atividade em questão (NUNES, 2007).

c) Mecanismos de inferência de contexto

Com base nas informações de contexto recuperadas, os mecanismos de inferência (Figura 7c) podem processar regras de contexto existentes para eleger o contexto considerado útil para a atividade e o executor. Por exemplo, através da captura de informações contextuais os mecanismos de inferência podem sugerir uma tática especifica para negociar com um cliente já conhecido na organização. Pode ainda sugerir a forma de tratar alguma exceção acontecida em contexto similar que foi bem sucedida. O resultado da inferência processada é apresentado ao profissional ou grupo que está executando a atividade (NUNES, 2007).

d) Mecanismos de visualização/apresentação de contexto

Os mecanismos de visualização (Figura 7d) devem levar em consideração as diversas informações de contexto e os relacionamentos existentes entre elas, se houver. A interação do indivíduo com a interface também pode gerar novas regras de contexto, contribuindo para que os mecanismos de inferência atuem no sentido de realizar uma seleção cada vez mais otimizada e proveitosa (NUNES, 2007).

e) Informações de contexto relevantes

Para definir as informações de contexto relevantes é preciso levar em consideração o foco de atenção (BREZILLON E POMEROL, 1999). O contexto não pode ser considerado de forma isolada, mas sempre de acordo com um foco de atenção que vai determinar o que é relevante em uma determinada situação. Neste trabalho, o foco de atenção é determinado pelos dois cenários de uso da abordagem descritos anteriormente, onde se destacam as atividades de definição do processo e adaptação do processo. Cada um dos cenários possui um conjunto de informações de contexto relevantes.

f) Modelo de representação de contexto

Para capturar e usar informações de contexto é necessário especificar como elas serão representadas em um modelo compreensível e aceitável por todos (Figura 7a). Uma representação formal do contexto permite que mecanismos de raciocínio lógico possam

ser usados para checar a consistência das informações de contexto, possam realizar comparações com outros contextos e possam inferir novas informações complexas a partir dos contextos existentes (NUNES, 2007).

Neste sentido, para usar informações de contexto é importante conhecer e representar a sua semântica. A elaboração de um modelo de contexto com riqueza semântica (o nível de riqueza deve ser tão alto quanto for a necessidade de expressividade desta semântica) ajuda a capturar a estrutura e o significado das informações de contexto e do relacionamento entre elas, e a identificar como essas informações devem ser manipuladas (FERNANDES, 2008). A semântica de uma linguagem não pode ser tratada de maneira informal e superficialmente (HAREL E RUMPE, 2004). É extremamente importante, estabelecer o significado e as relações semânticas entre os conceitos utilizados para representar informações de contexto criando uma visão unificada, pois isso viabiliza a sua utilização de maneira uniforme e permite a construção e utilização de mecanismos de raciocínio com mais segurança.

5.4. Informações de contexto

Como ponto de partida para definição das informações de contexto relevantes para este trabalho de pesquisa, Araujo et al. (2004) sugerem as seguintes dimensões de contexto para o desenvolvimento de software (Figura 10): indivíduo, papel, equipe, tarefa, projeto, organização, domínio da engenharia de software, produto de software, domínio do negócio e cliente. Em especial, as dimensões organização, projeto e equipe são relevantes para o presente trabalho de pesquisa. As dimensões organização e projeto devem ser consideradas, visto que a adaptação de processos, geralmente, é realizada nestes dois níveis (PEDREIRA ET AL., 2007). A dimensão equipe também é importante, dado o interesse deste trabalho de pesquisa no aspecto da colaboração.

Para cada uma destas dimensões de contexto selecionadas, pode-se pensar em um conjunto de informações de contexto, de acordo com os aspectos de colaboração e disciplina dos processos de desenvolvimento de software. A determinação das informações de contexto não é uma atividade trivial. Vários tipos de informações contribuem com o contexto e a relevância de cada informação depende do foco de atenção em questão (BREZILLON, 1999).

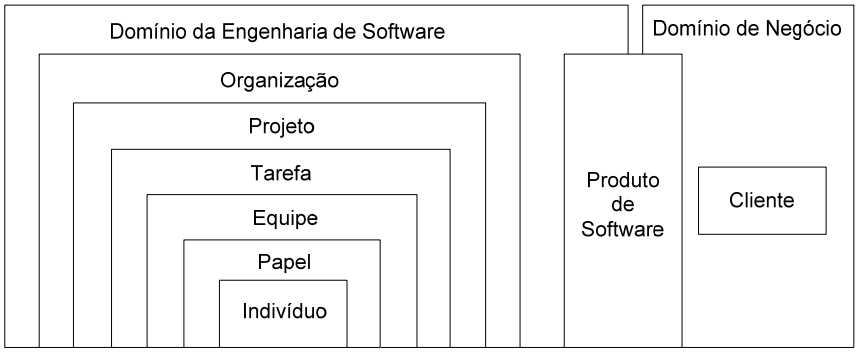

Figura 10 – Dimensões de contexto no desenvolvimento de software. Adaptado de: (ARAUJO ET AL., 2004)

Exemplos de informações de contexto são apresentados na Tabela 1. Estes exemplos foram obtidos a partir da adaptação de trabalhos publicados na literatura (ABRANTES, 2009, BOEHM E TURNER, 2003, GINSBERG E QUINN, 1995, LAANTI E KETTUNEN, 2005, LITTLE, 2005) ou através da experiência dos autores deste trabalho, mas ainda são necessários outros estudos para completar o levantamento e validar, junto aos especialistas, este conjunto de informações de contexto.

Além disso, também é necessário estabelecer uma metodologia para o levantamento sistemático destas informações de contexto, utilizando técnicas como revisão sistemática da literatura (BIOLCHINI ET AL., 2005, KITCHENHAM, 2004), *survey* com especialistas, *grounded theory* (GLASER E STRAUSS, 1967), levantamento de requisitos ou levantamento de conhecimento. A definição desta metodologia será alvo de trabalho futuros.

Há ainda outra questão relacionada à granularidade e complexidade da informação de contexto. Uma informação de contexto pode ser ou não atômica. Uma informação de contexto atômica já representa a menor unidade. Quando a informação de contexto não é atômica, ela é composta por outras informações de contexto.

Cada informação de contexto possui um conjunto de valores nominais possíveis de serem assumidos. Neste trabalho, os valores nominais foram organizados de acordo com a sua inclinação à colaboração, o que pode provocar inversões nas escalas. As próximas seções enumeram as informações de contexto, agrupadas de acordo com as suas respectivas dimensões.

a) Dimensão organização

A dimensão organização descreve os objetivos de negócio e o processo padrão da empresa (ARAUJO ET AL., 2004, SANTORO ET AL., 2007).

- **Estrutura organizacional:** a estrutura organizacional é utilizada para estabelecer responsabilidades, distribuir autoridade e alocar os recursos da organização (VASCONCELLOS E HEMSLEY, 2003). A estrutura organizacional é caracterizada pelo seu grau de complexidade. Esta complexidade pode ser medida pelo número de níveis hierárquicos existentes, desde o topo aos níveis mais baixos da organização. A quantidade e a rigidez dos níveis hierárquicos influenciam diretamente na colaboração, pois impactam nos aspectos de comunicação e coordenação.
  - **Complexidade**
    - Muito baixa;
    - Baixa;
    - Média;
    - Alta;
    - Muito alta.
  - **Rigidez**
    - Muito baixa;
    - Baixa;
    - Média;
    - Alta;
    - Muito alta.
- **Cultura organizacional:** Os valores e normas de uma organização são estabelecidos e reforçados ao longo do tempo. A cultura organizacional determina a forma de pensar e o comportamento de uma organização e difere de uma organização para outra (SCHEIN, 1990). A cultura organizacional exerce uma influência considerável no processo de tomada de decisões, nas estratégias de solução de problemas, nas práticas de inovação, nas negociações sociais, nos relacionamentos e nos mecanismos de planejamento e controle. Organizações que possuem culturas organizacionais mais informais, cooperativas e com maior autonomia de seus funcionários são ambientes onde práticas colaborativas são mais bem recebidas, do que organizações com culturas

formais, rígidas e burocráticas, que, em geral, prezam maior visibilidade e controle dos projetos e das equipes de desenvolvimento.

- **Formalismo**
  - Muito baixo;
  - Baixo;
  - Médio;
  - Alto;
  - Muito alto.

• **Gestão de conhecimento:** processo pelo qual o conhecimento é adquirido, transformado e disseminado. O conhecimento se distingue em tácito e explícito. O conhecimento tácito é altamente pessoal, específico do contexto e difícil de formalizar, o que dificulta sua transmissão e compartilhamento com outros. O conhecimento explícito é formal e fácil de ser comunicado ou registrado em diferentes tipos de artefatos (NONAKA E TAKEUCHI, 1995).

- **Formalismo**
  - Baixo - conhecimento tácito não disseminado;
  - Médio - socialização - conhecimento tácito transmitido;
  - Alto - externalização - formalização do conhecimento tácito em explícito.

• **Objetivo de negócio:** toda organização possui os seus objetivos de negócio, que estão relacionados ao seu planejamento estratégico. Os objetivos de negócio podem envolver diminuição de custos, aumento da qualidade, conquista de um novo segmento de mercado através do lançamento rápido de um produto, ou até inovação, através do uso de uma nova tecnologia. Todos estes aspectos devem ser levados em consideração no momento da adaptação do processo.

- Inovação;
- Conquista de um novo segmento de mercado;
- Aumento da qualidade;
- Diminuição de custos.

• **Nível de maturidade atual:** uma preocupação comum das organizações ao adotarem um novo processo de desenvolvimento de software é manter o seu nível de maturidade atual para que possam preservar as suas certificações nos modelos de maturidade. Como

cada nível de maturidade e modelo de qualidade impõe restrições significativas à adaptação do processo, esta informação de contexto também precisa ser observada.

- Nível 1 - Inicial;
- Nível 2 - Gerenciado;
- Nível 3 - Definido;
- Nível 4 - Gerenciado quantitativamente;
- Nível 5 - Otimizado.

• **Nível de maturidade desejado:** de forma análoga, se a organização tem o interesse em alcançar um novo nível de maturidade, tal fato também não pode ser ignorado durante a adaptação do processo, pois será necessário oferecer um processo que suporte esta iniciativa da empresa.

- Nível 1 - Inicial;
- Nível 2 - Gerenciado;
- Nível 3 - Definido;
- Nível 4 - Gerenciado quantitativamente;
- Nível 5 - Otimizado.

• **Processo padrão:** o processo padrão da organização impõe requisitos e restrições ao processo que será definido para o projeto.

- **Dificuldade de adaptação**
  - Muito baixa;
  - Baixa;
  - Média;
  - Alta;
  - Muito alta.

b) Dimensão projeto

A dimensão projeto compreende também o contexto do produto a ser construído, seus objetivos e o processo a ser seguido (ARAUJO ET AL., 2004).

•**Relacionamento com o cliente:** compreende o nível de participação e disponibilidade do cliente e o nível de confiança existente entre a equipe de desenvolvimento e o cliente. Também é importante levar em consideração experiências anteriores de trabalho

com este determinado cliente e a existência de um único ou de múltiplos representantes do cliente.

- **Envolvimento**
  - Muito alto;
  - Alto;
  - Médio;
  - Baixo;
  - Muito baixo.
- **Número de representantes**
  - Um;
  - Múltiplos.
- **Experiência no domínio**
  - Muito alta - especialista;
  - Alta - muito experiente;
  - Média - experiente;
  - Baixa - pouco experiente;
  - Muito baixa - sem experiência.
- **Habilidade de expressar requisitos**
  - Muito alta - expressivo;
  - Alta - muito comunicativo;
  - Média - comunicativo;
  - Baixa - pouco comunicativo;
  - Muito baixa - sem habilidade.

