UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JÚLIO HARTMANN

Utilizando Padrões Organizacionais e Avaliação de Risco para Adaptar a Metodologia de Desenvolvimento de Software

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Roberto Tom Price
Orientador

Porto Alegre, abril de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Hartmann, Júlio

Utilizando Padrões Organizacionais e Avaliação de Risco para Adaptar a Metodologia de Desenvolvimento de Software / Júlio Hartmann – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

154 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2005. Orientador: Roberto Tom Price.

1. Padrões Organizacionais. 2. Gerência de Riscos. 3. Processo de Desenvolvimento de Software. 4. Gerência de Projetos. 5. Engenharia de Software. I. Price, Roberto Tom. II. Título.

Nunca desprezes os teus amigos, por que se
um dia eles te esquecerem, só teus inimigos
se lembrarão de ti.

O sorriso enriquece os recebedores sem
empobrecer os doadores.

Quem não compreende um olhar tampouco
compreenderá uma longa explicação.

Se as coisas são inatingíveis... ora!
Não é motivo para não querê-las...

Que tristes os caminhos, se não fora
a presença distante das estrelas!

Livros não mudam o mundo,
quem muda o mundo são as pessoas.
Os livros só mudam as pessoas.

Mário Quintana

AGRADECIMENTOS

- À Universidade Federal Do Rio Grande do Sul, pela infraestrutura de ensino.
- Ao Instituto de Informática, pelo tempo em que freqüentei os seus corredores, no curso de Gradução e posteriormente no curso de Pós-Graduação.
- Aos Professores do Instituto, pelo ensino, pela sabedoria e pela dedicação.
- Ao meu orientador e ex-chefe, Prof. Dr. Roberto Tom Price, pelo suporte, pela inspiração, e também pelos debates metodológicos.
- À aluna de doutorado Lisandra Fontoura, por compartilhar seu conhecimento comigo.
- Aos brasileiros, que custearam a minha formação. Espero conseguir retribuir com meu trabalho este investimento.
- Aos amigos e colegas de curso. Espero encontrá-los por aí.
- À minha noiva Samara, pelo amor e pela compreensão nas longas horas de estudo.
- Aos meus familiares, pelo apoio e pelo incentivo permanente.

Estendo os agradecimentos a todos aqueles que de alguma forma contribuiram para que eu realizasse este trabalho, e que esqueci de citar acima.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AF	Accuracy Factor
CASE	Computer Aided Software Engineering
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CRC	Class Responsibility Collaboration
DIA	Denver International Airport
FDD	Feature Driven Development
Gof	Gang-of-Four
GQM	Goal/Question/Metric
GRS	Gerência de Riscos de Software
HQL	Hibernate Query Language
ISO	International Standards Association
JSP	Java Server Pages
KPA	Key Process Area
OML	OPEN Modeling Language
OMT	Object Modeling Technique
OPEN	Object-Oriented Environment and Notation
PC	Project Context
PMT	Pattern-based Methodology Tailoring
PPGC	Programa de Pós-Graduação em Computação
PSP	Personal Software Process
QA	Quality Assurance
RE	Risk Exposure
RPF	Risk Prioritization Factor
RUP	Rational Unified Process
SPICE	Software Process Improvement and Capability Determination
SCM	Software Configuration Management
SQA	Software Quality Assurance
SQL	Structured Query Language
TDD	Test Driven Development
TI	Tecnologia da Informação
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Unified Modelling Language
XML	eXtensible Markup Language
XP	eXtreme Programming
YAGNI	You Aren't Gonna Need It

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho descreve PMT – *Pattern-based Methodology Tailoring*, uma abordagem para a adaptação de metodologias de desenvolvimento de *software*, baseada em padrões e em critérios de risco. Seu principal objetivo é estabelecer meios de se adaptar uma linguagem de padrões organizacionais ao contexto de um projeto específico, o que é obtido através da seleção sistemática dos padrões organizacionais mais adequados aos requisitos do projeto. O trabalho é motivado pelo levantamento de que os arcabouços de processos de *software* existentes pouco fazem para compreender as necessidades de um projeto antes de definir a metodologia a ser aplicada. PMT utiliza uma análise dos riscos e do contexto de criticalidade para guiar o processo de adaptação. Padrões organizacionais que descrevem técnicas preventivas para os riscos identificados são selecionados por um mecanismo sistemático de seleção, o qual é suportado por uma ferramenta, chamada PMT-Tool.

Palavras-Chave: padrões organizacionais, gerência de riscos, processo de desenvolvimento de software, gerência de projetos, engenharia de software.

Tailoring the Software Development Methodology with Organizational Patterns and Risk Evaluation

ABSTRACT

This work presents PMT – Pattern-based Methodology Tailoring, an approach to tailoring software development methodologies, based on patterns and risk criteria. Its main goal is to provide means of adapting an organizational pattern language to the context of a given project, through the systematic selection of the most suitable organizational patterns to the requirements of the project. The work is motivated on the assessment that existing software process frameworks do little to understand the needs of a project before defining the development methodology to be used with it. PMT uses an analysis of the risks and of the criticality context of the project to guide the process adaptation. Organizational patterns which describe preventive techniques for the identified risks are selected by a systematic retrieval system which is supported by a tool, PMT-Tool.

Keywords: organizational patterns, risk management, software development process, project management, software engineering.

1 INTRODUÇÃO

Este trabalho descreve PMT – *Pattern-based Methodology Tailoring*, uma abordagem para a adaptação de metodologias de desenvolvimento de *software*, baseada em padrões e em critérios de risco. O trabalho se diferencia dos arcabouços de processos de *software* existentes, os quais não auxiliam no processo de tomada de decisões necessário à adaptação de uma metodologia ou um processo de desenvolvimento de *software* às necessidades específicas de um projeto. PMT utiliza como base um repositório de padrões organizacionais e uma análise dos riscos e do contexto de criticalidade do projeto desejado. Um mecanismo sistemático de seleção de padrões, implementado na ferramenta PMT-Tool, é utilizado para auxiliar o projetista de processos na tarefa de adaptação.

Como será estudado, há uma vasta literatura sobre metodologias de desenvolvimento de *software*. Foram realizados estudos sobre: *i.) a utilização de padrões (patterns) no desenvolvimento de software, ii.) as metodologias autodenominadas "ágeis" (agile), iii.) arcabouços de processos de desenvolvimento - como RUP (Rational Unified Process) e Open (Object-Oriented Process Environment and Notation), iv.) além de modelos para a melhoria de processos de software, como o CMM (Capability Maturity Model)*.

Primeiramente, no primeiro capítulo do trabalho, estudou-se o conceito de padrões (*patterns*) e linguagens de padrões, o qual foi explorado no final dos anos 70 pelo arquiteto de edificações Christopher Alexander [ALE 77]. No final dos anos 80 e início dos anos 90, surgiram as primeiras aplicações destes conceitos aplicados ao desenvolvimento de *software*; após a publicação do livro sobre os padrões de projeto GoF – *Gang of Four* – por Erich Gamma e outros autores em 95 [GAM 95], o conceito de padrões (*patterns*) se popularizou. Um estudo mais aprofundado sobre padrões foi realizado em [HAR 2004].

Há várias definições para um padrão no sentido do termo da língua inglesa *pattern*, a mais comum é a de Alexander – uma solução recorrente para um problema dentro de um contexto. Mas o termo padrão na língua portuguesa pode ser utilizado para traduzir também o termo *standard*, neste caso com um significado de "norma"; por isso, neste trabalho, sempre que o termo padrão for utilizado é com o sentido de *pattern*, e o termo norma é utilizado para traduzir *standard*.

Embora a aplicação mais conhecida de padrões no desenvolvimento de *software* sejam os padrões de projeto, o conceito pode ser aplicado para resolver diversos outros tipos de problemas. Há desde as expressões idiomáticas de programação (*idioms*), até os padrões organizacionais e de processos de James Coplien [COP 95], que buscam resolver problemas de organização do trabalho humano. É neste último tipo de aplicção que este trabalho concentra seu interesse, pois através de padrões e linguagens de padrões organizacionais é possível fazer a descrição de metodologias de desenvolvimento de *software*. Uma linguagem de padrões é uma coleção de padrões, que funcionam de forma conjunta de forma a gerar

um sistema inteiro, enquanto que um padrão isolado resolve apenas um problema de projeto isolado [COP 96]. Alguns autores preferem o termo “sistema de padrões”, já que o termo “linguagem” é normalmente utilizado na Ciência da Computação com outros sentidos.

James Coplien [COP 2004] mantém um *web site* onde disponibiliza uma extensa linguagem de padrões organizacionais. A estrutura de navegação da linguagem de padrões é representada na forma de um grafo. Cada nodo no grafo representa um padrão, enquanto que os arcos representam as referências de dependência entre os padrões. Ao se clicar sobre um nodo, navega-se para a descrição textual do padrão, que também inclui referências de hipertexto aos padrões relacionados. Para facilitar a navegação, os padrões são coloridos, mas não fica claro qual foi o critério de classificação que foi aplicado. Os padrões de Coplien capturam melhores práticas recorrentes em seu ambiente organizacional. Alguns destes padrões podem ser reutilizados para resolver riscos em outros projetos de *software*, mas em função da extensão da linguagem de padrões é um trabalho difícil realizar esta seleção, a qual precisa ser feita de forma empírica pelo projetista de processos ou pelo gerente de projetos.

O trabalho de Vasconcelos e Werner [VAS 98] utiliza padrões de processos para compor uma base de conhecimento sobre processos de *software*, conforme estudado em [HAR 2004]. Sua abordagem é suportada por uma ferramenta chamada Memphis – um ambiente de desenvolvimento de *software* baseado em reuso – a qual auxilia a se considerar diferentes alternativas quando os processos são compostos a partir de fragmentos descritos pelos padrões. No entanto, a ferramenta não auxilia o projetista com as decisões que este precisa tomar ao escolher entre diferentes alternativas de composição.

Já a abordagem Open (Processo, Ambiente e Notação Orientados a Objeto) [HEN 99], estudada na seção 2.3, utiliza um meta-modelo de processos a partir do qual um processo específico para um projeto pode ser instanciado. Permite um alto grau de flexibilidade para o usuário do processo, o qual precisa tomar várias decisões no processo de instanciação.

RUP (Processo Unificado da Rational) [JAC 2000] é um produto comercial, o qual pode ser considerado como uma instância completa do Processo Unificado de Desenvolvimento de *Software* [JAC 99], incluindo material disponibilizado em HTML na forma de um *web site*. É um processo pré-configurado, portanto é possível modificar somente algumas partes, como expandir, modificar ou remover etapas de atividades específicas, devendo ser considerado uma metodologia customizável ao invés de um arcabouço (*framework*) metodológico [HEN 99].

Ambas as abordagens Open e RUP não auxiliam com as decisões que um projetista de processos precisa tomar ao desenhar um processo específico. Ou seja, estas decisões, em maior ou menor grau, precisam ser realizadas de forma empírica pelo projetista de processos. Este trabalho procura auxiliar o projetista de processos em suas decisões, sugerindo práticas de metodologia, na forma de padrões, que podem ser adotadas em um projeto específico de desenvolvimento. A sugestão de padrões é feita através do mecanismo sistemática de seleção, que se baseia em regras de resolução de riscos, conforme será discutido na seção 6.5.

Outro assunto estudado neste trabalho, na seção 2.1, foi a melhoria de processos de *software*, para a qual foram desenvolvidos diversos modelos que servem para avaliar a qualidade dos processos de desenvolvimento sendo utilizados, de forma a permitir que sejam aprimorados. Estes modelos assumem que a qualidade do

software produzido é largamente influenciada pela qualidade dos processos utilizados para o seu desenvolvimento [MAN 2003a]. Vários modelos foram desenvolvidos baseados nesta idéia, os quais servem para a definição, avaliação e melhoria de processos de *software*.

O modelo CMM – *Capability Maturity Model* – propõe cinco estágios para a avaliação da maturidade dos processos de uma organização de desenvolvimento de software: *1. inicial; 2. repetível; 3. definido; 4. gerenciado*; 5. otimizado. Hoje o modelo CMM é amplamente conhecido e utilizado em muitas empresas para a melhoria de seus processos de desenvolvimento de *software*, não somente as grandes empresas envolvidas na terceirização de projetos de organizações militares norte-americanas. Também muitas empresas de grande porte nos Estados Unidos passaram e exigir algum nível de certificação CMM, o que contribuiu para disseminar sua utilização. Muitas grandes empresas de terceirização de desenvolvimento de software da Índia viram no CMM uma forma de aprimorar seus processos de desenvolvimento, e também de criar respeitabilidade para vender seu software a empresas americanas [ORR 2002].

Entretanto, alguns autores como Conradi [CON 2002] e Ward [WAR 2001], que estiveram envolvidos em diversos esforços de melhoria de processos de software, apontam alguns problemas no modelo CMM e nas abordagens existentes para SPI (*Software Process Improvement*) em geral. De forma geral, estes modelos são voltados excessivamente a estabelecer disciplina na forma de procedimentos e regras, além de um controle rigoroso sobre o projeto, o que contrasta com a perspectiva da criatividade, onde se estimula a participação dos usuários na freqüente clarificação e renegociação dos requisitos. Outra questão é a valorização excessiva da redução dos riscos do comprador sobre a satisfação do usuário. Além disso, são modelos voltados a grandes empresas desenvolvendo sistemas de alta criticalidade, e que implicam em um investimento bastante grande; investimento este que pode levar alguns anos para gerar retorno financeiro. Por isso é difícil de se adaptar este tipo de modelo em empresas pequenas e médias, envolvidas no desenvolvimento de produtos inovadores.

Por outro lado, nos últimos anos, tem chamado a atenção da comunidade de desenvolvimento de *software* a publicação de várias metodologias autodenominadas de ágeis (*agile*), estudadas no capítulo 3. Estas metodologias foram classificadas inicialmente como leves (*lightweight*), para diferenciá-las das metodologias tradicionais de desenvolvimento, consideradas pesadas (*heavyweight*), as quais seriam baseadas na produção de uma grande quantidade de documentação e de modelos para guiar a programação.

As metodologias ágeis de desenvolvimento de software têm sido abordadas em um grande número de livros técnicos recentemente. Além disso, conferências da indústria de desenvolvimento de software têm ocorrido em diversos países, inclusive no Brasil, com grande repercussão. Há diversos relatos em que se afirma a obtenção de melhores resultados com a utilização de metodologias ágeis do que com as metodologias tradicionais. No entanto, por terem estas metodologias, em sua maioria, uma publicação recente, é ainda incipiente a pesquisa e a comprovação acadêmica sobre o assunto.

Segundo Laurie Williams e Alistair Cockburn [WIL 2003], em tradução livre: "*As metodologias ágeis definitivamente atingiram um nervo da comunidade de desenvolvimento de software. Algumas pessoas argumentam ferozmente a seu favor, enquanto outras argumentam contra as mesmas com a mesma energia, e outras estão*

trabalhando para misturar as abordagens ágeis e guiadas a planos. Um número maior ainda de pessoas ainda está tentando descobrir o que a agilidade significa".

Nem todas as práticas relacionadas nas metodologias ágeis são novas. As principais práticas relacionadas a ciclos de desenvolvimento iterativos e incrementais, por exemplo, já eram mencionadas na metodologia descrita por Victor Basili em [BAS 75]. O artigo descreve uma metodologia chamada *Iterative Enhancement*, o qual foi aplicado com sucesso para a produção de uma família de compiladores. Alguns princípios valorizados em metodologias ágeis, como a iteração, a facilidade de se fazer mudanças e o redesenho, são utilizados. Em um artigo recente [LAR 2003], Craig Larman e Basili descrevem a história do desenvolvimento iterativo e incremental, na qual esta metodologia aparece em destaque.

Mesmo assim, inicialmente com as publicações da metodologia *Extreme Programming* por Kent Beck [BEC 99], e do manifesto ágil no ano de 2001, seguiram-se uma série de publicações de metodologias, como *Scrum*, por Ken Schwaber [SCH 2002], bem como alguns estudos relacionados a metodologias ou práticas ágeis, por exemplo [WIL 2002], de Laurie Williams, que serviram para tornar mais pública a existência e aplicabilidade das práticas. Contudo, considera-se que ainda faltam pesquisas relacionadas a estas práticas e metodologias, os quais foram inventados por consultores da indústria de software, e ainda estão por ser assimilados pela academia científica.

Neste trabalho, fez-se um estudo sobre as principais idéias presentes no manifesto ágil, e se estudaram três metodologias ágeis que têm sido destacadas, chamadas *Scrum*, *Feature-Driven Development* (FDD), e *Extreme Programming* (XP). Em [HAR 2003], realizou-se um estudo mais aprofundado, incluindo uma análise comparativa entre as mesmas.

Com tudo o que foi exposto até o momento, se percebe o quão difícil pode ser para uma organização ou um projeto saber que metodologia adotar e de que forma proceder para implementá-la. Cada metodologia foi normalmente desenvolvida considerando um determinado contexto, com seus próprios objetivos, por pessoas com diferentes conhecimentos e experiências, e frequentemente as asserções básicas não são claramente descritas nos seus textos. É difícil para um projetista de processos ou um gerente de projetos selecionar, elaborar ou combinar uma metodologia de desenvolvimento de *software* apropriada a um determinado projeto. Portanto, é necessária a pesquisa de abordagens sistemáticas que auxiliem o projetista de processos a executar estas tarefas.

Alistair Cockburn [COC 2000] descreve princípios e critérios para a seleção de uma metodologia adequada a um projeto de desenvolvimento de *software*. Os principais critérios são o número de pessoas envolvidas e a criticalidade do software sendo produzido. A criticalidade é classificada em quatro níveis, que significam o tipo de perda que um defeito no software pode causar (conforto, dinheiro discreto, dinheiro essencial, ou vida). Os objetivos e prioridades da organização também são considerados. Desta forma, é possível definir-se regiões em um gráfico, sendo que para cada região um tipo diferente de metodologia é indicado. Este conceito foi adaptado e explorado neste trabalho, sendo chamado de contexto de criticalidade.

Outro ponto interessante de Cockburn é a afirmação de que toda metodologia é desenhada baseada em medos. Os medos do projetista da metodologia, tanto no projeto atual, quanto em suas experiências passadas, tende a guiá-lo em seu trabalho. Por exemplo [COC 2000], "*Você tem medo que os programadores cometam erros de codificação? Realize revisões de código. Você está com medo que seus projetistas*

deixem o projeto antes do término? Faça com que eles escrevam documentações detalhadas conforme avançam.". Ao mesmo tempo, o autor afirma: "*Um aumento relativamente pequeno no tamanho ou densidade da metodologia adiciona um custo relativamente grande ao projeto"*. Ou seja, deve-se evitar adicionar práticas e processos na metodologia de um projeto sem necessidade, pois podem gerar um aumento grande no seu custo.

Este trabalho propõe que, ao invés de utilizar-se de medos inconscientes para o desenho de uma metodologia, utilize-se uma análise objetiva dos riscos do projeto para guiar este processo. A idéia de utilizar uma abordagem baseada em riscos para selecionar uma metodologia adequada a um projeto é explorada por Barry Boehm [BOE 2003a] [BOE 2003b]. A abordagem proposta pelo autor é semelhante à de Cockburn, utilizando fatores de risco para avaliar se um projeto deve utilizar uma metodologia ágil (*agile*) ou guiada a planos, ou ainda uma combinação de ambas. Ambas as abordagens – Cockburn e Boehm – são estudadas no capítulo 4.

O capítulo 5 introduz o tema de gerência de riscos em projetos de software. Segundo Tom DeMarco e Timothy Lister [DEM 2003], fugir dos riscos em projetos não é uma estratégia vencedora. Projetos que não tem riscos reais são perdedores, oferecem quase nenhum benefício. Portanto: "se um projeto não tem riscos, não o realize". Os riscos e os benefícios andam lado a lado.

Outro motivo para justificar a gerência de riscos em projetos de *software* é que, apesar de todo o esforço em metodologias de desenvolvimento, muitos destes projetos ainda falham. Terminam tarde, ultrapassam os seus custos, ou então deixam o usuário insatisfeito. Alguns destes projetos falham totalmente ou são cancelados antes mesmo de gerar qualquer resultado. Uma explicação para o alto índice de falhas é que os gerentes de projeto não estão gerenciando os riscos envolvidos nestes projetos [KEI 98]. Gerência de riscos é uma coleção de métodos que objetivam minimizar ou reduzir os efeitos de falhas nos projetos [ADD 2002].

A abordagem para a análise de riscos utilizada neste trabalho é baseada na proposta de Lizandra Fontoura [FON 2004] – aluna de doutorado do PPGC/UFRGS. Segundo a autora, Gerência de Riscos de Software (GRS) é uma abordagem que organiza as ações de tratamento de riscos em projetos de software em uma coleção de princípios e técnicas para analisar, preparar ações preventivas, tomar medidas corretivas e controlar os riscos de um projeto de desenvolvimento de software. GRS sugere medidas para prevenir os riscos de afetarem o projeto, ou para reduzir seu impacto no caso de isto ocorrer. Deve ser considerado um componente essencial dos processos de gerenciamento de projetos [ADD 2002]. O dicionário Webster define risco como a "possibilidade de perda ou ferimento". Esta definição implica em um conceito fundamental do risco: a exposição do risco [BOE 2002], algumas vezes chamada de "impacto do risco" ou "fator de risco" [BOE 91].

O gerenciamento de riscos envolve a avaliação do risco e o controle do risco [BOE 2002]. A avaliação do risco pode ser subdividida em identificação dos riscos, análise dos riscos, e priorização dos riscos [BOE 91]. A identificação dos riscos resulta em uma lista dos possíveis riscos de um projeto. A análise é uma estimativa da probabilidade de ocorrer e das conseqüências de cada um dos riscos que foram identificados [HAL 98]. Na priorização dos riscos, os riscos identificados são ordenados por sua importância [BOE 91]. Já o controle do risco é o processo de elaboração e implementação de planos de resolução de riscos, monitoramento do status de cada risco, e desenvolvimento e documentação de estratégias [HAL 98].

Resolução de um risco é a sua eliminação ou minimização, ao se executar ações descritas no plano e gerenciamento de riscos [BOE 91].

Este trabalho propõe uma abordagem dirigida por riscos para a adaptação de uma metodologia de desenvolvimento de *software*, descrita na forma de uma linguagem de padrões organizacionais. A abordagem PMT, descrita no capítulo 6, permite a estruturação de um repositório de padrões. Os padrões são classificados e são associados de forma a compor uma grande linguagem. Esta linguagem de padrões é adaptada para as necessidades de um determinado projeto de desenvolvimento. O processo de adaptação se baseia em um mecanismo sistemático de seleção, o qual sugere padrões para o projetista de processos, de acordo com o contexto de criticalidade e os riscos do projeto.

O repositório contém uma lista de riscos, a qual é utilizada como base (*checklist*) no processo de identificação dos riscos de um projeto. Os padrões são associados aos riscos que minimizam através de regras de resolução de riscos. Estas regras compreendem uma parte importante do conhecimento que é armazenado no repositório, suportando o processo de seleção sistemática.

O capítulo 7 descreve a ferramenta PMT-Tool, a qual suporta a abordagem. A ferramenta tem uma interface Web, tendo sido desenvolvida em Java, e acessando um banco de dados MySQL. O banco de dados armazena o repositório de padrões, riscos e regras. Armazena também os projetos e suas análises de exposição do risco. Com base nestas informações, a ferramenta executa a seleção sistemática, utilizando o algoritmo descrito na seção 6.5.

O capítulo 8 demonstra um exemplo de aplicação da abordagem PMT e da ferramenta PMT-Tool. Ao final, são apresentadas as conclusões do trabalho.

2 PADRÕES (PATTERNS) NO DESENVOLVIMENTO DE SOFTWARE

Este capítulo aborda o tema de padrões aplicados ao desenvolvimento de *software*. Serão descritos: a origem da aplicação de padrões no desenvolvimento de *software*, sua definição, sua característica geradora de soluções – *generatividade*, o conceito de linguagens de padrões, bem como se procurará fazer um estudo das principais aplicações e classificações dos padrões.

Uma observação importante é que neste trabalho o termo "padrão" será utilizado com a conotação da língua inglesa "*pattern*", ou seja, uma solução recorrente, genérica, dentro de um determinado contexto [COP 96] e [APP 97a]. Há outra palavra da língua inglesa – *standard* – a qual também pode ser traduzida para o termo "padrão", mas que neste caso assume o sentido de "norma", ou "regra".

Em [HAR 2004] desenvolveu-se um estudo mais aprofundado sobre o tema da aplicação de padrões no desenvolvimento de *software*.

2.1 Origens da aplicação de padrões

A definição de padrão á atribuída a Christopher Alexander, arquiteto de edifícios o qual publicou diversos trabalhos na área de padrões e linguagens de padrões, tais como [ALE 77]. Apesar de estes trabalhos tratarem de arquitetura, edifícios e construções, suas idéias podem ser aplicadas em diversas áreas de conhecimento. Acabaram por influenciar os trabalhos posteriormente publicados na área de engenharia de *software* e que criaram uma comunidade inteira dedicada a pesquisa e desenvolvimento de padrões. Segundo Alexander, "*um padrão descreve um problema que ocorre muitas e muitas vezes em nosso ambiente e então descreve a parte principal de uma solução para tal problema de tal forma que você possa utilizar esta solução um milhão de vezes sem nunca utilizar a solução da mesma forma por duas vezes*".

Além disso, um padrão descreve um problema de projeto e uma solução genérica para o problema, dentro de um determinado contexto [COP 96, citando Alexander]. Segundo Alexander, em tradução livre: "*Um padrão é uma regra de três partes, que expressa a relação entre um certo contexto, um problema, e a solução. Um padrão é, resumidamente, ao mesmo tempo algo o qual ocorre no mundo, e a regra que nos diz como criar este algo, bem como quando que devemos criá-lo. É tanto um processo como uma coisa; tanto uma descrição de algo que está vivo, quanto o processo que gera esta coisa*".

Segundo James Coplien [COP 96], um autor importante e um dos pioneiros na aplicação de padrões no desenvolvimento de *software*, as idéias de Alexander

inspiraram de tal forma a comunidade de *software* que as vendas de seus livros alcançaram altos patamares neste meio. O próprio vocabulário utilizado vem das definições deste autor, como o termo linguagem de padrões (*pattern language*). No entanto, atualmente boa parte da comunidade de *software* já não procura aplicar interpretações literais do trabalho de Alexander. Algumas necessidades pragmáticas do desenvolvimento de *software*, tais como o desenvolvimento iterativo e incremental, contrastariam com a visão de Alexander, que aplicava os padrões de forma mais monolítica. As analogias que são estabelecidas nem sempre são verdadeiras. De qualquer forma, mesmo que nem sempre se possa aplicar os conceitos desenvolvidos por Alexander de forma literal no desenvolvimento de *software*, seu legado é de grande valor: sua visão e seu sistema de valores.

Segundo Brad Appleton [APP 97a], embora as idéias de Alexander sejam ostensivamente sobre arquitetura e projeto urbanístico, elas podem ser aplicadas a diversas outras áreas do conhecimento, incluindo desenvolvimento de software.

Ainda segundo Appleton, padrões começaram a ser aplicados no desenvolvimento de *software* quando Ward Cunningham e Kent Beck (mais conhecido pela autoria do método *Extreme Programming*, ou XP [BEC 99]), estavam trabalhando em uma interface gráfica com o usuário implementada utilizando a linguagem de programação Smalltalk, em 87. Resolveram utilizar os conceitos desenvolvidos por Alexander para desenvolver uma pequena linguagem de padrões, com a intenção de guiar os programadores mais novos. Eles escreveram sobre os resultados obtidos e publicaram um artigo chamado de "*Utilizando linguagens de padrões para programas orientados e objeto*".

Alguns anos mais tarde (em 91), Jim Coplien compilou um catálogo de expressões idiomáticas em C++ (*idioms*) podem ser considerados um tipo de padrão – e publicou um livro de título "*Expressões idiomáticas e estilos de programação avançada em C++*". Além disso, no início dos anos 90, diversos *workshops* foram realizados nos Estados Unidos em que foram discutidos e compilados vários catálogos de padrões. Um pouco depois, em 95, foi publicado o livro dos padrões de projeto *GoF* [GAM 95], e a partir de então a aplicação de padrões se popularizou na comunidade de desenvolvimento de *software*.

2.2 Definindo o que é um Padrão

Nosso interesse é na aplicação de padrões ao desenvolvimento de *software*. No entanto, diversas outras áreas também utilizam o conceito de padrões. Segundo Peter Coad, em tradução livre [COA 92]: "*Muitas áreas do conhecimento utilizam padrões de diversas formas. Na música e na literatura, um padrão é uma estrutura coerente ou um projeto de uma música ou de um livro. (...) Na psicologia, um padrão é um mecanismo de pensamento que é fundamental ao funcionamento do cérebro*". Desta forma, pode-se imaginar que a definição de padrão seja também apresentada, discutida e interpretada de diferentes formas. Em relação ao desenvolvimento de *software*, uma boa definição é apresentada por [RIE 96], em tradução livre:

"*Um padrão é a abstração de uma forma concreta que continua a recorrer em contextos não-arbitrários específicos*".

Segundos os autores, esta definição seria mais genérica que outras encontradas na literatura de *software*, que normalmente se concentram na aplicação de padrões a

resolução de problemas de projeto (*design patterns*). Esta definição não seria restrita a uma forma específica de aplicação. Seria mais genérica que a definição mais popular de um padrão como "*uma solução para um problema recorrente em um contexto*", a qual seria guiada a resolver problemas de projeto.

A definição mais popular de um padrão é a definição de Alexander, apresentada na seção anterior, que define um padrão como uma regra de três partes dentro de um contexto. Esta noção de um padrão como uma regra influenciou diversos trabalhos como os de Erich Gamma [GAM 95] e de James Coplien [COP 96]. O último descreve em seu livro sobre padrões de *software* (tradução livre):

"*Eu gosto de relacionar esta definição com padrões para vestimentas. Eu poderia dizer-lhe como fazer um vestido especificando a rota da tesoura através de um pedaço de tecido através dos ângulos e comprimentos do corte. Ou então, eu posso te dar um padrão. Lendo a especificação, você não teria a menor idéia do que está sendo construído, ou se você construiu a coisa certa quando terminar. O padrão antevê ao produto: é uma regra para construir algo, mas também é, em muitos aspectos, o algo em si próprio*".

Através desta explicação, pode-se entender que um padrão não descreve uma seqüência de passos específica para resolver um problema. Descreve, isto sim, uma solução genérica de tal forma que a pessoa que irá aplicá-lo deverá compreender o problema e a solução. Por isso, os padrões são adequados para a descrição de soluções a problemas complexos, nos quais não é possível descrever-se uma seqüência de ações a qual baste ser seguida para que se solucione o problema.

Além disso, um padrão não é somente a solução abstrata, mas muitas vezes se confunde com a solução concreta do problema. Por exemplo, no padrão de projeto Adaptador (*Adapter*) [GAM 95], é descrita uma solução para a compatibilidade com classes legadas. Uma instância de solução para um problema específico também é chamada de Adaptador.

Outra metáfora para a aplicação de padrões como um instrumento a resolução de problemas e sistemas complexos é apresentada também por Coplien (citando Ward Cunningham):

"*Eu gosto de fazer a distinção entre um plano e uma receita. Um plano pode ser obtido por engenharia reversa de um edifício, mas uma receita não pode ser (facilmente) obtida por engenharia reversa de um bolo. Nosso genoma é uma receita, não um plano. Receitas tendem a servir melhor como um esquema para sistemas adaptativos complexos*".

Coplien [COP 96] acredita ser a comunicação humana o gargalo do desenvolvimento de *software*, mas que os padrões podem ser utilizados como um instrumento eficaz para comunicação com clientes, com fornecedores, e entre membros de um time de desenvolvimento, desta forma resolvendo um problema importante do desenvolvimento de *software* contemporâneo. O autor afirma que, ao final, um padrão nada mais é do que uma forma eficiente de documentação. Há algumas poucas formas literárias, relacionadas entre si, que os padrões assumiram ao longo do tempo.

Os padrões seriam mais adequados a descrição de atividades humanas, as quais não possam ser facilmente descritas como uma simples transformação de

artefatos de projeto e automatizadas. Eles não seriam inteligência artificial, pois celebram e encorajam a inteligência que separa os humanos dos computadores.

Um bom exemplo de um padrão desenvolvido por Cristopher Alexander no livro *A Pattern Language* [ALE 77], e citado por Coplien, está descrito abaixo:

"*Um local para esperar*

O processo de espera têm conflitos inerentes. Quando pessoas estão esperando por um doutor, um avião, ou um compromisso de negócio, há incertezas presentes, que inevitavelmente fazem com que gastem um tempo longo andando por perto, esperando, fazendo nada.

Por outro lado, normalmente eles não conseguem aproveitar este tempo. Porque é imprevisível, eles precisam estar ao lado da porta. Já que nunca sabem quando sua vez chegará, não podem dar uma caminhada ou sentar do lado de fora...

Portanto:

Em locais em que pessoas esperam, crie uma situação que faça a espera ser positiva. Disperse a espera com algum tipo de atividade como jornal, café, mesas de sinuca ou jogo da ferradura: algo que envolva as pessoas para que não fiquem simplesmente esperando. E também o oposto: crie um ambiente que envolva a pessoa esperando em uma reverência; calmo, um silêncio positivo". [ALE 77, páginas 707, 711]

Segundo James Coplien [COP 96], um dos maiores desafios no desenvolvimento de *software* contemporâneo é conseguir lidar adequadamente com a complexidade. De forma geral, esta é atacada com abstração. Muito do que utiliza-se no projeto de soluções é resolvido ao encontrar-se a abstração correta para um sistema, e ao dividir-se este sistema em pedaços menores, mais controláveis. Para isto recorre-se a paradigmas, que seriam conjuntos de regras, princípios e ferramentas. No entanto, há espaços na maior parte dos paradigmas e seus métodos, os quais não são preenchidos. Normalmente, os bons projetistas de *software* conseguem preencher estes buracos adequadamente, através de sua experiência passada ou da sua intuição. Este tipo de conhecimento, o qual não é normalmente descrito nos métodos e paradigmas de desenvolvimento de *software*, é que se busca capturar nos padrões.

Ainda segundo Coplien, um bom padrão:

- **Resolve um problema**: Padrões capturam soluções, não apenas princípios abstratos ou estratégias.
- **É um conceito comprovado**: Padrões capturam soluções que foram registradas, não teorias ou especulações. Uma boa regra, descrita em [CUN 2003], é de que um padrão deve ter sido aplicado na prática ao menos três vezes para poder ser considerado um padrão.
- **A solução não é óbvia**: Muitas técnicas de resolução de problemas (como paradigmas e métodos de projeto de *software*) tentam derivar as soluções a partir de princípios básicos. Os melhores padrões geram uma solução para um problema indiretamente – o que é uma abordagem necessária para os problemas mais difíceis do projeto.