• **Tamanho do problema:** o tamanho do problema a ser resolvido pelo projeto pode resultar em uma necessidade de maior gerenciamento do processo ou na necessidade de dividi-lo em problemas menores que serão resolvidos em fases.

  - Muito baixo;
  - Baixo;
  - Médio;
  - Alto;
  - Muito alto.

• **Complexidade do problema:** a complexidade do problema compreende o grau de dificuldade para a sua realização e está relacionado a natureza do projeto. Logo, quanto menos complexo, mais adequado à colaboração.

- Muito Baixa - projetos que não envolvem cálculos complexos;
- Baixa - projetos que envolvem cálculos pouco complexos;
- Média - projetos que envolvem cálculos complexos;
- Alta - projetos que envolvem cálculos razoavelmente complexos;
- Muito alta - projetos que envolvem cálculos muito complexos.

• **Duração:** o tempo previsto (em número de meses) para a duração de um projeto pode influenciar a adaptação do processo para o projeto. Em projetos longos, é necessário um cuidado maior com a gestão do conhecimento, pois aumenta o risco de mudanças nos membros da equipe de desenvolvimento. Além disso, também aumentam as chances do projeto ser afetado pela estabilidade dos requisitos e pela incerteza técnica.

- Até 2 meses;
- De 2 a 6 meses;
- De 6 a 12 meses;
- De 12 a 24 meses;
- Mais que 24 meses.

• **Criticidade:** a criticidade do projeto está relacionada ao risco envolvido em caso de falha do produto desenvolvido. Assim, em projetos onde existe um alto risco, seja de vidas humanas ou de funções críticas ao negócio, é necessário maior controle sobre o processo de desenvolvimento de software que será utilizado do que nos casos onde o risco envolve somente pequenas perdas financeiras.

- Muito baixa - em caso de falha do sistema, o trabalho precisará ser feito manual ou informalmente;
- Baixa - perda de valores financeiros discretos;
- Média - perda do investimento do projeto;
- Alta - perda de quantias significativas ou de funções críticas do negócio;
- Muito alta - perda de vidas humanas.

• **Novidade:** a novidade corresponde ao grau de inovação existente no projeto de desenvolvimento. Projetos com características de inovação não são o ambiente mais

adequado para impor muito formalismo ao processo de desenvolvimento de software, pois a equipe precisa de mais liberdade para trabalhar de forma colaborativa e criativa.

- Muito alta;
- Alta;
- Média;
- Baixa;
- Muito baixa.

• **Tecnologia de desenvolvimento do produto:** a escolha de uma tecnologia para o desenvolvimento do produto influencia na adaptação do processo de desenvolvimento, pois é necessário levar em consideração se esta tecnologia é uma novidade ou se já está difundida na organização e entre a equipe de desenvolvimento.

- **Maturidade**
  - Muito alta;
  - Alta;
  - Média;
  - Baixa;
  - Muito baixa.

• **Ambiente tecnológico de desenvolvimento:** o ambiente tecnológico de desenvolvimento corresponde ao suporte computacional (ferramentas, compiladores etc.) utilizado para a construção do produto.

- **Maturidade**
  - Muito alta;
  - Alta;
  - Média;
  - Baixa;
  - Muito baixa.

• **Estabilidade dos requisitos:** a estabilidade dos requisitos está relacionada à frequência de mudanças destes requisitos. Em projetos com requisitos voláteis e dinâmicos, são mais indicadas características de agilidade e colaboração com o cliente, do que tentar manter um excesso de formalismo no processo para gerenciar estas mudanças de requisitos.

- Muito Baixa - requisitos muito instáveis;

- Baixa - requisitos instáveis;
- Média - requisitos com grau médio de estabilidade;
- Alta - requisitos praticamente estáveis;
- Muito alta - requisitos estáveis.

• **Tipo de desenvolvimento:** o tipo de desenvolvimento do software pode ser exploratório, interno, para o mercado ou para um cliente específico. Dependendo do tipo de desenvolvimento pode existir mais espaço para a colaboração ou mais necessidade de disciplina.

- Projeto de pesquisa;
- Projeto interno à organização;
- Desenvolvimento do produto para o mercado;
- Desenvolvimento contratado por um cliente em particular.

• **Frequência de lançamento de versões:** assiduidade prevista para o lançamento de versões de acordo com a necessidade de *feedback* dos usuários. Esta informação ajuda na adaptação do processo ao apontar se as atividades deverão ser combinadas de forma sequencial ou iterativa.

- Muito alta;
- Alta;
- Média;
- Baixa;
- Muito baixa.

c) Dimensão equipe

O contexto de uma equipe reflete a agregação das características de seus participantes e papéis. As equipes são estabelecidas dentro de um projeto de desenvolvimento para executar tarefas específicas. Assim, equipes formadas por diferentes participantes irão planejar e executar as suas tarefas de formas diferentes (ARAUJO ET AL., 2004, SANTORO ET AL., 2007).

• **Tamanho da equipe:** o tamanho da equipe está relacionado à quantidade de pessoas que a compõem. Possivelmente, equipes maiores vão precisar de mais disciplina, enquanto equipes menores conseguem trabalhar de forma mais colaborativa.

- Até 6 pessoas;
- De 7 a 10 pessoas;
- De 11 a 39 pessoas;
- De 40 a 80 pessoas;
- Mais de 80 pessoas.

• **Experiência técnica:** a experiência técnica da equipe no desenvolvimento de produtos de software pode determinar a possibilidade de oferecer mais colaboração ou mais disciplina ao processo definido para o projeto. As equipes mais experientes tecnicamente, talvez já não precisem mais de tanto rigor no seu processo de desenvolvimento de software, pois já são capazes de antecipar as atividades, o que não acontece com uma equipe de novatos.

- **Experiência no domínio da aplicação**
  - Muito alta - especialista;
  - Alta - muito experiente;
  - Média - experiente;
  - Baixa - pouco experiente;
  - Muito baixa - sem experiência.
- **Experiência no desenvolvimento de software**
  - Muito alta - especialista;
  - Alta - muito experiente;
  - Média - experiente;
  - Baixa - pouco experiente;
  - Muito baixa - sem experiência.

• **Experiência no trabalho em conjunto:** da mesma forma, equipes onde os seus membros já estão acostumados a trabalhar em conjunto podem já estar mais voltadas à colaboração, enquanto outras podem precisar ainda de incentivo a colaboração para que os seus membros se integrem.

- Muito alta;
- Alta;
- Média;
- Baixa;
- Muito baixa.

• **Proximidade:** identificar se os membros da equipe estão localizados fisicamente próximos ou se esta equipe trabalha geograficamente distribuída é importante para balancear a dosagem de colaboração e disciplina necessárias. Uma equipe que trabalha reunida em um mesmo local tem maior facilidade de comunicação e coordenação do que uma equipe dispersa em locais diferentes.

- Localizada;
- Distribuída.

• **Estabilidade:** a alta rotatividade dos membros da equipe de desenvolvimento pode caracterizar uma necessidade de maior formalismo para evitar perda de conhecimento e diminuir os riscos para o projeto e a organização.

- Muito alta;
- Alta;
- Média;
- Baixa;
- Muito baixa.

A Tabela 1 resume as informações de contexto, porém vale ressaltar que nenhuma dessas informações de contexto sozinha é determinante para definir se um processo terá mais colaboração ou mais disciplina. O balanceamento será obtido pelas regras de contexto.

Dimensão	Informações de Contexto
Organização	Estrutura organizacional, Cultura organizacional, Gestão de conhecimento, Objetivo de negócio, Nível de maturidade atual, Nível de maturidade desejado e Processo padrão
Projeto	Relacionamento com o cliente, Tamanho do problema, Complexidade do problema, Duração, Criticidade, Novidade, Tecnologia de desenvolvimento do produto, Ambiente de desenvolvimento, Estabilidade dos requisitos, Tipo de desenvolvimento e Frequência de lançamento de versões
Equipe	Tamanho, Experiência técnica, Experiência no trabalho em conjunto, Proximidade, Estabilidade, Comunicação, Coordenação, Memória e Percepção.

Tabela 1 – Exemplos de informações de contexto

5.5. Situações de contexto

Para o entendimento destas informações de contexto dentro do domínio de processos de desenvolvimento de software, é necessário estabelecer algumas situações que impactem no estabelecimento de regras ou sugestões de combinações de práticas. Alguns exemplos destas situações são apresentados na Tabela 2.

Alguns projetos de desenvolvimento de software adotam uma política de controle de concorrência otimista. No momento do *check-in*, a ferramenta de controle de versão consegue detectar caso mudanças tenham sido feitas em paralelo no mesmo arquivo e, na integração, o desenvolvedor deve resolver os possíveis conflitos que não foram mesclados automaticamente pela ferramenta. Apesar do rápido ciclo de desenvolvimento e do grande número de desenvolvedores alterando o código, os conflitos ocorrem raramente, pois as listas de discussões são utilizadas para fornecer percepção sobre o trabalho que está em andamento por outros desenvolvedores. Assim, os desenvolvedores se comunicam e resolvem possíveis problemas, diminuindo o retrabalho na integração. Este cenário caracteriza a primeira situação de Conflitos raros (Tabela 2).

A segunda situação (Ambiente organizacional favorável a colaboração) corresponde ao caso em que a estrutura organizacional da empresa, utilizada para estabelecer responsabilidades, distribuir autoridade e alocar os recursos da organização (VASCONCELLOS E HEMSLEY, 2003), tem uma complexidade e uma rigidez baixas (Tabela 2). Esta complexidade pode ser medida pelo número de níveis hierárquicos existentes, desde o topo aos níveis mais baixos da organização. A quantidade e a rigidez dos níveis hierárquicos influenciam diretamente na colaboração, pois trazem dificuldades na coordenação e na comunicação. Além disso, a cultura organizacional é informal, ou seja, os funcionários têm maior autonomia e as práticas colaborativas são mais bem recebidas.

A terceira situação representa um cenário muito comum nas organizações que já trabalham orientadas a processos e possuem uma grande preocupação com a certificação dos seus processos de acordo com determinados modelos de maturidade ou normas de qualidade, tais como o CMMI (CHRISSIS ET AL., 2006), a ISO/IEC 12207 (ISO/IEC, 2007) e o MPS-BR (SOFTEX, 2009). Nesta situação, a organização já

alcançou um determinado nível de maturidade e tem a intenção de manter ou até mesmo evoluir nesta maturidade (Tabela 2). Para isso, é necessário certo grau de formalismo no controle dos processos e as organizações geralmente recorrem à definição de um processo padrão (GINSBERG E QUINN, 1995, PEDREIRA ET AL., 2007) inspirado em um ou mais modelos de referência, com as práticas que se deseja incorporar em todos os projetos para guiar o desenvolvimento dentro da organização.