- **Descreve um relacionamento**: Padrões não descrevem simplesmente módulos, mas descrevem estruturas e mecanismos mais profundos dos sistemas.
- **Possui um componente humano significativo**: Todo *software* serve para o conforto humano ou para sua qualidade de vida; os melhores padrões explicitamente apelam para a estética e para a utilidade.

2.3 Padrões generativos

Uma característica importante aos bons padrões é que estes seriam **generativos**. Conforme foi descrito no item anterior, os padrões seriam adequados para a resolução de problemas para os quais a solução não é óbvia, portanto os mesmos procuram atacar um problema de forma indireta. Na maior parte das abordagens para a resolução de problemas, tenta-se resolvê-los diretamente. No entanto, desta forma muitas vezes se resolvem apenas os sintomas, deixando-se o problema original não resolvido. Segundo Coplien [COP 96], Alexander percebeu que as boas soluções de arquitetura vão um nível adiante. As estruturas que o padrão apresenta não são uma solução por si somente, mas sim elas **geram** as soluções. Quando os padrões trabalham desta forma, eles seriam chamados de **padrões generativos**.

Segundo Alexander, conforme citado por Appleton [APP 97a], em tradução livre:

"*Os padrões em nossa mente são, mais ou menos, imagens mentais dos padrões no mundo; são representações abstratas das regras morfológicas que definem padrões em nosso mundo. No entanto, em um sentido eles são bastante diferentes. Os padrões no mundo meramente existem. Mas os mesmos padrões em nossa mente são dinâmicos. Eles têm força. Eles são generativos. Eles nos dizem o que fazer; eles nos dizem como nós devemos, ou podemos, gerá-los; e eles também nos dizem que, sob certas circunstâncias, nós devemos criá-los. Cada padrão é uma regra que descreve o que você deve fazer para criar a entidade que ele define*".

Padrões generativos seriam dinâmicos, ativos. Nos diriam como criar soluções, e poderiam ser observados como resultado do sistema que ajudaram a criar. Padrões não-generativos, no entanto, seriam passivos e estáticos; descreveriam um fenômeno recorrente sem necessariamente descrever como fazer para reproduzi-los [APP 97a]. Portanto, nós deveríamos tentar descrever padrões generativos porque estes não somente descrevem as características de um bom sistema, mas também nos ensinam como construí-lo.

Appleton afirma que Alexander vai ainda um passo além em relação ao conceito de **generatividade**. Padrões poderiam – quando combinados na forma de linguagens de padrões – gerar estruturas inteiras e vivas. O desejo de criar estruturas de arquitetura ocorreria pela habilidade das coisas vivas de evoluir e de se adaptar ao seu ambiente em freqüente modificação. Alexander queria que suas arquiteturas tivessem estas propriedades. De forma similar, em *software*, as melhores arquiteturas seriam aquelas que teriam a capacidade de se adaptar às mudanças em seu ambiente. Assim, o conceito de generatividade poderia ser entendido como a capacidade de se criar coisas "vivas", as quais conseguem evoluir e se adaptar as suas novas necessidades e demandas.

Por fim, outra propriedade dos padrões generativos seria a de **emergência**. A sucessiva aplicação de padrões, cada um deles separadamente resolvendo seu problema, criaria uma solução maior a qual seria emergente como resultado das soluções menores.

2.4 Linguagens de Padrões

O termo “linguagem de padrões” foi popularizado pelo arquiteto Alexander em seus trabalhos na construção de edifícios. Uma linguagem de padrões é uma coleção de padrões, que funcionam de forma conjunta de forma a gerar um sistema inteiro, enquanto que um padrão isolado resolve apenas um problema de projeto isolado.

Segundo Coplien [COP 96], uma linguagem de padrões não deve ser confundida com uma linguagem de programação. O sentido utilizado para o termo linguagem, neste caso, não é muito utilizado no campo de computação. Uma linguagem de padrões é uma peça de literatura que descreve uma estrutura de projeto, como uma arquitetura, um arcabouço (*framework*) ou uma técnica. Possui uma estrutura, mas esta não é tão formal quanto normalmente o é nas linguagens de programação. Por este motivo, alguns autores chamariam este termo de “sistema de padrões”.

Já de acordo com [DEV 2002], uma linguagem de padrões seria uma coleção de padrões inter-relacionados os quais provêem uma linguagem comum a respeito de um determinado problema. Não seriam descritos de forma formal, mas sim de uma forma literária com certa estrutura. Segundo os autores, as linguagens de padrão ajudariam os desenvolvedores de *software* a se comunicar melhor.

É importante a distinção entre uma linguagem de padrões e um catálogo de padrões. Em um catálogo, padrões independentes resolveriam problemas individualmente. Um exemplo de um catálogo de padrões é o livro de padrões de projeto GoF [GAM 95], nos quais diversos padrões são apresentados. Cada padrão resolve separadamente um problema relativamente comum no projeto de aplicações orientadas a objeto. Segundo [DEV 2002], nas linguagens de padrões, grupos de padrões inter-relacionados são aplicados a uma mesma área do projeto ou do problema de forma a resolve-lo de forma integrada.

Uma linguagem de padrões pode ser representada graficamente na forma de um grafo acíclico dirigido. O número de caminhos alternativos na linguagem de padrões é bastante grande, portanto esta não se limita a ser uma árvore de decisões [COP 96].

Coplien cita Alexander, em tradução livre, em relação ao fato de que as linguagens de padrões colocam os padrões em um contexto:

“*Então, cada padrão depende dos padrões menores que contém, e dos padrões maiores nos quais está contido. ...*

É a rede de conexões entre estes padrões que forma a linguagem.

Nesta rede, as conexões entre os padrões são quase tanto parte da linguagem quanto os próprios padrões.

De fato, é a estrutura da rede que faz com que os padrões individuais tenham sentido, pois os ancora, os ajuda a torná-los completos.

Mas mesmo quando eu tenho os padrões conectados uns aos outros, em uma rede, de tal forma que formem uma linguagem, como eu sei se esta linguagem é uma boa linguagem?

A linguagem é uma boa linguagem de padrões, capaz de fazer algo por completo, quando esta está completa morfologicamente e funcionalmente.

A linguagem é morfologicamente completa quando eu consigo visualizar os tipos de edifício que esta gera de forma bastante concreta.

E a linguagem é funcionalmente completa, quando o sistema de padrões que define é plenamente capaz de permitir a todas as suas forças interiores a resolverem a si mesmo. ”

Por fim, alguns autores têm uma visão bastante visionária da aplicação de linguagens de padrões. Brad Appleton [APP 97a] cita Mike Beedle [BEE 97]:

“*Michael Beedle, autor de Redesenhando o Processo de Desenvolvimento de Aplicações, liga os efeitos da utilização de linguagens de padrões com a geração de comportamentos emergentes: padrões recorrentes espontâneos de interação local entre as entidades de forma densa, resultando em sistemas dinâmicos e auto-organizáveis que são adaptativos, abertos, e capaz de efeitos em múltiplas escalas. Em outras palavras, linguagens de padrões provêem um processo dinâmico para a resolução ordenada de problemas dentro de seu domínio, que de uma forma indireta levam a resolução de um problema muito mais amplo. Os padrões e regras em uma linguagem de padrões combinam-se para formar um estilo arquitetural. Desta forma, linguagens de padrões guiam analistas, arquitetos, projetistas, e codificadores para produzir sistemas funcionando que resolvem problemas organizacionais e de desenvolvimento em muitos níveis de escala e diversidade”.*

2.5 Anti-padrões

Segundo [COP 96], anti-padrões seriam uma forma de documentar na forma de padrões práticas destrutivas ou que não funcionam. Podem ser uma descrição oposta a um determinado padrão, o qual apresenta, este sim, a solução recomendada para o problema. Os anti-padrões não apresentariam soluções como os padrões o fazem. Também não seriam recomendados como instrumento de ensino, pois recomenda-se que os alunos sejam ensinados através de exemplos positivos. Poderiam, isto sim, ser utilizados como instrumentos de diagnóstico de problemas.

Os anti-padrões representariam “lições aprendidas”, ao invés das “melhores práticas” representadas nos padrões. Segundo [APP 97a], haveria dois tipos de anti-padrões: *a. Aqueles que descrevem uma má solução para um problema, que resulta em uma má situação. b. Aqueles que descrevem como escapar de uma má situação e como proceder a partir de então para um boa solução.*

2.6 Formato literário de descrição de padrões

Padrões são descritos de uma forma literária, pois no fundo são apenas uma forma estruturada – mas não formal – de documentação. Existem diversos formatos que são utilizados na literatura para descrever padrões, portanto nesta seção será feito um estudo das formas que são mais utilizadas.

O formato de descrição de um padrão é importante, pois tem um propósito [COP 96]. Introduz o leitor ao problema, descreve o contexto em que o problema pode ocorrer, analisa o problema, e por fim apresenta uma solução que elucida o problema.

Os formatos em que os padrões são escritos são divididos em seções. Segundo Coplien, um bom padrão pode ser escrito ao se atender aos objetivos de cada uma das seções. Já escrever um excelente padrão é mais difícil, pois é um caminho mais subjetivo e heurístico. De acordo com o autor, as seguintes seções são comumente utilizadas para a descrição de padrões, e aparecem de uma forma explícita ou implícita nos formatos de descrição de padrões mais comuns, como o formato GoF (utilizado no livro de padrões de projeto), o formato Coplien, o formato de *Portland*, e o formato original utilizado por Christopher Alexander.

- **Nome:** É importante que o nome escolhido para descrever o padrão seja significativo, para que seja fácil de ser lembrado e identificá-lo com a resolução do problema a que se propõe. Assim, fica mais fácil para um projetista identificar um padrão que lhe seja útil, dentro de uma linguagem ou de um catálogo que não lhe seja familiar. Além disso, os nomes dos padrões rapidamente fazem parte do vocabulário de uma equipe; nomes curtos, mas com significado bem-escolhido, facilitam a comunicação.
- **Objetivo:** O objetivo é uma frase ou sentença que descreve brevemente o que um padrão faz, sumarizando o problema a que este se propõe a resolver. Assim como o nome, este tem como objetivo facilitar que um projetista encontre a solução para um problema que está procurando resolver, ao percorrer um catálogo, ou uma linguagem de padrões.
- **Problema:** Descreve qual o problema que o padrão se propõe a solucionar. O ideal é que esta descrição seja concisa, de forma a auxiliar o leitor a identificar se o padrão pode lhe ser útil ou não. Muitas vezes esta seção é combinada com o objetivo, e aparece como uma simples pergunta.
- **Contexto:** De forma geral, esta seção é descrita de forma explícita. Em uma linguagem de padrões, o contexto descreve o histórico de padrões que deve ter sido aplicado antes que o padrão atual seja aplicado. Especifica o escopo, o tamanho, o público alvo, a linguagem de programação. Caso alguma das variáveis descritas no contexto seja alterada, o padrão fica inválido. Nas linguagens de padrões o contexto é essencial para que possa-se fazer um encadeamento de padrões, que trabalham em conjunto para resolver um problema a nível do sistema como um todo. Segundo Coplien [COP 96], é difícil de escrever a seção de contexto, e na medida em que um padrão amadurece, a tendência é que os projetistas encontrem novas situações que invalidam o contexto, de forma que o mesmo tende a ficar mais restritivo. Em alguns casos, o contrário ocorre, ou seja, encontra-se novos contexto nos quais um mesmo padrão pode ser aplicado, e esta seção é estendida. É importante que a descrição do contexto de um padrão acompanhe o amadurecimento da utilização do mesmo.

- **Forças:** Padrões não são seqüência de passos que devem ser seguidas sem compreensão do racional envolvido. Os padrões devem ser bem entendidos, de forma que possamos adequá-los as nossas próprias necessidades. Entendendo as forças podemos entender melhor o problema com que estamos lidando, e a solução que é proposta. Aprendemos a tomar decisões e balancear entre as forças envolvidas, assim compreendendo parte do racional por trás de um padrão. O termo "forças" é herança da área de arquitetura e engenharia civil, onde diversas forças físicas estão envolvidas, como a força da gravidade. No desenvolvimento de *software*, há forças técnicas e humanas que podem ser consideradas. As forças auxiliam o projetista a entender a dificuldade de um problema. Esta seção nem sempre aparece explicitamente na descrição de padrões na literatura, mas aparecem também combinadas com a seção de problema.
- **Solução:** Segundo Coplien, uma boa solução é descrita de forma a ser suficientemente detalhada para explicar ao projetista o que fazer para resolver o problema, mas é genérica suficiente para atingir um contexto amplo, ao mesmo tempo. A solução deve resolver a tensão gerada pelas forças envolvidas, solucionando o problema descrito na seção específica. Alguns padrões resolvem o problema apenas parcialmente, deixando um novo contexto para que outros padrões possam resolver as forças ainda não solucionadas.
- **Desenho:** Segundo Alexander, "*se você não pode fazer um desenho, então não é um padrão*". Por isso, fazer um desenho na forma de um diagrama é muito importante para auxiliar no entendimento de um padrão. No caso de um padrão em *software*, podemos elaborar um diagrama de qualquer coisa que possa auxiliar um projetista a entender a solução do problema. Por exemplo, no livro de padrões de projeto GoF [GAM 95], os autores utilizam-se da linguagem de modelagem OMT (uma notação precursora da UML para a modelagem de objetos) para representar a estrutura dos objetos envolvidos no problema.
- ***Design Rationale***: Uma explicação sobre porque o padrão funciona, qual o histórico de raciocínio por trás do padrão. É uma seção para aprendizado, mais do que para a ação de resolução do problema.
- **Contexto Resultante:** Descreve o resultado da aplicação do padrão: quais as forças que foram resolvidas, quais que serão deixadas para serem resolvidas posteriormente, quais novos problemas podem vir a ocorrer, e quais padrões estão relacionados e podem ser aplicados em seguida.

Um padrão, dentro de uma linguagem de padrões, deve ser pensado de forma a transformar um sistema de um contexto para um novo contexto, a partir do qual novos padrões podem ser aplicados. É esta relação entre os contextos dos padrões que forma uma linguagem de padrões.

Os padrões são descritos dentro de uma forma literária. Há diversos formatos semelhantes ao apresentado acima com variações que são utilizados na literatura para descrevê-los. Alguns formatos apresentam de forma implícita ou omitem algumas seções.

2.7 Padrões organizacionais e de processos

Ward Cuningham [CUN 2003] afirma que os padrões de processos estão para a arquitetura social de uma organização assim como os padrões de projeto estão para a arquitetura e o projeto de *software*. Segundo esta fonte, Jim Coplien teria fundado um programa na empresa AT&T para estudar as organizações utilizando técnicas emprestadas da análise orientada a objetos, tais como a utilização de cartões CRC (*class-responsibility collaboration*). Os dados seriam visualizados como uma rede social, em que cada cartão seria um nodo nesta rede. Estas visualizações teriam ajudado a encontrar diversos padrões de processos sociais nesta organização, de forma semelhante a um estudo antropológico. Coplien afirmaria que os padrões de processos estariam mais próximos do espírito do trabalho de Alexander até mesmo do que os padrões de projeto, pois tratariam de aspectos sociais presentes no dia a dia das pessoas, o que Alexander tentava fazer com seus padrões.

Os padrões organizacionais seriam muito diferentes dos demais tipos de padrões utilizados na engenharia de *software*, pois capturam os aspectos gerenciais e sociais de uma organização. No desenvolvimento de *software*, podem ser utilizados para auxiliar no gerenciamento e na organização de um projeto. Podem ser também aplicados à modelagem de organizações, ao representar os processos sociais e de gerência, que não necessariamente estão relacionados ao *software*. Quando aplicados a organizações de desenvolvimento de *software*, endereçam problemas de desenvolvimento que todas organizações deste tipo têm de resolver, desta forma suportam o projeto do *software*, mas não o resolvem diretamente [DEV 2002]. Alguns padrões de organizacionais poderiam ser utilizados para definir novas organizações e seus processos, enquanto outros poderiam ser aplicados para evoluir organizações existentes.

Os padrões organizacionais seriam normalmente obtidos através do estudo de caso de organizações altamente produtivas [DEV 2002], ou através do senso comum. Mas seria difícil estabelecer uma métrica simples para julgar e avaliar este tipo de padrão. Por este motivo, o valor de um padrão organizacional seria sempre julgado empiricamente pelos gerentes das organizações que os aplicam.

Os padrões de processos, por sua vez, seriam relacionados aos métodos e técnicas que os profissionais utilizariam para resolver os problemas de uma forma recorrente em suas organizações. Na literatura, são geralmente voltados para os processos de desenvolvimento de *software*, embora possam ser aplicados também a outros tipos de processos. Normalmente os padrões de processos não seriam descritos de forma isolada, mas sim em conjunto, de forma a formarem uma linguagem de padrões. Os padrões de processos seriam normalmente obtidos através do estudo de padrões recorrentes em organizações de sucesso. Estes padrões focam mais no que é feito, mais do que como é feito, desta forma abstraindo detalhes e permitindo que uma determinada organização possa utilizá-los para desenvolver seus processos específicos de trabalho.

James Coplien publicou um artigo em que apresenta uma série de padrões organizacionais generativos que poderiam ser aplicados para aprimorar uma organização de desenvolvimento de *software* [COP 95]. Ao mesmo tempo em que propõe que estes padrões seriam uma forma poderosa para capturar as práticas adequadas de gerenciamento de projetos de *software*, ele descreve o que acredita serem as limitações deste tipo de abordagem (tradução livre):

"*(...) Há um crescente entendimento de que novas técnicas de estruturação de programação devem ser suportadas por técnicas de gerenciamento adequadas, e por*

estruturas de organização adequadas; padrões organizacionais são uma forma de capturá-las. (...) Nós entendemos que os padrões são particularmente adequados à construção e à evolução de uma organização.

(...) Não há nada de novo em aplicar padrões para a análise organizacional. O que é novidade aqui neste trabalho é a tentativa de aplicar padrões de uma forma generativa. (...) Não somente os padrões nos ajudariam a entender as organizações existentes, mas deveriam nos ajudar a construir novas organizações. Um bom conjunto de padrões organizacionais ajuda a (indiretamente) gerar o processo correto; esta forma indireta é a essência da generatividade de Alexander.

(...) Na data corrente, este trabalho é especulativo: somente uso limitado foi feito destes padrões na formulação de novas organizações. A qualidade ou a ruindade de tais padrões é difícil de testar por experimentos.

(...) Por esta razão, os padrões caem em estudos empíricos e senso comum. Nós procuramos por padrões recorrentes de interação nas organizações, notamos relações recorrentes entres estes padrões e alguma medida para qualidade, e então fazemos análise para explicar esta relação".

Um exemplo de uma linguagem de padrões organizacionais pode ser encontrado no *web site* de James Coplien [COP 2004]. Outro exemplo é a metodologia *Scrum* [SCH 2002] publicada na forma de padrões organizacionais [BEE 99].

3 PROCESSOS DE SOFTWARE

Neste capítulo será feito um estudo sobre modelos para a melhoria de processos de *software*, sobre o conceito de melhoria de processos de *software* de forma geral. Em [HAR 2004], aprofundou-se o estudo de alguns trabalhos que reportam problemas na aplicação dos modelos para melhoria de processos.

Os modelos para melhoria de processos de *software* têm como objetivo inicial avaliar a qualidade dos processos de desenvolvimento sendo utilizados, de forma a permitir sua melhoria. Estes modelos assumem que a qualidade do *software* produzido é largamente influenciada pela qualidade dos processos utilizados para o seu desenvolvimento [MAN 2003a]. Assim, estes modelos baseiam-se na idéia de que a melhoria da qualidade de *software* pode ser obtida pela melhoria da qualidade dos processos utilizados na organização. Vários modelos foram desenvolvidos baseados nesta idéia, os quais servem paraa definição, avaliação e melhoria de processos de *software*.

Na seção 2.2, serão estudados dois *frameworks* de processos de desenvolvimentos que podem também ser utilizados em esforços de melhorias de processos de *software*, Open e RUP. Será estudado de que forma estes *frameworks* podem ser adaptados para uma organização ou projeto específico.

3.1 Modelos para a melhoria de processos de software

3.1.1 ISO 9000-3

Esta norma foi desenvolvida com o propósito de guiar a aplicação de ISO 9001 para o desenvolvimento, fornecimento e manutenção de *software*. Cada item existente na ISO 9001 tem um item correspondente na ISO 9000-3 que o aplica ao *software* [MAN 2003a].

A norma ISO 9000-3 é dividida em três partes principais, que são: *1. estrutura, descrevendo os aspectos organizacionais. 2. atividades de ciclo de vida de desenvolvimento de software. 3. atividades de suporte que apóiam as atividades do ciclo de vida de desenvolvimento*.

As diretrizes propostas nesta norma cobrem questões como prover o entendimento comum entre a contratante e a contratada, em relação aos requisitos funcionais, e o uso de metodologias consistentes para o desenvolvimento e gerenciamento do projeto de *software* como um todo, desde sua concepção até a manutenção.

3.1.2 ISO/IEC 15504 (SPICE)

Esta norma foi gerada a partir de um estudo realizado em 91, feito pelo comitê de engenharia de *software* da ISO, o qual apontou um consenso internacional sobre a necessidade e a importância de implementação de uma norma para a avaliação do processo de *software*.

O projeto SPICE (*Software Process Improvement and Capability Determination*) foi criado em 93 com o objetivo de gerar normas para a avaliação dos processos de desenvolvimento de *software* de uma organização, possibilitando a melhoria contínua de tais processos, e determinar sua capacidade. A norma ISO/IEC 15504 é também chamada de SPICE, e pode ser utilizada por uma organização de desenvolvimento de *software* para planejar, gerenciar, monitorar, controlar e melhorar a aquisição, fornecimento, desenvolvimento, operação, evolução e suporte de *software* [MAN 2003a].

Os objetivos da norma ISO 15504 são: *a. entender o estado dos processos de uma organização, buscando sua melhoria. b. determinar a adequação dos processos de uma organização a um requisito em particular, ou a uma classe de requisitos. c. determinar a adequação dos processos a um determinado contrato ou classe de contratos.*

A norma ISO 15504 define um modelo de referência, o qual serve de base para a avaliação dos processos. Este modelo estabelece um conjunto universal dos processos que seriam fundamentais para a engenharia de *software*, e, para cada processo, um modelo racional sobre como melhora-lo. O modelo é dividido em cinco grandes categorias de processos: cliente-fornecedor, engenharia, suporte e organização, sendo que cada uma destas categorias é descrita em processos mais específicos. A seguir será feito um detalhamento destas categorias, de acordo com [MAN 2003a]:

- **Cliente-fornecedor**: estes processos tratam sobre o relacionamento do cliente com o fornecedor, facilitando o suporte e a transição do *software* para o cliente, e provendo a operação e uso correto do *software*.
- **Engenharia**: estes processos especificam, implementam, ou mantém um sistema ou produto de software e sua documentação.
- **Projeto**: consiste nos processos que estabelecem o projeto, coordenam e gerenciam seus recursos para produzir um produto e prover serviços que satisfaçam o cliente.
- **Suporte**: define processos que podem ser empregados por qualquer dos outros processos.
- **Organização**: estabelecem os objetivos do negócio da organização.

Além disso, no modelo de referência são definidos seis níveis de capacitação, os quais são utilizados para a avaliação de uma organização existente. A avaliação é feita selecionando-se os processos relevantes e para cada um deles é atribuído um perfil que é composto pela percentagem de adequação a cada um dos níveis de capacitação. Os seguintes níveis de capacitação são definidos [MAN 2003a]:

- **Processo Incompleto (nível 0)**: não há processo implementado e há falha no atendimento dos objetivos.

- **Processo Executado (nível 1)**: o processo que está sendo executado atinge seus objetivos.
- **Processo Gerenciado (nível 2)**: o processo sendo executado é capaz de entregar produtos de trabalho com uma qualidade definida, obedecendo a cronogramas e a recursos previamente definidos.
- **Processo Estabelecido (nível 3)**: o processo gerenciado é executado de acordo com bons princípios de engenharia de *software*.
- **Processo Previsível (nível 4)**: o processo estabelecido é executado de acordo com limites definidos de controle para atingir seus objetivos.
- **Processo Otimizado (nível 5)**: o processo previsível otimiza o seu desempenho para atender as necessidades de negócio atuais e futuras e consegue ser repetido para atender novamente os objetivos de negócio previamente definidos.

Segundo [MAN 2003a], o modelo SPICE – ISO/IEC 15504 é bastante flexível, possibilitando sua utilização conforme as necessidades de negócio das organizações, sempre objetivando a melhoria contínua dos processos.

3.1.3 CMM – Capability Maturity Model

Segundo [MAN 2003a], o modelo *CMM* ou *SW-CMM – Capability Maturity Model for Software*, foi desenvolvido em 91, com o objetivo de avaliar a capacidade e a maturidade dos processos de desenvolvimento de *software* de uma organização. Este modelo foi desenvolvido pelo *SEI – Software Engineering Institute*, e foi apoiado pelo departamento de defesa dos Estados Unidos, o qual é um grande consumidor de *software* e precisava de um modelo formal o qual permitisse selecionar seus fornecedores de forma adequada.

Segundo [ORR 2002], as origens do modelo CMM são projetos de larga escala em desenvolvimento de *software* militar, principalmente os projetos do Departamento de Defesa americano (DoD). Por muitas décadas, este departamento vem promovendo a utilização de metodologias rigorosas em seus projetos, e, como este é um grande consumidor de *software*, muitas outras empresas militares e aeroespaciais adotaram (ou foram forçadas a adotar) as metodologias empregadas pelo mesmo, como o modelo em cascata (*waterfall*). Como resultado, as organizações utilizando métodos rigorosos são freqüentemente grandes empresas de contratos militares. Como muitas organizações atualmente precisam mais e mais gerenciar projetos que são desenvolvidos por organizações externas, estas se espelham no Departamento de Defesa e aplicam CMM e forçam sua adoção por suas empresas contratadas.

O CMM se caracterizaria por buscar tornar as organizações mais maduras em relação a seus processos. Uma organização pode utiliza-lo para determinar sua classificação com relação à maturidade de seus processos e assim definir suas prioridades para melhoria. Há cinco níveis de maturidade que são definidos neste modelo, os quais são baseados nos princípios de qualidade utilizados na indústria de manufatura. Com exceção do nível 1, cada nível de maturidade é composto por áreas-chave de processo, os quais priorizam ações de melhoria quer podem ser utilizadas. Os cinco níveis de maturidade podem ser caracterizados da seguinte forma:

1. **Inicial**: neste nível, o processo de *software* é caracterizado como *ad hoc* ou mesmo caótico. Há poucos processos definidos e o sucesso depende de esforços individuais e heróicos.
2. **Repetível**: neste nível, há processos básicos para o gerenciamento de projeto os quais estão estabelecidos, permitindo assim que se acompanhe custo, cronograma e funcionalidade. É possível repetir o sucesso de um processo utilizado anteriormente em outros projetos semelhantes.
3. **Definido**: as atividades de gerenciamento e de engenharia do processo estão documentadas, padronizadas e integradas em um processo de *software* da organização. Todos os projetos seguem este processo, para desenvolver e manter o *software*.
4. **Gerenciado**: neste nível, são coletadas medidas de qualidade detalhadas dos produtos e processos de *software*. Tanto o produto quanto os processos são entendidos e controlados quantitativamente.
5. **Otimizado**: A melhoria contínua do processo é possibilitada pela realimentação quantitativa do processo, e conduzida a partir de idéias e tecnologias inovadoras.

Cada um dos níveis de maturidade dos processos, com exceção do nível 1, é composto por várias áreas chave de processo. Cada área-chave é organizada em cinco seções, as quais são chamadas de características-comuns. Os principais componentes do CMM seriam [MAN 2003a]:

- **Níveis de Maturidade**: servem para estabelecer um platô evolucionário bem definido com o objetivo de se alcançar um processo maduro. Os níveis seriam uma forma de priorizar as ações de melhoria.
- **Capacidade de Processo**: a capacidade de um processo descreve o conjunto de resultados que são esperados, e que podem ser alcançados, ao se seguir o processo de *software*. Através da capacidade de um processo de *software* seria possível predizer os resultados esperados para o próximo projeto da organização.
- **Áreas-chave de Processo**: para cada nível de maturidade, são definidas as áreas-chave que, quando executadas coletivamente, alcançam as metas consideradas importantes para estabelecer a maturidade almejada.
- **Metas**: são utilizadas para determinar se uma organização ou projeto implementou efetivamente uma área-chave de processo.
- **Práticas-chave**: as áreas-chave de processos são descritas em termos de práticas-chave que, quando implementadas, ajudam a satisfazer suas metas. Estas descrevem a infraestrutura e as atividades que contribuem para a efetiva implementação e institucionalização da área-chave de processo. No modelo CMM completo seriam descritas 316 práticas-chave.
- **Características-comuns**: seriam atributos que serviriam para indicar se a implementação de uma área-chave de processo é efetiva, repetível

> e duradoura. Representariam uma forma de organizar as práticas-chave. As cinco características comuns seriam as seguintes: *1. compromisso de executar, o qual descreve as ações que a organização deve adotar para garantir que o processo está estabelecido e vai perdurar. Normalmente, envolve políticas organizacionais e envolvimento direto da gerência sênior. 2. capacidade para executar, que descreve as pré-condições que garantiriam que o projeto ou organização implemente o processo de forma competente. 3. atividades executadas: atividades, papéis e procedimentos necessários para implementar a área-chave. 4. Medição e análise: como determinar o status relativo ao processo, com o objetivo de controlar e melhorar o processo. 5. Verificação da implementação, que descreve as etapas utilizadas para assegurar que as atividades são executadas de acordo com o processo estabelecido*.

A partir do modelo CMM original, foram desenvolvidas variações, como os modelos para *Software* (SW-CMM), *Software Acquisition* (SA-CMM), *Systems Engineering* (SE-CMM), *Integrated Product Development* (IPD-CMM) e *People* (P-CMM). Como existiam diferenças estruturais e conceituais entre os modelos, o que gerava dificuldades para sua implementação em organizações, houve motivação para a criação de um modelo integrado, chamado de CMMI – *Capability Maturity Model Integration*. Este modelo tem como objetivo integrar os diferentes modelos em uma única estrutura, com a mesma terminologia e os mesmos processos de avaliação, facilitando assim a sua adoção pela empresa.

3.2 Processos orientados a objetos dirigidos a planos

Esta seção tem como objetivo estudar resumidamente dois processos de desenvolvimento de *software* os quais são classificados como dirigidos a planos, para diferenciá-los das metodologias ágeis, que serão estudadas no capítulo 3. Na seção 2.2.1 será estudado o Processo Unificado da Rational (RUP – *Rational Unified Process*), enquanto que na seção 2.2.2 será estudado o processo Open (*Object-oriented Process Environment and Notation*). Por fim, na seção 2.2.3 estes dois processos serão avaliados quanto a sua capacidade de adaptação e flexibilidade.

3.2.1 Rational Unified Process (RUP)

O Processo Unificado da Rational (RUP) [JAC 2000] é um processo genérico para engenharia de software utilizando-se o paradigma da orientação a objetos. Procura impor um enfoque disciplinado para a distribuição das tarefas e das responsabilidades dentro de uma organização de desenvolvimento de software, com o objetivo de assegurar a produção de produtos de alta qualidade que satisfaçam as necessidades de seus usuários finais, dentro de cronogramas e orçamentos previsíveis. O Processo Unificado da Rational captura as melhores práticas modernas em desenvolvimento de *software*, e pode ser adaptado a diferentes projetos e organizações.

O RUP utiliza a Linguagem de Modelagem Unificada (UML) para descrever os modelos e diagramas do sistema, e descreve as atividades que são necessárias para

transformar os requisitos do usuário em um sistema de *software*. As principais características deste processo são descritas abaixo:

- **Dirigido a casos de uso**: Casos de uso são utilizados para descrever os requisitos do usuário, e podem ser apresentados visualmente de forma sintética, no formato de um diagrama de casos de uso. Os casos de uso são utilizados não somente para especificar os requisitos do sistema, mas também para dirigir os demais processos de projeto, teste e implementação [MAN 2003]. Os desenvolvedores criam os modelos de projeto e de implementação a partir do modelo de casos de uso. Os testadores do sistema utilizam os casos de uso para validar o sistema. O termo dirigido por casos de uso significa que o processo de desenvolvimento segue um fluxo, especificado por uma série de disciplinas que derivam dos casos de uso. Casos de uso são especificados, casos de uso são projetados, e, no final, são construídos casos de teste a partir dos casos de uso. Os casos de uso devem ser definidos junto com a arquitetura do sistema, e um influencia o outro, e ambos amadurecem enquanto o ciclo de vida evolui.

- **Centrado na arquitetura**: Várias visões do sistema que está sendo construído são utilizadas para descrever a arquitetura do sistema. A arquitetura descreve os aspectos dinâmicos e estáticos mais importantes na construção do sistema. A arquitetura é originada das necessidades da organização, da visão do usuário, e dos casos de uso. Porém, ela é também influenciada por vários outros fatores, como a plataforma de *software* e de *hardware* onde o *software* vai executar, os componentes reusáveis disponíveis (como *frameworks* e bibliotecas de classes), considerações de implementação, necessidade de integração com sistemas legados, e requisitos não-funcionais (desempenho, segurança). A arquitetura deve ser pensada de forma que o sistema evolua, não somente pensando em seu desenvolvimento inicial, mas pensando em gerações futuras. Para encontrar esta forma, o arquiteto deve trabalhar com o entendimento dos casos de uso mais importantes do sistema, possivelmente os 5 a 10% mais importantes casos de uso, os quais constituem o núcleo das funções do sistema.

- **Iterativo e incremental**: Como o desenvolvimento de um produto de *software* é uma atividade que pode durar vários meses ou mesmo anos, é necessário que o trabalho seja dividido em partes ou em mini-projetos. Cada mini-projeto é uma iteração a qual resulta em um incremento. Iteração se refere aos passos em uma disciplina, e incrementos ao crescimento do produto [MAN 2003]. Em cada uma das iterações os desenvolvedores identificam e especificam os casos de uso relevantes, criam um projeto usando a arquitetura escolhida como um guia, implementam o projeto em componentes, e testam o mini-projeto (iteração).

O Processo Unificado da Rational é descrito através de melhores práticas, de um ciclo de vida, de um meta-modelo e de disciplinas. As melhores práticas descrevem como efetivamente implantar abordagens comprovadas para equipes de

desenvolvimento de *software* e são: *a. desenvolvimento iterativo e incremental, b. gerência de requisitos, c. uso de arquitetura baseada em componentes, d. modelagem visual, e. verificação contínua da qualidade, f. gerência de alterações*.