Situação	Dimensão	Informação de Contexto
Conflitos raros	Equipe	Tamanho da equipe: >= 40 pessoas Comunicação da equipe: alta Percepção da equipe: alta
Ambiente organizacional favorável a colaboração	Organização	Estrutura organizacional – Complexidade: baixa Estrutura organizacional – Rigidez: baixa Cultura organizacional – Formalismo: baixo
Preocupação com certificação	Organização	Nível de maturidade atual: >= 2 Nível de maturidade desejado: >= 2 Processo padrão: definido
Projeto governado por definições contratuais	Projeto	Risco do projeto: alto Tamanho do projeto: alto Complexidade do projeto: alta Relacionamento com cliente – confiança: baixa
Desenvolvimento distribuído	Equipe	Tamanho da equipe: >= 40 pessoas Proximidade da equipe: distribuída
Carência de gestão de conhecimento	Organização	Gestão de conhecimento - formalismo: baixo
	Projeto	Duração: >=30 meses
	Equipe	Estabilidade: baixa

Tabela 2 – Exemplos de situações de contexto

A quarta situação foi criada tendo como inspiração o cenário histórico que motivou a criação dos modelos de qualidade. Neste cenário, o projeto tem um alto risco, seja de perda de vidas humanas ou de comprometimento de funções críticas para o negócio.

Além disso, trata-se de um projeto grande e complexo, onde o relacionamento entre cliente e equipe de desenvolvimento se caracteriza pelo baixo nível de confiança (Tabela 2). Desta forma, o projeto acaba sendo governado por definições contratuais (GLAZER ET AL., 2008).

Na quinta situação foi colocado um caso mais simples que manipula apenas duas informações de contexto. Nesta situação, existe uma grande equipe de desenvolvimento trabalhando de forma distribuída (Tabela 2), ou seja, trata-se de um cenário típico de desenvolvimento distribuído de software.

Por fim, a sexta e última situação representa um caso mais complexo que envolve informações de contexto das três dimensões. Neste cenário existe a Carência de Gestão de Conhecimento, pois ela é feita informalmente na organização e a equipe tem uma estabilidade baixa, ou seja, nesta equipe é comum a rotatividade dos seus membros (Tabela 2). Além disso, trata-se de um projeto de longa duração que tem maiores chances de ser afetado por estas mudanças na equipe.

6. Linguagens de representação

A representação de conhecimento é uma área originada na inteligência artificial (MIRA, 2008) preocupada com a formalização semântica do raciocínio, o que significa compreender como utilizar símbolos para representar um domínio de conhecimento. De maneira geral, deve ser possível aplicar algum tipo de lógica a essa representação, de forma a permitir que um sistema raciocine em cima dos símbolos descritos e responda perguntas.

As informações de contexto devem ser representadas através de uma linguagem de representação com semântica e formalismo que atendam as necessidades de representação. A expressividade sintática e semântica de uma linguagem de representação é extremamente importante, pois ela determina o nível de adequabilidade da linguagem para representar algumas abstrações da realidade (GUIZZARDI, 2005), tornando “mais claro dizer alguma coisa”. Em contrapartida, uma linguagem altamente expressiva dificulta o raciocínio lógico automatizado.

Na presente análise, foram levados em consideração requisitos associados à inteligibilidade tanto pelos humanos quanto por máquinas. Uma linguagem de modelagem, conforme afirma Guizzardi (2005), por um lado, deve ser suficientemente expressiva para caracterizar de forma adequada (em termos de riqueza sintática e semântica para representar definições, propriedades, relacionamentos, relações de dependência) a conceituação de um domínio (e do contexto do domínio). Por outro deve possuir uma semântica clara e não ambígua, ou seja, deve ser possível para um analista reconhecer o significado dos construtos existentes e saber usá-los na modelagem. Além disso, a linguagem deve ser computacionalmente interpretável.

Neste trabalho, deseja-se adotar linguagens que possuam graus de completude e consistência necessários e suficientes de forma a serem interpretadas por sistemas capazes de prover uma representação lógica e gráfica e que possuam mecanismos para realizar raciocínio lógico baseado em regras aplicadas às informações de contexto modeladas.

As formas de representação de contexto podem ser classificadas em diferentes categorias, com base na estrutura de dados utilizada para representar as informações de contexto. Fernandes (2008) e Santos (2008) descrevem cada uma dessas técnicas em detalhes e analisam as vantagens e desvantagens de cada uma. Levando-se em consideração os resultados desta análise, as duas linguagens de representação de informações de contexto, selecionadas para estudo e análise de viabilidade de uso, foram: ontologias e modelos de características.

6.1. Ontologias

Noy e McGuinness (2001) reconhecem o uso de ontologias como forma de compartilhar um entendimento comum entre pessoas e agentes de software. De acordo com (GRUBER, 1995), ontologias podem ser vistas como uma especificação formal de conceitos e termos do domínio e definem regras que regulam a combinação entre termos.

Existem diversas classificações para ontologias, mas a classificação adotada aqui é a sugerida por Guarino (1998) quanto à *generalidade*, na qual, como mostra a Figura 11, ontologias podem ser classificadas em: (i) ontologias de fundamentação, que descrevem

conceitos muito gerais, como espaço, tempo, problema, objeto, evento, ação etc.; (ii) ontologias de domínio, que descrevem o vocabulário relacionado a um domínio genérico como, por exemplo, desenvolvimento de software; (iii) ontologias de tarefa, que descrevem o vocabulário relacionado a uma tarefa genérica, como, por exemplo, diagnose, venda etc. e (iv) ontologias de aplicação, que descrevem conceitos dependentes de um domínio e uma tarefa particulares, os quais são, frequentemente, especializações de ontologias relacionadas. Dentre estes tipos, neste trabalho será utilizada a ontologia de domínio.

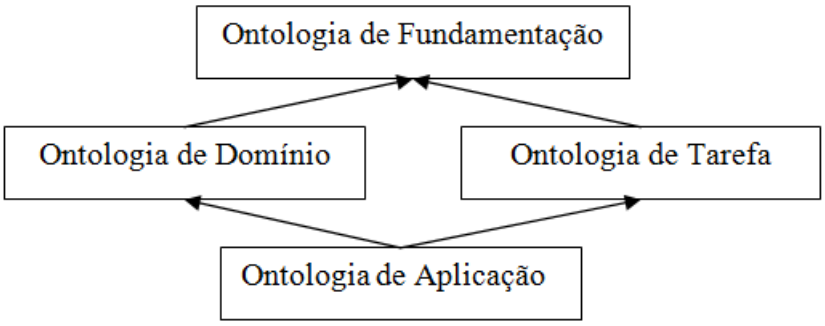

Figura 11 - Classificação de ontologias proposta por (GUARINO, 1998)

Ontologias de domínio explicitam o entendimento do domínio, permitindo reutilização, separação do conhecimento operacional e análise sobre os conceitos do domínio. Elas são formadas por abstrações que refletem o consenso entre os participantes de um domínio: conceitos (ou classes), relacionamento entre classes, restrições e axiomas do domínio, utilizam organização taxonômica baseada em generalização e especialização (superclasse/subclasse), possuem aspectos composicionais (relacionamentos do tipo todo/parte) e funções não taxonômicas (propriedades) (CAPPELLI ET AL., 2007).

A aplicação de ontologias é uma área em constante estudo e com vastas possibilidades já identificadas na literatura (GUIZZARDI, 2005). Neste trabalho, a proposta do uso de ontologias na análise de viabilidade surgiu a partir da definição proposta por (FALBO, 1998), na identificação dos conceitos e dos relacionamentos entre eles presentes no domínio de processos de desenvolvimento de software, e da definição de uma ontologia de contexto para processos de trabalho proposta por Nunes (2007).

Para representar ontologias, Guizzardi (2007) afirma que um dos principais fatores de sucesso no uso de uma linguagem de representação reside na habilidade da própria

linguagem prover aos usuários primitivas de modelagem que sejam capazes de expressar diretamente conceitos relevantes de um domínio.

Para este estudo de viabilidade, foi selecionada a linguagem OWL (*Web Ontology Language*) que é a linguagem de representação de ontologias mais utilizada, segundo pesquisa realizada em (CARDOSO, 2007) e recomendação do World Wide Web Consortium (W3C, 2004a), e possui um conjunto de construtos, descritos de forma completa no OWL Guide (SMITH ET AL., 2004).

Existem três tipos de linguagens em que OWL se classifica (SMITH ET AL., 2004, W3C, 2004a): OWL Lite, que possui os construtos mais simples da linguagem (LOPES, 2009); OWL DL (*Description Logics*), que possui o conjunto completo de construtos (LOPES, 2009); e OWL Full, que assim como a OWL DL, possui o conjunto completo de construtos OWL, porém permite que uma classe seja identificada como uma instância, assim como uma instância como uma classe, impedindo que seu tratamento computacional tenha um tempo definido (LOPES, 2009).

Neste trabalho foi utilizada a linguagem OWL DL (*Description Logics*, ou Lógicas de Descrição) (W3C, 2004a), que atende as necessidades de representação, pois possui restrições de cardinalidade completas, operações de conjunto (união, complemento, interseção e classes enumeradas), disjunção, axiomas de intervalo de valor para propriedade de dados, restrição de um conjunto de instâncias específicas para uma propriedade (allValuesFrom, someValuesFrom), definindo classes com essa restrição, e regra de classificação de uma instância caso possua um valor específico em uma propriedade (hasValue), características que não estão presentes na OWL Lite. Esta linguagem foi utilizada tanto para a modelagem dos conceitos em um domínio de processos de desenvolvimento de software quanto para a modelagem de informações de contexto do domínio.

A motivação, segundo Nunes (2007), para o uso de ontologias na formalização de modelos de contexto está fundamentada nas seguintes questões:

- **Compartilhar um entendimento comum em relação à estrutura da informação, entre pessoas ou agentes computacionais**: é uma das principais razões para o desenvolvimento de ontologias. Ela é mais que um vocabulário

padrão, pois assegura que os termos escolhidos sejam suficientes para especificar e definir conceitos e permitir relacionamentos adequados a partir da escolha terminológica realizada.

- **Permitir reutilização dentro do domínio de conhecimento:** a construção de ontologias a partir de ontologias pré-existentes tem sido largamente pesquisada e visa aproveitar conceituações previamente estabelecidas. Se for necessário construir uma ontologia de grande porte, podem ser integradas diversas ontologias já existentes que descrevem parte de um domínio mais amplo. Pode-se também reutilizar uma ontologia mais geral e estendê-la para descrever um domínio de interesse.

- **Tornar concepções acerca do domínio (e do contexto do domínio) explícitas:** desta forma, é possível alterar essas concepções facilmente se o conhecimento sobre o domínio (e seu contexto) se modificar. Especificações explícitas do domínio de conhecimento são úteis para os novos usuários que desejam aprender o significado dos seus termos.

- **Analisar o conhecimento do domínio (e do contexto do domínio):** isto é possível, pois uma especificação declarativa dos termos está disponível. Segundo McGuinness et al. (2000) análises formais de termos possuem grande valor quando tentam reutilizar ontologias existentes e as estender.

- **Permitir o compartilhamento do conhecimento:** a utilização de ontologias de contexto permite que entidades computacionais como agentes ou serviços apresentem informações de contexto enquanto interagindo com o usuário.

- **Permitir o uso de mecanismos de inferência para raciocinar sobre vários contextos:** mecanismos computacionais baseados em ontologias podem explorar várias formas existentes de mecanismos de raciocínio lógico para deduzir informações e regras acerca dos contextos existentes.

Conforme afirma Santos (2008), um grande número de modelos de contexto baseados em ontologias já foram propostos em diferentes áreas. Porém, as ferramentas e padrões

para manipular ontologias ainda não são de fácil uso e os mecanismos de inferência impactam diretamente no desempenho das aplicações.

6.2. Modelagem de características

A modelagem de características é uma abordagem que trata da complexidade em expressar requisitos em forma de características e da sua estruturação hierárquica em diagramas de características (MASSEN E LICHTER, 2004). Características (do inglês *features*) são definidas por Czarnecki e Eisenecker (2000) como um "um aspecto, uma qualidade, ou uma característica visível ao usuário, proeminente ou distinta, de um sistema (ou sistemas) de software". As características são tipicamente organizadas em um diagrama de características, através de uma estrutura hierárquica, como uma árvore ou grafo acíclico, dependendo da notação utilizada.