O RUP pode ser visto através de duas perspectivas diferentes, porém integradas [MAN 2003]. A perspectiva gerencial aborda os aspectos financeiros, estratégicos, comerciais, e humanos, enquanto que a perspectiva técnica aborda aspectos de qualidade, engenharia e projeto. Da perspectiva gerencial, o ciclo do *software* é decomposto em quatro fases – concepção, elaboração, construção e transição, cada uma concluída por um marco. Da perspectiva técnica, o ciclo de vida de *sofware* consiste em várias disciplinas de desenvolvimento, as quais são aplicadas durante cada passo de desenvolvimento, ou iteração. Ou seja, o ciclo de vida do *software* é uma sucessão de iterações, e o *software* é desenvolvido incrementalmente. Cada iteração finaliza com uma versão (*release*) de um produto executável, o qual é acompanhado de artefatos de suporte, como a descrição da versão e documentação para o usuário. Uma iteração é um ciclo de vida completo, o qual resulta em uma subversão, um subconjunto do produto final, que cresce incrementalmente de versão para versão para se transformar no produto final.

RUP é baseado em componentes e utiliza a Linguagem de Modelagem Unificada (UML), integrando ciclos, fases, disciplinas, suavização de riscos, controle de qualidade, gerenciamento de projetos e gerência de configuração. O RUP pode ser representado visualmente através de elementos de um meta-modelo, o qual inclui componentes como o trabalhador (*worker*), atividades (*activities*), guias de trabalho (*work guidelines*), conceitos (*concepts*), guias de ferramentas (*tools mentor*) e artefatos (*artifacts*). Atividades tem artefatos como entradas e saídas. Um artefato é um produto de trabalho do processo. Trabalhadores utilizam artefatos para executar suas atividades, e produzem artefatos durante a execução de suas atividades.

Uma mera enumeração dos artefatos, trabalhadores e atividades não constituem um processo, no entanto. É necessário se descrever a seqüência das atividades que produz algum resultado e mostrar a interação entre os trabalhadores. Uma disciplina é uma seqüência de atividades que produz um resultado de um valor considerável. No RUP existem nove disciplinas, que são divididas em seis disciplinas de processos e três de suporte. As disciplinas de processos são: Modelagem de Negócios, Requisitos, Análise e Projeto (desenho), Implementação, Teste e Implantação. As disciplinas de suporte são: Gerenciamento da Configuração e de Alterações, Gerenciamento de Projetos, e Ambiente. As disciplinas são expressas em termos de diagramas de atividades. Cada passo em um disciplina é descrito com mais detalhes, nas unidades da disciplina (*workflow detail*). Estas unidades são grupos de atividades as quais são executadas juntas, e estão relacionadas a artefatos de entrada e de saída. Em UML, um processo de trabalho (*workflow*) pode ser representado por meio de um diagrama de seqüência, um diagrama de colaboração ou um diagrama de atividades. No RUP optou-se por descrever as disciplinas através de diagramas de atividades [JAC 2000].

3.2.2 Ambiente, Notação e Processo Orientados a objeto (OPEN)

Object-oriented Process, Environment and Notation é um processo de domínio público os qual consiste em vários elementos [MAN 2003]: *framework de desenvolvimento, linguagem de modelagem (OML – OPEN Modeling Language), procedimentos, padrões e guias, papéis de desenvolvimento recomendados, e métricas*.

Open é essencialmente um *framework* para métodos de desenvolvimento de *software* orientados a objetos, de terceira geração. Provê suporte para a modelagem de processos e elicitação de requisitos e oferece a habilidade para modelar agentes inteligentes. Inclui um *framework* para o gerenciamento de projetos e reuso. Suporta a modelagem de processos de negócio, ofrece guias, e suporta questões relacionadas às pessoas [MAN 2003]. Open se preocupa primariamente com a qualidade de *software* e com o uso de métricas.

Open é considerado um método orientado a objetos de terceira geração, por ter sido originado a partir da elaboração, em 97, da união de três métodos de segunda geração (MOSES, SOMA, e Firesmith). É desenvolvido e mantido pelo consórcio Open, que é uma organização sem fins lucrativos, a qual é composta pelos autores dos três métodos recém descritos, além de outros pesquisadores, metodologistas e pela comunidade do mundo todo.

O processo para um projeto ou uma organização é criado através da instanciação do *framework*, também chamado de meta-modelo. Combina as pessoas dentro de um determinado contexto de cultura organizacional, ferramentas e tecnologias disponíveis. Utiliza unidades de trabalho e produtos de trabalho, que são documentados pelo uso de uma linguagem (natural, modelagem, ou codificação), em um conjunto de estágios que são seqüenciados através do tempo. Uma instância de processo OPEN basicamente consiste de um conjunto em seqüência de atividades. As atividades são associadas aos produtos de trabalho, os quais são criados, pelo menos em parte, pela aplicação de uma ou mais técnicas, dentro deste *framework* de processo.

O processo de instanciação consiste em dois estágios: instanciar as meta-classes, usando componentes já fornecidos em uma biblioteca de processo, e então montar as classes de acordo com a configuração requerida. A escolha dos componentes corretos é influenciada por vários fatores, como nível de habilidade da equipe, o nível CMM da organização, e o nível de criticalidade do projeto.

3.2.3 Suporte a adaptação com RUP e OPEN

Segundo [HEN 99], criar um processo implica em instanciar um meta-modelo de processos. Há diversas opções de decisão que podem ser tomadas durante o processo de instanciação. Uma abordagem seria a de tomar nenhuma decisão e somente documentar os elementos do meta-modelo, descrevendo como que estes se relacionam uns com os outros. Esta seria a forma com que o Open é descrito. Uma outra alternativa seria a de já prescrever um modelo de ciclo de vida em particular, ou prescrever uma ou mais técnicas obrigatórias. Esta última forma seria a utilizada pelo RUP.

Se as decisões não estão previamente tomadas, então estas decisões precisam ser tomadas pelo engenheiro de processos. Isto tem a vantagem de proporcionar alto grau de flexibilidade de forma que uma organização pode escolher exatamente quais os componentes de Open que deseja selecionar. Por outro lado, isto tem a desvantagem do trabalho extra.

Por outro lado, no outro extremo, se o autor da metodologia toma todas as decisões para o usuário e obriga a adotar todos os passos do método, então este é considerado um método empacotado (*out of the box*). Não há a necessidade de este ser adaptado, e há nenhuma ou muito pouca flexibilidade. Se um método empacotado

corresponder exatamente às necessidades da organização, então este é a resposta perfeita.

Quando um método empacotado não atende exatamente as necessidades da organização, então esta tentará adaptá-lo. A possibilidade de isto ocorrer depende do quanto o método permite que decisões específicas da organização ocorram. Algumas decisões são inalteráveis: RUP é dirigidos por casos de uso e Shlaer-Mellor é uma abordagem transitiva. Negar alguma destas restrições leva o usuário a abandonar total ou parcialmente a abordagem do método que escolheu.

No entanto, RUP permite alguns ajustes e adaptações que podem ser realizadas. Portanto podemos considerar RUP um método adaptável mais do que propriamente um *framework* de processos. Não há uma razão técnica pela qual o RUP não possa ser desmontado em uma série de componentes separados, mas esta não é claramente uma intenção desta abordagem.

Diversas publicações foram escritas sobre o Processo Unificado da Rational (RUP) [JAC 2000] e o Processo de Desenvolvimento Unificado [JAC 98]. Estaria claro que o segundo é uma descrição geral, e o RUP seria uma instância totalmente convertida em um produto [HEN 99]. RUP é essencialmente um processo pré-configurado, mas retém um pequeno grau de adaptabilidade. Pode-se modificar certas durações; pode-se modificar ou remover passos de certas atividades; pode-se adicionar logotipos específicos da empresa nos padrões (*templates*); e pode-se adicionar novos marcos para atividades de revisão.

4 METODOLOGIAS ÁGEIS

4.1 Introdução

Nos últimos anos, têm chamado a atenção da comunidade de desenvolvimento de *software* a publicação de várias metodologias auto-denominadas de ágeis (*agile*). Estas metodologias foram classificadas inicialmente como leves (*lightweight*), para diferenciá-las das metodologias e processos tradicionais de desenvolvimento, considerados pesados (*heavyweight*), os quais seriam baseados na produção de uma grande quantidade de documentação e de modelos para guiar a programação.

As metodologias ágeis de desenvolvimento de *software* têm sido abordadas em um grande número de livros técnicos recentemente. Além disso, conferências da indústria de desenvolvimento de *software* têm ocorrido em diversos países, inclusive no Brasil, com grande repercussão. Há diversos relatos em que afirma-se a obtenção de melhores resultados com a utilização de metodologias ágeis do que com os processos tradicionais. No entanto, por terem estas metodologias, em sua maioria, uma publicação recente, é ainda incipiente a pesquisa e a comprovação acadêmica sobre o assunto.

Segundo Laurie Williams e Alistair Cockburn [WIL 2003], em tradução livre: "As metodologias ágeis definitivamente atingiram um nervo da comunidade de desenvolvimento de software. Algumas pessoas argumentam ferozmente a seu favor, enquanto outras argumentam contra os mesmos com a mesma energia, e outras estão trabalhando para misturar as abordagens ágeis e guiadas a planos. Um número maior ainda de pessoas ainda está tentando descobrir o que a agilidade significa".

Nem todas as práticas relacionadas nas metodologias ágeis são novas. As principais práticas relacionadas a ciclos de desenvolvimento iterativos e incrementais, por exemplo, já eram mencionadas no método descrito por Victor Basili em [BAS 75]. O artigo descreve uma metodologia chamada *Iterative Enhancement*, o qual foi aplicada com sucesso para a produção de uma família de compiladores. Alguns princípios valorizados em metodologias ágeis, como a iteração, a facilidade de se fazer mudanças e o redesenho, são apresentados. Em um artigo recente [LAR 2003], Craig Larman e Basili descrevem a história do desenvolvimento iterativo e incremental, na qual este trabalho aparece em destaque.

Mesmo assim, inicialmente com as publicações da metodologia *extreme programming* por Kent Beck [BEC 99], e do manifesto ágil no ano de 2001, seguiram-se uma série de publicações de metodologias, como Scrum, por Ken Schwaber [SCH 2002], bem como alguns estudos relacionados a metodologias ou práticas ágeis, por exemplo [WIL 2002], de Laurie Williams, que serviram para tornar mais pública a existência e aplicabilidade das práticas. Contudo, considera-se que ainda faltam pesquisas relacionadas a estas práticas e metodologias, os quais foram

inventados por consultores da indústria de *software*, e ainda estão por ser assimilados pela academia científica.

Este capítulo do trabalho tem como objetivo fazer um estudo sobre as principais idéias presentes no manifesto ágil, bem como estudar três metodologias ágeis que têm sido destacados, chamados *Scrum, Feature-Driven Development*, e *Extreme Programming (XP)*. Ao final do capítulo, serão realizadas comparações entre as metodologias estudadas. Outros trabalhos foram realizados nesta universidade, com o objetivo de comparar as metodologias ágeis com as metodologias tradicionais, classificadas como pesadas (*heavyweight*), ou também como dirigidas a planos [MAN 2003a], o que não será aprofundado.

Nas próximas seções são descritas resumidamente algumas idéias presentes nas metodologias ágeis, particularmente *Scrum*, *Feature-Driven Development (FDD) e Extreme Programming (XP)*. Um estudo mais detalhado, que inclui uma análise comparativa, pode ser encontrado em [HAR 2003].

4.2 Desenvolvendo software de forma ágil

Em 2001 um grupo de desenvolvedores de *software* experientes e consultores em métodos de desenvolvimento de *software* "leves" se reuniu e compôs um manifesto. Neste manifesto o grupo se auto-denominou a aliança ágil (*agile alliance*), a qual valoriza um conjunto de valores e práticas comuns "ágeis" para o desenvolvimento de *software*. Estes valores diferem dos processos até então tidos como tradicionais para o desenvolvimento de *software*. Os principais termos do manifesto para o desenvolvimento de *software* ágil são os seguintes [BEC 2001], em tradução livre:

Estamos descobrindo melhores formas para o desenvolvimento de software fazendo isto e ajudando outros a fazerem. Através deste trabalho nós viemos a valorizar:

~ **Indivíduos e interações** sobre processos e ferramentas ~

~ **Software funcionando** sobre documentação completa ~

~ **Colaboração com o cliente** sobre negociação do contrato ~

~ **Resposta à mudança** sobre seguir um plano ~

Isto é, embora haja valores nos itens á direita, nós valorizamos mais os itens à esquerda.

Kent Beck Mike Beedle Arie van Bennekum Alistair Cockburn
Ward Cunningham Martin Fowler James Grenning Jim Highsmith
Andrew Hunt Ron Jeffries Jon Kern Brian Marick
Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas

Nós seguimos estes princípios:

Nossa maior prioridade é satisfazer o cliente através da entrega freqüente e o mais cedo possível de *software* com valor agregado.

Alterações sobre os requisitos são bem vindas, mesmo que tarde no desenvolvimento. Processos ágeis suportam a mudança, para a vantagem competitiva do cliente.

Entrega de *software* com freqüência, de algumas semanas a alguns poucos meses, com preferência para a escala de tempo mais curta.

Os especialistas no negócio e os desenvolvedores devem trabalhar juntos diariamente durante o projeto.

Monte projetos ao redor de indivíduos motivados. Dê a eles o ambiente e o suporte que eles precisam, e confie neles para que façam o serviço.

A maneira mais eficiente e eficaz de trocar-se informação com e através de um time de desenvolvimento é a comunicação face-a-face.

Software funcionando é a medida primária de progresso.

Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem poder manter um ritmo freqüente indefinidamente.

A atenção contínua a excelência técnica e ao bom projeto melhoram a agilidade.

Simplicidade, a arte de maximizar a quantidade de trabalho não realizado, é essencial.

As melhores arquiteturas, requisitos, e projetos emergem de times auto-organizados.

O time reflete, a intervalos regulares, sobre como se tornar mais efetivo, então se ajusta e otimiza o seu comportamento de acordo.

O termo “ágil” foi utilizado para identificar e publicar para o público em geral de forma comum um novo tipo de métodos que até então era pouco difundido, estando seu conhecimento restrito principalmente a consultores da indústria de *software*. Até então o termo “leve” (*light*) era utilizado para designar alguns dos métodos. No entanto, segundo Alistair Cockburn, citado em [FOW 2001], “Eu não me importo que minha metodologia seja chamada leve em seu peso, mas não estou certo de eu queira ser chamado de um ‘peso-leve’ comparecendo a uma reunião de metodologistas ‘peso-leves’. Parece uma porção de pessoas magrelas e de baixo QI tentando lembrar que dia é hoje”.

Segundo [FOW 2001], o manifesto tem uma porção de aspectos bastante interessantes. Um grupo de desenvolvedores de *software* experientes que está descobrindo novas e melhores formas para desenvolver *software*. Isto mostraria que as pessoas que assinaram o manifesto ainda não sabem tudo e estão sempre prontas a aprenderem novas lições. Além disso, estas pessoas estão aprendendo como fazer melhor *software* através da prática e da experimentação.

Os princípios reconhecidos pelo manifesto, ainda de acordo com [FOW 2001], mostram que certos valores e princípios continuam sendo importantes, ainda que outros valores e princípios sejam mais valorizados. A modelagem é importante, mas não somente para preencher alguma lacuna em um repositório não utilizado. A documentação é importante, mas não somente para gastar montanhas de papéis que raramente são lidos. O planejamento é utilizado, mas com o reconhecimento que há limitações sobre um plano, principalmente em ambientes conturbados. O movimento ágil não seria contra metodologias, mas sim restauraria certos valores de forma a buscar-se a credibilidade nos métodos de desenvolvimento.

Uma das principais diferenças dos métodos ágeis em relação aos métodos tradicionais, segundo [FOW 2003], seria a questão previsibilidade *versus* adaptabilidade. A inspiração dos métodos tradicionais busca reproduzir no desenvolvimento de *software* métodos aplicados na engenharia civil e mecânica. Estes métodos, quando aplicados nas áreas de engenharia, colocariam sua ênfase no planejamento prévio do projeto, focando em desenvolver-se um projeto o qual consiste de uma série de modelos e diagramas os quais descrevem o que deve ser construído. Na indústria de construção civil ou mecânica, este projeto é normalmente entregue a um grupo diferente, até mesmo outra empresa, a qual se encarregará da construção. Provavelmente os construtores irão passar por algumas dificuldades e dúvidas em relação ao projeto, mas que normalmente serão suficientemente pequenas para não atrapalharem. Esta abordagem permite um planejamento detalhado de como a construção será feita, permitindo uma alta previsibilidade sobre variáveis como prazo e custo.

Haveria, portanto, nos métodos tradicionais, uma clara separação entre as atividades de projeto e de construção. A atividade de projeto seria difícil de prever, mas, uma vez concluída, levaria a uma fase de construção que poderia ser executada por indivíduos com baixa qualificação intelectual, com relativa previsibilidade.

Contudo, o autor [FOW 2003] questiona a aplicação de um princípio semelhante na engenharia de *software*, pois, segundo ele, embora a previsibilidade não seja impossível em um projeto de *software*, esta é muito difícil de ser alcançada. Um dos maiores perigos de se buscar a previsibilidade seria tentar aplicá-la em situações em que esta não pode ser obtida. Isto freqüentemente ocorre em projetos de software, onde a variação constante dos requisitos, tanto em termos de funcionalidade quanto em termos de tecnologia, é muito rápida. Segundo [FOW 2003]: “Se você está em uma situação que é imprevisível, então não adianta utilizar uma metodologia que espera previsibilidade. Isto é uma mudança difícil. Isto significa que muitos dos modelos que são hoje utilizados para controlar projetos, e muitos dos modelos para a relação inteira com o cliente, simplesmente não são mais verdadeiros. Os benefícios da previsibilidade são tão bons, que é difícil abandoná-los. Como muitos problemas, a pior parte é simplesmente admitir que o problema existe”.

No entanto, deixar de esperar previsibilidade não significaria abandonar um projeto ao caos, mas sim utilizar mecanismos adaptáveis de controle. Tais mecanismos seriam baseados em ciclos de respostas, ou retroalimentação (*feedback*). A palavra chave para a obtenção de mecanismos de retroalimentação, onde freqüentemente avalia-se qual a posição em que o projeto se encontra seria o desenvolvimento de software iterativo, baseado em ciclos curtos de desenvolvimento.

Esta mudança, de um modelo baseado em previsibilidade para um modelo adaptável, não seria tão simples, contudo, pois requer mudanças na própria forma com

que as empresas de software se relacionam com seus clientes, e como os gerentes se relacionam com os programadores.

4.3 Scrum

A primeira referência a um método de desenvolvimento de produtos inspirado no jogo de *rugby* é apresentada em [TAK 86], após os autores analisarem como as mais inovadoras companhias da época criavam novos produtos. Segundo os autores, esta abordagem se caracterizaria pela sobreposição das fases de desenvolvimento, onde times multidisciplinares trabalham juntos do início ao fim do projeto. Suas principais características seriam: *1. Estabilidade obtida de dentro para fora do time. 2. Times de projeto auto-organizáveis. 3. Fases de desenvolvimento sobrepostas. 4. "Multiaprendizado". 5. Controle sutil do projeto. 6. Transferência do conhecimento dentro da organização.*

O termo *Scrum* é uma metáfora para uma situação em um jogo de Rugby, onde um denso círculo de pessoas é formado pelos times quando lutando pela posse da bola [SCH 2002]. O jogo de rugby se caracteriza pela corrida em direção a uma meta, com os jogadores de um mesmo time constantemente passando a bola entre si.

Na primeira metade dos anos 90, o método *Scrum* começou a ser aplicado no desenvolvimento de produtos de software, mais especificamente para gerenciar projetos de desenvolvimento de software, por Ken Schwaber e Jeff Sutherland [SCH 2002]. A história do método *Scrum* está ligada a história de um de seus autores, Ken Schwaber, que em 90 era fabricante de uma metodologia e de uma ferramenta CASE chamada MATE. As requisições para modificações em sua ferramenta eram muitas, e em um determinado momento ele se encontrava em desespero: se continuasse no ritmo de desenvolvimento de novas funcionalidades em que se encontrava, não haveria como atender as demandas de seus clientes.

Alguns anos mais tarde, quando Ken foi questionado por Jeff Sutherland sobre qual era a metodologia que utilizava para desenvolver tão rapidamente sua ferramenta CASE, este o respondeu de uma forma totalmente inesperada. Este respondeu que utilizava um método empírico, o qual havia sido inspirado nos métodos utilizados por companhias de software de sucesso como a Borland. Não mais utilizava o processo que ele mesmo pregava e vendia. *Scrum* foi introduzido mais tarde por Sutherland em diversas empresas [SUT 2001].

O processo *Scrum* evoluiu a partir de sua publicação inicial [SCH 95], tendo sido posteriormente publicado na forma de um conjunto de padrões organizacionais para a produção de *software* de forma hiper-produtiva [BEE 99], até finalmente ser publicado na forma de livro [SCH 2002], em seu formato atual. A definição de *Scrum* como um conjunto de padrões para o desenvolvimento de *software* utilizando pequenos times é reforçada em [RIS 2000], onde o resultado de experiências práticas com o método é examinado. Segundo [ALE 77], "um padrão descreve um problema que ocorre muitas e muitas vezes em nosso ambiente e então descreve a parte principal de uma solução para tal problema de tal forma que você possa utilizar esta solução um milhão de vezes sem nunca utilizar a solução da mesma forma por duas vezes."

Scrum não é um método de software completo, pois não define práticas para as atividades clássicas de um ciclo de vida de desenvolvimento de software, tais como análise, projeto, construção e testes. Segundo [ABR 2002], "*Scrum* não requer que seja utilizada nenhuma prática ou técnica de desenvolvimento de software. Ao

contrário, requer certas práticas de gerência e ferramentas nas várias fases do *Scrum* para escapar do caos causado pela incerteza e pela complexidade". É um método para o gerenciamento de um projeto de software, com o objetivo de controlar seu caos e produzir resultados de forma incremental e contínua.

4.4 Feature Driven Development (FDD)

O processo chamado de *Feature-Driven Development* (desenvolvimento guiado por características), ou simplesmente FDD, foi concebido na implementação de um projeto de grande porte em Singapura [PAL 2002], em que Peter Coad atuou no papel de Arquiteto, e Jeff De Luca atuou como Gerente de Projeto.

Segundo [COA 99], FDD é um processo que agrada aos desenvolvedores, aos gerentes e aos clientes. Os desenvolvedores gostam porque estão constantemente recebendo novas atividades, no mínimo a cada duas semanas, e também porque tem a sensação de fechamento das atividades, também no mínimo a cada duas semanas. Duas semanas é o tempo máximo do ciclo de implementação de uma *feature*, ou característica.

FDD provê aos gerentes uma forma acurada de medir e reportar o progresso. O risco do projeto é reduzido, pois este produz resultados de forma constante e de forma tangível, através da entrega de produtos funcionando. Já os clientes também gostam do método, pois através do mesmo eles conseguem ver planos cujas etapas intermediárias conseguem entender. Eles conseguem ver resultados intermediários e conseguem saber em que ponto o projeto está a qualquer momento.

FDD é baseado em modelos, e guiado por características – *features*. Inicia com um modelo global, que é implementado em uma série de iterações, ou ciclos de desenvolvimento, bastante curtos. Cada característica é uma funcionalidade que contém valor aos olhos do cliente.

De forma geral, FDD não é um método completo, pois não descreve todas as atividades necessárias para construir o software [PAL 2002]. O método concentra-se nas cinco principais etapas do processo, de construir um modelo inicial, identificar, planejar e classificar as características, e de iterativamente fazer o projeto e a construção de cada característica. Provê mecanismos para acompanhar e reportar o progresso do projeto. Além disso, o método inclui quais os papéis, os artefatos e os objetivos seriam necessários a um projeto.

4.5 Extreme Programming (XP)

> *"XP é o mais importante movimento em nosso campo hoje em dia. Eu estou prevendo que ele será tão essencial para a geração atual quanto o SEI e o Capability Maturity Model (CMM) foram para a passada".*
>
> Tom DeMarco, no prefácio de [BEC 2000].

Segundo [BEC 2000], *extreme programming*, ou XP, mais do que um método, é um conjunto de práticas, não necessariamente novas, mas que quando aplicadas em conjunto se complementam de tal forma a auxiliar uma pequena equipe de programadores a entregarem *software* de forma rápida e com qualidade em um ambiente turbulento.

As práticas de XP individualmente não são novas [BEC 99]. A separação clara entre as decisões de negócio e as decisões técnicas, a evolução rápida de um plano em

respostas às mudanças, a especificação do projeto em relação às necessidades dos usuários, o estabelecimento de metáforas, bem como outras das práticas do XP, foram inspiradas em outros métodos e descobertas. Por exemplo, a utilização de estórias do usuário (*user stories*) como forma de especificação foi inspirada nos casos de uso (*Use Cases*) do método *Objectory* [JAC 94]. A rapidez de reação às mudanças seria inspirada nas primeiras publicações relativas ao método *Scrum* [TAK 86], anteriormente descrito.

O método XP consiste em doze práticas que devem idealmente ser adotadas em conjunto. O método é particularmente voltado a ambientes de desenvolvimento de *software* onde um grupo pequeno ou médio de desenvolvedores deve criar suas aplicações a partir de requisitos vagos ou em constante mudança. O principal objetivo desta metodologia é o de abraçar as mudanças, desenvolvendo *software* de forma freqüente e o mais cedo possível, e absorvendo a retroalimentação (*feedback*) na forma de mudanças em relação aos requisitos do *software*, em relação ao ambiente de desenvolvimento, e finalmente em relação ao código [WIL 2000]. O nome do método, que utiliza o termo *extreme* (extremo), mostraria que a proposta do método é o de utilizar conceitos já utilizados e provados como efetivos, e levá-los ao extremo.

A figura 4.1 faz uma comparação do método XP, colocando-o como uma evolução do modelo de desenvolvimento iterativo. Segundo [BEC 99], no início utilizava-se o modelo em cascata (*waterfall*), no qual o desenvolvimento de um projeto de *software* é dividido em etapas seqüenciais. Este modelo de desenvolvimento assume que os usuários conseguem nos dizer tudo aquilo que eles precisam em um *software* no início do projeto. No entanto, este modelo seria falho. Os usuários e clientes não conseguem dizer tudo o que querem no início do projeto. Eles não sabem, eles descobrem ao longo do projeto, e às vezes inclusive se contradizem.

Uma evolução do modelo de desenvolvimento em cascata foi o modelo iterativo. Neste caso, já que um ciclo de desenvolvimento muito longo traz problemas, opta-se por dividir-se um projeto em alguns ciclos menores, ou iterações. Em cada iteração seguem-se as fases de desenvolvimento de forma seqüencial.

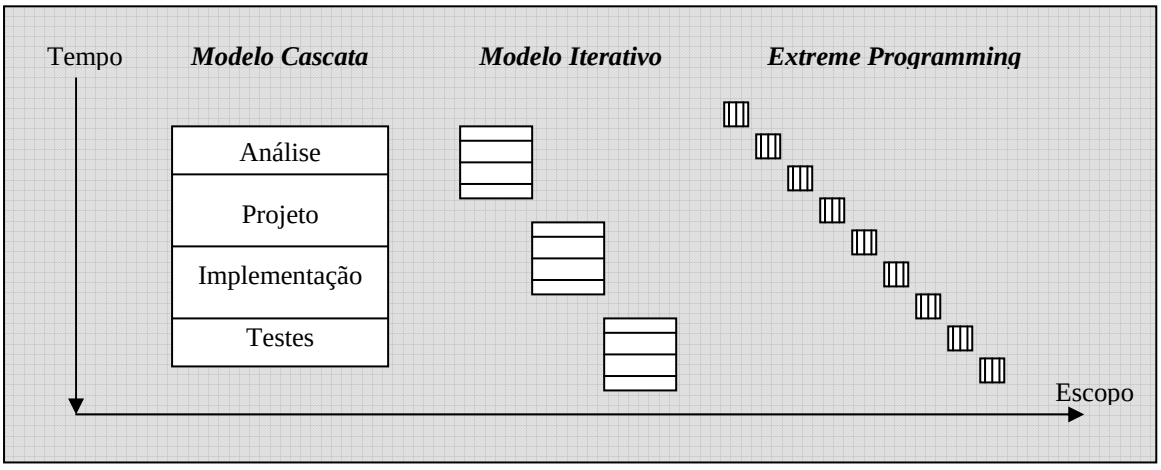


Figura 4.1: XP comparado a desenvolvimento em cascata e a desenvolvimento iterativo, retirada de [BEC 99]

Extreme Programming, conforme se verifica na figura 4.1, leva o conceito de desenvolvimento iterativo ao extremo. O *software* é construído através de múltiplos ciclos bastante curtos de desenvolvimento, durante os quais as fases de desenvolvimento são sobrepostas, e não seqüenciais como no modelo iterativo de desenvolvimento.

O autor [BEC 99] observa que o modelo de desenvolvimento em cascata não surgiu por acaso. Foi, isto sim, uma reação ao custo de realizar-se mudanças tardias no *software*, baseada em estudos de que o custo de uma mudança aumenta ao longo do tempo no projeto; isto justificaria um modelo onde procura-se tomar todas as decisões importantes no início do projeto. Contudo, muitas mudanças tecnológicas teriam ocorrido desde que o modelo foi concebido, tais como a criação de bancos de dados relacionais e de linguagens de programação orientadas a objeto, e que o custo de realizar-se mudanças ao longo do projeto já não seria mais um problema.

Segundo o autor [BEC 99], XP não é uma idéia totalmente terminada, e os limites de sua aplicação ainda não estão bem definidos. Ele sugere que inicialmente o método deva ser aplicado nos domínios onde é claramente aplicável: desenvolvimento de sistemas pequenos e médios, onde os requisitos são vagos e estão em constante mudança. As práticas do método não precisam ser adotadas todas de uma vez, mas sim incrementalmente adotadas para resolver os problemas que existirem em cada ambiente de desenvolvimento de software.

5 GERENCIANDO RISCOS EM PROJETOS DE SOFTWARE

Neste capítulo será feito um estudo sobre o gerenciamento de riscos em projetos de desenvolvimento de *software*.

5.1 Dançando Valsa com Ursos

Esta seção busca introduzir os conceitos de gerenciamento de riscos em projetos de *software*, e é baseada no livro de Tom DeMarco e Timothy Lister intitulado (tradução livre) "*Dançando com Ursos: Gerenciando Riscos em Projetos de Software"* [DEM 2003].

Segundo os autores, fugir dos riscos não é uma estratégia vencedora. Projetos que não tem riscos reais são perdedores, oferecem quase nenhum benefício. Portanto: "se um projeto não tem riscos, não o realize". Os riscos e os benefícios andam lado a lado. Se um projeto tem riscos, é porque ele leva a águas ainda não exploradas. Utiliza suas capacidades ao extremo, e se você tiver sucesso, desenvolverá vantagem competitiva sobre seus concorrentes.

No mundo atual, estamos no meio de mudanças que vão provavelmente nos afetar pelo resto de nossas vidas. O mundo está cada vez mais conectado. Indivíduos estão mais conectados a suas empresas e a seus fornecedores. As empresas estão mais conectadas a seus clientes, seus empregados, seu mercado, e as agências do governo que afetam seu trabalho. E tudo isto está evoluindo. Neste período, a habilidade de correr riscos é essencial. Importa mais do que a eficiência.

Embora correr riscos seja algo essencial no mundo atual, os autores [DEM 2003] alertam que a atitude de aceitar riscos muitas vezes leva a um comportamento não desejado. Empresas tendem a incentivar o pensamento positivo ao ignorar as conseqüências indesejadas dos riscos que estão correndo. Esta é uma variação extrema de uma atitude "eu posso fazer". Correr riscos envolve um pouco de uma atitude "eu posso fazer". No entanto, eles de forma cega se recusam a enxergar o lado negativo de sua atitude. Se há coisas que podem ocorrer de errado, eles preferem simplesmente ignorá-las totalmente.

Ninguém é tão estúpido a ponto de ignorar totalmente os riscos. De forma geral, as pessoas fazem isto de forma seletiva: gerenciam e equacionam os riscos mais simples, e escolhem ignorar os riscos mais graves do projeto. Esta não seria uma boa fórmula para o gerenciamento de riscos. Se você decide correr em direção aos riscos ao invés de para longe dos mesmos, é melhor manter os olhos firmes no que vêm pela frente.

5.1.1 Definições básicas

Os autores começam definindo o gerenciamento de riscos em projetos de *software* como gerenciamento de projetos para adultos. Quando somos crianças, somos protegidos das ameaças do mundo por nossos pais, e por isso podemos viver na ignorância dos problemas do mundo. Ameaças como guerra nuclear, poluição do meio ambiente, violência urbana, e injustiça não são problemas que normalmente aterrorizam uma criança pequena. Quando crescemos, no entanto, somos forçados a enfrentar a dura realidade. É isto que significa ser um adulto.

Da mesma forma, quando os gerentes de projeto não estão gerenciando os riscos, eles estão agindo como crianças. Toda a indústria de desenvolvimento de *software* tem sido criançola neste sentido. Há uma prevalência da atitude "eu posso fazer", em que se planeja sempre pensando somente no melhor resultado possível para o projeto, e se ignora as diversas ameaças que podem fazer com que tal resultado seja impossível de ser atingido.

Os autores definem risco primeiramente como sendo: *a. um possível evento futuro o qual leva a um resultado não desejado. b. o resultado insatisfatório propriamente dito.* A primeira definição é a causa, enquanto que a segunda é o efeito. Ambos são importantes, mas normalmente podemos gerenciar somente as causas.

Uma alternativa de definição é que um risco é algo que ainda não ocorreu, enquanto que um problema é um risco que já se materializou. Antes de acontecer, um risco é somente uma abstração. É algo que pode afetar o projeto, mas também pode não vir a afetar. Há a possibilidade de que se você ignorar o risco, ele não o atingirá. Mesmo assim, você não é inocente se você ignorar o risco, somente sortudo; o risco mal gerenciado somente não foi "descoberto". O gerenciamento de riscos envolve tomar ações corretivas antes que um problema venha a ocorrer. Seu oposto é o gerenciamento de crises: neste caso, se procura descobrir uma forma de solucionar um problema quando o mesmo já ocorreu.

Há um momento de transição em relação a um risco: ele costumava ser um risco, mas de repente se transforma em um problema. Este é o ponto em que se diz que o risco se *materializou*. Este é o momento da *transição do risco*. A transição é um evento importante para o gerente de projeto. É o gatilho que dispara o plano que foi feito para lidar com o risco. A transição em si pode ser invisível, portanto o que se observa é o *indicador de transição*. Os indicadores de transição precisam ser monitorados pelo gerente de projeto.

A razão pela qual o gerente de projeto precisa se preocupar com a transição do risco é que quando um indicador é disparado, é necessária alguma ação. Antes de o evento ocorrer, pode ser muito cedo para tomar a ação – pode ser caro e consumir tempo – portanto normalmente se torce para que isto não seja necessário. Algumas ações podem não ser possível de serem postergadas, no entanto. Há ações que precisam ser tomadas antes da transição, de forma a possibilitar que quando a transição ocorra haja uma opção corretiva menos custosa. O trabalho que deve ser realizado de forma preventiva antes que o risco ocorra é chamado de *suavização* do risco.

A suavização de riscos custa tempo e dinheiro. Em um cenário cor-de-rosa em que nenhum dos riscos se materializa, este gasto de tempo e dinheiro pode parecer ter sido desnecessário. Este tipo de pensamento muitas vezes torna o gerenciamento de riscos impossível, no entanto.

As principais atividades que estariam envolvidas no gerenciamento de riscos são as seguintes:

- **Descoberta dos riscos**: uma sessão inicial de pensamento (*brainstorm*) e triagem subseqüente.
- **Análise de exposição**: quantificação de cada risco em termos de sua probabilidade de materialização e de seu impacto em potencial.
- **Planejamento de contingência**: o que você espera fazer se e quando o risco se materializar.
- **Suavização**: passos que precisam ser tomados antes que a transição do risco ocorra, de forma que as ações de contingência preparadas possam ser efetivas quando requeridas.
- **Monitoramento constante da transição**: verificação dos riscos sendo gerenciados, procurando-se identificar sua materialização.

Um exemplo de um projeto que falhou por falta de gerenciamento de riscos é citado pelos autores [DEM 2003], como o "infame" sistema automatizado de transporte da bagagem do Aeroporto Internacional de Denver. Em 88 a cidade de Denver, no Colorado, EUA, fez um plano para substituir seu aeroporto que não suportaria mais futuras expansões. O novo aeroporto, DIA - Aeroporto Internacional de Denver, seria mais moderno, a poluição seria reduzida, e o tráfego aéreo apresentaria menores atrasos. O plano para sua construção previa sua entrada em operação no dia 31 de outubro de 93. Quando esta data foi atingida, tudo estava bem, com exceção da "porcaria" do *software* que ainda não estava pronto.

Particularmente, o que não estava pronto no projeto do DIA era o sistema automatizado para controle das bagagens. Sem este *software*, o aeroporto não poderia abrir. Como a construção do aeroporto incluía um grande gasto de capital, este investimento não poderia ser recuperado enquanto o pessoal de *software* não conseguisse atingir seu objetivo. O aeroporto somente entrou parcialmente em operação dois anos após o planejado, o que causou grandes prejuízos financeiros para a cidade de Denver. O time de desenvolvimento de *software* foi considerado o culpado. Na época, o fracasso deste projeto foi amplamente divulgado em jornais e revistas. Um artigo na revista *Scientific American* colocou a responsabilidade na indústria de *software* e na sua falta de padrões e processos.

Os autores questionam a avaliação de que o problema que causou a falha do projeto de construção do aeroporto foi a falta de processos de desenvolvimento. Digamos que houvesse um processo perfeito para entregar-se *software*. Isto removeria a incerteza destes projetos? Na verdade, na opinião dos autores, a resposta é não. Há incertezas em projetos de *software* em relação a diversos fatores, como requisitos, ambiente em mudanças, recursos humanos, política, inovação tecnológica, escala, etc. Mesmo o mais perfeito processo de desenvolvimento não removerá a incerteza de um projeto de *software*. Onde há incerteza, há risco. Onde há risco, há a necessidade de que este seja gerenciado.

Para concluir o argumento de que faltou no projeto do DIA, na realidade, o gerenciamento de riscos, e não os processos de desenvolvimento de *software*, os autores fazem alguns questionamentos. A questão é que o *software* de controle de bagagens foi colocado no caminho crítico do projeto. No entanto, havia um risco

bastante grande de que este *software* não ficaria pronto a tempo, risco que foi identificado pela empresa contratada para a construção do mesmo, mas ignorada pela contratante (a cidade de Denver). Assim, não foram tomadas medidas preventivas para tirar-se o *software* do caminho crítico. Os túneis para passagem da bagagem foram construídos de forma muito estreita, de tal forma que a única forma de a bagagem ser transportada era através do sistema automatizado. Se o risco de que o *software* poderia atrasar houvesse sido gerenciado, os túneis poderiam ter sido construídos um pouco mais largos, de tal forma que os meios convencionais de manipulação de bagagem poderiam ser utilizados como uma estratégia de contingência. Este meio convencional é a utilização de pequenos carros de transporte, como é feito em boa parte dos aeroportos. O custo de tomar esta medida preventiva teria sido muito pequeno em comparação com o prejuízo que a materialização do risco causou.

5.1.2 Porque realizar gerência de riscos

Há vários motivos para que a gerência de riscos seja um aspecto necessário e muito importante na gerência de um projeto de *software*. Nesta seção examinaremos alguns destes motivos, a partir do trabalho de DeMarco e Lister [DEM 2003].

Um primeiro motivo é que a gerência de riscos torna possível se correr riscos de forma agressiva. Uma razão para o fato de a gerência de riscos ser difícil de ser realizada no típico ambiente corporativo é que encoraja a se enfrentar as incertezas. Com gerência de riscos, você pode encontrar-se dizendo a seu cliente que, de acordo com sua análise de riscos, há uma janela de incerteza em relação à data de entrega do projeto. Na verdade há diversas possibilidades de datas em que o projeto pode ser finalizado, o que depende de uma série de fatores incerteza. Sem a gerência de risco, você provavelmente teria simplesmente citado uma data e cruzado os dedos.

Há gerentes de projeto que dizem que se explicassem os riscos envolvidos em um projeto para seus clientes, estes nunca iriam realizar nenhum projeto. Estes gerentes se vêem prestando um serviço positivo a seus clientes ao esconder destes a feiúra do projeto. Após o cliente aceita o projeto, este pode ser lentamente introduzido às notícias ruins uma a uma. O problema é que os clientes têm memória, e lembram de outros projetos que iniciaram com cenários cor-de-rosa e depois afundaram. O resultado é que eles sempre esperam o pior e ficam com aversão a riscos.

A abordagem mais saudável para a gerência de riscos é abordar o cliente com uma lista de incertezas que podem ocorrer no projeto, e que podem afetar a data de entrega. Ao mesmo tempo, apresenta-se um plano de como se irá agir para minimizar os vários riscos, e como se medir para identificar se um risco ocorrer. Desta forma, o cliente sabe que está correndo um risco, e sabe quanto risco está correndo. A sua vontade de comprometer-se com um projeto arriscado está relacionada à conclusão de que os riscos foram identificados, quantificados e confrontados.

Outra questão é que a atitude de “eu posso fazer” está bastante difundida na indústria de *software*. O resultado desta atitude é ignorar qualquer tipo de análise que possa sugerir “não posso fazer”. Sem a infra-estrutura de gerenciamento de risco, anunciar um risco pode colocar a pessoa em uma posição desconfortável. Ela pode ser considerada desmotivada, pessimista, ou descomprometida com o projeto.

Com o gerenciamento de riscos, realiza-se um esforço controlado de pensamentos pessimistas. Quando se coloca a estrutura de gerenciamento de riscos no lugar, se autorizam as pessoas a pensarem de forma negativa, pelos menos uma parte

do tempo. As empresas que fazem isto entendem que o pensamento negativo é a única forma de se evitar a cegueira em relação aos riscos conforme o projeto progride.

Quando um projeto não declara haver incertezas, qualquer resultado que não seja o melhor resultado possível pode ser considerado uma falha. Sem o gerenciamento de riscos, os projetos não conseguem distinguir entre suas metas e as expectativas efetivamente realizáveis. Desta forma, acaba-se adotando as metas mais apertadas como sendo os prazos do projeto, e assim as chances de que o projeto atrase em relação ao plano inicial ficam sendo bastante grandes.

Outro ponto em que a gerência de riscos ajuda é ao colocar limites para as incertezas. Quando não há limites para as incertezas, estas se tornam ainda mais assustadoras, e isto leva as pessoas a dois tipos de comportamentos que são desastrosos: aversão a riscos ou ignorância total dos riscos.

Quando se conhece a incerteza, sabe-se o quanto que é necessário se reservar de forma a se proteger do risco. A reserva é o que você gasta em suavização do risco, mais o que você guarda para apagar o fogo quando ele surge. Esta reserva significa dinheiro e tempo que você pode não precisar. É necessário ter coragem para colocar esta reserva em seu plano de projeto. Mas não ter uma reserva pode se mostrar muito pior, ou seja, pode custar muito mais caro no caso de o risco se materializar.

Além de apresentarem motivos pelos quais é necessário se fazer a gerência de riscos, os autores [DEM 2003] analisam alguns motivos que são freqüentemente apontados para se evitar a gerência de riscos.

Um primeiro motivo é que os clientes e envolvidos com o projeto (*stakeholders*) no projeto não seriam maduros o suficiente para saber dos riscos envolvidos. A mentira, neste caso, seria um serviço público. Esta atitude estaria relacionada ao fato de que no início da indústria de *software* os sistemas automatizados envolviam apenas as funções mais rotineiras das empresas. Era comum um analista de sistemas ganhar muito mais do que seus usuários, de tal forma que este adotava uma atitude paternalista do tipo “eu sei mais”. No entanto, hoje e dia não somente os projetos de TI assumem riscos. Muitos projetos de TI estão enquadrados dentro de projetos maiores, que também tem os seus próprios riscos fora do escopo de TI. Assim, os clientes conhecem o risco, e não é uma boa idéia tentar mentir sobre ele.

Um outro pretenso motivo para evitar a gerência de riscos é que as janelas de incerteza seriam muito grandes. Muitos gerentes de projeto gostariam que estas janelas de incerteza em relação a datas de entrega fossem pequenas, por isso criam ilusões de controle com estimativas de alto grau de precisão, que logo se mostram muito imprecisas. A questão é que as incertezas são grandes, e mesmo que nós não gostemos, elas estarão lá. Há gerentes ainda que acreditam que explicitar janelas de incerteza grandes é uma desculpa para a má performance. Estes gerentes não entendem a diferença entre a definição de metas internas para a equipe de desenvolvimento e os prazos acordados com os clientes. Os autores aconselham a sempre se definir metas para a equipe de desenvolvimento trabalhar com a melhor performance possível. Ao mesmo tempo, se utiliza uma forma de planejamento completamente diferente ao se fazer promessas aos clientes e à gerência.

5.1.3 Mecanismos do gerenciamento de riscos

Nesta seção serão estudados os principais mecanismos utilizados no gerenciamento de riscos em projetos de *software*.

Para se identificar se um projeto de desenvolvimento de *software* está gerenciando seus riscos, uma forma simples seria perguntar ao gerente de projeto se este pode indicar quais as tarefas contidas em seu plano de projeto as quais podem não ser necessárias. Geralmente a resposta é que não há nenhuma tarefa que pode não ser necessária. Ou seja, na visão destes gerentes, somente as tarefas que serão estritamente necessárias é que estão contidas no plano de projeto [DEM 2003]. Esta visão é contrária ao gerenciamento de risco. Muitas vezes quando um projeto atrasa, não é porque o trabalho que foi planejado levou mais tempo do que o esperado; uma explicação muito mais comum é que o projeto ficou sobrecarregado com atividades que não foram previstas inicialmente.

Para se encontrar quais os riscos que podem vir a afetar um projeto de *software*, a forma proposta pelos autores [DEM 2003] é a de se utilizar os problemas que afetaram os projetos passados da organização. Esta seria uma forma bem mecânica para se introduzir o gerenciamento de riscos em uma organização: fazer uma análise *post-mortem* de alguns projetos bons e ruins, avaliando os problemas que os afetaram de suas expectativas iniciais. A partir dos problemas que foram identificados, procura-se identificar as suas causas, e chama-se cada causa de um risco. O nome desta abordagem seria "*os problemas de ontem são os riscos de amanhã*".

A abordagem de se realizar análises *post-mortem* não seria nova. O que seria novo na proposta dos autores é utilizar o resultado deste tipo de análise como entrada para o processo de gerenciamento de riscos de um novo projeto. Os problemas dos projetos tendem a se repetir com freqüência, portanto estudando-se cerca de meia dúzia de projetos passados já se tem dados suficientes. A identificação de riscos, como os próprios autores afirmam, não deve se limitar aos dados de projetos passados. Nas seções 4.2 e 4.3 deste trabalho, serão estudadas algumas outras formas de identificação dos riscos.

Quando se identifica um risco e este é adicionado à lista de riscos do projeto, há uma pressão grande para que este seja removido. Um risco é uma incomodação, um distúrbio no projeto. Se esta incomodação permanece na lista de riscos durante algum tempo, a gerência sênior tende a ficar impaciente. A gerência sênior se sentiria bem melhor se o risco fosse removido da lista. O quanto mais assustador o risco for, maior a pressão para que este seja eliminado.

Alguns riscos **expiram** durante a execução de um projeto. Ao final do projeto, todos os riscos que não se materializaram são considerados expirados. No entanto, ao invés de esperarmos que os riscos expirem, quais as ações que podemos tomar em relação a um risco? Os autores [DEM 2003] identificaram quatro coisas que se pode fazer em relação a um risco.

- Pode-se **evitar** o risco.
- Pode-se **conter** o risco.
- Pode-se **suavizar** o risco.
- Pode-se **torcer** que o risco não se materialize.

Um risco é **evitado** quando as atividades as quais o risco está relacionado não são executadas. A consequência natural desta abordagem é que ao evitar-se o risco evita-se também o benefício que a realização das atividades poderia oferecer.

Um risco é **contido** quando se separa dinheiro e tempo no projeto suficientes para bancá-lo no caso de ele se materializar. De forma geral, não faz muito sentido se

conter um risco individualmente, mas sim se contém o conjunto inteiro de riscos. Alguns deles irão se materializar, e outros não. Uma estratégia para conter os riscos é a de se separar recursos suficientes, na média, para bancar os riscos que provavelmente se materializarão.

Um risco é **suavizado** quando se realizam ações antes do risco se meterializar, de forma a reduzir um eventual impacto do mesmo. Muitas vezes a estratégia de suavização dos riscos deve ser combinada com a estratégia utilizada para conter os riscos. A suavização do risco torna possível a sua contenção no caso de este se materializar.

Outras coisa que se pode fazer em relação a um risco é **torcer** para que este não se materialize. Isto significa que não foi feita nenhuma das três coisas descritas acima, e mesmo assim o risco não gerou impacto para o projeto, ou seja, não se materializou. Neste caso, é comum cruzar-se os dedos.

Todos nós torcemos em relação a alguns riscos em algum momento. No entanto, esta não é uma boa estratégia de planejamento. Mesmo uma pequena lista de riscos com doze riscos carrega uma probabilidade muito pequena de que nenhum dos riscos irá se materializar, o que leva a um conceito importante em relação ao risco: a exposição do risco.

A **exposição do risco** é a expectativa dos custos de conter-se o risco. É uma combinação da probabilidade de o risco se materializar com os custos associados no caso de este ocorrer. De forma geral, pode-se afirmar que:

Exposição do risco = custo x probabilidade

Ou seja, um risco que tem uma probabilidade de 20% de ocorrer, e que irá custar 1 milhão de dólares no caso de se materializar, terá um valor de exposição de risco de 200 mil dólares. O custo exato que ocorre para um risco nunca será exatamente igual à exposição do risco. O risco de exemplo pode se materializar ou não. Se ocorrer, irá custar 1 milhão. Se não ocorrer, irá custar nada. Ao se calcular a exposição para todos os riscos de um projeto e separar uma reserva igual a exposição total, na média esta será suficiente para bancar os riscos que se materializarem. Em alguns projetos a reserva pode ser um pouco curta demais, e em outros projetos poderá ser mais do que o suficiente. Na média de vários projetos, será adequada.

Uma reserva para um risco é uma certa quantidade de tempo ou de dinheiro que se separa para conter riscos. Uma estratégia é se alocar uma reserva igual a exposição total dos riscos do projeto. Isto pode resultar em uma reserva não completamente utilizada, ou então pode ser que seja necessário um extra. Uma estratégia mais defensiva seria alocar algo mais do que a exposição total calculada, enquanto que uma estraté menos defensiva seria alocar menos.

Os autores [DEM 2003] alertam que estimar a exposição do risco não é uma ciência bem-definida. É uma aproximação que pode ser obtida a partir dados da indústria, listas de problemas prévios, um repositório de riscos, ou simplesmente uma estimativa. Mas isto não deve ser considerado uma desculpa para se deixar de realizar a estimativa da exposição do risco, pois esta atividade é extremamente importante.

Os autores ainda afirmam que a exposição do risco também pode ser medida em termos de tempo (prazo) com que o risco pode afetar o projeto. Por exemplo, um risco que irá custar 5 meses de atraso ao projeto caso se materialize, e cuja probabilidade de ocorrer é de 20%, então a exposição de risco (medida em tempo, neste caso) será de 1 mês.

Há ainda que se tomar cuidado com riscos que podem ser considerados *show-stoppers*, ou seja, no caso de se materializarem causam um impacto tão grande, seja em termos de custo ou de prazo, que podem causar o cancelamento do projeto. A identificação deste tipo de risco não invalida o esforço de gerência de riscos, embora seja difícil de quantificar estes riscos. A única forma de gerenciá-los é através do que se chama de **asserções do projeto**. Para que o gerente de projeto possa continuar seu trabalho, este deve assumir que o risco *show-stopper* não ocorrerá. Caso alguma asserção não seja verdadeira, o problema resultante deverá ser escalado para um gerente de nível superior. Este tipo de risco está acima da autoridade e da responsabilidade do projeto.

Além disso, há outros custos de gerência de riscos além da reserva de contenção dos riscos, que são os custos envolvidos na suavização dos riscos. A suavização é por definição algo que é feito antes da materialização do risco, portanto a sua ocorrência não é condicional. Seus custos não podem ser recuperados no caso de o risco não se materializar, mas tornam o impacto do risco menor no caso de este ocorrer.

Uma atividade que também é bastante importante no gerenciamento de riscos é o monitoramento de indicadores de materialização. Para cada risco sendo gerenciado, deve-se definir um ou mais indicadores de materialização, e então monitorá-los com frequencia para ativar um plano de contingência tão cedo quanto for necessário.

Nem sempre o indicador que pode ser detectado o mais cedo possível será o indicador mais adequado de ser monitorado, no entanto. O indicador pode não ser muito confiável e gerar um alarme falso. Por outro lado, o indicador mais confiável pode aparecer tarde demais para que se possa fazer algo a respeito. Por exemplo, há a máxima dos caminhoneiros: “toda bola rolando precede uma criança correndo”. Mesmo que nem sempre que aparece uma bola rolando em frente a um caminhão venha uma criança correndo atrás, este é um indicador adequado, pois, caso se decida esperar para visualizar a criança antes de freiar o caminhão, pode ser tarde demais e causar um acidente sério. Assim, para se definir indicadores de transição, é necessário um balanceamento entre a urgência do risco e o custo de disparar-se um alarme falso.

5.2 Um arcabouço para a identificação de riscos em projetos de software

Mark Keil e outros autores [KEI 98] apresentam um arcabouço (*framework*) para a identificação de riscos em projetos de desenvolvimento de *software*. O arcabouço foi criado com base em painéis realizados com gerentes de projetos em diferentes partes do mundo – Finlândia, Estados Unidos e Hong Kong. Os gerentes primeiramente identificaram uma lista de riscos, e depois os priorizaram em termos de importância. Por fim, os resultados foram combinados e analisados.

Segundo os autores, a quantidade de dinheiro movimentada atualmente nos Estados Unidos em projetos de desenvolvimento de *software* é bastante grande. Mesmo assim, uma grande quantidade de projetos tem falhado. Uma das explicações para estas falhas seria que os gerentes não estariam tomando medidas prudentes para identificar e gerenciar os riscos nestes projetos.

Possivelmente o principal resultado do estudo é que, após a realização de três painéis diferentes, em três diferentes continentes, foram selecionados 11 fatores de risco como sendo os itens mais importantes. Abaixo estes são listados, em ordem de importância:

- Falta de comprometimento da gerência sênior do projeto.
- Falha na obtenção de comprometimento por parte do cliente.
- Mal entendimento dos requisitos.
- Falta de envolvimento adequado por parte dos usuários.
- Falha no gerenciamento da expectativa dos usuários finais.
- Mudança de escopo/objetivos.
- Falta de conhecimento/habilidade no pessoal do projeto.
- Falta de requisitos estáticos.
- Introdução de novas tecnologias.
- Montagem de equipe insuficiente/inadequada.
- Conflitos entre os departamentos dos usuários.

O artigo, por questões de espaço, não pôde descrever a discussão que foi realizada sobre cada um dos riscos, por isso discutiu apenas os três mais importantes.

Em relação ao comprometimento da gerência sênior do projeto, muitos dos participantes dos painéis demonstrou acreditar que este item é tão importante que pode sombrear todos os demais caso não seja bem endereçado. Isto significa que a gerência sênior deve ter um papel de comprometimento do início ao fim do projeto.

Outra área de preocupação para os participantes foi a obtenção de comprometimento por parte dos usuários. Este comprometimento seria importante por garantir que os usuários participem ativamente da definição dos requisitos, a também por criar um senso de propriedade, assim minimizando o risco de o *software* ser rejeitado. Diversos participantes concordaram na opinião de que um comprometimento bastante forte por parte dos usuários pode compensar até mesmo a falta de comprometimento da gerência sênior.

Outro ponto crítico em um projeto seria o entendimento dos seus requisitos, pois estes guiariam todo o projeto.

5.2.1 Categorizando os riscos

Uma descoberta nos painéis realizados pelos autores [KEI 98] com gerentes de projetos foi que os riscos considerados mais importantes não estão muitas vezes sob o seu controle. Na verdade, o fato de um gerente não ter controle sobre um determinado risco faz com que este risco sejam considerado mais importante que os demais. A noção de controle e a percepção de como esta afeta a importância de um risco permitiu aos autores o desenvolvimento de uma categorização dos riscos.

Os riscos foram categorizados através de um “grid” 2x2. Uma de suas dimensões é a percepção de importância, que foi definida de forma relativa de um risco em relação aos demais riscos. A importância é alguma combinação da freqüência (probabilidade de que o risco venha a ocorrer) e o impacto do risco (isto é, qual a seriedade da ameaça no caso de o risco vir realmente a ocorrer). A segunda dimensão é a percepção do nível de controle, e representa o grau em que os gerentes

consideram que suas ações podem prevenir o risco de ocorrer. Estas dimensões, que são obviamente contínuas, foram reduzidas a um “grid” para efeito de simplicidade.

O resultado do “grid” formulado pelos autores foram os quadrantes, descritos abaixo:

- **Quadrante 1: O cliente determina**. Vários dos principais riscos dos participantes do painel entraram neste quadrante. Exemplos são o comprometimento da gerência sênior e do cliente. Estes itens são críticos mas estão fora do controle do gerente do projeto. Requerem estratégias que envolvem manter um bom relacionamento com os clientes e com os gerentes. Este tipo de relacionamento não seria obtido a curto prazo, mas deveria ser uma preocupação de longo prazo. Um elemento essencial para este relacionamento funcionar no longo prazo seria o cumprimento dos compromissos por parte do gerente de projetos. Outra preocupação seria não somente obter o compromisso no início do projeto, mas também manter o comprometimento dos clientes em seu decorrer. Uma estratégia útil neste sentido seria a de obter o comprometimento de diversos clientes e usuários chave, não deixando que este venha de apenas um usuário ou cliente. De forma geral, os riscos deste quadrante não podem ser controlados pelo gerente de projetos, mas podem ser influenciados.
- **Quadrante 2: Escopo e requisitos.** Os riscos neste quadrante estão relacionados à incerteza dos requisitos e da definição do escopo do projeto. Exemplos de riscos neste quadrante são o mal entendimento dos requisitos e o não gerenciamento adequado das mudanças. Estes riscos também são importantes, mas seriam considerados mais dentro do controle do gerente de projetos. As estratégias para o gerenciamento destes riscos envolvem o gerenciamento da ambigüidade e das mudanças. Com maior ou menor freqüência, é impossível determinar completamente os requisitos do projeto em seu início, o que levaria à adoção de modelos de ciclo de vida evolucionários de desenvolvimento de *software*, como o modelo em espiral. Outra estratégia útil seria a de listar o que não estaria incluído no projeto para definir o seu escopo. De forma geral, os riscos deste quadrante podem ser bastante controlados pelo gerente de projetos, mas requerem habilidade na interface com os clientes e usuários. O maior perigo dos riscos deste quadrante é que os gerentes de projeto podem falhar em perceber suas próprias limitações.
- **Quadrante 3: Execução**. Os riscos deste quadrante estão relacionados à execução do projeto propriamente dita, ou seja, uma vez definido um escopo e determinados os objetivos do projeto, este pode ser concluído com a equipe e os demais recursos dos quais o projeto dispõe? Exemplos de risco neste quadrante são uma equipe de projeto inadequada, a má definição de papéis, e a falta de uma metodologia de desenvolvimento. Os participantes dos painéis conduzidos pelos autores do trabalho consideraram estes riscos como de média importância, pois acreditam ter controle suficiente sobre os mesmos. As principais estratégias envolveriam a condução de avaliações internas e externas, a definição de uma metodologia de desenvolvimento e dos papéis e responsabilidades, além do

desenvolvimento de planos de contingência para lidar com a saída de membros da equipe e com novas tecnologias.

- **Quadrante 4: Ambiente**. Os riscos deste quadrante dizem respeito ao ambiente tanto interno quanto externo à organização. Exemplos de riscos neste quadrante são mudanças na gerência sênior que impliquem em mudanças de escopo e objetivos, e conflitos que possam surgir entre os departamentos dos usuários. Sobre estes riscos o gerente de projetos têm muito pouco controle. Normalmente, têm pouca possibilidade de ocorrer, por isso não são considerados como muito importantes. No entanto, quando ocorrem, podem ser bastante impactantes e perigosos para o projeto. As estratégias para lidar com estes riscos são difíceis, mas incluem um planejamento para o desastre, e a avaliação de cenários.

A figura abaixo mostra o "grid" que foi desenvolvido pelos autores [KEI 98].

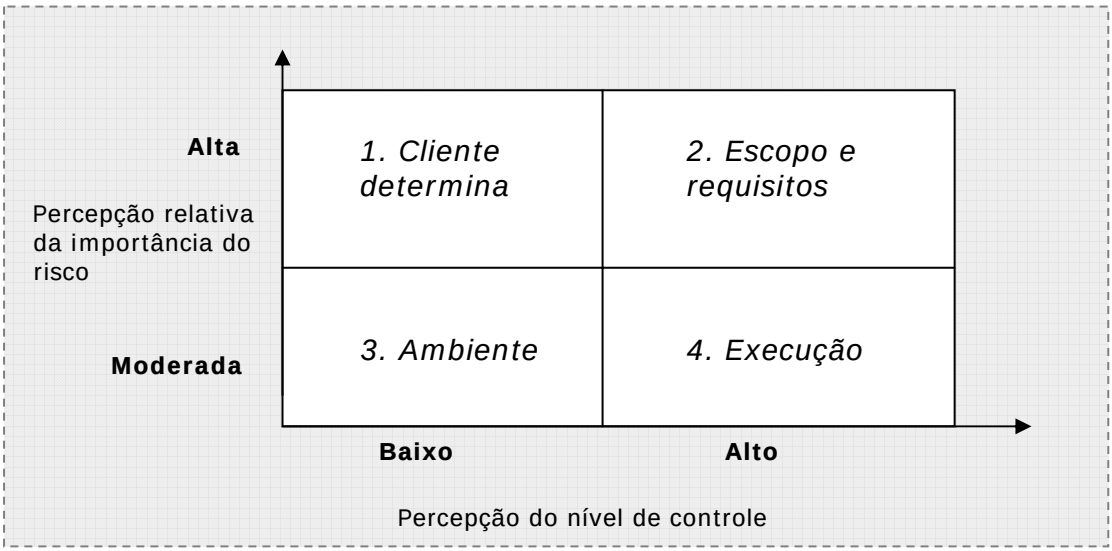


Figura 5.1: Arcabouço para categorização de riscos, adaptada de [KEI 98]

Por fim, os autores fazem algumas conclusões. Uma de suas descobertas seria o fato de que muitos dos riscos que afetam um projeto de desenvolvimento de *software* não estão diretamente sob o controle do gerente de projetos. Além disso, os autores realizaram uma comparação com o trabalho de Barry Boehm [91], o qual não incluía muito dos riscos identificados como sendo importantes. Esta diferença se deveria ao fato de Boehm ter trabalhado apenas com riscos de execução de um projeto. Uma contribuição do trabalho seria, portanto, que os gerentes de projeto não deveriam se restringir aos riscos de execução, mas sim utilizar o arcabouço de quadrantes para explorar um conjunto maior de fatores de risco.

5.3 Gerenciando riscos no desenvolvimento de produtos e processos e evitando surpresas

John Coppendale [COE 95] afirma que há muitos exemplos de projetos de desenvolvimento de *software* que atrasam ou que excedem seu orçamento, e que,

algumas vezes, as conseqüências são tão sérias que podem comprometer a própria existência das companhias envolvidas. Por isso é necessário um adequado gerenciamento dos riscos destes projetos. Todos os projetos carregam risco e incerteza. Mesmo que nem sempre o risco possa ser eliminado inteiramente, ele pode ser gerenciado e resolvido.

Ao desenvolver novos produtos e processos de manufatura para seus clientes, a organização do autor criou uma abordagem estruturada para o gerenciamento de riscos. Esta abordagem foi utilizada com sucesso em uma variada gama de setores, incluindo aeroespacial, defesa, manufatura de materiais e bens de consumo duráveis. Utiliza principalmente três etapas, descritas a seguir.

5.3.1 Identificar os riscos

Nesta etapa, os membros do projeto são entrevistados individualmente e uma ou mais reuniões em grupo são realizadas. Pessoas não diretamente envolvidas com o projeto também devem participar, pois muitos riscos potenciais não estão sob a responsabilidade imediata da equipe de desenvolvimento.

Através das entrevistas e de sessões de *brainstorming* durante as reuniões em grupo, é identificada uma lista de riscos. Esta lista pode ficar longa, contendo até mesmo centenas de riscos. Alguns deles podem ser óbvios para o time, mas podem não ter sido comunicados para o gerente de projeto. Outros poderiam não ser lembrados por ninguém caso o time não tivesse passado pelo processo. A longa lista de riscos é organizada em categorias, como “riscos externos”, “riscos de gerenciamento de projetos”, “riscos comerciais”, etc. Dependendo do tamanho e da complexidade do projeto, pode haver entre 5 a 15 categorias.

Tabela 5.1: Avaliando os riscos em termos de probabilidade e impacto, adaptada de [COE 95]

Probabilidade de ocorrer	0 representa uma probabilidade de ocorrência de menos de 5%
	5 representa uma probabilidade de aproximadamente 50% que o risco ocorra
	10 representa uma probabilidade maior de 95% de que o risco venha a ocorrer
Impacto	0 representa nenhum aumento no custo ou prazo do projeto
	10 representa um aumento significativo de custo (como 100%) e/ou um acréscimo significativo de prazo (mais de um ano, digamos)

5.3.2 Avaliar a probabilidade de ocorrer e o impacto em potencial dos riscos

Nesta etapa, se trabalha com o time de projeto para se avaliar tanto a probabilidade de ocorrer e o impacto em potencial de cada risco. Estas medidas podem ser feitas em uma escala de 0 a 10, aplicando-se algumas regras, como mostra a tabela 5.1.

O autor afirma que, embora seja importante atingir-se um nível de concordância razoável, não vale a pena gastar muito tempo debatendo pequenas diferenças de avaliação dos riscos. Quando um nível razoável de concordância foi atingido, os resultados podem ser pontuados graficamente em uma matriz, como na figura 5.2.

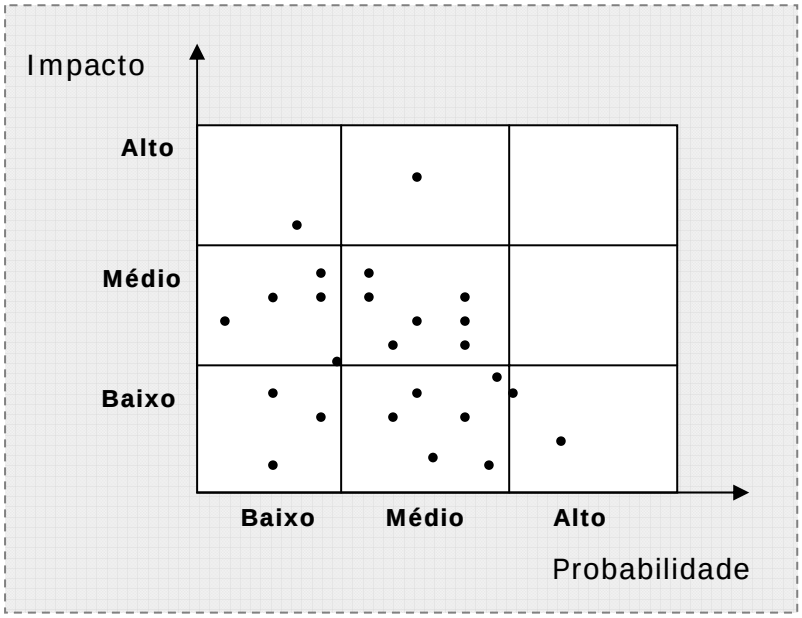


Figura 5.2: Distribuição da avaliação dos riscos, em termos de probabilidade e impacto, adaptada de [COE 95]

5.3.3 Desenvolver planos para o gerenciamento dos riscos

A matriz de riscos da figura 5.2 mostra imediatamente os riscos que têm ao mesmo tempo uma alta probabilidade de ocorrer e um alto potencial de impacto. Um plano de gerenciamento de riscos irá idealmente endereçar todos os riscos – ignorar um risco pode ser considerado negligência, se houver atitudes que possam ser tomadas e que sejam benéficas em termos de custo comparando-se com as implicações que pode haver para o projeto caso nada seja feito. No entanto, nem todos os riscos necessitam de uma atenção imediata. Podem haver restrições, como de orçamento, que impeçam todos os riscos de serem resolvidos simultaneamente. Nestes casos, aqueles riscos que têm uma alta probabilidade de ocorrer, e um alto potencial de impacto, são os que necessitam atenção imediata.

Exemplos de ações que podem reduzir o impacto dos riscos incluem:

- Realizar testes cedo no projeto, para verificar a viabilidade das soluções técnicas mais críticas.
- Realizar desenvolvimento concorrente para ter uma solução de contingência.
- Ter uma opção alternativa de fornecimento para componentes ou materiais críticos.