O propósito da modelagem de características é "capturar e gerenciar as similaridades e diferenças, de forma a facilitar o entendimento de clientes e desenvolvedores no que se refere às capacidades gerais de um domínio, que são expressas através de características" (KANG ET AL., 1990). Uma vez que o modelo de características é um modelo de alto nível de abstração e o ponto de partida para o recorte necessário à instanciação dos processos, existem diversas notações (OLIVEIRA, 2006, TEIXEIRA, 2008) que se propõem a representar a variabilidade no modelo de características para lidar com as diferenças e características comuns entre os produtos de software.

A variabilidade pode ser entendida como a habilidade que um artefato de software possui de ser alterado, customizado ou configurado para um contexto em particular (BOSCH, 2004). Alguns conceitos são inerentes à variabilidade e a sua modelagem, tais como (OLIVEIRA, 2006):

- **Pontos de variação:** um ponto de variação é a característica que reflete a parametrização no domínio de uma maneira abstrata. Um ponto de variação é configurável através das variantes.

- **Variantes:** as variantes são as características que atuam como alternativas para se configurar um ponto de variação.

• **Invariantes:** as invariantes são as características fixas, que representam elementos não configuráveis em um domínio.

Ortogonalmente, em relação à opcionalidade, as características podem ser classificadas como (BLOIS, 2006, LINDEN ET AL., 2007, OLIVEIRA, 2006)

• **Elementos opcionais:** os elementos opcionais podem ou não estar presentes no domínio.

• **Elementos mandatórios:** os elementos mandatórios devem obrigatoriamente estar presentes em um domínio.

Um dos conceitos relacionados à modelagem de características é a cardinalidade, que é utilizada para definir o número mínimo e máximo de características que podem ser escolhidas a partir de um conjunto de alternativas de um ponto de variação. Outro conceito importante é a especificação de restrições entre as características, que inclui os relacionamentos de dependência e mútua exclusividade. Esta é uma forma de indicar a necessidade ou incompatibilidade da seleção conjunta de características. Estas regras de composição podem ser de dois tipos (OLIVEIRA, 2006):

•**Inclusivas:** definem relações de dependência entre duas ou mais características, indicando que elas devem ser selecionadas em conjunto.

•**Exclusivas:** definem relações de mútua exclusividade entre características, indicando que duas ou mais características não devem ser escolhidas em conjunto.

Neste trabalho, o conceito de características foi utilizado em um domínio de processos de desenvolvimento de software. Uma vez descrito o domínio em termos de características, estas são combinadas de forma a construir configurações viáveis de práticas ou atividades relacionadas aos três modelos de desenvolvimento de software tratados neste trabalho.

Para a modelagem de domínio foi selecionada a linguagem Odyssey-FEX (FEX – *Feature Extended*) (OLIVEIRA, 2006), que estende os elementos do modelo de características do ambiente Odyssey (ODYSSEY, 2010) definidos originalmente na proposta de Miler (2000), para abranger elementos que representam conceitos,

funcionalidades e tecnologias, incluindo a variabilidade e os relacionamentos entre eles. Oliveira (2006) realizou uma análise entre as notações mais comumente utilizadas (FODA, FORM, FeatureRSEB, Svanhberg & Bosh, Riebisch, Cechticky, Czarnecki e Odyssey) de forma a resolver na Odyssey-FEX algumas das deficiências existentes nestas linguagens.

Além disso, para a modelagem de contexto, foi utilizada a notação UbiFEX (FERNANDES, 2008). A UbiFEX é uma notação para a modelagem de características que inclui de forma explícita elementos para modelagem das entidades e informações de contexto relevantes para um determinado domínio. Além disso, inclui elementos para representar a influência dessas informações na configuração dos produtos. A UbiFEX-estende a notação Odyssey-FEX (OLIVEIRA, 2006) com o propósito de permitir a representação de contexto em modelos de características.

Abordagens que focam em sistematizar a reutilização, além de representar os conceitos e informações fundamentais e obrigatórios para o entendimento do domínio, devem fornecer uma ampla perspectiva de reutilização e adaptação. Segundo Lee et al. (2002), existem algumas razões para o uso extensivo e popularidade deste tipo de modelo, tanto na indústria quanto na academia:

- Características funcionam como uma **forma de comunicação efetiva**, sobre as características do produto, entre os diferentes *stakeholders*. Eles comunicam requisitos ou funções em termos de características, pois estas representam uma abstração, capaz de ser entendida tanto pelos clientes e usuários quanto pelos desenvolvedores, do que deve ser implementado, testado, implantado e mantido;

- Uma análise de domínio orientada a características é uma forma natural e intuitiva de **identificar variabilidades e similaridades** entre diferentes produtos de um domínio;

- O modelo de características pode fornecer uma base para **desenvolver, parametrizar e configurar vários ativos reutilizáveis**. Este modelo tem um papel fundamental na gerência e configuração de múltiplos produtos em um domínio.

Resumindo, não existe uma linguagem de representação de conhecimento que seja sempre necessariamente melhor do que outra. A escolha de uma linguagem de representação depende do uso que se pretende fazer dela. A partir da análise das semelhanças e diferenças entre o modelo de características e as ontologias, podemos considerar que o modelo de características implementa em parte os conceitos e restrições utilizados nas ontologias. O modelo de características é uma descrição dos termos e o relacionamento entre eles, formando uma rede semântica. Esta conceituação de modelo de características é exatamente a conceituação clássica utilizada para a descrição de uma ontologia.

No entanto, o modelo de características não possui o formalismo de uma ontologia, pois não são utilizados axiomas formais para se descrever os termos e relacionamentos do domínio. Apesar do modelo de características representar restrições entre características, não é utilizado nenhum formalismo nesta modelagem, o que dificulta sua verificação. Todavia, em relação ao propósito final, que é o entendimento do domínio e seus termos, ambas as abordagens são similares (BRAGA, 2000).

Analisando o modelo de características sob uma ótica de recuperação de informação, podemos dizer que este modelo é bastante semelhante a uma ontologia, no sentido de ser um modelo que descreve um determinado domínio em um nível conceitual alto e onde o mapeamento entre as informações a serem recuperadas e os termos do modelo ontológico são especificados. Desta forma, o modelo de características atende a este propósito, tendo como objetivo a modelagem de um domínio em alto nível de abstração, agregando aos termos informações que permitam o seu entendimento e o mapeamento destas para outras informações do domínio (rastreabilidade entre modelos) (BRAGA, 2000).

7. Ferramentas analisadas

A análise da linguagem mais adequada para realizar a modelagem de conceitos relacionados ao domínio de processos de desenvolvimento de software, envolvendo os três modelos (orientado ao planejamento, ágil e livre) e as informações de contexto referentes às dimensões de contexto equipe, projeto e organização, deve levar em consideração o equilíbrio entre a qualidade semântica da linguagem de representação

implementada e as funcionalidades oferecidas por uma ferramenta que a implemente. O ferramental de apoio adequado é necessário para gerenciar os conceitos do domínio e as informações de contexto e os seus relacionamentos e dependências para ajudar a garantir a consistência entre os modelos (HUBAUX ET AL., 2010, MASSEN E LICHTER, 2004).

Baseado na ontologia para o domínio de processos de software**Erro! Fonte de referência não encontrada.**, nas dimensões de contexto para o desenvolvimento de software (Figura 10) e nos exemplos de informações de contexto definidos na tabela 1, foi realizada a modelagem de parte dos conceitos de domínio e das informações de contexto em ferramentas que implementam ontologias em OWL-DL, utilizando a ferramenta Protégé (PROTEGE, 2010), e modelos de características em UbiFEX, utilizando a ferramenta Odyssey (ODYSSEY, 2010).

7.1. Ferramenta de ontologia

Para a modelagem da ontologia em OWL, foi selecionada a ferramenta Protégé (PROTEGE, 2010) analisada em trabalho de Azevedo et al. (2008).

O Protégé é um editor de ontologias *open source* bem como um *framework* baseado em conhecimento que foi desenvolvido, inicialmente, pelo departamento de informática médica da Universidade de Stanford[2] para tratar conceitos relacionados ao domínio de oncologia. Hoje, trata-se de uma ferramenta que permite construir ontologias de domínio, personalizar formulários de entrada de dados, inserir e editar dados, possibilitando a criação de bases de conhecimento, de onde é possível inferir informações baseadas em regras formais.

Considerando os conceitos relacionados ao contexto, as dimensões de contexto propostas por Araujo et al. (2004) e as informações de contexto de cada umas das três dimensões trabalhadas (Tabela 1), foram traduzidas em classes na ontologia. Como forma de simplificar o uso da ferramenta, optou-se pela modelagem dos dois conjuntos de conceitos, de processos de desenvolvimento de software e de contexto, em um único projeto dentro da ferramenta, ou seja, neste caso, em um único modelo. Para efeito de

[2] Site: http://bmir.stanford.edu/

apresentação, parte da ontologia de domínio é apresentada na Figura 12, parte da ontologia de características é apresentada na Figura 13 e parte da ontologia de contexto é apresentada na Figura 14.

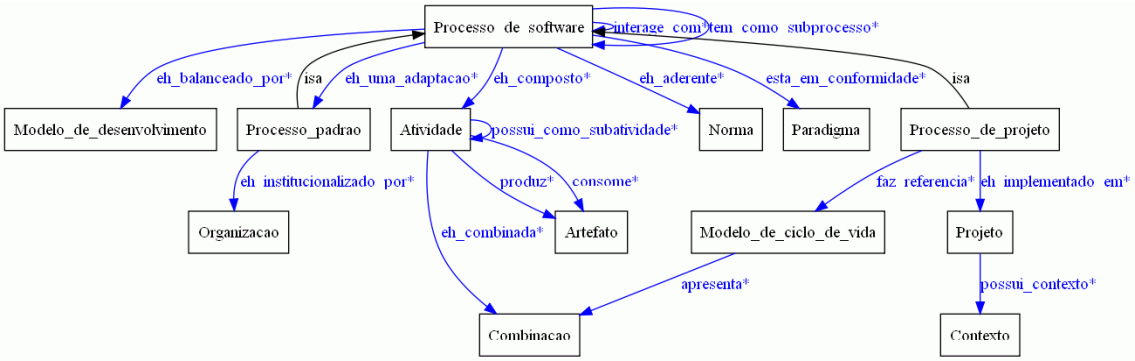

Figura 12 – Parte da ontologia de domínio de processos de desenvolvimento de software