Exemplos de ações que podem reduzir a probabilidade dos riscos ocorrerem são:

- Obter uma segunda opinião em áreas de tecnologia pouco familiares ou críticas.
- Impor condições contratuais em um fornecedor sub-contratado que estiver suprindo materiais ou componentes críticos.

As ações de prevenção dos riscos devem ser atribuídas a indivíduos que ficarão responsáveis por estas, e o progresso destas atividades deve ser monitorado pelo gerente de projeto. Há o perigo de que, uma vez que o tenha passado por este exercício uma vez, o time acredite que os riscos podem ser ignorados pelo restante do projeto. O plano de gerenciamento de riscos deve ser revisto regularmente e o exercício completo de avaliação e gerenciamento deve ser realizado novamente em intervalos regulares.

6 ESTUDO DE ABORDAGENS PARA A SELEÇÃO DE METODOLOGIAS

Nesta seção será feito um estudo sobre abordagens existentes na literatura que procuram, a partir de uma análise das necessidades e dos riscos existentes em um projeto de desenvolvimento, definir uma metodologia adequada. O estudo será baseado principalmente no trabalho de Alistair Cockburn [COC 2000], [COC 2001b], e Barry Boehm [BOE 2003a], [BOE 2003b].

6.1 Selecionando a Metodologia de um Projeto

Alistair Cockburn [COC 2000] discute alguns princípios que seriam úteis para que se determinem as necessidades para vários processos ou metodologias, e que ajudariam a escolher uma metodologia adequada para um projeto.

Para começar, o autor discute o vocabulário utilizado em seu trabalho. Em 95, Sam Adams teria descrito as diferenças encontradas entre as metodologias aplicadas pelas grandes firmas de consultoria e as descritas nos livros de métodos de desenvolvimento de *software* orientado a objetos, como os de Grady Booch e James Rumbaugh. Ele teria dito que estes livros descreveriam algumas poucas técnicas e alguns poucos papéis. As grandes firmas de consultoria, por outro lado, utilizariam o que foi chamado de “Metodologias com um grande M” (*Big-M methodologies*), nas quais procurariam descrever todo o possível sobre sua forma de trabalhar – processos, técnicas, e normas, tudo isto sendo apenas uma parte da figura global.

Segundo o autor, alguns autores gostam de utilizar o termo *processo*. No entanto, um processo tipicamente é uma série de passos, e não teria todo o significado necessário para se descrever a forma de um time trabalhar. Por exemplo, não cobrem valores culturais, organização do posicionamento das pessoas no ambiente de trabalho, entre outras questões. Portanto, o trabalho do autor cobriria a diversidade de metodologias de forma universal, o que naturalmente incluiria a diversidade de processos como um subconjunto.

Outros termos do vocabulário do autor:

- **Tamanho (*size*)**: o tamanho de uma metodologia estaria relacionado ao número de elementos de controle, como entregáveis intermediários, atividades, medidas de qualidade, e assim por diante.
- **Densidade (*density*)**: a densidade é o detalhe e a consistência requerida pelos elementos produzidos.

- **Peso (*weigth*)**: o peso de uma metodologia seria seu tamanho vezes sua densidade (apenas conceitualmente).
- **Tamanho de um projeto**: o tamanho de um projeto é o número de pessoas que uma organização aloca para um determinado projeto.
- **Tamanho do problema**: o tamanho do problema diz respeito a dificuldade do *software* a ser produzido. Segundo o autor, este é difícil de ser medido, pois uma nova pessoa pode ver um padrão que simplifica o problema. Por isso, seria importante separar o tamanho do projeto do tamanho do problema.

Uma metodologia, no ponto de vista do autor, incluiria ao menos os elementos apresentados na figura 6.1: pessoas, papéis, habilidades, times, ferramentas, técnicas, processos, atividades, marcos intermediários (*milestones*), produtos de trabalho, normas, medidas de qualidade, e valores do time.

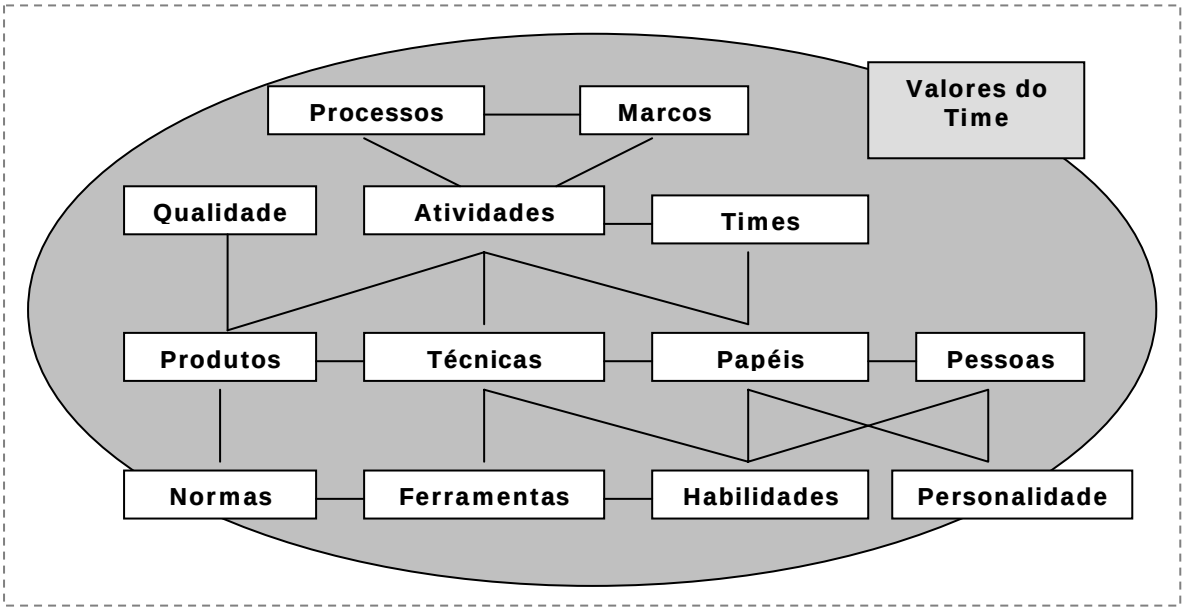


Figura 6.1: Elementos de uma metodologia “grande-M”,
adaptada de [COC 2000]

A figura 6.1 apresenta os relacionamentos entre os diversos elementos de uma metodologia. As pessoas, que têm habilidades particulares, preenchem papéis no projeto, trabalhando em diferentes tipos de times. Elas utilizam técnicas para construir os produtos de trabalho que seguem certas normas e atendem aos critérios de qualidade selecionados. As técnicas requerem certas habilidades e ferramentas; as ferramentas auxiliam a garantir o seguimento das normas. O time se envolve em atividades que fazem parte dos processos; cada atividade indica marcos que definem o quanto o processo está se movendo em frente. Todos estes elementos operam sob os valores do time, que precisam estar alinhados com os valores individuais e os processos sendo utilizados.

6.1.1 Princípios envolvidos na seleção de metodologias

O autor [COC 2000] apresenta alguns princípios que seriam importantes de serem entendidos para suportar a seleção de uma metodologia adequada a um projeto, os quais teriam sido desenvolvidos com base em dezenas de entrevistas em projetos, e após vários desenhos de metodologias pelo autor:

1. ***Um grupo maior precisa de uma metodologia maior***. Uma metodologia seria maior quando esta contém mais elementos, como papéis, produtos de trabalho, revisões, etc. O autor afirma que as metodologias existem primariamente para coordenar as pessoas, por isso, quanto maior for o projeto, maior será a metodologia que será considerada apropriada ao mesmo.
2. ***Um sistema mais crítico, ou seja, um no qual defeitos não detectados irão produzir maior estrago, precisa de uma metodologia de maior densidade em sua construção***. O autor separa a avaliação de criticalidade de um projeto em quatro categorias. O que este princípio significa é que um time pode justificar um maior custo para o projeto para proteger contra erros, conforme for a criticalidade do projeto.
    a. *Perda de conforto*: significa que com uma falha no sistema, as pessoas vão somente estar menos confortáveis, terão de fazer mais trabalho manualmente, ou entrar em contato entre si para reparar problemas de comunicação. Sistemas de suporte a compras e infraestrutura corporativa entrariam nesta zona.
    b. *Perda de dinheiro discreto,* que significa que um defeito no sistema produzirá a perda de dinheiro ou valores relacionados, mas somente na região do desconforto. Sistemas de controle de faturas tipicamente entrariam neste item.
    c. *Perda de dinheiro essencial*, que significa que um defeito no sistema irá causar a perda de dinheiro ou de valores similares que podem levar à quebra da empresa. Sistemas de automação bancária se enquadrariam nesta categoria.
    d. *Perda de vidas*: neste caso um defeito pode causar a perda de uma ou mais vidas humanas. Sistemas de controle de aviões, naves espaciais, ou plantas atômicas se enquadram nesta categoria.
3. ***Um aumento relativamente pequeno no tamanho ou densidade da metodologia adiciona um custo relativamente grande ao projeto***. Interromper o desenvolvimento para coordenar com outras pessoas custa não somente tempo, mas também concentração. Manter documentos de requisitos, projeto, e testes atualizados são também grandes consumidores de tempo. Este princípio não questiona o fato de as atividades de coordenação do trabalho serem benéficas ou ruins. Ele somente endereça o custo de se adicionar elementos de controle a uma metodologia.
4. ***O mais efetivo meio de comunicação para a transmissão de idéias é interativamente face-a-face, em frente a um quadro-branco.*** Este princípio significa que pessoas as quais estão sentadas de forma próxima, com contato freqüente e fácil, irão desenvolver *software* mais facilmente. Ou seja, o *software* será mais barato de ser desenvolvido. No entanto, conforme um projeto cresce em tamanho, a comunicação interativa, face-a-face, fica mais

difícil de ser obtida, de forma que a eficiência de comunicação no projeto decresce, e conseqüentemente seu custo aumenta.

Com base nos princípios apresentados, o autor [COC 2000] apresenta alguns relacionamentos entre as dimensões sobre as quais as metodologias variam.

Uma primeira análise que o autor mostra estaria relacionado ao tamanho do problema, ao tamanho do projeto, e, conseqüentemente, ao tamanho da metodologia. Um time pequeno tipicamente precisa de pouca metodologia para coordenar seu trabalho. Com menos trabalho necessário, o time consegue trabalhar de forma mais produtiva. Como conseqüência, o time consegue endereçar um problema de tamanho maior, o que forma um ciclo positivo.

Por outro lado, um projeto com um time maior precisa de uma metodologia maior para coordenar os esforços do time. Em conseqüência, a produtividade do time diminui, portanto, para completar o mesmo trabalho do time menor, o time com a metodologia maior precisa de mais pessoas.

Portanto, para um mesmo tamanho de problema, tipicamente precisa-se de um time menor se você utilizar uma metodologia menor (ver figura 6.2). Mas há um limite para o tamanho do problema que esta metodologia será capaz de resolver. Metodologias maiores têm um limite mais alto para o tamanho de problema que conseguem resolver, do que uma metodologia pequena aplicada a um time pequeno de desenvolvimento.

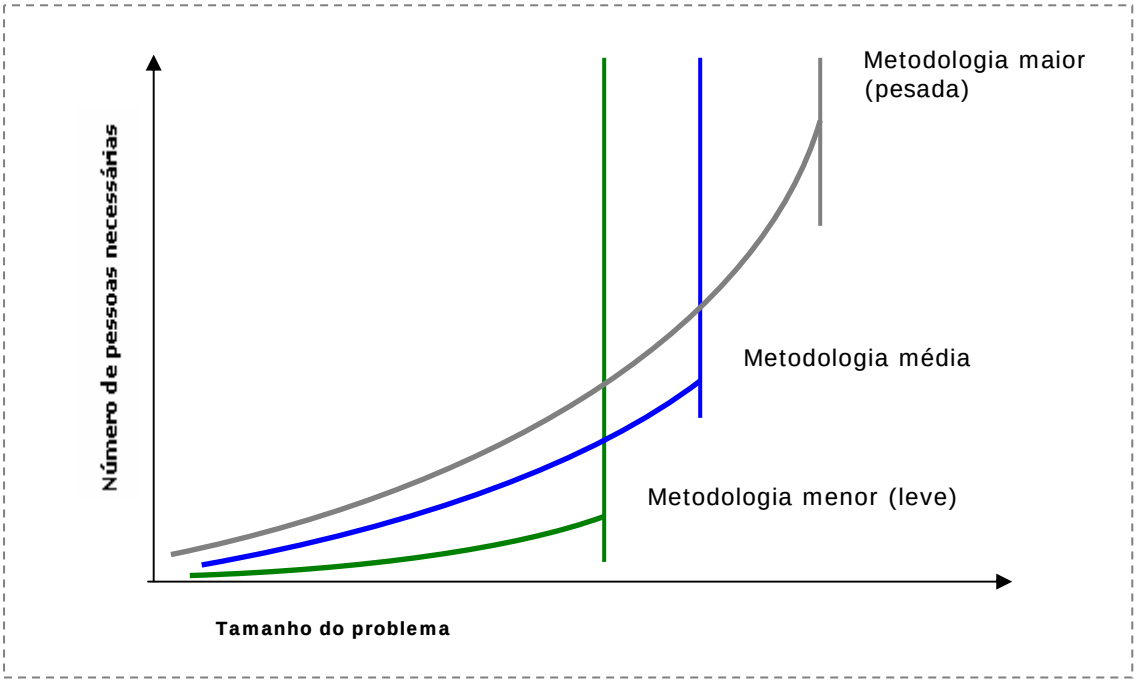


Figura 6.2: Relação entre o tamanho do problema, o número de pessoas envolvidas, e o tamanho da metodologia, adaptada de [COC 2000]

Segundo Cockburn [COC 2000], para cada tamanho de problema haveria uma combinação ótima de metodologia e tamanho de projeto. A dificuldade é que esta combinação poderia variar de acordo com quais as pessoas que estão no time, ou seja, a habilidade dos componentes do time teria influência.

Outra análise sobre os princípios que foram apresentados diz respeito a efetividade das formas de comunicação humana. A figura 6.3 mostra a temperatura e riqueza da comunicação por diferentes canais. A figura acrescenta ao princípio 4, de que a comunicação interativa face-a-face é a forma de comunicação mais efetiva. Segundo [COC 2001b], este princípio não significaria que a comunicação fica impossibilitada de outra forma, nem que todo *software* pode ser desenvolvido por um pequeno grupo co-locado. O que o princípio implica é que um projetista de metodologia, ou metodologista, caso deseje melhorar a produtividade e reduzir o custo de um projeto, pode enfatizar pequenos grupos de desenvolvimento, com bastante contato pessoal. O princípio seria suportado por pesquisas na área de gerenciamento.

Um exemplo de como os princípios apresentados pelo autor teriam sido aplicados com sucesso seria o caso do sistema C3 (*Chrysler Comprehensive Compensation*) [C3T 98], o qual foi a implementação original do método Programação Extrema [BEC 99]. Após 26 pessoas falharem em desenvolver o que era considerado um grande sistema, um subconjunto formado por oito pessoas reiniciou o projeto. Este pequeno grupo de desenvolvedores foi capaz de em um ano o que o time maior com uma metodologia pesada não foi capaz de realizar. Parte do sucesso seria a aderência ao princípio 4 de Cockburn.

Outra questão despertada pelo princípio 4 seria a seguinte [COC 2001b]: “de que forma os diferentes canais de comunicação afetam a avaliação da conformidade de um time com um contrato?”. Esta pergunta introduziria a questão de *visibilidade* em uma metodologia, e produziria provavelmente um resultado bastante diferente, em que se documentos escritos seriam enfatizados.

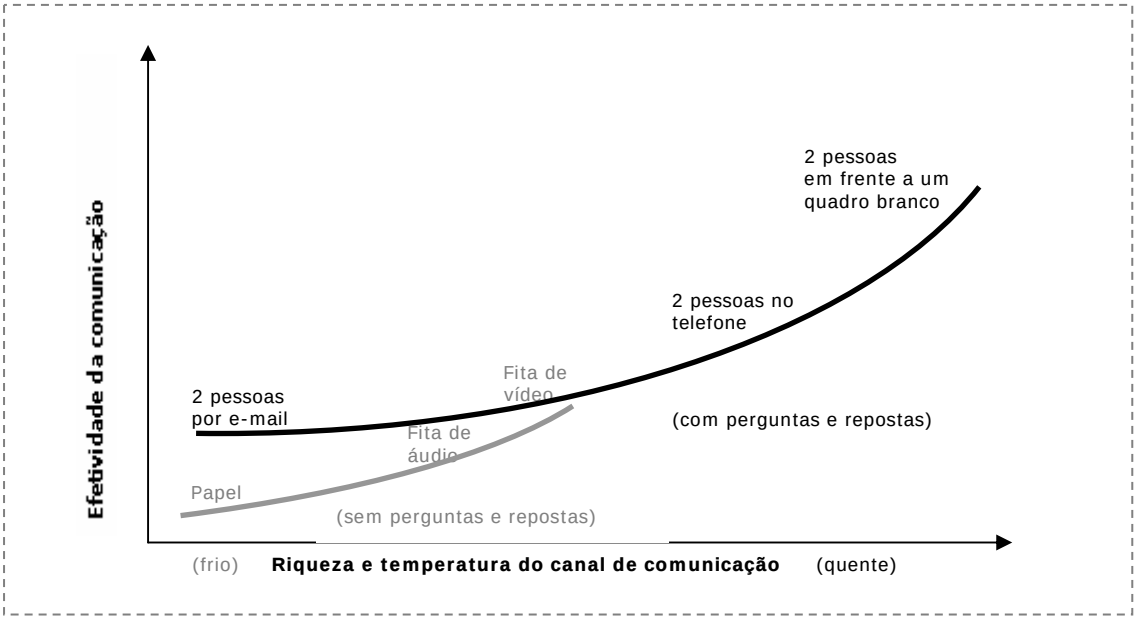


Figura 6.3: Comparação da efetividade de diferentes canais de comunicação, adaptada de [COC 2001b]

6.1.2 Outros fatores para a seleção de metodologias

Alistair Cockburn [COC 2000] descreve ainda dois outros fatores que afetariam a seleção de uma metodologia adequada a um projeto:

- **Prioridades do projeto**: importa bastante na seleção da metodologia os interesses de quem financia o projeto: se estes desejam o *software* pronto rapidamente, se o desejam livre de defeitos, ou se desejam que o processo tenha visibilidade. As diferentes metodologias existentes na literatura teriam diferentes prioridades. Por exemplo, o método PSP (*Personal Software Process*) claramente priorizaria para previsibilidade; já a família Crystal de metodologias priorizaria para produtividade e tolerância.
- **Os medos do projetista da metodologia**: Cockburn cita Kent Beck que teria afirmado que "toda metodologia é baseada em medos" em uma discussão. Os elementos existentes em um processo podem ser considerados como preventivos contra uma má experiência pela qual a pessoa passou. Por exemplo, se você está com medo que os desenvolvedores irão cometer erros de projeto, realize revisões de projeto; se você está com medo que seus projetistas deixem o projeto antes de seu final, faça com que os mesmos escrevam uma documentação extensiva. No entanto, é importante que se faça uma distinção entre os riscos reais existentes em um projeto, e a bagagem do metodologista. O metodologista utiliza sua experiência para desenhar a metodologia. No entanto, os riscos variam de projeto para projeto, por isso a metodologia desenhada irá ser útil a um projeto na medida em que as asserções do metodologista corresponderem a riscos reais do mesmo.

6.1.3 Selecionando uma metodologia adequada a um projeto

A abordagem para a seleção de uma metodologia proposta pelo autor [COC 2000] utiliza-se de três dimensões de análise: número de pessoas x criticalidade x prioridades do projeto, o que resultou na figura 6.4.

A vantagem desta abordagem é que ela seria objetiva. As divisões em zonas são arbitrárias, mas plausíveis. Pode-se contar o número de pessoas do projeto e levantar-se a criticalidade e as prioridades.

As metodologias tipicamente ficam maiores nas zonas que se situam para o lado direito da figura, e ficam mais densas conforme a zona aproxima-se da parte superior (um vocabulário de termos é provido no início desta seção do trabalho). De acordo com o princípio 3 do autor, mover-se para o lado direito ou para cima neste desenho implica em um grande aumento de custo para o projeto, o que significa que há um incentivo financeiro para que um time permaneça no lado esquerdo e abaixo no gráfico da figura 6.4.

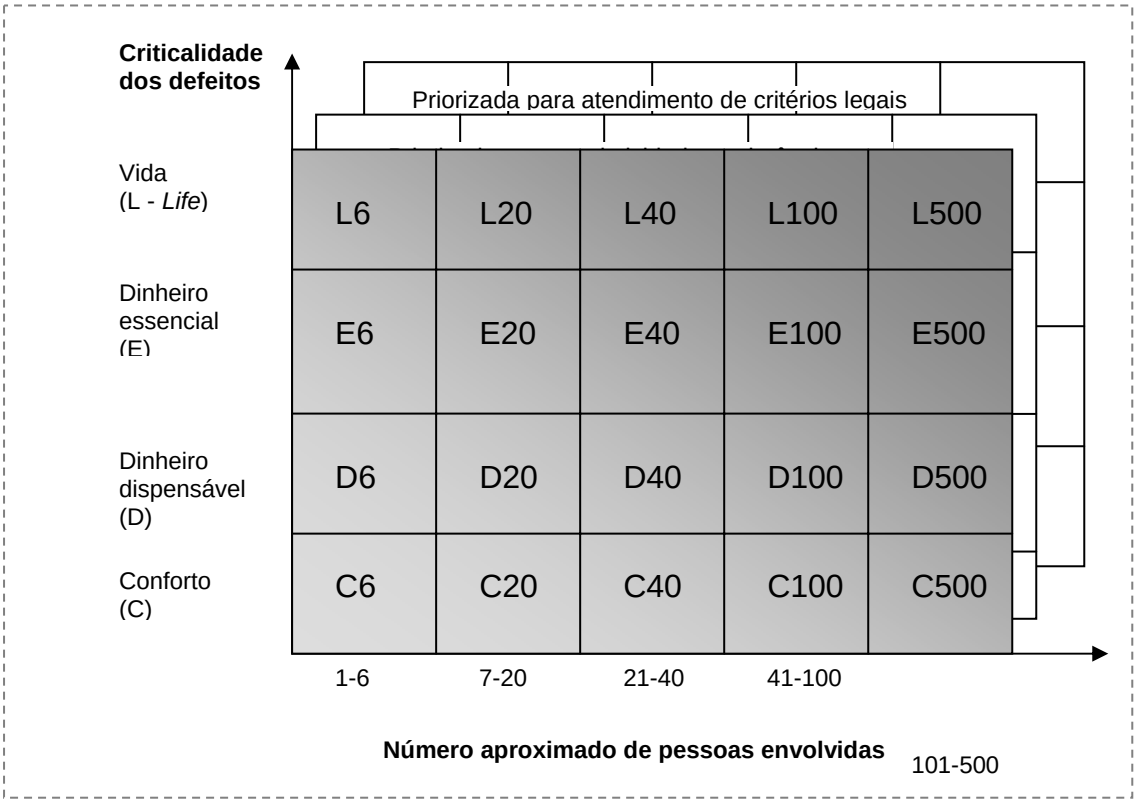


Figura 6.4: Abordagem para a seleção de metodologias, baseada em criticalidade x número de pessoas x prioridades do projeto, adaptada de [COC 2000]

Para cada célula no desenho, haveria várias metodologias que poderiam ser adequadas, de acordo com as prioridades do projeto, como visibilidade, produtividade, correção, etc. Após utilizar-se o desenho, mais os princípios apresentados pelo autor, pode-se tomar decisões básicas sobre qual metodologia utilizar. A partir de então, as preferência e experiências pessoais são utilizadas.

No artigo [COC 2000] são ainda discutidas experiências de aplicação da abordagem do autor no desenho de metodologias para projetos com características bem distintas. Em seu livro [COC 2001b], os princípios apresentados nesta seção são discutidos em maior profundidade.

6.2 Utilizando risco para balancear métodos ágeis e guiados a planos

Barry Boehm [BOE 2003a] [BOE 2003b] propôs uma abordagem, em alguns aspectos semelhantes à de Alistair Cockburn, para selecionar uma metodologia adequada a um projeto, balanceando entre os métodos ágeis e os métodos guiados a planos. Segundo o autor, os métodos ágeis, como a Programação Extrema [BEC 99] e Scrum [SCH 2002] prometeriam maior satisfação do usuário, menor número de defeitos, tempos de desenvolvimento mais curtos, e uma solução para requisitos em constante mudança. Por outro lado, abordagens guiadas a planos, como PSP (*Personal Software Process*) e métodos baseados no modelo CMM prometem previsibilidade, estabilidade, e garantia de qualidade.

O autor explica que tanto as abordagens ágeis, quanto as abordagens guiadas a planos têm problemas, que se não forem endereçados, podem levar o projeto a falhar. O desafio seria balancear ambas as abordagens, de forma a aproveitar seu potencial em determinadas situações, e ao mesmo tempo compensar suas fraquezas. Por isso, o autor propõe uma técnica baseada em riscos para montar projetos que incorporem ambas as abordagens, proporcionalmente as suas necessidades.

O método proposto pelo autor utiliza-se de uma análise de risco e de um arcabouço (*framework*) de processo para definir processos baseados em risco dentro da estratégia geral de desenvolvimento. Este método seria baseado em um modelo chamado de *pontos de âncora de um modelo espiral baseado em risco,* o qual seria um conjunto de critérios de decisão para o comprometimento dos interessados (*stakeholders*) em pontos específicos do processo de desenvolvimento. O método é aplicado em cinco etapas:

- **Passo 1**: Inicia-se aplicando uma análise de risco para áreas específicas de risco, que são divididas em três categorias: *ambientais, ágeis*, e *guiados a plano*s.
- **Passo 2**: Neste passo, se avalia o resultado da análise de risco efetuada no passo 1, para verificar se o projeto se qualifica para ser puramente ágil ou guiado a planos. Se isto ocorrer, pula-se diretamente ao passo 4.
- **Passo 3**: Utiliza-se este passo se o projeto em análise não situa-se claramente nem no campo típico de atuação de um método ágil, nem no caso típico de utilização de uma abordagem guiada a planos. Se possível, desenvolve-se uma arquitetura de organização do projeto de forma a utilizar os métodos ágeis onde possam oferecer maiores benefícios e seus riscos possam ser minimizados. Os métodos guiados a planos fazem o resto do trabalho, sendo considerados como padrão quando uma arquitetura deste tipo não puder ser criada.
- **Passo 4**: Neste passo, é desenvolvida uma estratégia para endereçar os riscos que foram identificados. Isto inclui identificar estratégias individuais para cada um dos riscos e depois integrá-las.
- **Passo 5**: Neste passo a estratégia é implementada, enquanto os gerentes mantém um monitoramento e uma avaliação freqüente de sua performance. Este passo suporta o refinamento e ajuste do balanço entre os métodos ágeis e guiados a planos que foi definido inicialmente.

6.2.1 Revisando os Três Níveis de Entendimento de Cockburn

Alistair Cockburn, em seu livro [COC 2001b], define três níveis de entendimento que as diferentes pessoas têm. Com base nestes níveis de entendimento, pode-se avaliar o grau de habilidade dos componentes de um projeto, e assim considerar esta variável ao desenhar uma metologia.

Barry Boehm [BOE 2003a] utiliza a proposta de Cockburn como uma das dimensões de análise de um projeto, mas acrescenta um novo nível intermediário de entendimento. Assim sendo, o resultado é apresentado na tabela 6.1.

Tabela 6.1: Níveis de entendimento de Cockburn, revisados por Boehm, tabela adaptada de [BOE 2003a].

Nível	Características
3	Tem a habilidade necessária para revisar um método, inclusive quebrando suas regras para atender a uma nova necessidade não prevista.
2	Tem a habilidade de adaptar um método para atender a uma nova situação com precedentes.
1A	Com treinamento, é capaz de executar certas etapas de um método como estimar o tamanho de estórias para compor um determinado incremento, ou compor padrões ou fazer um redesenho composto. Com experiência, pode se tornar nível 2.
1B	Com treinamento, é capaz de seguir passos de procedimentos como codificar um método simples de uma classe, fazer um redesenho simples, seguir normas de codificação e de gerência de configuração, ou rodar testes.
-1	Pode ter algumas habilidades técnicas, mas não colabora ou não é capaz de seguir um método.

Os autores tomaram a liberdade de dividir o nível 1 de Cockburn para melhor capturar as diferenças entre os métodos ágeis e guiados a planos. Além disso, acrescentaram o nível –1. As pessoas situadas neste último nível deveriam ser rapidamente identificadas e encaminhadas a outro tipo de trabalho.

As pessoas de um nível 1B seriam capazes de realizar um desenvolvimento de *software* simples em uma situação estável. No entanto, tipicamente irão atrasar um time ágil o qual esteja lidando com uma situação em rápida mudança, ainda mais de formarem a maior parte do time. Poderiam, no entanto, formar a maior parte de um time em uma situação estável, bem estruturada, e guiada por planos.

Em um time ágil, as pessoas de um nível de entendimento 1A podem ser funcionais, desde que guiadas por uma pessoa de nível 2. Os agilistas, como Cockburn, mencionam ser possível trabalhar com equipes de pessoas de nível 1, desde que haja uma razão de ao menos 1 pessoa de nível 2 para cada pessoa de nível 1. Boehm corrige um pouco esta percepção, afirmando que de forma geral, neste caso, estes estão falando de pessoas de nível 1A.

As pessoas de nível 2 seriam capazes de guiar pequenos times utilizando métodos ágeis ou guiados a planos, mas precisariam de uma pessoa de nível 3 a guiá-los em projetos maiores ou sem precedência. Algumas pessoas de nível 2 poderiam atingir um nível de entendimento 3 com a experiência, mas outras não.

6.2.2 Utilizando cinco critérios para a seleção de uma metodologia adequada

Na abordagem proposta por Barry Boehm [BOE 2003a], são utilizados cinco critérios como determinantes a seleção de uma metodologia ágil ou guiada a planos para um projeto, ou ainda uma metodologia mista. A tabela 6.2 descreve estes cinco fatores, e como eles afetam as metodologias ágeis ou as guiadas a planos.

Estes fatores são utilizados para balancear entre os métodos ágeis e guiados a planos de acordo com as características de um projeto específico, e podem ser melhor visualizados graficamente na forma apresentada na figura 6.5. Dos cinco eixos

apresentados, o tamanho e a criticalidade são os mesmos fatores utilizados por Cockburn, onde tipicamente os métodos mais leves ou ágeis são aplicados para projetos menores e menos críticos, enquanto que métodos mais pesados são utilizados para projetos mais críticos e com um número grande de participantes.

O eixo de cultura significa que tipicamente um método ágil irá funcionar melhor em um ambiente onde o caos e a liberdade são valorizados, mas não funcionará tão bem em um ambiente cultural que valorize a ordem; o oposto é verdadeiro para os métodos guiados a planos.

Tabela 6.2: Fatores de risco utilizados por Boehm [BOE 2003a].

Fator	**Metodologia Ágil**	**Metodologia Guiada a Planos**
Tamanho	Metodologias que são mais adequadas a pequenos times. A dependência de conhecimento tácito limita a escalabilidade.	Métodos que são aplicados para controlar grandes times e produtos, mas são difíceis de serem adaptados a pequenos times.
Criticalidade	Não foram testadas em produtos críticos para a segurança; podem apresentar riscos em potencial devido ao desenho simples e à falta de documentação.	Métodos que são aplicados para endereçar produtos de alta criticalidade, mas são difíceis de serem adaptados para produtos de baixa criticalidade.
Dinamismo	O projeto simples e o redesenho contínuo podem ser uma boa prática para ambientes com alto grau de mudança, mas podem ser um risco potencial de re-trabalho em projetos estáveis.	Planos e projetos detalhados no início do projeto são bons para ambientes estáveis, mas um risco grande de re-trabalho para projetos com mudanças muito freqüentes.
Pessoal	Requer a presença de pessoas com um talento alto – os níveis 2 e 3 de Cockburn – que são escassas; é arriscado utilizar pessoas nível 1B.	Os métodos guiados a planos precisam de uma massa crítica de pessoas de nível 2 e 3 durante a definição do projeto, mas pode trabalhar com poucos mais adiante no projeto; pode acomodar pessoas de nível 1B.
Cultura	Utiliza uma cultura onde as pessoas se sentem confortáveis com o caos e por terem vários níveis de liberdade.	Utiliza uma cultura onde as pessoas se sentem confortáveis por terem seus papéis claramente definidos por procedimentos e políticas.

Os dois eixos restantes são o dinamismo e o pessoal, que são assimétricos no sentido de que em um extremo normalmente ou um método ágil ou um método guiado a plano irá ter sucesso, mas o método oposto falhará. No outro extremo ocorre o mesmo, mas ao contrário. Em relação ao dinamismo, os métodos ágeis podem funcionar bem tanto com alto quanto com baixo grau de mudanças, mas os métodos guiados a planos funcionam melhor com um baixo grau de modificações.

O eixo de pessoal corresponde à classificação de habilidades baseada em níveis de entendimento, que foi descrita na seção anterior. A assimetria neste caso é

que, enquanto os métodos guiados a planos podem funcionar bem tanto com pessoas mais qualificadas quanto com indivíduos de menor habilidade, os métodos ágeis precisam de uma quantidade maior de pessoas habilidosas.

6.2.3 Aplicando a abordagem de Boehm

A abordagem é explicada pelo autor estabelecendo um contexto realista através de três aplicações de exemplo. Para cada uma destas três aplicações de exemplo, os riscos do projeto sugerem uma mistura diferente entre os métodos ágeis e dirigidos a planos. São utilizados sistemas de planejamento baseados em agentes como aplicações de exemplo. As três seguintes aplicações seriam representativas:

- **Pequena, relativamente não crítica**: Esta aplicação baseada em agentes é utilizada para gerenciar eventos como convenções ou conferências, e apresenta os padrões de risco observados em pequenos sistemas de serviço na Web.
- **Intermediária**: Uma aplicação baseada em agentes para o gerenciamento de uma cadeia de suprimentos, interligando uma rede de produtores e consumidores. Os riscos desta aplicação foram estudados baseados na experiência de um projeto da empresa *ThoughtWorks* (uma grande empresa de desenvolvimento de *software* dos Estados Unidos, onde Martin Fowler é o principal cientista), onde o método Programação Extrema teve de ser adaptado para atender as necessidades de um projeto de 50 pessoas.
- **Muito grande, muito crítica**: Um sistema baseado em agentes para o planejamento do gerenciamento de crises nacionais americanas, é baseado nos padrões de risco observados em projetos do departamento de defesa e do exército americanos; um sistema sendo desenvolvido por mais de 2000 pessoas.

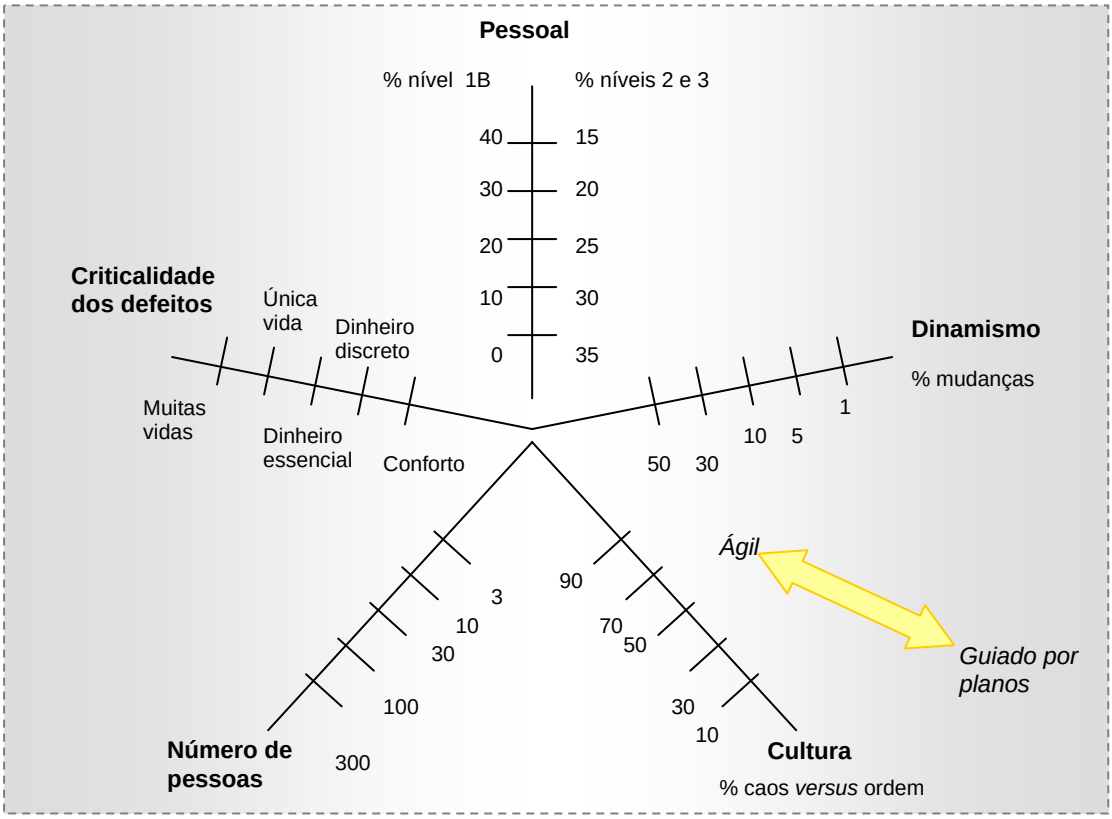


Figura 6.5: Cinco eixos para seleção de metodologias combinando métodos ágeis e métodos guiados por planos, figura adaptada de [BOE 2003a]

Enquanto a aplicação pequena tipicamente é um projeto adequado a uma metodologia ágil, a aplicação muito grande, por sua vez, é tipicamente suportada por uma abordagem dirigida a planos. Por isso, a aplicação intermediária é a mais interessante de ser analisada, pois permite examinar como que seus padrões de risco auxiliam a determinar uma mistura adequada.

A aplicação intermediária utilizada como exemplo seria um projeto de gerenciamento de cadeias de suprimentos, envolvendo cerca de 50 pessoas possivelmente em diferentes localidades. O objetivo do projeto é prover um rápido incremento de valor para a companhia de manufatura, através do aumento da velocidade, e da adaptabilidade da cadeia. Algumas partes desta aplicação são estáveis; já outras partes são bastante voláteis. Já os riscos associados a defeitos no sistema são altos, podem envolver perdas grandes para o negócio. Ou seja, esta aplicação possui características ágeis, como necessidade de agregar valor rapidamente, e a volatilidade. Mas também possui fatores guiados a planos, como a escalabilidade do projeto e a criticalidade.

Aplicando-se a seqüência de passos da abordagem de Boehm, inicia-se com uma avaliação dos riscos do projeto (passo 1). As principais fontes de risco no projeto sendo analisado são:

- **Riscos ambientais**: Os riscos ambientais deste projeto estão relacionados ao desconhecimento da tecnologia de planejamento baseados em agentes pelo time do projeto, e à existência de uma rede de fornecedores e compradores distribuída, a qual precisa ser coordenada.

- **Riscos ágeis**: É difícil conciliar o desejo de se freqüentemente produzir incrementos de funcionalidade com a necessidade de se manter as regras de dependência entre os diversos componentes do sistema. Além disso, é difícil conciliar o desejo dos desenvolvedores ágeis de aplicar a regra "você não vai precisar disto" (*You Aren't Gonna Need It – Yagni*) e de manter o projeto simples, com o fato de que algumas partes da aplicação são mais estáveis e, portanto, poderiam se beneficiar de um projeto antecipado, também é difícil. Por outro lado, a baixo risco relacionado a dependência de conhecimento tácito, pois a força de trabalho da empresa é razoavelmente estável.

- **Riscos guiados a planos**: Os riscos estão largamente relacionados à atrasos significativos que podem ocorrer pela necessidade de re-trabalho de planos elaborados, em um mercado volátil em constante evolução. Adaptar planos detalhados em um ambiente onde há rápidas mudanças na tecnologia, nas organizações, e nas condições de mercado, provavelmente será lento e custoso. A falta de habilidade de produzir novas funcionalidades rapidamente pode significar a perda de fatias de mercado. Depender demasiadamente de requisitos pré-especificados em áreas nas quais os requisitos emergem de acordo com a familiarização dos usuários pode afetar a arquitetura e os planos de forma imprevisível.

Na segunda etapa da análise, o time avalia se os riscos do projeto se encaixam mais dentro do perfil de um método ágil ou de um método guiado a planos. Como o projeto não se enquadra totalmente no perfil de uma das abordagens, é necessário escolher uma de duas opções: basear-se em uma abordagem ágil, e seletivamente acrescentar práticas guiadas a planos para endereçar riscos ágeis como a falta de escalabilidade, criticalidade e o projeto simplificado; ou então, basear-se em um modelo de projeto guiado a planos, com uma aplicação seletiva de práticas ágeis para endereçar riscos como a rápida mudança de requisitos e a necessidade de respostas rápidas.

Devido a fatores como a cultura e o ambiente da empresa, o time seleciona aplicar a abordagem ágil dirigida por riscos. Se, por outro lado, houvesse condições mais estáveis, melhor compreendidas, e um grande ritmo de troca de pessoal, seria mais apropriada uma abordagem guiada a planos dirigida por riscos.

Pula-se o passo 3 diretamente ao passo 4, onde se identificam estratégias individuais para cada um dos riscos identificados nos passos 1 e 2. A tabela 6.3 mostra os riscos individuais que foram identificados, e a estratégia que foi definida para resolvê-lo.

Tabela 6.3: Exemplo de estratégias para resolução de riscos, adaptada de [BOE 2003a].

Classe	**Risco**	**Estratégia**
Ambiental	Muitas redes de cadeias de suprimento separadas	Selecionar os representantes dos usuários (*stakeholders*), que devem ser colaborativos, representativos, autorizados, comprometidos, e com conhecimento.
	Incertezas técnicas; desconhecimento de sistemas baseados em agentes; riscos na utilização de componentes ainda não testados.	Antes de se comprometer com um pacote específico de componentes, o time deve detalhar o que espera do pacote, em termos de objetivos, restrições, e prioridades, e utilizar esta definição para avaliar várias alternativas de solução, através de validações e prototipações.
Ágeis	Escalabilidade do projeto em termos do número de pessoas: problemas de integração entre as diversas equipes distribuídas e com a dependência de conhecimento tácito e da responsabilidade compartilhada.	Os intervalos de liberação precisam ser maiores, sendo que na medida em que se atinge o final do intervalo, trabalha-se com a implementação incremental de funcionalidade de baixa granularidade, de forma a evitar que um time dependa de características que estão em desenvolvimento por outro time.
	Projeto simples e redesenho em situações em que se podem prever as mudanças antecipadamente.	Neste caso, podem ser utilizados padrões de projeto para desenvolver arcabouços que enderecem as modificações de forma mais eficiente do que com um projeto simples e redesenho.
	Rotação de pessoal e perda de conhecimento tácito	Rotativamente trocar os pares de programadores e trocar os desenvolvedores de times pode auxiliar a reduzir a perda de conhecimento tácito. Para evitar a perda dos usuários-chave (*stakeholders*), o que também pode prejudicar o projeto, o autor sugere a bonificação dos projetos completados com sucesso.

Guiados por planos	Atrasos e redução da competitividade resultantes da pré-especificação detalhada de planos e projetos.	Não escrever especificações quando o risco de não utilizá-las for baixo, e o risco de utilizá-las for alto. Deve-se procurar escrever especificações quando o risco de isto não ser feito for alto, e o risco de faze-lo for baixo. Um bom exemplo seria um protocolo de interface entre cadeias de suprimento, obtido através de extensiva negociação com os usuários-chave.

Por fim, os autores [BOE 2003a] afirmam que os desenvolvedores podem utilizar o processo adaptável para balancear entre métodos ágeis e guiados por planos de uma forma adequada a seu projeto específico. Desta forma, seria possível que as organizações se aproveitem dos benefícios oferecidos por ambas abordagens, ao mesmo tempo reduzindo os riscos de seus maiores problemas.

O exemplo apresentado seria uma aplicação que utilizaria a maior parte das práticas do método Programação Extrema [BEC 99], ao mesmo tempo utilizando uma arquitetura de componentes, auditorias de situações de risco, análises do negócio, e geração automática de documentação sob demanda.

7 PATTERN-BASED METHODOLOGY TAILORING

7.1 Introdução

Conforme foi estudado nos capítulos anteriores, há uma vasta literatura sobre metodologias de desenvolvimento de *software*. Cada uma das metodologias, como CMMI [SEI 2002], Scrum [SCH 2002] ou RUP [JAC 2000], foi normalmente desenvolvida por pessoas com diferentes experiências e conhecimentos, e freqüentemente as pressuposições não são claramente descritas nos textos que as descrevem. Definir ou aprimorar a metodologia de desenvolvimento de um projeto de *software* envolve uma série de decisões, muitas possivelmente empíricas, isto é, baseadas no senso comum ou na experiência do responsável. É difícil selecionar, elaborar ou combinar os elementos das diversas metodologias existentes de forma a estabelecer a metodologia do projeto, portanto é necessária a pesquisa de abordagens sistemáticas que auxiliem o gerente de projeto ou o projetista de processos ao executar estas tarefas.

Mesmo com todo o esforço em metodologias e processos de desenvolvimento, muitos projetos de *software* ainda falham em atingir os seus objetivos, terminando com atraso, acima do custo previsto, ou com um usuário insatisfeito. Alguns projetos falham inteiramente ou são cancelados antes mesmo de produzirem qualquer resultado. Uma explicação para o alto índice de falhas é que os gerentes de projeto não estão agindo para avaliar e gerenciar os riscos envolvidos nestes projetos [KEI 98]. Gerenciamento de riscos é uma coleção de métodos os quais buscam evitar, minimizar ou reduzir os efeitos da falha de um projeto [ADD 2002].

Gerência de Riscos de *Software* (GRS) é uma abordagem que organiza as ações de tratamento de riscos em projetos de *software* em uma coleção de princípios e técnicas para analisar, preparar ações preventivas, tomar medidas corretivas e controlar os riscos de um projeto de desenvolvimento de *software* [FON 2004]. GRS sugere medidas para prevenir os riscos de afetarem o projeto, ou para reduzir seu impacto no caso de isto ocorrer. Deve ser considerado um componente essencial dos processos de gerenciamento de projetos [ADD 2002]. O dicionário Webster define risco como a “possibilidade de perda ou ferimento”. Esta definição implica em um conceito fundamental do risco: a exposição do risco [BOE 2002], algumas vezes chamada de “impacto do risco” ou “fator de risco” [BOE 91].

O gerenciamento de riscos envolve a avaliação do risco e o controle do risco [BOE 2002]. A avaliação do risco pode ser subdividida em identificação dos riscos, análise dos riscos, e priorização dos riscos [BOE 91]. A identificação dos riscos resulta em uma lista dos possíveis riscos de um projeto. A análise é uma estimativa da probabilidade de ocorrer e das conseqüências de cada um dos riscos que foram

identificados [HAL 98]. Na priorização dos riscos, os riscos identificados são ordenados por sua importância [BOE 91]. Já o controle do risco é o processo de elaboração e implementação de planos de resolução de riscos, monitoramento do status de cada risco, e desenvolvimento e documentação de estratégias [HAL 98]. Resolução de um risco é a sua eliminação ou minimização, ao se executar ações descritas no plano e gerenciamento de riscos [BOE 91].

A abordagem apresentada neste trabalho, chamada PMT – *Pattern-based Methodology Tailoring*, é dirigida por riscos e seleciona construções metodológicas que sejam apropriadas a um determinado projeto, na forma de uma linguagem de padrões organizacionais. Conforme foi descrito no primeiro capítulo, um padrão descreve a parte essencial da solução para um problema o qual é recorrente em um determinado contexto. Padrões organizacionais e processuais capturam as práticas de sucesso no gerenciamento de projetos de desenvolvimento de *software* [DEV 2002], e podem ser utilizados para definir o processo de desenvolvimento de uma nova organização, ou então melhorar o processo de uma organização já existente. Uma linguagem de padrões é uma coleção de padrões inter-relacionados, os quais são aplicados uns sobre os resultados prévios dos outros de forma a gerar um sistema [COP 96]. Em PMT, utilizam-se padrões organizacionais como forma de resolver os riscos de projetos de *software*.

Há diversos padrões e linguagens de padrões organizacionais documentados, como os padrões de Coplien [COP 95] [COP 2004], a metodologia Scrum descrita na forma de padrões [BEE 99], a linguagem de padrões para testes de DeLano [DEL 2004], os padrões para a reengenharia de sistemas de Stevens [STE 98] e os padrões para organização de times de Harrison [HAR 96a]. Este trabalho define formas de estruturação de um repositório de padrões, através da classificação e da associação entre os padrões. Regras de resolução de riscos são utilizadas para associar cada padrão com os riscos que estes resolvem. Cada regra é associada a um determinado contexto de projeto. O contexto de criticalidade do projeto é estabelecido neste trabalho de uma forma similar aos trabalhos de Alistair Cockburn [COC 2000] e Barry Boehm [BOE 2003] – descritos no capítulo 4, analisando-se três critérios: a criticalidade de defeitos (a possibilidade de perda causada pela ocorrência de um defeito no *software*), o número de pessoas envolvidas no projeto, e o nível de habilidade do time de desenvolvimento. Outros critérios poderiam ser definidos, como as prioridades do projeto em termos de produtividade e tolerância, ou em termos de atendimento de critérios legais [COC 2001b], mas os três critérios foram selecionados por serem mais representativos ao se balancear entre as metodologias ágeis e dirigidos a planos [BOE 2003].

A abordagem PMT é suportada por uma ferramenta, que foi desenvolvida para a seleção sistemática de padrões pelo gerente de projeto. O gerente de projeto cria sua própria linguagem de padrões ao selecionar a partir do repositório os padrões que são mais apropriados, e é auxiliado pelo mecanismo de seleção. O sistema de seleção utiliza o repositório completo de padrões e uma lista priorizada de riscos do projeto como sua entrada. A lista de riscos é obtida através da identificação dos riscos, da análise de exposição dos riscos, e da priorização dos riscos. O resultado da seleção é uma lista de padrões sugeridos que podem ser aplicados como ações para prevenir ou minimizar os riscos que foram identificados. O gerente de projeto escolhe alguns dos riscos sugeridos e os adiciona a linguagem de padrões do projeto. A linguagem de padrões do projeto é navegada pelos membros do time do projeto, que a utilizam

como uma fonte de aprendizado para resolver e prevenir os riscos do projeto de *software*.

Na seção 6.2 será analisado de que forma a abordagem se compara a alguns trabalhos relacionados que foram previamente descritos nos capítulos anteriores. Na seção 6.3 será descrito como que o repositório de padrões é estruturado. Na seção 6.4, a abordagem sistemática para a análise de riscos de um projeto será explicada. A seção 6.5 explica o mecanismo sistemático de seleção.

7.2 Trabalhos relacionados

Conforme descrito no primeiro capítulo, James Coplien disponibilizou em seu *web site* uma extensa linguagem de padrões organizacionais [COP 2004]. A estrutura navegacional da linguagem de padrões organizacionais é apresentada em um formato de grafo acíclico dirigido. Cada nodo do grafo representa um padrão, enquanto que os arcos representam as referências de dependência entre os padrões. Ao clicar em um nodo, o usuário navega para a descrição textual do padrão, a qual também inclui referências (*links*) de hipertexto para os padrões relacionados. Os padrões incluem descrições de processos como *Early and Regular Delivery* (entrega cedo e freqüente) e *Scenarios Define Problem* (o problema é definido por cenários), assim como descrições de papéis como *Architect Controls Product* (o arquiteto controla o produto) e *Mercenary Analyst* (analista mercenário). Técnicas e valores do time como *Code Ownership* (propriedade de código), *Compensate Sucess* (compense o sucesso) e *Public Character* (personalidade pública) também estão incluídos na linguagem de padrões. Para auxiliar os usuários a navegar na linguagem, os padrões são coloridos, embora não fique claro qual o critério de classificação que foi utilizado. Os padrões de Coplien capturam melhoras práticas recorrentes que foram aplicadas em sua organização. Alguns de seus padrões podem ser reutilizados para atacar riscos em outros projetos de software, mas por causa da extensão da linguagem de padrões é um trabalho duro selecionar os padrões apropriados, e esta seleção precisa ser feita através de decisões empíricas do gerente de projeto ou do projetista de processos.

O trabalho de Vasconcelos e Werner [VAS 98] utiliza padrões de processos para compor uma base de conhecimento sobre processos de *software*. Sua abordagem é suportada por uma ferramenta chamada Memphis – um ambiente de desenvolvimento de *software* baseado em reuso – a qual auxilia a se considerar diferentes alternativas quando os processos são compostos a partir de fragmentos descritos pelos padrões. No entanto, a ferramenta não auxilia o projetista com as decisões que este precisa tomar ao escolher entre diferentes alternativas de composição.

A abordagem Open (Processo, Ambiente e Notação Orientados a Objeto) [HEN 99], estudada na seção 2.3, utiliza um meta-modelo de processos a partir do qual um processo específico para um projeto pode ser instanciado. Permite um alto grau de flexibilidade para o usuário do processo, o qual precisa tomar várias decisões no processo de instanciação.

RUP (Processo Unificado da Rational) [JAC 2000] é um produto comercial, o qual pode ser considerado uma instância completa do Processo Unificado de Desenvolvimento de *Software* [JAC 99], incluindo material disponibilizado em HTML na forma de um *web site*. É um processo pré-configurado, portanto é possível modificar somente algumas partes, como expandir, modificar ou remover etapas de

atividades específicas, devendo ser considerado uma metodologia customizável ao invés de um arcabouço (*framework*) metodológico [HEN 99].

Ambas as abordagens Open e RUP não auxiliam com as decisões que um projetista de processos precisa tomar ao desenhar um processo específico. Ou seja, estas decisões, em maior ou menor grau, precisam ser realizadas de forma empírica pelo projetista de processos. Este trabalho procura auxiliar o projetista de processos em suas decisões, sugerindo práticas de metodologia, na forma de padrões, que podem ser adotadas em um projeto específico de desenvolvimento. A sugestão de padrões é feita através do mecanismo sistemática de seleção, que se baseia em regras de resolução de riscos, conforme será discutido na seção 6.5.

No capítulo 5, foram descritos dois trabalhos que procuram auxiliar nas decisões necessárias para a seleção de uma metodologia adequada a um projeto. Alistair CockBurn [COC 2000] propõe um arcabouço (*framework*) de seleção para que esta escolha seja realizada. O autor afirma que é apropriada e necessária a existência de múltiplas metodologias. A metodologia adequada pode ser escolhida através de duas dimensões: o tamanho da equipe do projeto e a criticalidade do sistema. Boehm e Turner [BOE 2003] estendem o arcabouço de Cockburn e utilizam critérios de risco de forma a balancear entre as metodologias ágeis e dirigidas por planos. Este trabalho utiliza um arcabouço semelhante de forma a selecionar padrões apropriados ao contexto de um determinado projeto.

7.3 Estruturando um repositório de padrões organizacionais

Esta seção descreve como estabelecer um repositório de padrões organizacionais. O repositório age como uma base de conhecimento sobre metodologias de desenvolvimento de *software*. Os padrões são selecionados a partir do repositório de forma a resolverem riscos de um projeto de desenvolvimento de software. O repositório considera alternativas de soluções para um mesmo problema.

Em primeiro lugar, é necessário que os padrões sejam descritos textualmente utilizando um formato comum. PMT utiliza um formato que é baseado no trabalho de James Coplien [COP 96] – descrito no primeiro capítulo, reforçando a utilização de campos separados nas descrições textuais. Cada padrão é descrito pelos seguintes campos: *Nome, Classificação, Problema, Contexto, Forças, Solução, Relações, Justificativa, e Fonte*. O campo de solução pode incluir um diagrama no caso de isto auxiliar na compreensão da solução do problema. O campo de fonte especifica o autor e a publicação a partir da qual o padrão foi obtido ou elaborado, como os padrões organizacionais da metodologia Scrum [BEE 99] e o *web site* de James Coplien [COP 2004].

Os padrões são classificados utilizando dois critérios, descritos abaixo. A classificação é útil para o gerente de projeto quando este utiliza o mecanismo de seleção. Ele pode pesquisar por padrões em todas as categorias, ou então pode restringir a pesquisa a uma categoria específica na qual ele está necessitando de ajuda.

- **Disciplina de processo**: os padrões são classificados de acordo com a disciplina de processo em que se enquadram. As disciplinas de processo que são utilizadas para a classificação neste trabalho correspondem às disciplinas do Processo Unificado da Rational (RUP) [JAC 2000], porque é um processo bem conhecido e razoavelmente completo [MAN 2003]: *Business Modeling* (Modelagem de Negócios), *Requirements* (Requisitos), *Analysis and Design* (Análise e Projeto), *Implementation* (Implementação),

Test (Testes), *Deployment* (Liberação), *Configuration and Change Management* (Gerenciamento de Configuração e de Mudanças), *Project Management* (Gerenciamento de Projetos), e *Environment* (Ambiente).

- **Mecanismo**: um padrão organizacional é ainda classificado pelo mecanismo que utiliza para descrever a solução para a organização do trabalho de desenvolvimento de *software*. Os mecanismos estabelecidos neste trabalho são: processos, papéis, técnicas ou valores do time. Um padrão de processo é um padrão organizacional que estrutura a sua solução como uma seqüência de atividades, i.e. um *workflow*. Um padrão de papel descreve um ou mais papéis em que as pessoas atuam ao desenvolverem *software*. Uma técnica descreve os procedimentos específicos que as pessoas utilizam para completar determinadas tarefas [COC 2001b]. Algumas técnicas são aplicáveis a uma única pessoa (projeto de uma classe ou de um caso de testes), enquanto outras se aplicam a grupos de pessoas (propriedade de código, sessões de planejamento). Os valores do time governam os elementos da metodologia. Um exemplo de um valor do time é a convenção de 40 horas semanais de Extreme Programming.

De forma a exemplificar a aplicação dos critérios de classificação, a tabela 7.1 mostra como alguns dos padrões organizacionais de Coplien [COP 2004] podem ser classificados. Embora possa haver outras possibilidades de classificação, os dois critérios descritos acima são utilizados porque representam o problema alvo do padrão (disciplina de processo) e o tipo de solução que descreve (mecanismo). A ferramenta apresentada na seção 6.5 permite ainda a configuração de um critério de classificação de usuário.

Este trabalho explora em maior profundidade a semântica do relacionamento entre os padrões. Tipicamente na literatura, uma linguagem de padrões relaciona a dependência entre padrões através de seu contexto, quando o contexto de aplicação de um padrão é o contexto resultante do padrão anterior [COP 96]. Permite a linguagem de padrões ser navegada através de referências (links) de hipertexto como no web site de James Coplien [COP 2004]. PMT utiliza relacionamentos tipados e uma notação baseada nos diagramas de dependência de James Martin [MAR 87] para representar uma linguagem de padrões. A tabela 7.1 mostra as possibilidades de relacionamentos entre padrões utilizando-se a abordagem PMT.

Tabela 7.1: Exemplo de padrões organizacionais

Padrão organizacional	**Disciplina de processo**	**Mecanismo**
SizeTheSchedule	Project Management	Process
EarlyAndRegularDelivery	Project Management	Process
HolisticDiversity	Project Management	Role
ArchitectAlsoImplements	Implementation	Role
ArchitectControlsProduct	Analysis and Design	Role
CodeOwnership	Implementation	Technique
IncrementalIntegration	Configuration Management	Process
PrivateVersioning	Configuration Management	Process
ScenariosDefineProblem	Analysis and Design	Process
SacrificeOnePerson	Project Management	Role
StandUpMeeting	Project Management	Technique
PublicCharacter	Environment	Team Values

Tabela 7.2: Tipos de relacionamentos entre os padrões

Tipos de relacionamentos entre os padrões	Símbolo
Dependência: um padrão deveria ser aplicado após o outro.	Seta.
Alternativa: ambos os padrões são diferentes alternativas de solução a um mesmo problema.	Círculo "ou".
Similaridade: os padrões são alternativas similares para a solução de um mesmo problema.	Círculo "ou" com linha tracejada.
Generalização / especialização: um padrão mais genérico, o qual pode ser aplicado a um contexto de problemas mais amplo, é especializado em uma solução mais específica.	Triângulo.
Trabalho em conjunto: os padrões deveriam ser aplicados em conjunto, sem necessidade de precedência.	Linha.
Composição: um padrão é decomposto em outros padrões os quais detalham alguns de seus aspectos.	Losango.

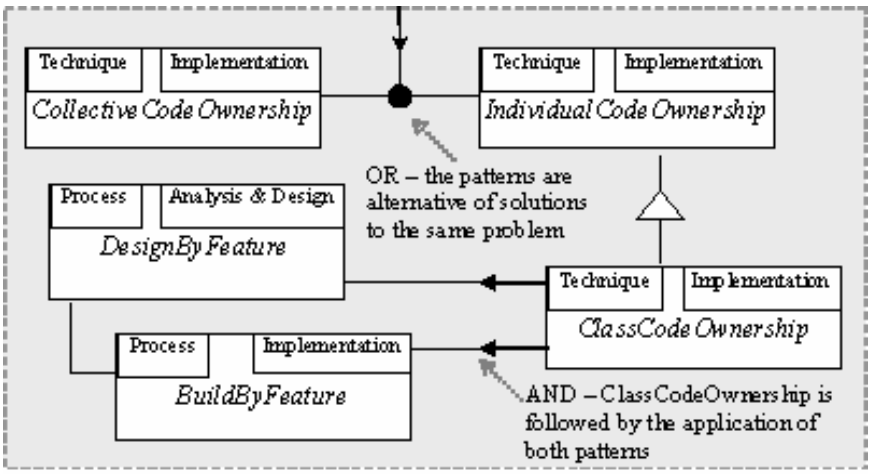


Figura 7.1: Um exemplo de um diagrama PMT aplicado a decidir uma técnica para propriedade de código

A figura 7.1 mostra um exemplo de um diagrama PMT, o qual explora a decisão de adotar uma técnica de propriedade do código fonte em um projeto de desenvolvimento de software. Os padrões foram elaborados a partir de duas metodologias publicadas: Extreme Programming (XP) [BEC 99] e Feature Driven Development (FDD) [PAL 2002] – estudadas em maior detalhe em [HAR 2003]. Duas alternativas são apresentadas: CollectiveCodeOwnership (propriedade coletiva do código-fonte) – ou seja, qualquer programador pode modificar qualquer parte do código a todo momento – e IndividualCodeOwnership (propriedade individual do código-fonte) – neste caso, cada programador é responsável por algumas partes do código. Uma especialização possível para a propriedade individual é

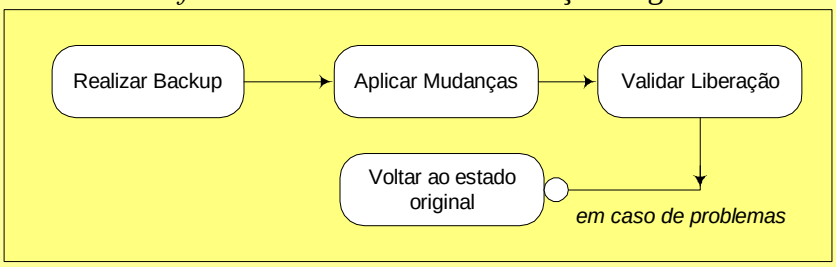
ClassCodeOwnership (propriedade de classes do código-fonte) – esta técnica define que um programador é responsável por algumas classes de código, por isso deve ser aplicada com uma linguagem de programação orientada a objetos.

Os autores da metodologia FDD afirmam que a técnica de propriedade de classes do código-fonte auxilia a metodologia a suportar grandes projetos, enquanto que a técnica de XP de propriedade coletiva do código-fonte somente funcionaria em projetos envolvendo pequenos times co-locados. Aplicar a primeira técnica, no entanto, não é tão simples porque novos problemas surgem. Como lidar com a situação em que um programador precisa completar a implementação de um método em uma classe A, mas antes é necessário criar um novo método na classe B, embora o programador não seja o responsável pela classe B? Uma possível solução é utilizar os processos de FDD de *DesignByFeature* (Projetar por característica) e *BuildByFeature* (Construir por característica). Ambos os processos trabalham em conjunto para resolver o problema de dependência que surge quando se aplica a propriedade de classes do código-fonte. O objetivo destes processos é o de incrementalmente implementar uma nova característica no sistema de *software*. No primeiro processo as mudanças em cada classe são projetadas e documentadas, enquanto que, no segundo processo, cada programador realiza as mudanças respectivas às suas próprias classes.

A tabela abaixo apresenta um exemplo de padrão organizacional descrito no formato da abordagem PMT: um padrão de processo para a liberação de novas versões e de modificações sobre um *software* o qual já está sendo utilizado.

Tabela 7.3: Exemplo de um padrão organizacional

Nome	***LiberaçãoSeguraFreqüente***
Categoria	Processo para gerência de configuração e de mudanças
Problema	Como realizar a liberação de uma nova versão de um *software* garantindo que erros durante este processo não comprometam os dados, nem a disponibilidade do sistema aos usuários? Com que freqüência realizar as liberações?
Contexto	Este padrão foi elaborado a partir da experiência do autor em um projeto de criticalidade dos defeitos nível 2, habilidade da equipe nível 4, e tamanho nível 2. A prioridade primária do projeto era a produtividade (prazo) e a secundária o custo.
Forças	1. Os usuários utilizam o *software* com freqüência e querem o menor tempo possível de indisponibilidade. 2. Os procedimentos de mudanças a serem realizadas no ambiente de produção não podem ser previamente testados. 3. Manter uma versão em produção muito diferente da versão em desenvolvimento gera um custo extra de gerência de configuração. 4. As mudanças geram um impacto de aprendizado nos usuários.
Solução	Faça liberações freqüentes de versão, desta forma diminuindo a quantidade de mudanças a serem realizadas a cada vez. Isto minimizará sua complexidade, seu risco, e o seu impacto para os

	usuários do *software*. Uma liberação deve iniciar com a criação de uma cópia *backup* dos dados, esquemas e programas que serão modificados. Após a sua realização, deve-se revisar as funcionalidades afetadas. Caso a liberação não seja bem-sucedida, deve ser possível retornar rapidamente o *software* e os seus dados a situação original. 
Relações	A aplicação deste padrão assume que *TestesRegressao* (*link*) estão sendo aplicados. Uma solução complementar é utilizar um *AmbientePreProducao* (*link*) para validar os procedimentos das mudanças antes de efetivá-las.
Justificativa	A liberação de uma nova versão de um *software* o qual está sendo utilizado pelos usuários é uma tarefa complexa e arriscada, que pode envolver mudanças sobre esquemas de dados, dados, e programas. O risco inerente de uma liberação pode levar os gerentes a adiaram ao máximo sua realização. Esta decisão pode ter um efeito contrário ao desejado, pois aumenta o número de mudanças necessárias, aumentando a complexidade da liberação e o impacto sobre os usuários.

A próxima seção apresenta o mecanismo sistemático para identificar e analisar os riscos de um projeto de *software*. A seção 6.5, por sua vez, mostra como o mecanismo de seleção funciona. Para que este mecanismo de seleção funcione, o repositório de padrões precisa relacionar os padrões com os riscos que estes se propõe a resolver. Isto é obtido através de regras de resolução de riscos. Uma regra de resolução de risco relaciona riscos com os padrões que podem ser aplicados para minimizar ou resolver o risco. Cada regra é associada com o contexto de projeto no qual funciona melhor, e tem um fator de eficácia. Um contexto de projeto é estabelecido em termos de três critérios, como melhor explicado na próxima seção: a criticalidade do sistema e o tamanho da equipe de Cockburn [COC 2000], e o nível de habilidade do time conforme sugerido por Boehm [BOE 2003]. O fator de eficácia (AF – *accuracy factor*) é um valor de 0 a 10 que é utilizado para ajustar o mecanismo de seleção. Pode ser ajustado pelos usuários da ferramenta após estes terem experimentado com os padrões. Exemplos de regras de resolução de riscos são apresentadas na próxima seção.

7.4 Analisando os riscos e o contexto de um projeto

Vários estudos foram conduzidos de forma a identificar os principais riscos de projetos de *software*. Alguns destes estudos consideram somente os riscos que o gerente de projeto tem controle, enquanto outros consideram todos os riscos. Lisandra

Fontoura [FON 2004] realizou uma comparação e uma compilação dos riscos identificados por Keil [KEI 98], Boehm [BOE 91], Addison [ADD 2002], e o CMMI [SEI 2002], de forma a obter uma lista de riscos contendo os principais riscos de projetos de *software*.

A lista de riscos para projetos de *software* foi classificada utilizando um critério que foi adaptado do arcabouço de classificação de riscos apresentado por Keil [KEI 98] – estudado na seção 4.1. Utilizando este critério, os riscos podem ser:

- **Riscos do cliente**: estes riscos são originados pelo envolvimento do cliente com o sistema que está sendo desenvolvido.
- **Riscos de requisitos**: estabelecer os requisitos do *software* é uma das partes mais importantes do processo, porque se o *software* não endereçar os seus requisitos, não atenderá as necessidades do cliente.
- **Riscos de planejamento**: estes riscos monitoram o planejamento inicial do projeto de *software*.
- **Riscos de execução**: os riscos de execução descrevem situações que podem ocorrer enquanto o *software* está sendo desenvolvido, e podem causar impacto substancial na qualidade final do *software*.