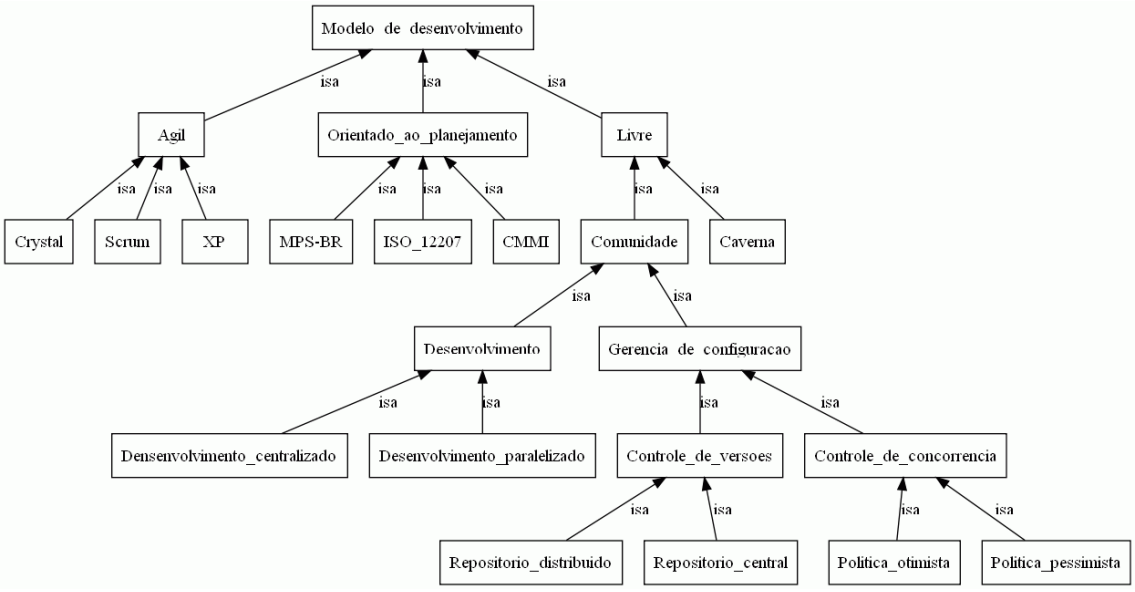

Figura 13 – Parte da ontologia de características de processos

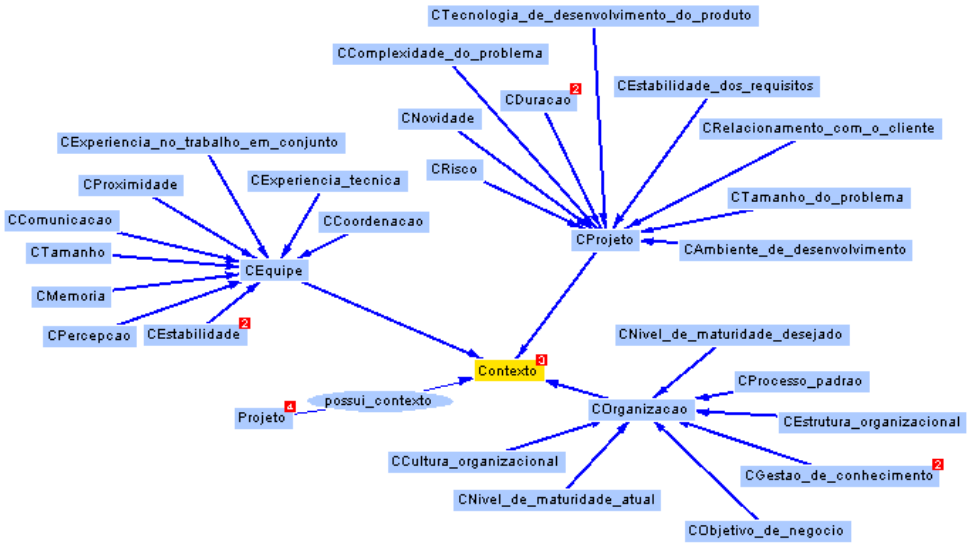

Figura 14 – Parte da ontologia de contexto para processos de desenvolvimento de software

No exemplo analisado, apenas alguns conceitos do domínio e algumas informações de contexto foram modelados. Para a construção de regras que permitam realizar a combinação de conceitos, é necessário utilizar uma linguagem de regras. Estas oferecem facilidades para especificar regras de transformações de dados que definem como sintetizar novos fatos a partir daqueles armazenados na base de dados.

A linguagem de regras inicialmente testada foi a SQWRL (*Semantic Query-Enhanced Web Rule Language*) (SQWRL, 2009) baseada na SWRL (*Semantic Web Rule Language*) (W3C, 2004b), que une a lógica matemática (cláusulas de Horn) a linguagem OWL. As regras são construídas com base na implicação entre um antecedente e um consequente. Para exemplificar a seleção pela prática Ágil em função de um contexto, pode-se caracterizar uma situação onde ocorram conflitos raros que é identificada quando o tamanho da equipe é grande (>= 40 pessoas) e a comunicação e percepção da equipe são altas. Quando esta situação ocorre, o uso da prática Política Otimista, o que em SWRL se traduz na regra apresenta na Figura 15.

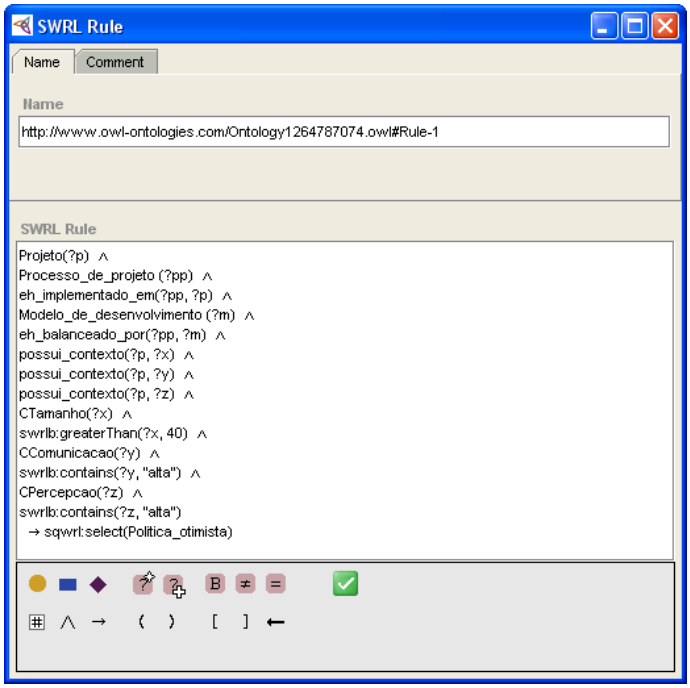

Figura 15 – Exemplo de regra em SQWRL no Protégé

Para que essas regras possam ser aproveitadas a partir de inferências, é necessária a utilização de um *reasoner*, ou raciocinador, um aplicativo que retorna novas informações sobre as instâncias presentes na ontologia a partir das regras definidas nesta. O *reasoner* infere características das instâncias da ontologia que não foram diretamente definidas pelo seu criador, identificando novas informações e retratando de forma mais fiel o domínio representado. O Protégé permite o uso de diversos *reasoners*, dentre eles, Pellet (já incorporado a ferramenta), Racer[3] e Jess[4]. Esta regra, quando executada, retorna a sugestão de uso da prática Politica_otimista quando encontra a situação mencionada anteriormente.

7.2. Ferramentas de modelagem de características

Para a modelagem de características utilizando a notação Odyssey-FEX (OLIVEIRA, 2006), foi utilizada a ferramenta que a implementa, Odyssey (ODYSSEY, 2010), desenvolvido pelo Grupo de Reutilização de Software da COPPE/UFRJ. O ambiente Odyssey é uma infraestrutura de reutilização baseada em modelos de domínio.

[3] Site Racer: http://www.racer-systems.com

[4] Site Jess: http://www.jessrules.com

Contempla atividades de Engenharia de Domínio (ED)[5] e atividades de Engenharia de Aplicação (EA)[6], com foco no reaproveitamento e adaptação dos componentes do domínio à aplicação.

Este ambiente foi escolhido para o estudo de viabilidade deste trabalho, pois permite que se tenha, de forma unificada, no mesmo ambiente, o suporte para a modelagem do domínio através do modelo de características e para a modelagem contextual. Além disso, a ferramenta oferece a possibilidade de interação entre as duas abordagens.

Assim, considerando os conceitos relacionados ao contexto, foi adotada a proposta de Fernandes (2008) que implementa uma extensão da notação Odyssey-FEX (OLIVEIRA, 2006) e da ferramenta Odyssey, com o propósito de permitir a representação de contexto em modelos de características. Neste trabalho, foram criadas duas novas características: entidade de contexto e informação de contexto. As entidades de contexto representam as dimensões de contexto relevantes para o domínio. As informações de contexto são as características que representam as informações que devem ser coletadas para caracterizar as entidades de contexto do domínio (FERNANDES, 2008).

Assim, as dimensões e as informações de contexto deste trabalho foram traduzidas, respectivamente, para as entidades de contexto (*context entity*) e informações de contexto (*context information*), como apresentado pelo exemplo na Figura 16.

[5] A Engenharia de Domínio (ARANGO E PRIETO-DIAZ, 1991, BRAGA, 2000, WERNER E BRAGA, 2005) é uma "abordagem baseada em reutilização para definição do escopo, especificação da estrutura, e construção de recursos para uma classe de sistemas, subsistemas ou aplicações" (ISO/IEC, 2007).

[6] A Engenharia de Aplicação "consiste na reutilização de artefatos gerados na Engenharia de Domínio" (MILER, 2000).

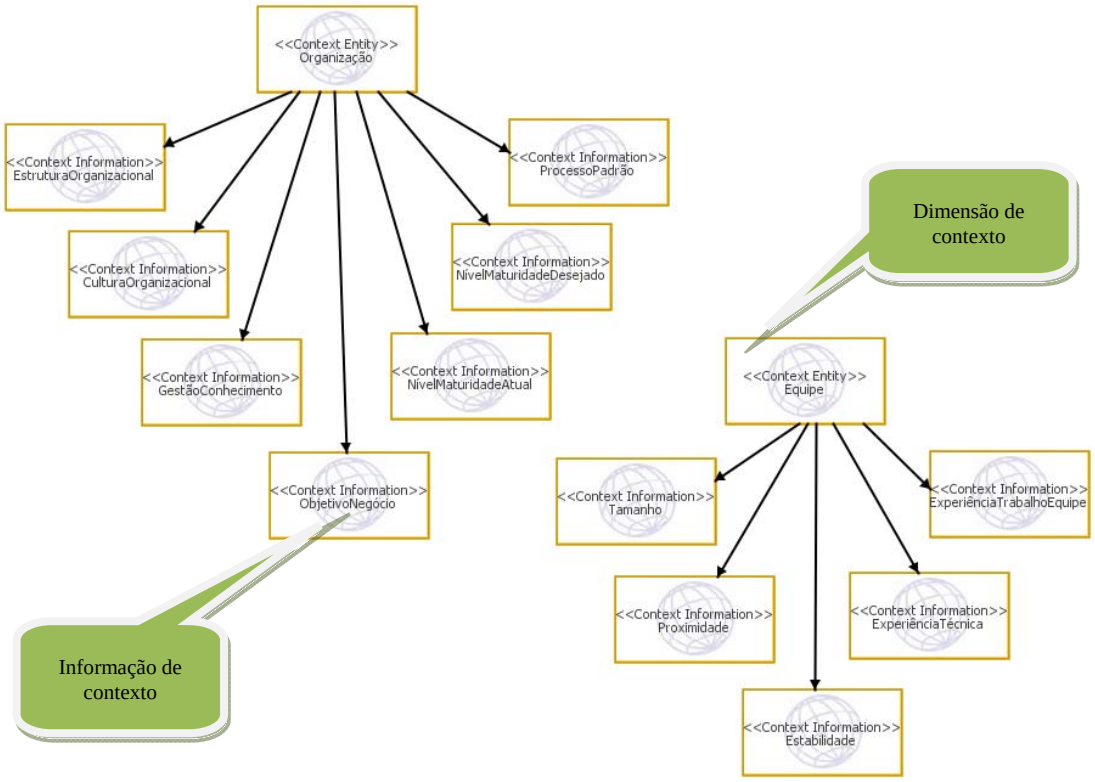

Figura 16 – Exemplo parcial de entidades e informações de contexto no Odyssey

Considerando as situações de contexto apresentadas na Tabela 2, foi selecionada para a modelagem a situação de Conflitos raros. Esta situação ocorre quando o tamanho da equipe é grande (>= 40 pessoas) e a comunicação e percepção da equipe são altas. Na ferramenta, esta definição de contexto é apresentada na Figura 17 e sua semântica é definida como: “Conflitos Raros = (((Equipe.Tamanho >= 40) AND (Equipe.Comunicação = alta)) AND (Equipe.Percepção = alta)).