Tabela 7.4: O checklist de riscos

Cliente	**Requisitos**
Falta de envolvimento dos usuários.	Não-entendimento dos requisitos.
Falha em obter compromisso dos usuários.	Escopo/objetivos não claros.
Falha na gerência das expectativas dos usuários.	Instabilidade de requisitos.
Conflito entre os departamentos dos usuários.	
Planejamento	**Execução**
Falta de compromisso da gerência sênior com o projeto.	Introdução de novas tecnologias.
Falta de conhecimento/habilidade pela equipe.	Requisitos criados pelos desenvolvedores (*gold plating*).
Equipe insuficiente ou inadequada.	Desenvolvimento errado das funções ou interfaces.
Cronograma e orçamento não realistas.	Subcontratação.
Falta de uma metodologia de projeto.	Utilização de recursos e performance do sistema.
	Projeto (desenho) inviável.

A compilação de riscos é apresentada na tabela 7.4. Os riscos de gerenciamento de projeto foram separados em riscos de planejamento e riscos de execução, porque alguns destes devem ser resolvidos no início do projeto, enquanto outros devem ser monitorados e resolvidos durante a execução do projeto.

A lista de riscos da tabela 7.4 é utilizada neste trabalho como um *checklist* dos riscos que mais freqüentemente ocorrem, para se identificar os riscos de projetos de *software*. O processo de identificação dos riscos não deve ficar limitado a esta lista de riscos, contudo, e deve considerar muito mais riscos. Para realizar a identificação dos riscos, os membros do time devem ser entrevistados e sessões de grupo devem ser efetuadas com o time do projeto e com outras pessoas envolvidas com o projeto, de forma a se identificar todas as coisas que podem ocorrer de errado no projeto [ADD 2002]. O resultado pode ser uma longa lista de riscos que deve ser organizada nas quatro categorias de Cliente, Requisitos, Planejamento ou Execução. A classificação de riscos pode ser utilizada com a classificação dos padrões como parâmetros para o sistema de seleção de padrões, como será explicado na seção 6.5.

Outra forma que também pode ser utilizada para a identificação dos riscos é utilizar-se dos principais problemas que ocorreram em projetos passados da organização, como sugerido por DeMarco e Lister [DEM 2003]. Segundo os autores: "*Os problemas de ontem são os riscos de amanhã*". Portanto, uma forma de popular a lista de riscos do projeto é através da utilização dos resultados de análises *post-mortem*.

Após a identificação dos riscos, estes devem ser analisados e priorizados utilizando-se a técnica de exposição de risco [FON 2004]. A exposição do risco, também chamada de impacto do risco ou fator de risco, é o produto da probabilidade de um resultado não satisfatório ocorrer, e a perda associada com este resultado não satisfatório [BOE 2002]. Este trabalho utiliza uma nota de 0 a 10 para que a probabilidade e a perda de cada risco sejam estimadas. Coppendale [COP 95] sugere algumas regras simples para que se determine a probabilidade e a perda de um risco:

- **Probabilidade**: 0 (zero) representa a probabilidade de até 5% de um risco ocorrer. 5 representa a probabilidade de cerca de 50%, enquanto 10 representa a probabilidade de 95% ou mais.
- **Perda**: 0 (zero) representa nenhum incremento em termos do prazo ou do custo do projeto, enquanto que 10 representa um impacto bastante significativo no prazo ou no custo do projeto, como uma acréscimo de 100%.

Quando cada um dos riscos que foram identificados for quantificado, a lista de riscos é ordenada pela exposição do risco. O gerente de projeto deve definir o critério para a priorização dos riscos mais importantes, estabelecendo o Fator de Priorização de Riscos (RPF – *Risk Prioritization Factor*). Isto significa que somente os riscos cuja exposição do risco seja maior ou igual ao fator RPF irão receber atenção imediata. O sistema de seleção sistemático descrito na próxima seção permite ao gerente de projeto configurar o fator RPF.

Após os riscos terem sido priorizados, ações preventivas devem ser selecionadas e elaboradas [FON 2004]. A abordagem PMT seleciona padrões organizacionais a partir de um repositório de padrões, como ações que são sugeridas para o gerente de projeto aplicar, de forma a prevenir, resolver ou minimizar cada um dos riscos priorizados. A seleção de padrões é dirigida pelo resultado da análise de riscos, com a aplicação de regras de resolução de riscos. PMT também utiliza uma análise do contexto do projeto, porque é importante que este seja considerado quando se seleciona uma metodologia, como discutido por Cockburn [COC 2000] e Boehm [BOE 2003]. O contexto do projeto é identificado em termos de três critérios.

- **Criticalidade dos defeitos**: a possibilidade de perda associada com a ocorrência de um defeito. Varia de uma perda de conforto (1) até a perda de muitas vidas (5). De acordo com Cockburn [COC 2000], quanto mais crítico um sistema, maior a densidade necessária na metodologia, i.e. os artefatos produzidos precisam ser detalhados, revisados e verificados uns em relação aos outros com maior cuidado.
- **Tamanho da equipe**: o número de pessoas envolvidas no projeto também é um fator importante considerado por Cockburn [COC 2000]**.** Quanto maior o time do projeto, maior a metodologia que é necessária, i.e. mais documentos intermediários precisam ser produzidos para coordenar o trabalho.
- **Habilidade da equipe**: Barry Boehm [BOE 2003] estende a classificação das pessoas em níveis de entendimento de Alistair Cockburn [COC 2001b], e a utiliza como um importante fator para o balanceamento entre metodologias ágeis, como Extreme Programming [BEC 99], e metodologias guiadas por planos, como as que seguem o modelo CMM [SEI 95]. A classificação é: -*1. Pessoas que não são capazes ou não estão dispostas a colaborar; 1B. Pessoas que são capazes de contribuir em um ambiente guiado por planos, ao executar tarefas simples seguindo passos de procedimentos, mas não conseguem contribuir em um time ágil; 1A. Pessoas que são capazes de contribuir tanto em um time ágil quanto um time guiado por planos; 2. Pessoas que podem liderar um pequeno time de desenvolvedores e adaptar uma metodologia para atender uma nova situação com precedente; 3. Pessoas que podem revisar uma metodologia, quebrando suas regras para atender a uma nova situação sem precedente.* Boehm utiliza o percentual de pessoas de nível 1B como um fator para balancear a metodologia do projeto. PMT utiliza uma medida da habilidade do time ao considerar a percentagem de pessoas classificadas em todos os níveis de entendimento, de 1B a 3. Uma nota de 1 para a habilidade da equipe significa um time altamente habilidoso, com a maior parte das pessoas nos níveis 2 e 3, enquanto que um nota de 5 significa que a maior parte das pessoas do time está no nível 1B.

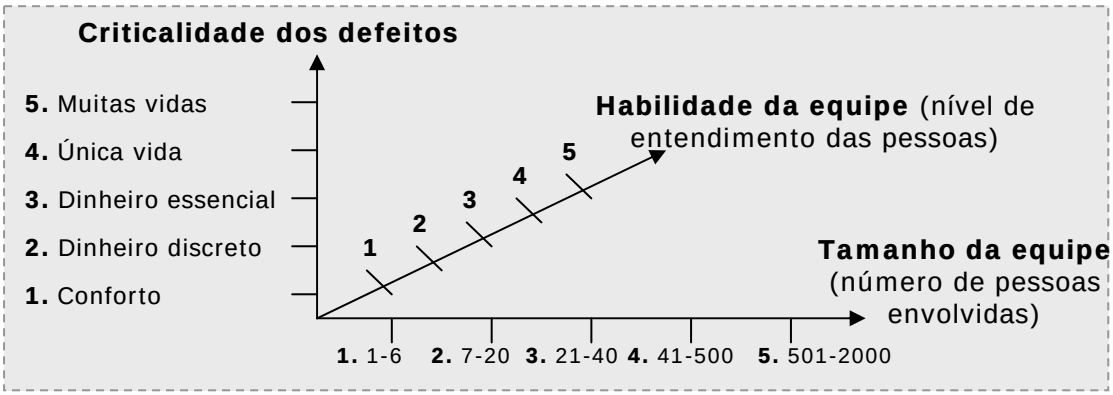


Figura 7.2: Analisando o projeto em termos de criticalidade, tamanho e habilidade da equipe

O contexto de criticalidade do projeto é estabelecido em PMT ao se identificar sua posição nos eixos da figura 7.2. Esta análise auxilia a abordagem a selecionar padrões que são mais prováveis de terem sucesso em um ambiente ágil ou em um ambiente guiado por planos. Se o contexto do projeto estiver localizado próximo da origem dos três eixos, o que significa uma equipe pequena altamente habilidosa trabalhando em um projeto de baixa criticalidade, os padrões mais apropriados irão tipicamente representar práticas das metodologias ágeis. Para um projeto com um alto nível de criticalidade dos defeitos, e um time com habilidade abaixo da média, o sistema de seleção irá escolher padrões organizacionais que reflitam práticas de metodologias guiadas por planos. Para projetos que não se localizam em nenhum dos extremos opostos no espaço tridimensional, a abordagem auxilia a balancear a metodologia ao selecionar os padrões que sejam mais adequados ao contexto de criticalidade do projeto e que enderecem seus mais importante riscos.

Na próxima seção o sistema de seleção de padrões será examinado. A seleção é baseada em regras de resolução de riscos, as quais precisam ser armazenadas no repositório junto com os padrões e o *checklist* de riscos. Cada regra associa um ou mais padrões com um ou mais riscos que pretendem prevenir, resolver ou minimizar. Uma regra também está associada com o contexto de criticalidade do projeto no qual funciona melhor. O contexto é um ponto marcado no espaço tridimensional da figura 7.2. Uma regra também tem um fator de eficácia (AF –*Accuracy Factor*) o qual mede o quanto que os padrões associados à regra auxiliam a resolver os riscos que estão associados à mesma regra, assim auxiliando a ajustar o mecanismo de pesquisa de padrões. Um fator de 0 (zero) significa que os padrões não auxiliam em nada, enquanto que um fator de 10 significa que os padrões podem ser aplicados para totalmente eliminar os riscos correspondentes.

A tabela 7.5 apresenta exemplos de regras de resolução de riscos. Os exemplos de regras e de padrões foram elaborados a partir de práticas descritas em metodologias existentes, a partir da experiência do autor, e a partir de outros trabalhos [FON 2004].

Diferentes regras podem endereçar o mesmo risco em contextos diferentes (a coluna PC). Como um exemplo, a regra de número 1 endereça o risco de falta de compromisso da gerência sênior com o projeto, no contexto de um projeto com características ágeis: baixa criticalidade dos defeitos, e um time pequeno e hábil. A participação da gerência sênior durante reuniões de planejamento de iterações pode ser utilizada para minimizar seu risco [FON 2004], como na técnica de *PlanningGame* (Jogo de Planejamento) de Extreme Programming [BEC 2000] e na técnica de *SprintPlanningMeeting* (“Reunião de planejamento de Sprint”) da metodologia Scrum [SCH 2002]. A regra de número 2 endereça o mesmo risco em um contexto diferente – um projeto com características de guiado por planos – com a prática do CMM de a gerência sênior revisar os compromissos do projeto junto com o gerente de projeto [SEI 95].

Tabela 7.5: Exemplo de regras de resolução de riscos. Cada regra é aplicada em um contexto de projeto (PC) e tem um fator de eficácia (AF).

#	Comentários	Criticalidade dos defeitos	Tamanho da equipe	Habilidade da equipe
A	Características ágeis	2 (Dinheiro discreto)	1 (1-6)	2 (Alta)
B	Características guiadas por planos	5 (Muitas vidas)	4 (41-500)	4 (Baixa)

#	Risco(s)	PC	Padrão(ões)	AF
1	Falta de compromisso da gerência sênior com o projeto.	A	*SprintPlanningMeeting* *PlanningGame*	7
2	Falta de compromisso da gerência sênior com o projeto.	B	*SeniorManagementReview*	5
3	Instabilidade dos requisitos; Não entendimentos dos requisitos; Conflitos entre os departamentos dos usuários.	A	*EarlyAndRegularDelivery* *ScenariosDefineProblem*	8
4	Não entendimento dos requisitos.	B	*ScenariosDefineProblem* *RequirementsAreValidated* *DocumentedChangeMgmt*	6
5	Requisitos criados pelos desenvolvedores (*gold plating*).	A	*EarlyAndRegularDelivery* *ScenariosDefineProblem*	6

7.5 O Mecanismo de Seleção de Padrões Organizacionais

Esta seção descreve o mecanismo de seleção sistemático implementado na ferramenta PMT-Tool. A ferramenta registra e mantém o repositório de padrões, com as características descritas na seção 6.3. Ela também permite a identificação de riscos, a análise da exposição do risco a priorização dos riscos, além da análise do contexto do projeto, conforme descrito na seção 6.4. A ferramenta utiliza o repositório, os resultados da análise do projeto, e as regras de resolução de riscos, de forma a realizar a seleção de padrões, resultando em uma lista de padrões organizacionais que são sugeridos ao gerente de projeto. O gerente de projeto pode utilizar os padrões selecionados para endereçar riscos do projeto. O mecanismo de seleção é complementado com a navegação exploratória no repositório de padrões.

A ferramenta PMT-Tool é implementada em Java utilizando um banco de dados MySQL para armanezar os seus dados. O diagrama orientado a objetos da figura 7.3, o qual representa os principais conceitos registrados pela ferramenta, é mapeado para o banco de dados utilizando uma biblioteca de mapeamento objeto-relacional [HIB 2004]. A interface com o usuário da ferramenta utiliza o paradigma *web*, facilitando a

navegação exploratória dos padrões organizacionais, porque os relacionamentos entre os padrões são navegados através de referências (*links*) de hipertexto.

O sistema de seleção assume que o repositório de padrões já está armazenado (a classe *BasePatternLanguage*). Os padrões organizacionais (a classe *Pattern*) na linguagem-base de padrões (o repositório) têm uma descrição textual, uma classificação, e são relacionado uns com os outros. Há também: uma lista de riscos (a classe *ProjectRisk*), uma lista com diferentes combinações de contextos de projeto (a classe *ProjectContext*), e as regras de resolução de riscos (a classe *SelectionRule*).

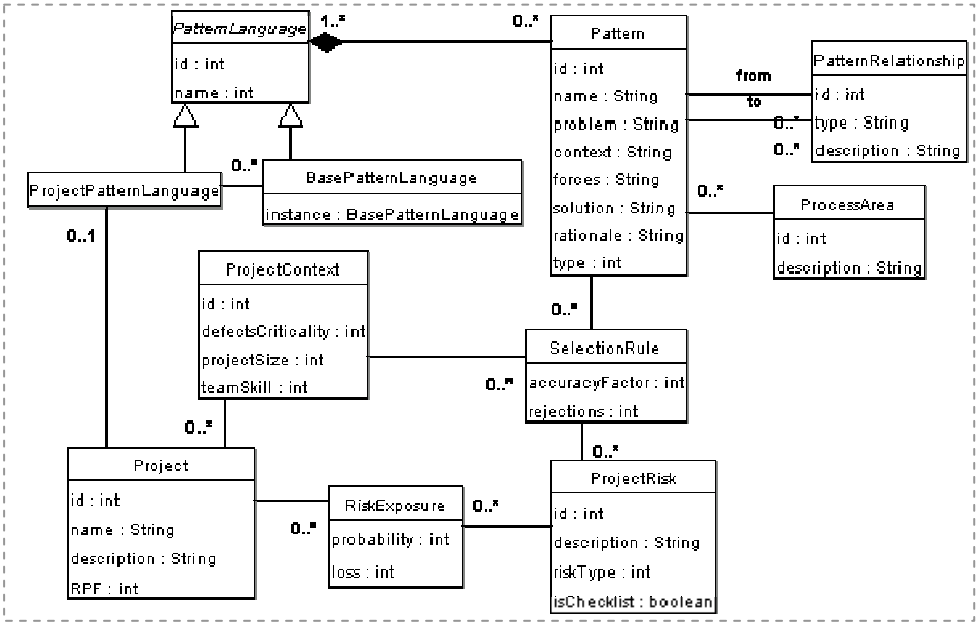


Figura 7.3: O diagrama conceitual da ferramenta

Uma sessão de seleção inicia com o gerente de projeto registrando um novo projeto na ferramenta (a classe *Project*), ou utilizado um projeto já previamente armazenado. Para um novo projeto, o seu contexto de criticalidade deve ser estabelecido e os riscos precisam ser identificados e priorizados utilizando a exposição do risco e o fator de priorização do risco (RPF), como explicado na seção 6.4. As informações de riscos e de contexto podem ser atualizadas para um projeto existente, antes de uma nova sessão de seleção ser iniciada.

O sistema de seleção é configurado com parâmetros. O fator de priorização de riscos (RPF) é obrigatório. Conforme explicado na seção 6.4, os riscos do projeto são identificados e são ordenados pela exposição do risco, mas somente os riscos com valor de exposição maior que o RPF são considerados pelo mecanismo de seleção. Os outros parâmetros utilizam a classificação dos padrões e dos riscos, conforme descrito nas seções 6.3 e 6.4, e são opcionais. Os padrões são classificados pela disciplina de processos que endereçam, pelo tipo de mecanismo que sua solução utiliza, e possivelmente por um critério de classificação do usuário. O tipo do risco também pode ser utilizado como um parâmetro para o mecanismo de seleção. Os parâmetros de classificação ajudam o gerente de projeto a restringir a pesquisa para resolver um problema em uma área específica onde ele está necessitando de aconselhamento.

O primeiro passo do algoritmo de seleção dos padrões é avaliar cada uma das regras de resolução de riscos que estão associadas aos riscos priorizados. Para cada regra, dois fatores são avaliados:

- **Exposição do risco**: corresponde à média do valor de exposição do risco para os riscos associados com a regra. A exposição do risco é um valor de 0 a 100, pois é a probabilidade (0-10) multiplicada pela perda (0-10).
- **Proximidade do contexto**: a regra está associada ao contexto de projeto no qual funciona melhor, mas este contexto pode ser diferente do contexto do projeto-alvo. Um contexto de projeto é avaliado utilizando três critérios (cada um sendo um valor de 1-5), como explicado na seção 6.4: a criticalidade do sistema, o tamanho e a habilidade da equipe. A proximidade do contexto avalia o quão próximo o contexto da regra está do contexto do projeto. A proximidade é calculada utilizando a fórmula $p = 100 - (\sqrt[2]{(d1^2+d2^2+d3^2)} * 100/8)$, na qual $d1$, $d2$, e $d3$ são as distâncias do contexto da regra ao contexto do projeto em cada um dos eixos, variando de 0 a 4, porque o valor mínimo para cada critério é 1 e o valor máximo é 5. A fórmula utiliza o teorema de Pythagoras para calcular a distância no espaço tridimensional, e converte o resultado para um valor de 0 a 100. O resultado é arredondado por baixo para o número inteiro mais próximo. Contextos idênticos (distância zero) resultam em um valor de proximidade igual a 100, enquanto que a distância máxima ($d1=d2=d3=4$) resulta em proximidade igual a zero.

Tabela 7.6: Calculando a pontuação resultante para cada regra de resolução de riscos

Regra#	Exposição (0-100)	Proximidade (0-100)	AF (0-10)	Pontuação (0-100.000)
1	65	59	7	26.845
2	82	23	6	11.316
3	91	68	8	49.504

Os dois fatores (exposição e pontuação) são avaliados para cada regra, resultando um uma tabela de avaliação de regras, como exemplificado pela tabela 7.6. A pontuação resultante para cada regra é calculada ao se multiplicar o valor da exposição do risco, a proximidade do contexto, e o fator de eficácia da regra. A tabela resultante é ordenada pelas pontuações de cada uma das regras. O resultado-alvo, no entanto, não é uma lista de regras, mas uma lista de padrões ordenada pela relevância de sua aplicação a um determinado projeto. Portanto, o próximo passo do algoritmo de seleção é a criação de uma tabela de comparação de padrões.

Uma tabela de comparação de padrões lista todos os padrões os quais estão associados com ao menos uma das regras de resolução de riscos, e a pontuação resultante para cada um dos padrões. Para padrões que aparecem apenas uma vez na lista de regras, a pontuação para o padrão é a pontuação da regra correspondente. Quando um padrão aparece em mais de uma das regras aplicadas, a pontuação resultante para o padrão é a soma das pontuações das regras em que este aparece. A tabela de padrões resultante, como exemplificado pela tabela 7.7, é ordenada pela

coluna de pontuação e é apresentada ao gerente de projeto em ordem decrescente – as maiores pontuações aparecem em primeiro lugar.

Seguindo-se uma referência de hipertexto no nome do padrão, o usuário pode navegar para a descrição textual completa de cada padrão. Outra referência de hipertexto existe na coluna de pontuação, a qual detalha a informação sobre a pontuação do padrão correspondente, como quais regras que foram aplicadas, quais os riscos que o padrão pretende resolver, e como que a pontuação foi calculada.

Tabela 7.7: Um exemplo de uma lista sugerida de padrões para o gerente de projeto

Padrão	**Pontuação (0-100.000)**
EarlyAndRegularDelivery	89.023
SprintPlanningMeeting	49.302
ScenariosDefineProblem	11.983

Uma sessão de seleção é executada no contexto de um projeto de desenvolvimento de *software* específico. A ferramenta cria um espaço de trabalho (*workspace*) em separado, para que o gerente de projeto elabore uma linguagem de padrões organizacionais para o projeto. Este espaço de trabalho é chamado em PMT de linguagem de padrões do projeto, que é um sistema de padrões em separado. Isto auxilia o gerente de projeto a documentar a metodologia do projeto na forma de uma linguagem de padrões organizacionais. O repositório de padrões é chamado de a linguagem de padrões base. Após uma sessão de seleção, o gerente de projeto pode selecionar alguns dos padrões sugeridos pela ferramenta para que sejam incluídos na linguagem de padrões do projeto. Um relacionamento entre padrões também será transferido, no caso de ambos os padrões relacionados estarem incluídos na linguagem de padrões alvo.

Após o espaço de trabalho do projeto (a linguagem de padrões do projeto) ter sido preenchida com padrões, novas sessões de seleção podem ser executadas. O gerente de projeto pode experimentar com diferentes parâmetros para a seleção, como diminuir o fator de priorização de riscos (RPF) de forma a considerar mais riscos como alvo. Os padrões que já foram incluídos na linguagem de padrões do projeto não serão considerados novamente em novas sessões de seleção.

O mecanismo de seleção da ferramenta PMT-Tool é complementado com navegação exploratória no repositório de padrões. O gerente de projetos pode navegar no repositório utilizando referências de hipertexto para seguir os relacionamentos entre os padrões, e empiricamente selecionar padrões para inclusão na linguagem de padrões do projeto.

Outra possibilidade é a seleção de todos os padrões que pertencem a mesma fonte de padrões. Esta operação é útil quando o gerente de projeto quer utilizar todos os padrões de uma determinada metodologia como um começo, como os padrões de *Scrum* [BEE 99]. Após preencher a linguagem de padrões com os mesmos, o gerente de projeto pode complementar a metodologia ao endereçar os riscos ainda não tratados, através de uma sessão de seleção de padrões.

Quando o gerente de projeto acredita que completou a adaptação da linguagem de padrões do projeto, ele permitirá acesso a esta linguagem de padrões aos demais membros do projeto. A maior parte dos membros do projeto irá navegar apenas através dos padrões e dos relacionamentos que existem neste espaço de trabalho

separado para o projeto, e não irá necessitar saber da existência do repositório de padrões. Eles irão utilizar os padrões como fonte de aprendizado, de forma a resolver os problemas particulares que ocorrerem no projeto de desenvolvimento de *software*. O gerente de projeto irá continuar a monitorar os riscos do projeto e pode executar novas sessões de seleção, e publicar novos padrões na linguagem de padrões do projeto, no caso de isto se mostrar necessário.

Nas próximas seções do trabalho, o funcionamento da ferramenta PMT-Tool será detalhado, e serão apresentados exemplos mais detalhados de aplicação da abordagem.

7.6 Aplicando PMT

A aplicação da abordagem PMT enfatiza a tomada de decisões gerenciais no processo de adaptação da metodologia de desenvolvimento de software para um projeto. Consiste em uma seqüência de passos os quais são executados iterativamente, conforme mostra a figura 7.4. Estas etapas são executadas já assumindo que há uma base de conhecimento, a qual foi previamente cadastrada, na forma de padrões organizacionais, riscos e regras. Este trabalho é executado pelo projetista de processos, que é o responsável por definir a metodologia para o projeto. É importante que haja comunicação e cooperação entre o projetista de processos e o gerente do projeto. A metodologia deve ser formulada considerando também as prioridades do projeto, em termos de prazos e custos. Outras prioridades também podem ser consideradas, como a necessidade de atendimento a critérios legais, ou a modelos para a avaliação da qualidade de processos. Em muitos casos o projetista de processos pode ser o próprio gerente do projeto, o que é bastante recomendável para que haja sinergia entre as necessidades do projeto e a metodologia.

Na primeira etapa, são analisados os requisitos metodológicos do projeto, que correspondem neste trabalho ao contexto de criticalidade e aos riscos, conforme foi descrito na seção 6.4. Com base nos requisitos que foram definidos, é feita a seleção de uma lista de padrões candidatos. Os padrões são selecionados utilizando o mecanismo sistemático de seleção descrito na seção anterior, ou então através da navegação exploratória no repositório de padrões. A lista de padrões candidatos não representa diretamente os padrões que serão aplicados no projeto. Os padrões precisam ainda ser priorizados pelo projetista de processos, de forma que somente os mais importantes sejam primeiramente aplicados.

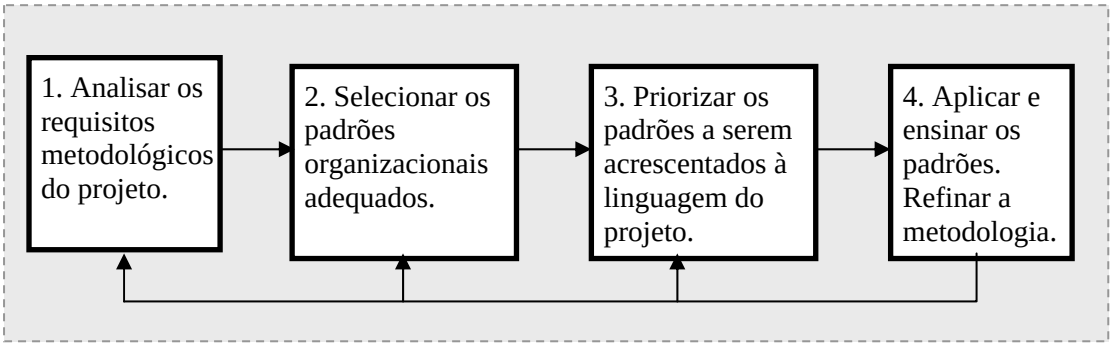


Figura 7.4: Etapas da aplicação da abordagem PMT

Os padrões candidatos priorizados na etapa 3 são efetivamente aplicados na etapa 4. Nesta última etapa, os padrões são ensinados aos membros da equipe de desenvolvimento de *software*. Isto pode ser feito de várias formas, principalmente: *a. através de consulta pela web ao site da ferramenta PMT-Tool (ver próxima seção). b. através de seminários internos à equipe de desenvolvimento. c. através do trabalho em conjunto dentro da equipe, em que um novo membro aprende com os membros mais antigos.*

Alguns padrões, ao serem aplicados, envolvem uma mudança cultural por parte da equipe do projeto. Outros, como os padrões de processos, são utilizados para desenhar processos específicos para o projeto, auxiliando o projetista do processo de forma semelhante ao apoio que um padrão de projeto [GAM 95] [DAN 2002] proporciona ao projetista de um diagrama de classes. Na última etapa, a metodologia pode ser refinada com descrições de procedimentos e com exemplos de documentos e diagramas. O conhecimento obtido através dos padrões pode ser customizado para necessidades específicas do projeto. Um exemplo desta última situação podemos considerar com a aplicação do padrão descrito na seção 6.3, *LiberaçãoSeguraFrequente*. Este padrão descreve um processo comum para ser aplicado na gerência de liberações de um sistema. O padrão descreve as atividades comuns, recorrentes, que ocorrem em muitos projetos de *software*. Um projeto específico, no entanto, terá as suas atividades particulares que precisam ser executadas durante uma liberação. Passos específicos do ambiente do projeto, como fazer logon com determinados usuários e senhas em determinadas máquinas, e executar determinados comandos, precisam ser descritos e acrescentados dentro do procedimento genérico. Desta forma, é produzido um documento descrevendo o procedimento de liberação específico do projeto.

Quando a gerência do projeto estiver satisfeita com a aplicação e o entendimento dos padrões por parte da equipe, pode então repetir quaisquer dos passos anteriores para selecionar novos padrões a serem aplicados.

A abordagem PMT foi pensada de forma a ser aplicada iterativamente. O desenvolvimento iterativo e incremental de software não é uma prática nova, como apontado por Larman e Basili [LAR 2003]. A prática vem sendo adotada, ainda que de diferentes formas, pelo menos a partir dos anos 70. A principal motivação para sua aplicação é evitar uma passada única e seqüencial de documentação, a qual guiará todo o restante do projeto. O desenvolvimento é feito de forma evolucionária, incorporando retroalimentação sobre os resultados que são parcialmente obtidos. Desta forma, se lida melhor com mudanças que ocorrem sobre os requisitos do software que está sendo desenvolvido. Da mesma forma, esta prática é adotada neste trabalho, mas como um meio de se lidar com mudanças nos requisitos metodológicos do projeto durante a sua execução. Considera-se que durante um mesmo projeto o tamanho e a habilidade da equipe podem variar, por exemplo, o que pode criar a necessidade de aplicação de novos padrões.

Outro motivo para a abordagem ser aplicada incrementalmente é a curva de aprendizado da equipe, que precisa absorver o conhecimento existente nos padrões, antes que novos padrões sejam aplicados. Por isso, deve-se iniciar com a aplicação dos padrões mais importantes. Conforme o time compreende e aplica o conhecimento de forma efetiva, novos padrões podem ser selecionados e priorizados para serem aplicados.

A abordagem PMT pode ser aplicada a um projeto já existente, mas neste caso deve-se acrescentar uma nova etapa no ciclo iterativo da figura 7.4: análise da

metodologia existente. A nova etapa deve ser realizada antes que se possa iniciar o trabalho (antes da etapa 1). É preciso que a metodologia existente seja compreendida na forma de uma linguagem de padrões. Os padrões desta linguagem podem existir no repositório PMT ou não. No segundo caso, precisarão ser cadastrados. Uma vez que a metodologia já existente seja compreendida como uma linguagem de padrões, deve-se prosseguir com o processo descrito nesta seção, analisando os requisitos metodológicos do projeto, e acrescentando padrões de forma a endereçar os riscos do projeto que ainda não estejam sendo endereçados.

7.6.1 Monitorando os riscos

Uma vez definida uma metodologia com base nos riscos e iniciado o projeto, os seus riscos devem ser monitorados. Com base no resultado do monitoramento dos riscos, a metodologia do projeto pode precisar ser ajustada de acordo com a nova realidade.

A seção 4 descreve alguns conceitos importantes no gerenciamento de riscos. Um risco é um problema em potencial o qual ainda não ocorreu. Existe um momento no qual o risco se transforma em um problema, e que precisa ser monitorado. Este momento é chamado de transição - ou materialização - do risco. Um risco é monitorado através de indicadores de transição, ou seja, o que se monitora não é a transição do risco, mas sim os seus indicadores [DEM 2003].

Uma estratégia através da qual os riscos podem ser monitorados é o paradigma GQM (*Goal/Question/Metric*). Este paradigma é baseado é uma abordagem orientada a metas e utilizada em engenharia de software para a medição de produtos e processos de software [BAS 88]. GQM é baseado no requisito de que toda a coleta dos dados deve ser baseada num fundamento lógico, em um objetivo ou meta, que é documentado explicitamente. O primeiro passo nessa abordagem é definir metas a serem alcançadas no programa de medição. Após a identificação das metas, um plano GQM é elaborado para cada meta selecionada. O plano consiste, para cada meta, em um conjunto de questões quantificáveis que especificam as medidas adequadas para sua avaliação. As questões identificam a informação necessária para atingir a meta e as medidas definem operacionalmente os dados a serem coletados para responder as perguntas.

Manzoni [MAN 2004] propôs uma abordagem utilizando GQM para o monitoramento dos riscos de um projeto. Para cada risco, uma meta é definida com o objetivo de monitorá-lo. As questões são elaboradas de forma a se monitorar os indicadores de transição do risco. As questões são respondidas numericamente através da contabilização de métricas. O exemplo abaixo, retirado de [MAN 2004], mostra algumas questões e métricas que podem ser utilizadas para monitorar riscos relacionados aos requisitos do *software*. [MAN 2004] descreve um conjunto de métricas que permitem monitorar os principais riscos de projetos de *software*.

A medição das métricas tem como objetivo monitorar os riscos, provendo retroalimentação para a análise e a melhoria dos processos da organização [MAN 2004]. Isto implica que o monitoramento dos riscos causa impacto na análise inicial dos riscos em que foi baseado o desenho da metodologia do projeto na abordagem PMT. Conforme os riscos são monitorados, os fatores de exposição do risco devem ser revistos. Isto pode mudar o resultado da análise de riscos feita inicialmente. Caso isto ocorra, novos padrões podem ser selecionados e priorizados para o projeto, com base na nova análise de riscos.

Tabela 7.8: Exemplo de utilização de GQM para monitorar riscos

Riscos: Não entendimento dos requisitos. Requisitos instáveis. Falha na gerência da expectativa dos usuários. Criação de requisitos pela equipe de desenvolvimento (Gold Plating)	
Meta 3: Analisar o projeto com a finalidade de monitorar com respeito à definição dos requisitos do ponto de vista da equipe de desenvolvimento.	
Questão 3.1: Os usuários validaram os documentos de requisitos do projeto?	*Métrica 3.1a*: Percentual de validação de requisitos pelo Cliente PVRC = (número de documentos de requisitos validados/ número total de documentos de requisitos) * 100 *Métrica 3.1b*: Percentual de requisitos elaborados pelo Cliente PRE = (número de requisitos elaborados pelos usuários/requisitos elaborados pelo pessoal de sistema) * 100
Questão 3.2: Os requisitos estão mudando durante a fase de execução do projeto?	*Métrica 3.2*: Percentual de Alteração dos requisitos PAR = (número de solicitações de mudança de requisitos / número total de requisitos) * 100
Questão 3.3: Os requisitos implementados satisfizeram as necessidades dos clientes?	*Métrica 3.3a*: Requisitos aceitos RA = (número de requisitos aceitos / número total de requisitos) * 100 *Métrica 3.3b*: Requisitos rejeitados RR = (número de requisitos rejeitados / número total de requisitos) * 100
Questão 3.4: O tempo para incorporar as mudanças está adequado?	*Métrica 3.4*: Percentual de mudanças realizadas no prazo Classificar as mudanças em simples, média e complexa. Estabelecer uma política de prazo máximo para que a mudança seja incorporada no sistema. PMRP = (número de mudanças realizadas no prazo / número de mudanças solicitadas) *100

7.7 Evolução do repositório de conhecimento

Esta seção descreve como que o repositório de conhecimento PMT pode evoluir ao longo do tempo, conforme a abordagem for aplicada em um número maior de projetos. O conhecimento armazenado no repositório foi pensado de forma a evoluir, conforme novos padrões e regras forem registrados, e conforme a abordagem for aplicado em um número grande de projetos.