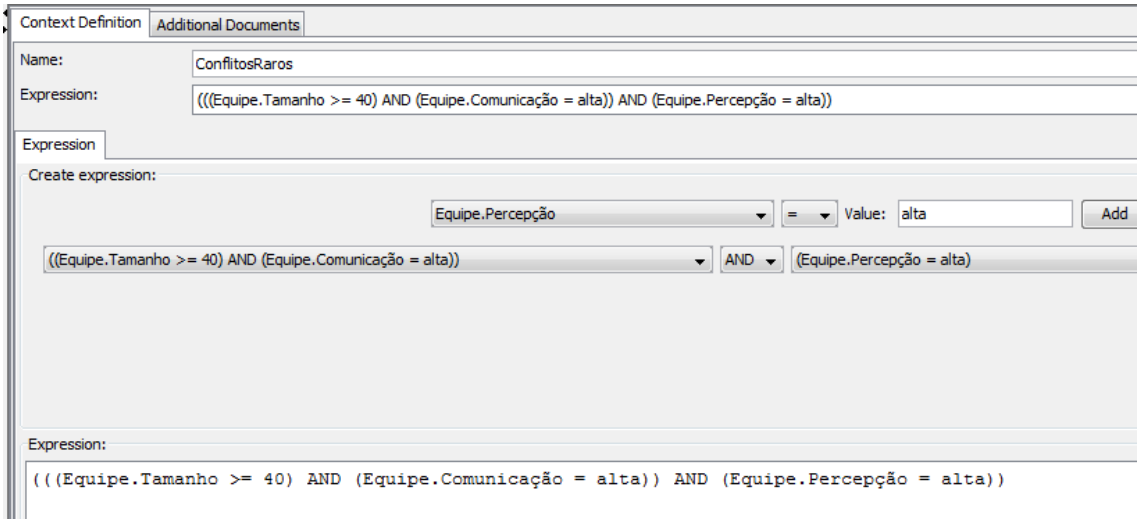

Figura 17 – Exemplo de definição de contexto no Odyssey

Após a definição dos contextos relevantes para o domínio, as ações que devem ser tomadas para uma determinada situação podem ser especificadas por meio das regras de contexto (*context rule*). Estas regras representam como uma situação de contexto impacta na configuração de um produto (no caso deste trabalho do processo de desenvolvimento de software), indicando, por exemplo, decisões a respeito da seleção de variantes em um ponto de variação.

Por exemplo, podem existir as seguintes regras de contexto:

(i) CR1: “Conflitos raros” implica na seleção da *feature* Política Otimista;

(ii) CR2: “Desenvolvimento distribuído” implica na seleção da *feature* Desenvolvimento Paralelizado.

O próprio diagrama de características, após o estabelecimento de situações e regras, apresenta os conceitos sobre os quais algum tipo de raciocínio pode ser realizado como apresentado na Figura 18. As características que fazem parte do consequente de uma regra de contexto são marcadas no canto superior esquerdo com o identificador da regra, permitindo de forma clara a identificação das características que sofrem influência de contexto.

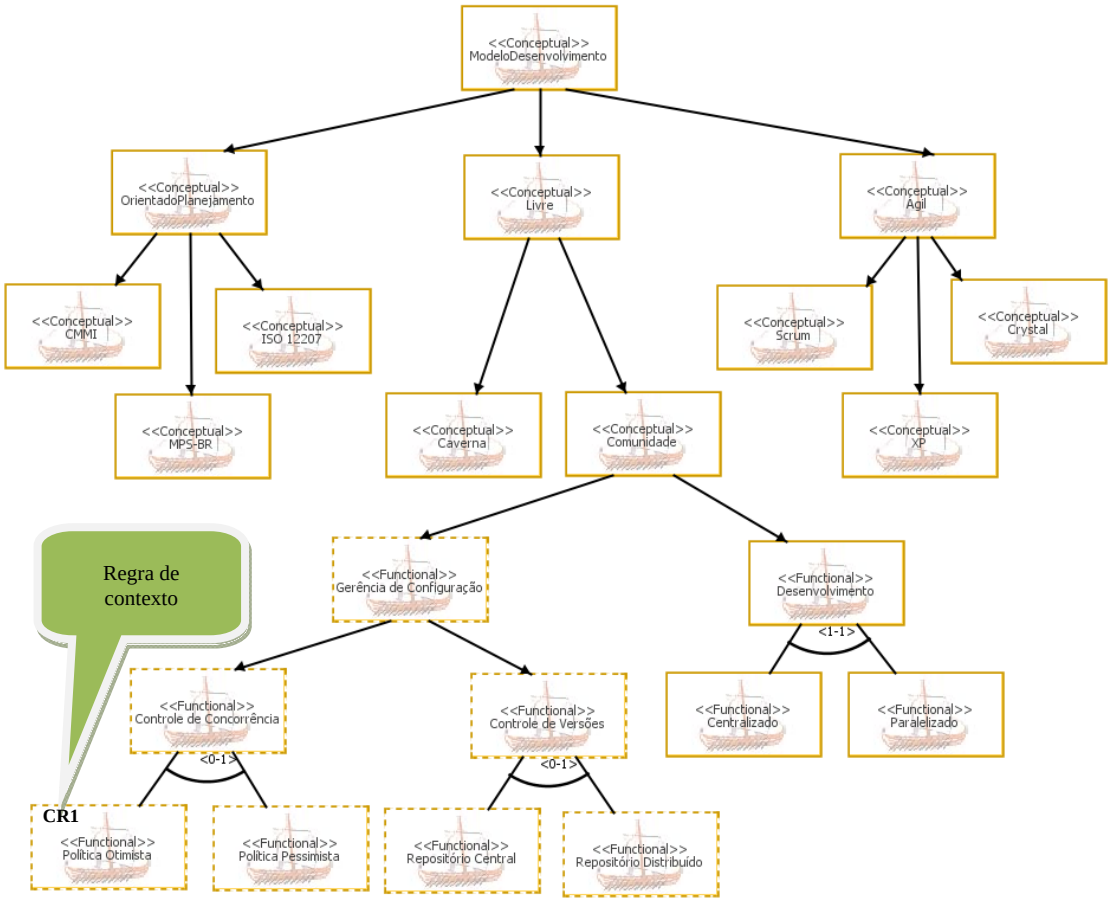

Figura 18 – Diagrama de características parcial após estabelecimento de situações e regras contextuais

8. Discussões e Conclusões

Este estudo analisou a combinação de uma linguagem de representação e da ferramenta que a implementa, uma vez que algumas das necessidades do trabalho necessitavam de apoio computacional e não somente de uma linguagem semanticamente rica.

Não existe uma linguagem de representação de conhecimento necessariamente melhor do que outra e sim aquela cuja semântica provê significado útil a parte do mundo real que se quer representar. Segundo Guizzardi (2005), “a adequabilidade de uma linguagem para criar especificações em um determinado domínio depende de quão próximas as estruturas das especificações construídas com esta linguagem estão da estrutura dos modelos que se pretende representar”.

O presente trabalho não teve como objetivo avaliar a representatividade das linguagens de representação de ontologias e de modelos de características e das ferramentas utilizadas, mas tão somente avaliar, para o estudo de adaptação de processos de

software, qual a melhor alternativa. Neste sentido, a linguagem OWL, e os mecanismos de raciocínio oferecidos pela ferramenta Protégé, demonstraram uma complexidade de representação e capacidade de raciocínio que estão além dos objetivos do trabalho.

Por outro lado, a linguagem de representação Odyssey-FEX (OLIVEIRA, 2006) e a linguagem de representação de contexto UbiFEX (FERNANDES, 2008), apoiadas pela ferramenta Odyssey (ODYSSEY, 2010), apresentaram um grau de representatividade suficiente para abarcar as necessidades do trabalho que será realizado sobre adaptação de processos de software, sem aumentar a complexidade da modelagem. Estas linguagens demonstraram possuir uma complexidade adequada de utilização e uma semântica suficientemente rica que atende aos propósitos do trabalho.

Como oportunidade de trabalhos futuros, pode-se considerar o estudo de formas de representação do conhecimento que tratem incertezas e probabilidades. O raciocínio lógico utilizado como forma de sistematização computacional, apesar de ser uma abordagem poderosa, pode não ser útil em situações onde não se conhece previamente todo o escopo do problema. Para estes casos, o raciocínio probabilístico surge como uma boa opção. “A principal vantagem de raciocínio probabilístico sobre raciocínio lógico é o fato de que agentes podem tomar decisões racionais mesmo quando não existe informação suficiente para se provar que uma ação funcionará” (CHARNIAK, 1991). Em particular, as redes bayesianas (JAYNES E BRETTHORST, 2003) são adequadas para tarefas de descoberta de conhecimento em bases de dados.

Agradecimentos

Este trabalho é parcialmente financiado pelo CNPq sob o processo no. 142006/2008-4.

Referências

ABRAHAMSSON, P.; WARSTA, J.; SIPONEN, M.; ET AL., 2003, "New directions on agile methods: a comparative analysis". In: *Proceedings of the 25th International Conference on Software Engineering*, pp. 244-254, Portland, Oregon, USA.

ABRANTES, J. F., 2009, *Uma Abordagem para Agilidade em Processos de Teste de Software*. Exame de Qualificação, COPPE/UFRJ, Rio de Janeiro, Brasil.

ABRANTES, J. F.; TRAVASSOS, G. H., 2007, *Revisão quasi-Sistemática da Literatura: Caracterização de Métodos Ágeis de Desenvolvimento de Software*, Relatório Técnico ES-714/07, PESC-COPPE. Disponível em: http://www.cos.ufrj.br.

AGOSTINI, A.; MICHELIS, G.; GRASSO, M. A.; ET AL., 1996, "Contexts, work processes, and workspaces", *Computer Supported Cooperative Work (CSCW)*, v. 5, n. 2 (Jun.), pp. 223-250.

ARANGO, G.; PRIETO-DIAZ, R., 1991, "Domain analysis: Concepts and research directions", *Domain Analysis: Acquisition of Reusable Information for Software Construction*, IEEE Computer Society Press, pp. 9-32.

ARAUJO, R. M. D.; SANTORO, F. M.; BRÉZILLON, P.; ET AL., 2004, "Context Models for Managing Collaborative Software Development Knowledge". In: *International Workshop on Modeling and Retrieval of Context (MRC)*, pp. 61-72, Ulm.

ARAUJO, R. M. D.; BORGES, M. R. S., 2007, "The role of collaborative support to promote participation and commitment in software development teams", *Software Process: Improvement and Practice*, v. 12, n. 3, pp. 229-246.

ASKLUND, U.; BENDIX, L., 2002, "A study of configuration management in open source software projects", *IEE Proceedings – Software*, v. 149, pp. 40-46.

AZEVEDO, L. G.; SOUZA, J.; LOPES, M.; ET AL., 2008, *Inspeção de Ferramentas de Ontologias*, Relatório Técnico 003/2008, RelaTe-DIA, UNIRIO. Disponível em: http://seer.unirio.br/index.php/monografiasppgi.

BARABASI, A. L., 2003, *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life*. Cambridge, Plume.

BARNETT, L., 2004, "Applying Open Source Processes In Corporate Development Organizations", *Forrester Research*, pp. 1-15.

BARRETO, A.; MURTA, L. G. P.; ROCHA, A. R., 2009, "Componentizando Processos Legados de Software Visando a Reutilização de Processos". *Simpósio Brasileiro de Qualidade de Software (SBQS)*, pp. 189-203, Ouro Preto, Minas Gerais, Brasil.

BAZIRE, M.; BREZILLON, P., 2005, "Understanding Context Before Using It", *Modeling and Using Context*, , pp. 29-40.

BECK, K., 1999, *Extreme Programming Explained: Embrace Change*. Boston, MA, USA, Addison-Wesley.

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; ET AL., 2001. Manifesto for Agile Software Development. Disponível em: http://agilemanifesto.org/. Acesso em: 15 Dez 2008.

BERTOLLO, G.; SEGRINI, B.; FALBO, R. D. A., 2006, "Definição de Processos de Software em um Ambiente de Desenvolvimento de Software Baseado em Ontologias". *Simpósio Brasileiro de Qualidade de Software (SBQS)*, pp. 72-86, Vila Velha, Espírito Santo, Brasil.

BIOLCHINI, J.; MIAN, P. G.; NATALI, A. C. C.; ET AL., 2005, *Systematic Review in Software Engineering*, Relatório Técnico ES-679, PESC-UFRJ.

BLOIS, A. P. T. B., 2006, *Uma Abordagem de Projeto Arquitetural Baseado em Componentes no Contexto de Engenharia de Domínio*. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro.

BOEHM, B., 2002, "Get ready for agile methods, with care", *IEEE Computer*, v. 35, n. 1, pp. 64-69.

BOEHM, B.; TURNER, R., 2003, *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA, Addison-Wesley.

BOSCH, J., 2004, "Software Variability Management". In: *International Conference on Software Engineering (ICSE)*, pp. 720-721, Scotland, UK.

BRAGA, R. M. M., 2000, *Busca e Recuperação de Componentes em Ambientes de Reutilização de Software*. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Brasil.

BREZILLON, P.; POMEROL, J., 1999, "Contextual Knowledge Sharing and Cooperation in Intelligent Assistant Systems", *Le Travail Humain*, v. 62, pp. 223-246.

BREZILLON, P., 1999, "Context in problem solving: a survey", *Knowledge Engineering Review*, v. 14, n. 1, pp. 47-80.

BREZILLON, P., 2003, "Making context explicit in communicating objects", In: C. Kantzig, G. P. [org.] (eds), *Communicating with Smart Objects*, London: Logan Page Science, pp. 45–59.

CAPILUPPI, A.; LAGO, P.; MORISIO, M., 2003, "Characteristics of Open Source Projects". In: *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, pp. 1-11, Washington, USA.

CAPPELLI, C.; BAIAO, F.; SANTORO, F. M.; ET AL., 2007, "Uma abordagem de construção de ontologia de domínio a partir do modelo de processos de negócio". *Workshop on Ontologies and Metamodels in Software and Data Engineering (WOMSDE)*, pp. 85-96, João Pessoa, PB, Brasil.

CARDOSO, J., 2007, "The Semantic Web Vision: Where Are We?", *Intelligent Systems, IEEE*, v. 22, n. 5, pp. 84-88.

CHARNIAK, E., 1991, "Bayesian networks without tears: making Bayesian networks more accessible to the probabilistically unsophisticated", *AI Mag.*, v. 12, n. 4, pp. 50-63.

CHEN, G.; KOTZ, D., 2000, *A Survey of Context-Aware Mobile Computing Research*, Technical Report TR2000-381, Dartmouth College. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.3717.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S., 2006, *CMMI: Guidelines for Process Integration and Product Improvement*. 2 ed. Boston, MA, Addison-Wesley.

COCKBURN, A.; HIGHSMITH, J., 2001, "Agile Software Development: The People Factor", *IEEE Computer*, v. 34, n. 11, pp. 131-133.

COCKBURN, A., 2001, *Agile Software Development*. Boston, Massachussetts, Addison-Wesley Professional.

COCKBURN, A., 2004, *Crystal Clear: A Human-Powered Methodology for Small Teams*. 1 ed. Boston, MA, USA, Addison-Wesley.

CUBRANIC, D.; BOOTH, K. S., 1999, "Coordinating open-source software development". In: *Proceedings of the 8th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 61-66, Stanford, USA.

CUGOLA, G.; GHEZZI, C., 1998, "Software processes: A retrospective and a path to the future", *Software Process Improvement and Practice (SPIP) Journal*, v. 4, pp. 101-123.

CZARNECKI, K.; EISENECKER, U., 2000, *Generative Programming: Methods, Tools, and Applications*. Boston, MA, USA, Addison-Wesley.

DAVENPORT, T. H.; PRUSAK, L., 1998, *Working Knowledge: How Organizations Manage What They Know*. 1 ed. Harvard Business Press.

DEY, A.; ABOWD, G.; SALBER, D., 2001, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications", *Human-Computer Interaction*, v. 16, n. 2-4, pp. 97-166.

EBERT, C., 2007, "Open Source Drives Innovation", *IEEE Software*, v. 24, n. 3, pp. 105-109.

FALBO, R. A., 1998, *Integração de Conhecimento em um Ambiente de Desenvolvimento de Software*. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Brasil.

FALBO, R. A.; BERTOLLO, G., 2005, "Establishing a Common Vocabulary for Software Organizations Understand Software Processes". *EDOC International Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE)*, pp. 1-8, Enschede, The Netherlands.

FEI DAI; TONG LI, 2007, "Tailoring Software Evolution Process". *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD)*, pp. 782-787, Nagoya, Japan.

FELLER, J.; FITZGERALD, B., 2001, *Understanding Open Source Software Development*. Boston, MA, USA, Addison-Wesley.

FERNANDES, P. C. C., 2008, *UbiFEX: Uma Abordagem para Modelagem de Características de Linha de Produtos de Software Sensíveis ao Contexto*. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil.

FRITZSCHE, M.; KEIL, P., 2007, "Agile Methods and CMMI: Compatibility or Conflict?", *e-Informatica Software Engineering Journal*, v. 1, n. 1, pp. 9-26.

FSF, 2008. The Free Software Definition. Disponível em: http://www.gnu.org/philosophy/free-sw.html. Acesso em: 31 Maio 2008.

FUGGETTA, A., 2000, "Software process: a roadmap". In: *Proceedings of the Conference on The Future of Software Engineering*, pp. 25-34, Limerick, Ireland.

GACEK, C.; ARIEF, B., 2004, "The Many Meanings of Open Source", *IEEE Software*, v. 21, n. 1, pp. 34-40.

GAO, Y.; FREEH, V.; MADEY, G., 2003, "Analysis and Modeling of Open Source Software Community". *North American Association for Computational Social and Organization Sciences Conference (NAACSOS)*, pp. 1-4, Pittsburgh, PA, United States.

GINSBERG, M.; QUINN, L., 1995, *Process Tailoring and the Software Capability Maturity Model* CMU/SEI-94-TR-024, SEI-CMU. Disponível em: http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.024.html.

GLASER, B. G.; STRAUSS, A., 1967, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Transaction.

GLASS, R. L., 2001, "Agile Versus Traditional: Make Love, Not War!", *Cutter IT Journal*, v. 14, n. 12, pp. 12-18.

GLAZER, H.; DALTON, J.; ANDERSON, D.; ET AL., 2008, *CMMI or Agile: Why Not Embrace Both!*, SEI-CMU. Disponível em: http://www.sei.cmu.edu/publications/documents/08.reports/08tn003.html.

GOTO, C. S.; ROSA, M. P.; DE SOUZA, C. D., 2008, "Um Estudo Exploratório sobre os Efeitos da Refatoração na Coordenação das Atividades de Desenvolvimento

de Software Livre". *Workshop de Manutenção de Software Moderna (WMSWM)*, pp. 1-8, Florianópolis, SC, Brasil.

GRUBER, T. R., 1995, "Toward principles for the design of ontologies used for knowledge sharing", *Int. J. Hum.-Comput. Stud.*, v. 43, n. 5-6, pp. 907-928.

GUARINO, N., 1998, *Formal Ontology in Information Systems*. Trento, Italy, IOS Press.

GUIZZARDI, G., 2005, *Ontological foundations for structural conceptual models*. Doctoral thesis, University of Twente Disponível em: http://doc.utwente.nl/50826/.

GUIZZARDI, G., 2006, "On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models". In: *Proceeding of the Conference on Databases and Information Systems IV: Selected Papers from the Seventh International Conference (DBandIS)*, pp. 18-39, Vilnius, Lithuania.

HAEFLIGER, S.; VON KROGH, G.; SPAETH, S., 2007, "Code Reuse in Open Source Software", *Management Science*, v. 54, n. 1, pp. 180-193.

HANSSON, C.; DITTRICH, Y.; GUSTAFSSON, B.; ET AL., 2006, "How agile are industrial software development practices?", *Journal of Systems and Software*, v. 79, n. 9, pp. 1295-1311.

HAREL, D.; RUMPE, B., 2004, "Meaningful modeling: what's the semantics of "semantics"?", *Computer*, v. 37, n. 10, pp. 64-72.

HEALY, K.; SCHUSSMAN, A., 2003, "The Ecology of Open Source Software". In: *Annual Meeting of the American Sociological Association*, pp. 1-20, Atlanta, GA, USA.

HEIDEGGER, M., 1978, *Being and time*. Wiley-Blackwell.

HIGHSMITH, J.; COCKBURN, A., 2001, "Agile Software Development: The Business of Innovation", *IEEE Computer*, v. 34, n. 9, pp. 120-127.

HUBAUX, A.; CLASSEN, A.; MENDONÇA, M.; ET AL., 2010, "A Preliminary

Review on the Application of Feature Diagrams in Practice". In: *International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pp. 53-59, Linz, Austria.

HUMPHREY, W. S., 1989, *Managing the Software Process*. Boston, MA, USA, Addison-Wesley.

ISO/IEC, 2007, *ISO/IEC 12207:1995, Information technology - Software life cycle processes*.

JAYNES, E. T.; BRETTHORST, G. L., 2003, *Probability theory: The logic of Science*. Cambridge University Press.

JIN XU; YONGQIN GAO; CHRISTLEY, S.; ET AL., 2005, "A Topological Analysis of the Open Source Software Development Community". In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS)*, pp. 198-208, Waikoloa, HI, United States.

KANG, K.; COHEN, S.; HESS, J.; ET AL., 1990, *Feature-Oriented Domain Analysis* CMU/SEI-90-TR-21, CMU-SEI. Disponível em: http://www.sei.cmu.edu/domain-engineering/FODA.html.

KITCHENHAM, B., 2004, *Procedures for Performing Systematic Reviews*, Technical Report TR/SE-0401, Software Engineering Group, Department of Computer Science, Keele University. Disponível em: http://www.idi.ntnu.no/emner/empse/papers/kitchenham_2004.pdf.

KRISHNAMURTHY, S., 2002. Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. *First Monday*. Disponível em: http://firstmonday.org/issues/issue7_6/krishnamurthy/. Acesso em: 25 Maio 2008.

LAANTI, M.; KETTUNEN, P., 2005, "How to Steer an Embedded Software Project: Tactics for Selecting Agile Software Process Models", *Information and Software Technology*, v. 47, n. 9, pp. 587-608.

LEE, K.; KANG, K. C.; LEE, J., 2002, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering". In: *Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools*, pp. 62-77

LINDEN, F. J.; SCHMID, K.; ROMMES, E., 2007, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. 1 ed. Berlin / Heidelberg, Springer.

LINDVALL, M.; BASILI, V.; BOEHM, B.; ET AL., 2002, "Empirical Findings in Agile Methods", *Extreme Programming and Agile Methods (XP/Agile Universe)*, Springer-Verlag, pp. 81-92.

LITTLE, T., 2005, "Context-adaptive agility: managing complexity and uncertainty", *Software, IEEE*, v. 22, n. 3, pp. 28-35.

LOPES, M., 2009, *Construção de Modelos Conceituais de Domínio Ontologicamente Bem-fundamentados a partir de Regras de Negócio*. Dissertação de Mestrado, UNIRIO, Rio de Janeiro, Brazil.

LOPEZ-FERNANDEZ, L.; ROBLES, G.; GONZALEZ-BARAHONA, J. M.; ET AL., 2004, "Applying Social Network Analysis to the Information in CVS Repositories". *Mining Software Repositories Workshop - International Conference on Software Engineering (ICSE)*, pp. 101-105, Edinburgh, Scotland.

MACHADO, L. F. D. C., 2000, *Modelo para Definição de Processos de Software na Estação TABA*. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

MADEY, G.; FREEH, V.; TYNAN, R., 2002, "The open source software development phenomenon: An analysis based on social network theory". *Americas Conference on Information Systems (AMCIS)*, pp. 1806-1813, Dallas, TX, USA.

MAGDALENO, A. M., 2010, "Balancing Collaboration and Discipline in Software Development Processes". In: *Doctoral Symposium of International Conference on Software Engineering (ICSE)*, Cape Town, South Africa (to appear).

MAGDALENO, A. M.; WERNER, C. M. L., 2008, *Introdução aos processos de colaboração e reutilização em software livre*, Relatório Técnico 01/2008, FAPERJ/COPPE. Disponível em: http://www.uniriotec.br/padct/.

MAGDALENO, A. M.; WERNER, C. M. L.; ARAUJO, R. M. D., 2009, *Revisão Quasi-Sistemática da Literatura: Conciliação de processos de desenvolvimento*

de software, Relatório Técnico ES-730/09, PESC-COPPE. Disponível em: http://www.cos.ufrj.br.

MASSEN, T.; LICHTER, H., 2004, "Deficiencies in Feature Models". In: *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, pp. 59-72, Boston, MA, USA.

MCGUINNESS, D. L.; FIKES, R.; RICE, J.; ET AL., 2000, "An Environment for Merging and Testing Large Ontologies". In: *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 483-493, Breckenridge, Colorado, USA.

MILER, N., 2000, *A Engenharia de Aplicações no Contexto da Reutilização Baseada em Modelos de Domínio*. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil.

MILLER, G. G., 2001, "The Characteristics of Agile Software Processes". In: *Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS)*, pp. 1-3, Santa Barbara, CA, USA.

MIRA, J. M., 2008, "Symbols versus connections: 50 years of artificial intelligence", *Neurocomputing*, v. 71, n. 4-6 (Jan.), pp. 671-680.

NERUR, S.; MAHAPATRA, R.; MANGALARAJ, G., 2005, "Challenges of migrating to agile methodologies", *Communications of ACM*, v. 48, n. 5, pp. 72-78.

NONAKA, I.; TAKEUCHI, H., 1995, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, USA.

NOY, N. F.; MCGUINNESS, D. L., 2001, "Ontology Development 101: A Guide to Creating Your First Ontology". *Semantic Web Working Symposium*

NUNES, V. T., 2007, *Um Modelo de Suporte à Gestão de Conhecimento Baseado em Contexto*. Dissertação de Mestrado, UFRJ/IM/NCE, Rio de Janeiro, Brasil.

NUNES, V. T.; WERNER, C.; SANTORO, F. M., 2010, "Context-Based Process Line". In: *International Conference on Enterprise Information Systems (ICEIS)*, p. (to

appear), Funchal, Madeira, Portugal.

ODYSSEY, 2010. Odyssey SDE Homepage. Disponível em: http://reuse.cos.ufrj.br. Acesso em: 12 Jan 2010.

OLIVEIRA, R. F. D., 2006, *Formalização e Verificação de Consistência na Representação de Variabilidades*. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil.

OMG, 2008. Software Process Engineering Metamodel. Disponível em: http://www.omg.org/technology/documents/formal/spem.htm. Acesso em: 23 Ago 2008.

OMG, 2009. Business Process Management Notation (BPMN) Version 1.2. Disponível em: http://www.bpmn.org/. Acesso em: 8 Jun 2009.

OSTERWEIL, L., 1987, "Software processes are software too". In: *International Conference on Software Engineering (ICSE)*, pp. 2-13, Monterey, CA, USA.

PALMER, S. R.; FELSING, J. M., 2002, *A Practical Guide to Feature-Driven Development*. USA, Prentice Hall.

PATEL, C.; LYCETT, M.; MACREDIE, R.; ET AL., 2006, "Perceptions of Agility and Collaboration in Software Development Practice". In: *Hawaii International Conference on System Sciences (HICSS)*, pp. 1-7, Kauai, Hawaii, USA.

PAULK, M., 2001, "Extreme programming from a CMM perspective", *Software, IEEE*, v. 18, n. 6, pp. 19-26.

PEDREIRA, O.; PIATTINI, M.; LUACES, M. R.; ET AL., 2007, "A systematic review of software process tailoring", *SIGSOFT Software Engineering Notes*, v. 32, n. 3, pp. 1-6.

PENG XU; RAMESH, B., 2008, "Using Process Tailoring to Manage Software Development Challenges", *IT Professional*, v. 10, n. 4, pp. 39-45.

PRESSMAN, R. S., 2001, *Software Engineering: A Practitioner's Approach*. 5 ed. McGraw-Hill.

PROTEGE, 2010. The Protégé Ontology Editor and Knowledge Acquisition System. Disponível em: http://protege.stanford.edu/. Acesso em: 25 Jan 2010.

QUMER, A.; HENDERSON-SELLERS, B., 2008, "An evaluation of the degree of agility in six agile methods and its applicability for method engineering", *Information and Software Technology*, v. 50, n. 4 (Mar.), pp. 280-295.

RAMAN, S., 2000, "It is software process, stupid: next millennium software quality key", *Aerospace and Electronic Systems Magazine, IEEE*, v. 15, n. 6, pp. 33-37.

RAYMOND, E. S., 2001, *The Cathedral & the Bazaar*. O'Reilly Media.

REIS, C. R., 2003, *Caracterização de um Processo de Software para Projetos de Software Livre*. Dissertação de Mestrado, USP, São Paulo, SP, Brasil. Disponível em: http://www.async.com.br/~kiko/dissert.pdf.

SANTANA, C. A.; TIMÓTEO, A. L.; VASCONCELOS, A. M. L., 2006, "Mapeamento do modelo de Melhoria do Processo de Software Brasileiro (MPS.Br) para empresas que utilizam Extreme Programming (XP) como metodologia de desenvolvimento". In: *Anais do V Simpósio Brasileiro de Qualidade de Software (SBQS)*, pp. 130-143, Vila Velha, Espírito Santo, Brasil.

SANTORO, F.; BRÉZILLON, P.; ARAUJO, R., 2007, "Context Dynamics in Software Engineering Process", *Computer Supported Cooperative Work in Design III*, , pp. 377-388.

SANTOS, V. V., 2008, *CEManTIKA: A Domain-Independent Framework for Designing Context-Sensitive Systems, Ph.D. Thesis, Federal University of Pernambuco*. Tese de Doutorado, Universidade Federal de Pernambuco (UFPE), Recife, Brasil.

SCACCHI, W., 2007, "Free/open source software development: recent research results and emerging opportunities". *Meeting on European software engineering conference*, pp. 459-468, Dubrovnik, Croatia.

SCHEIN, E. H., 1990, "Organizational Culture", *American Psychologist*, v. 45, n. 2, pp. 109-119.

SCHMIDT, A.; AIDOO, K. A.; TAKALUOMA, A.; ET AL., 1999, "Advanced Interaction in Context". In: *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pp. 89-101, Karlsruhe, Germany.

SCHWABER, K., 2004, *Agile Project Management with Scrum*. Washington, DC, USA, Microsoft Press.

SMITH, M.; WELTY, C.; MCGUINNESS, D. L., 2004. OWL Web Ontology Language Guide. Disponível em: http://www.w3.org/TR/owl-guide/. Acesso em: 26 Jan 2010.

SOFTEX, 2009, *Melhoria de Processo do Software Brasileiro – Guia Geral*, Modelo de Qualidade Versão 2.0 Disponível em: http://www.softex.br.

SQWRL, 2009. Semantic Query-Enhanced Web Rule Language. Disponível em: http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL. Acesso em: 12 Jan 2010.

STAPLETON, J.; CONSTABLE, P., 1997, *DSDM Dynamic Systems Development Method: The Method in Practice*. Boston, MA, USA, Addison Wesley.

TEIXEIRA, E. N., 2008, *Flexibilização para Representação de Características no Ambiente Odyssey*. Projeto Final, UFRJ/IM

THEUNISSEN, M.; KOURIE, D.; BOAKE, A., 2008, "Corporate-, Agile- and Open Source Software Development: A Witch's Brew or An Elixir of Life?", *Balancing Agility and Formalism in Software Engineering: Second IFIP TC 2 Central and East European Conference on Software Engineering Techniques (CEE-SET)*, Springer-Verlag, pp. 84-95.

TURK, D.; FRANCE, R.; RUMPE, B., 2002, "Limitations of agile software processes". In: *Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering*, pp. 43-46, Alghero, Italy.

VASCONCELLOS, E. P. G. D.; HEMSLEY, J. R., 2003, *Estrutura das organizações*. São Paulo, Brasil, Pioneira Thomson.

VINEKAR, V.; SLINKMAN, C. W.; NERUR, S., 2006, "Can Agile and Traditional Systems Development Approaches Coexist? An Ambidextrous View",

Information Systems Management, v. 23, n. 3, pp. 31-42.

W3C, 2004a. Web Ontology Language (OWL). Disponível em: http://www.w3.org/2004/OWL/. Acesso em: 14 Jan 2010.

W3C, 2004b. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Disponível em: http://www.w3.org/Submission/SWRL/. Acesso em: 12 Jan 2010.

WARSTA, J.; ABRAHAMSSON, P., 2003, "Is Open Source Software Development Essentially an Agile Method?". In: *Proceedings of the Workshop on Open Source Software Development*, pp. 143-147, Portland.

WASHIZAKI, H., 2006, "Building Software Process Line Architectures from Bottom Up", *Product-Focused Software Process Improvement*, , chapter 4034, Berlin / Heidelberg: Springer, pp. 415-421.

WASSERMAN, S.; FAUST, K., 1994, *Social Network Analysis: Methods and Applications*. 1 ed. Cambridge, United Kingdom, Cambridge University Press.

WERNER, C.; BRAGA, R., 2005, "A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes", In: Gimenes, I. M. D. S., Huzita, E. H. M. [orgs.] (eds), *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*, Rio de Janeiro, Brasil: Ciência Moderna, pp. 57-103.

YAMAUCHI, Y.; YOKOZAWA, M.; SHINOHARA, T.; ET AL., 2000, "Collaboration with Lean Media: how open-source software succeeds". In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pp. 329-338, Philadelphia, PA, USA.