PMT baseia-se em um repositório de conhecimento o qual armazena padrões, riscos, e regras de resolução de riscos. Os padrões armazenados neste repositório podem ser obtidos ou elaborados a partir de várias fontes, principalmente:

- Publicações de padrões já existentes, em livros, periódicos (como [COP 95] e [BEE 99]), ou web sites de Internet (como [COP 2004]).
- Metodologias já publicadas, ainda que não estejam exatamente na forma de padrões. Os padrões podem ser elaborados a partir das descrições de técnicas, papéis e processos existentes nestas metodologias, como Extreme Programming [BEC 99] ou Feature Driven Development [STE 2002].
- A experiência de gerentes e projetistas de processos, ao resolverem problemas recorrentes em mais de um projeto.

A partir das fontes descritas acima, pode-se compor um repositório de padrões organizacionais bastante amplo. Este repositório pode combinar o conhecimento existente em diversas metodologias que foram estudadas ao longo deste trabalho (capítulos 2 e 3). A seguir são listadas as principais fontes que foram estudadas e que podem ser utilizadas para compor um repositório de conhecimento sobe metodologias, na forma de padrões organizacionais, bastante amplo:

Tabela 7.9: Fontes de padrões organizacionais que foram estudadas

Fonte / Metodologia	Tipo de Metodologia	Tipos de padrões	Referências
Scrum	Ágil, limitada a gerenciamento de projetos	Papéis, técnicas e processos.	[BEE 99] [SCH 95] [SCH 2002]
Feature Driven Development	Ágil, intermediária.	Papéis, técnicas e processos.	[COA 99] [STE 2002]
Extreme Programming	Ágil	Papéis, técnicas e valores.	[BEC 99] [CUN 2005] [WEL 2005]
Rational Unified Process	Guiada por planos	Papéis e processos.	[JAC 2000]
Capability Maturity Model	Guiada por planos	Processos.	[SEI 95]

Padrões organizacionais	Ágil	Papéis, técnicas, processos e valores.	[COP 95] [COP 2004] [HAR 96a]

Alguns padrões foram, ainda, capturados pelo autor deste trabalho, a partir de sua experiência na solução de problemas que recorrentes nos projetos em que trabalhou.

Como fonte para os riscos a serem armazenados no repositório utilizou-se a lista de riscos descrita na seção 6.4. A lista foi elaborada por Fontoura [FON 2004] a partir da combinação do trabalho de diversos autores, e inclui os principais riscos que costumam ocorrer em projetos de *software*.

As regras de resolução de riscos vinculam os riscos com os padrões que os resolvem. Estas regras também foram extraídas a partir da literatura, ainda que não estivessem descritas explicitamente como regras, ou que não mencionassem explicitamente o risco sendo resolvido. Um exemplo é o padrão *EngageCustomer* [COP 95], cujo objetivo é manter a satisfação do cliente fazendo o usuário participar ativamente do desenvolvimento e interagindo com os desenvolvedores e os arquitetos. O padrão claramente ajuda a minimizar o risco de "Falta de envolvimento do usuário". Outro exemplo é o padrão "*ProductOwner*", o qual descreve o papel do gerente de produto, o qual deve balancear entre as diferentes necessidades dos departamentos de usuários, sendo o ponto focal do projeto na definição de prioridades de desenvolvimento. Estabelecer este papel em um projeto claramente minimiza o risco de "conflitos entre departamentos dos usuários" afetar o projeto. Outras regras, ainda, foram elaboradas pelo autor deste trabalho de acordo com a sua experiência.

A base de conhecimento – o repositório PMT - foi elaborada de forma a evoluir ao longo do tempo. O conhecimento sobre metodologias está aumentando, e, conforme novas metodologias, técnicas ou padrões sejam publicados, este conhecimento pode ser acrescentado ao repositório.

O conhecimento existente no repositório, capturado na forma de padrões, riscos e regras de resolução de riscos, também deve evoluir através da experimentação e do ajuste. Ou seja, conforme os padrões e as regras forem experimentadas, este conhecimento pode ser ajustado. As descrições dos padrões podem ser melhoradas. Por exemplo, a descrição do problema pode ser aprimorada para melhor refletir o problema genérico que o padrão se propõe a resolver. Ou ainda, a descrição do contexto de aplicação de um padrão pode evoluir e amadurecer conforme este padrão for experimentado, e melhor forem estabelecidos os limites de sua aplicação. As regras podem ser ajustadas redefinindo-se o seu fator de eficácia, ou qual é um valor de 0 a 10 que reflete o quanto que um ou mais padrões colaboram para minimizar os riscos associados à regra. Este fator pode ser ajustado para mais ou para menos de acordo com o resultado da aplicação da regra em projetos.

Uma oportunidade para melhoria na ferramenta descrita na próxima seção seria o desenvolvimento de mecanismos de auto-ajuste. Price e Girardi descrevem em seu artigo [PRI 90] uma ferramenta para suporte ao reuso de software em ambientes orientados a objeto. Um mecanismo de busca de classes que integra a procura sistemática (sobre descritores das classes) com a exploração através de navegação é sugerido. São utilizados fatores auto-adaptáveis que refletem a experiência dos usuários na reutilização das classes existentes. Os fatores são obtidos por mecanismos

de aquisição de conhecimento, e são incorporados no mecanismo sistemático de busca. Mecanismos similares poderiam ser desenvolvidos de forma a capturar a experiência dos usuários na sua utilização e auto-ajustar o fator de eficácia das regras de resolução de risco de acordo.

7.7.1 Validação de padrões organizacionais

Esta seção descreve um modelo para suportar a validação de padrões organizacionais, de forma que os padrões no repositório PMT possam ser avaliados de acordo com o seu *status* de validação. As idéias presentes nesta seção não foram implementadas na ferramenta descrita na próxima seção, mas podem ser utilizadas em trabalhos futuros.

É difícil estabelecer-se uma métrica simples para julgar ou avaliar um padrão organizacional. Por isso, normalmente o valor de um padrão organizacional é julgado empiricamente pelos gerentes das organizações que o aplicam [DEV 2002]. Este problema pode ser endereçado através da retroalimentação de experiências e resultados na aplicação dos padrões. Considera-se que a descrição de um padrão possa ser obtida ou elaborada a partir de diversas fontes, principalmente:

- Publicações de padrões organizacionais existentes na literatura;
- Web sites na Internet;
- Metodologias de desenvolvimento de software;
- Experiência de gerentes ou metodologistas na resolução de problemas recorrentes.

Considera-se ainda que muitas descrições de padrões obtidas a partir das fontes acima podem ser de qualidade duvidosa. Ainda pior, mesmo que um padrão seja efetivo e bem descrito, um erro na descrição de seu contexto de aplicação pode torná-lo inadequado se este for aplicado no contexto errado. Este problema pode ser endereçado ao se manter o controle do seu status de validação. O status de validação é um atributo do padrão, o qual pode ser armazenado junto com o mesmo na base de conhecimento sobre metodologias de desenvolvimento de software. Um padrão, ao ser cadastrado na base, é considerado em um status inicial (não validado). Conforme a retroalimentação é coletada, o padrão pode evoluir para outro status de validação. Neste processo, seu contexto e sua descrição são refinados. Se muita experiência negativa for coletada na aplicação de um determinado padrão, este pode passar ao status de “Anti-Padrão”.

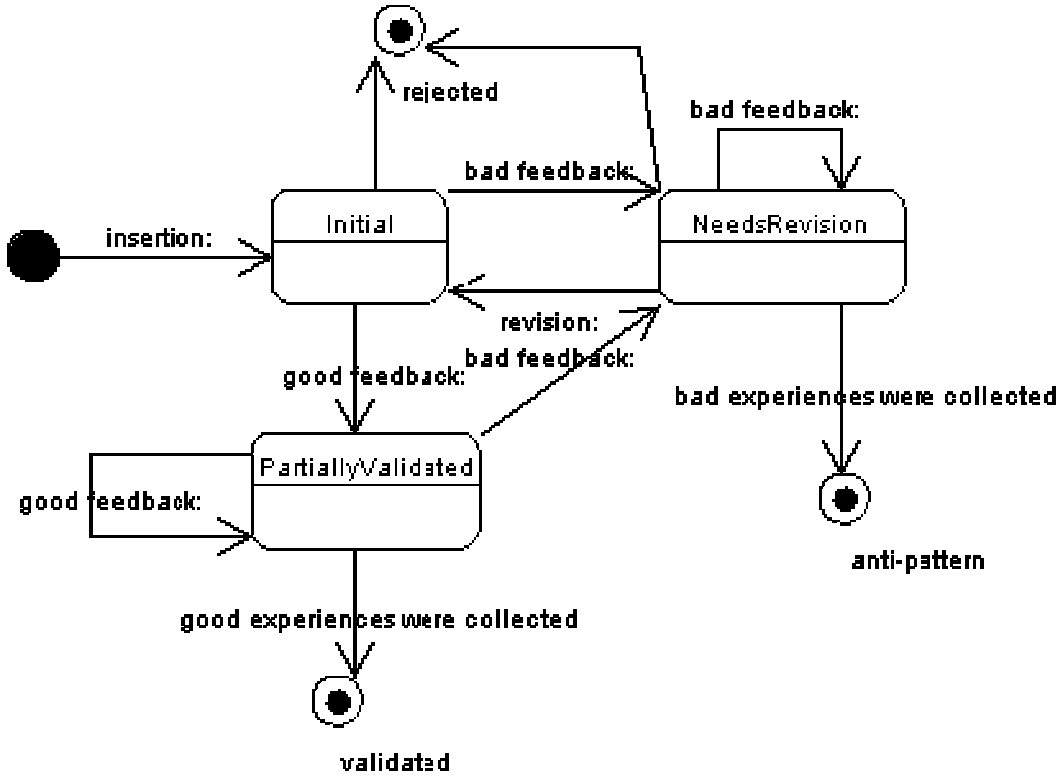


Figura 7.5: Diagrama de estados possíveis para um padrão organizacional

A figura 7.5 explora a transição de estados que os padrões poderiam sofrer uma vez que fossem cadastrados no repositório PMT. Um padrão teria desta forma um ciclo de vida, de acordo com seu status de validação. Quando um padrão organizacional é inserido na base de conhecimento, este é considerado em um status inicial (ainda não validado). Uma vez cadastrado o padrão, este permanecerá no status inicial até que seja recebida retroalimentação (*feedback*) em relação a uma experiência de aplicação do mesmo. Se for relatada uma experiência de sucesso, este passará ao estado de *ParcialmenteValidado*. Mas se, por outro lado, for reportado insucesso ou algum defeito em sua descrição, este ficará em um estado de *PrecisaReparos*. Permanecerá neste estado até que seja editado por seu autor ou uma pessoa habilitada, quando voltará ao estado inicial novamente. Ainda, se forem coletadas três experiências de sucesso na aplicação de um padrão, este último é considerado *Validado*. Se ocorrer o contrário, ou seja, o insucesso for regra, então este será considerado um *Anti-Padrão*.

Como sugestão de trabalho futuro, poderiam ser desenvolvidos mecanismos de aquisição do conhecimento, o qual capturassem as experiências de sucesso ou insucesso dos usuários de PMT na aplicação dos padrões. A aquisição do conhecimento poderia ser utilizada de forma a suportar a validação dos padrões conforme sugerido nesta seção, e ainda o auto-ajuste dos fatores de eficácia das regras de resolução de riscos conforme discutido na seção anterior. Outra melhoria, ainda, é que o estado de validação dos padrões poderia ser considerado pelo mecanismo de seleção sistemático descrito na seção 6.5. O mecanismo poderia ser aprimorado para atribuir uma melhor nota na pesquisa aqueles padrões que estiverem mais experimentados e validados. Assim, a tendência do mecanismo de seleção seria sugerir padrões já comprovados ao invés dos padrões ainda não experimentados.

8 A FERRAMENTA PMT-TOOL

Este capítulo aborda a ferramenta PMT-Tool, a qual suporta a abordagem PMT – *Pattern-based Methodology Tailoring* que foi apresentada no capítulo anterior. Neste capítulo a ferramenta será mais detalhada.

A ferramenta PMT-Tool foi inspirada no web site *wiki* de Ward Cuningham [CUN 2004]: “Este web site é um wiki no qual o foco primário são pessoas, padrões e projetos em desenvolvimento de *software.* (...) Observe que a idéia de “Wiki” é bastante estranha a princípio, mas mergulhe e explore suas referências (*links*). Wiki é um sistema composto; é um meio de discussão; é um repositório; é um sistema de mensagens; é uma ferramenta para colaboração. Na verdade nós não sabemos de fato o que ele é, mas é uma forma divertida de comunicação assíncrona através de uma rede. A palavra *wiki* é o havaiano para rápido (*quick*).” O funcionamento básico de um *wiki* é permitir a edição colaborativa das páginas, com uma grande facilidade para a criação e a manutenção da integridade das referências (*links*). O web site de Cuningham foi utilizado para capturar um grande número de padrões (*patterns*), e está ligado à formação da comunidade de padrões aplicados ao desenvolvimento de *software*. A ferramenta PMT-Tool facilita também a edição colaborativa dos padrões, mas de forma mais estruturada.

A figura 8.1 mostra a tela inicial da ferramenta, a qual apresenta um menu de opções. A figura 8.2, por sua vez, mostra as opções disponíveis no menu inicial da ferramenta, na forma de uma hierarquia. As opções de menu são divididas em três partes. A primeira parte, que será apresentada na seção 7.1, oferece as funcionalidades relacionadas ao gerenciamento do repositório de padrões, riscos e regras. A segunda parte, que será discutida na seção 7.2, oferece opções para o gerenciamento de projetos, análise de riscos e navegação na linguagem de padrões específica de um projeto. A terceira parte, que será discutida na seção 7.3, permite ao usuário criar uma sessão de seleção.

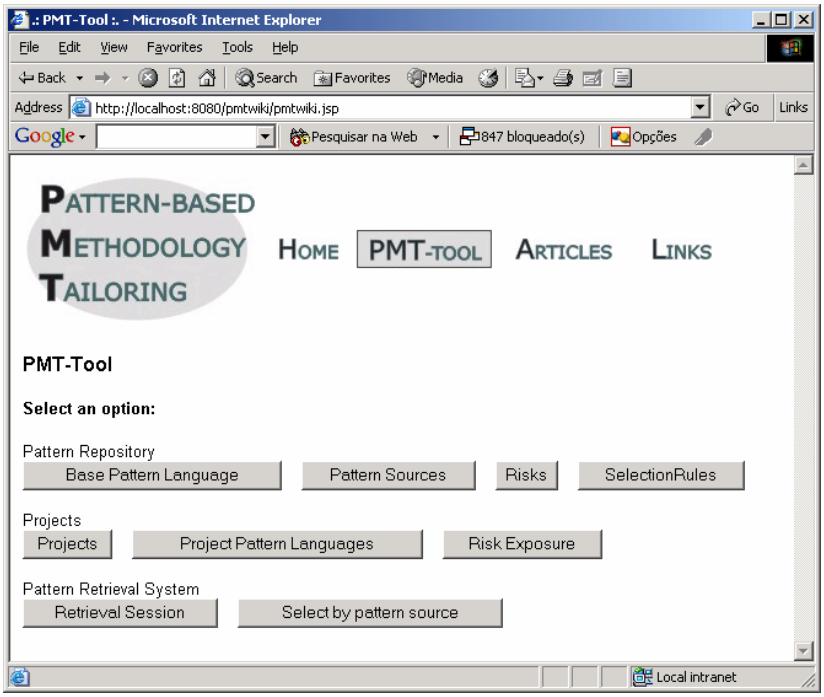


Figura 8.1: Menu inicial da ferramenta PMT-Tool

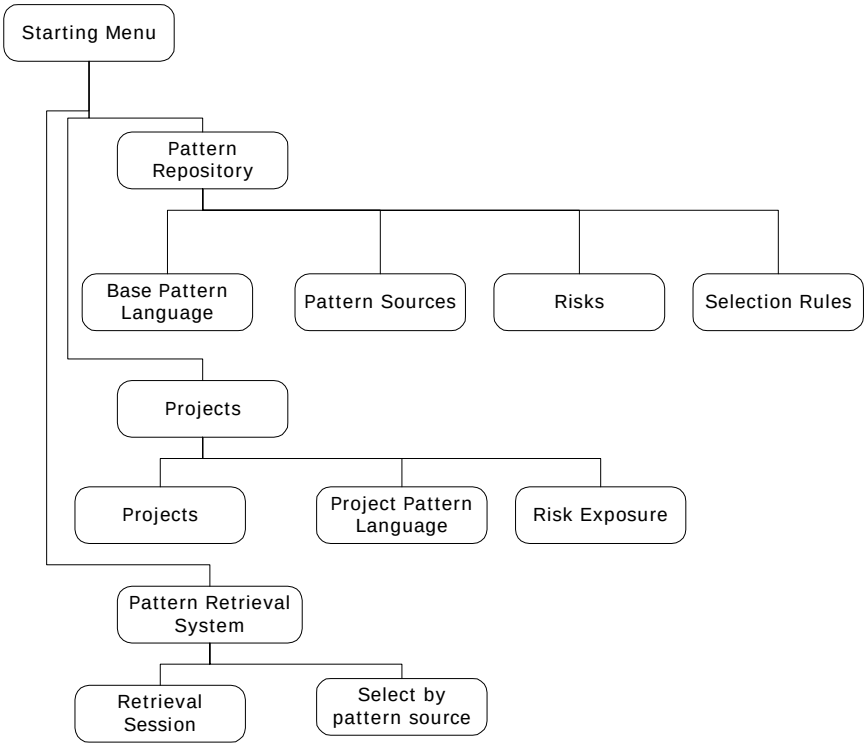


Figura 8.2: o menu inicial da ferramenta (diagrama hierárquico)

8.1 O Repositório de Padrões

Será explorado inicialmente o repositório de padrões, o qual pode ser navegado a partir da opção “Base Pattern Language” da figura 8.2.

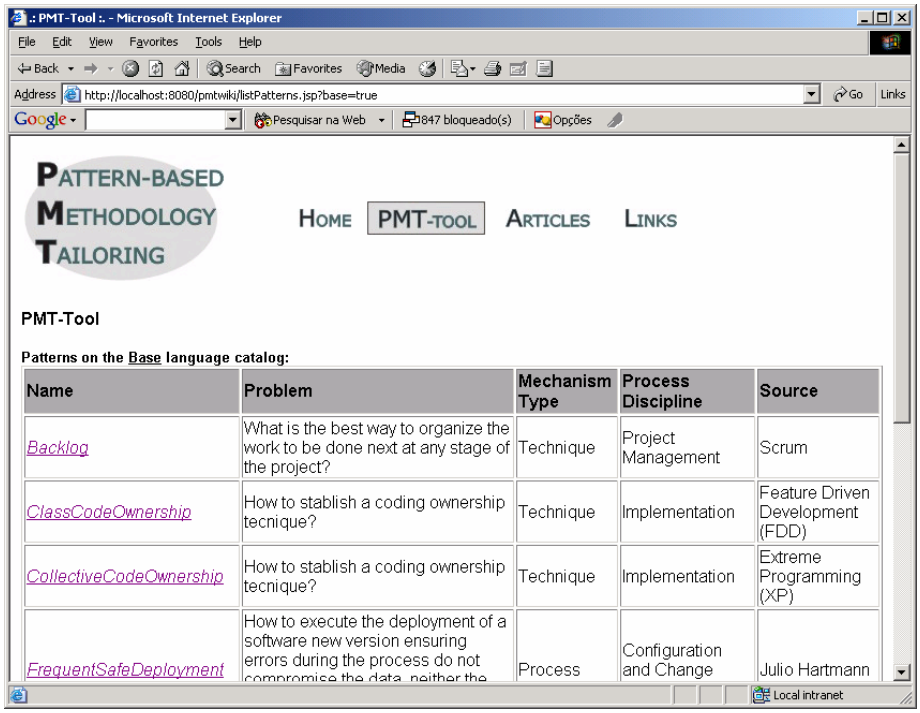


Figura 8.3: Exemplo de lista de padrões existentes na linguagem de padrões base (repositório)

A tela de entrada do repositório de padrões lista todos os padrões existentes, como exemplificado na figura 8.3. As opções disponíveis nesta tela são clicar sobre a referência no nome do padrão, que leva a tela de detalhamento do padrão, ou clicar em um botão para adicionar um novo padrão. A figura 8.4 mostra o fluxo de navegação a partir destas opções.

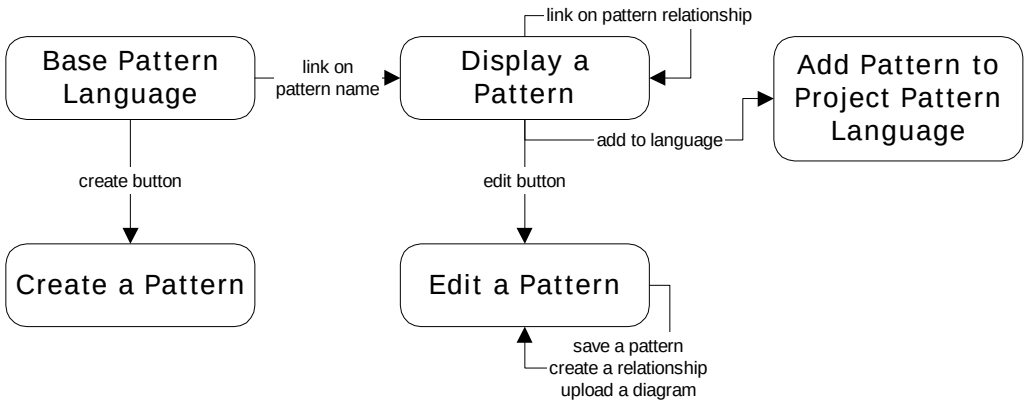


Figura 8.4: navegação a partir da listagem de padrões da linguagem base (repositório)

A criação de um novo padrão é feita preenchendo-se os campos de nome, classificação, fonte, problema, contexto, forças, solução e justificativa, como exemplificado pela figura 8.5.

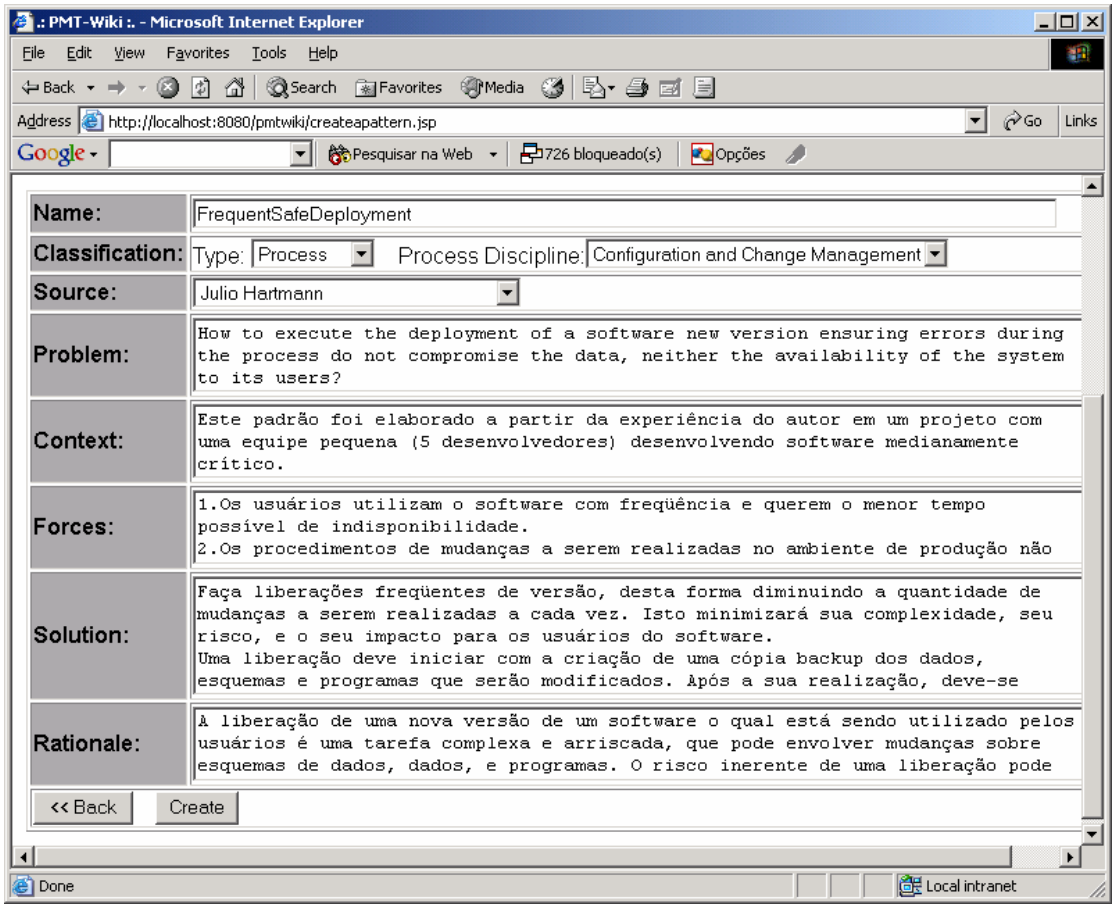


Figura 8.5: Exemplo de adição de um novo padrão ao repositório

Após o padrão ter sido adicionado ele aparecerá na listagem de padrões existentes no repositório. Pode-se visualizar suas informações detalhadas ao clicar-se sobre a referência de hipertexto em seu nome. A figura 8.6 mostra o detalhamento do padrão de exemplo recém cadastrado, ainda sem relacionamentos com outros padrões, e sem um diagrama para representar sua solução.

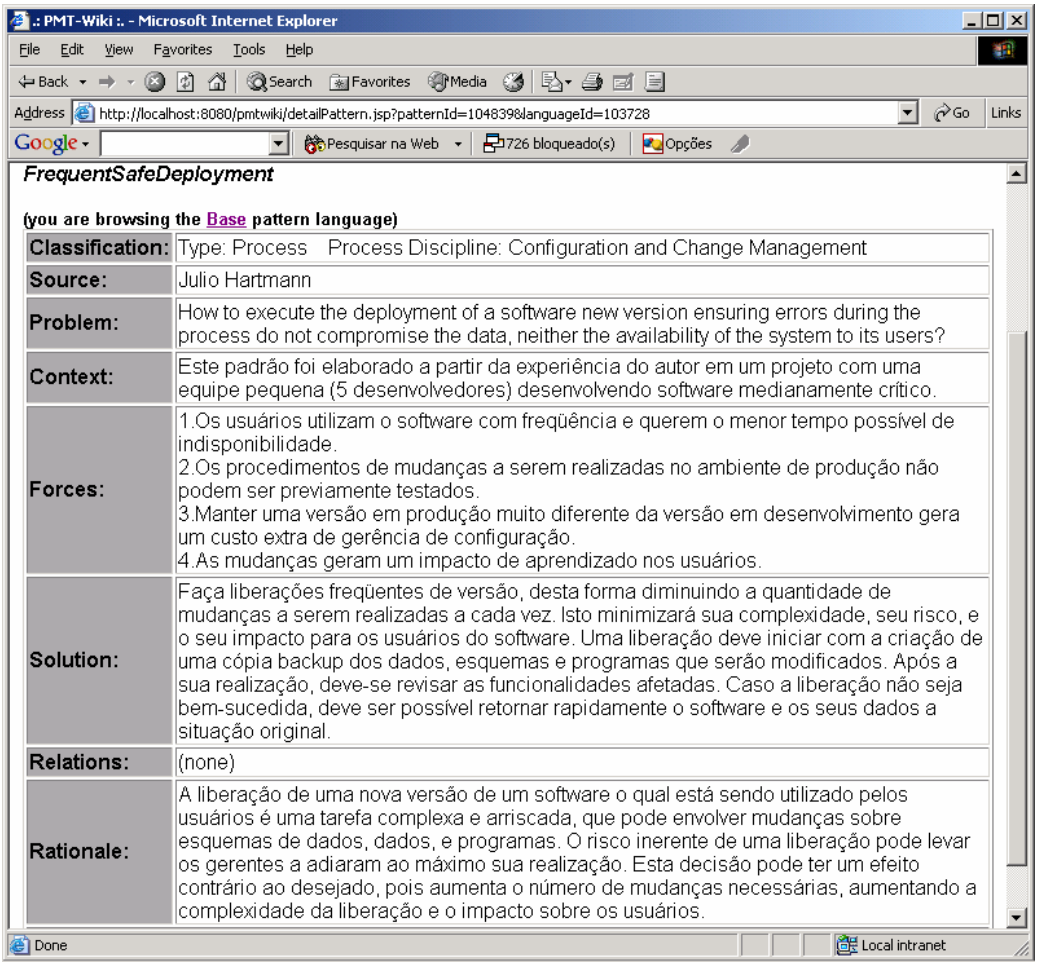


Figura 8.6: Tela de detalhamento de um padrão

Uma opção que é acessível na tela de detalhamento de um padrão é a adição do padrão a uma linguagem de padrões de projeto, conforme exemplificado na figura 8.7.

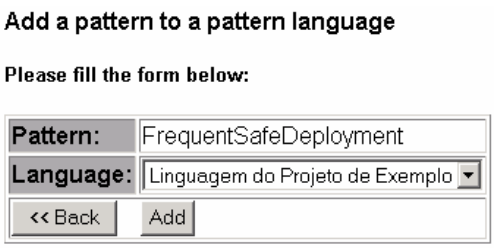


Figura 8.7: adicionando um padrão a uma linguagem de padrões de projeto

A tela de edição de padrões é semelhante à tela de criação, mas inclui duas novas possibilidades: a adição de relacionamentos. Os relacionamentos são adicionados selecionando-se o tipo de relacionamento, conforme discutido na seção 6.3 (dependência, trabalho em conjunto, alternativa, similaridade, generalização / especialização, composição), o padrão de origem e o padrão de destino (a origem e o destino não podem ser o mesmo padrão, e o padrão sendo editado deve ser ou origem

ou destino). A figura 8.8 mostra uma parte da tela de edição de um padrão, que permite a edição dos relacionamentos. Os relacionamentos já existentes são listados da mesma forma na tela de detalhamento de um padrão. Ao se clicar sobre o nome do padrão relacionado, navega-se para o seu respectivo detalhamento.

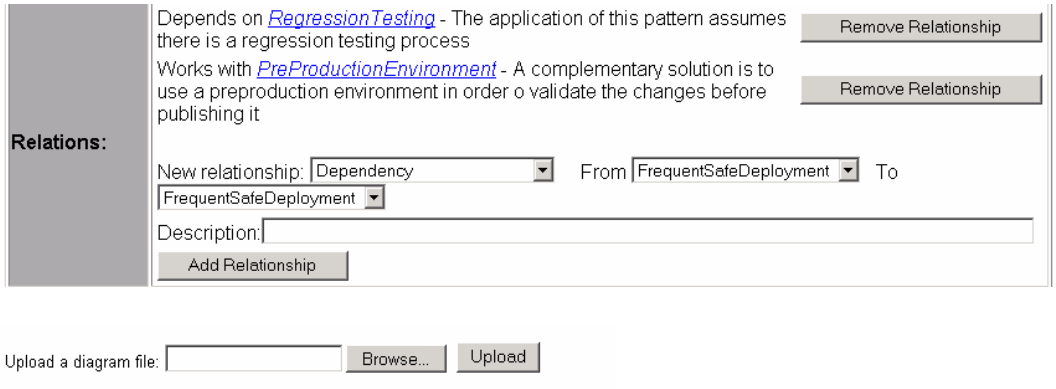


Figura 8.8: Editando os relacionamentos de um padrão, ou fazendo upload de um arquivo de diagrama

Outra opção permitida na tela de edição é o carregamento (*upload*) de um arquivo de diagrama. O diagrama é carregado junto com o texto da seção de "solução" do padrão, e serve para auxiliar na descrição desta seção. No padrão utilizado como exemplo, o resultado no campo solução, após o carregamento de um arquivo de diagrama, ficou como mostrado pela figura 8.9.

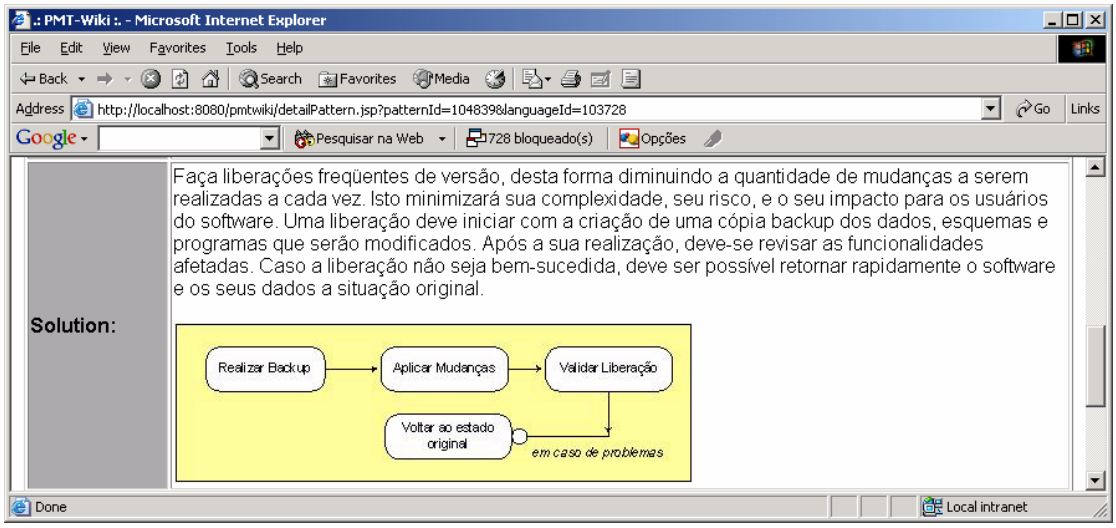


Figura 8.9: Campo solução incluindo um diagrama de processo

Um outro cadastro que é utilizado como auxílio para classificar os padrões é o cadastro de fontes (*Pattern Sources*), como exemplificado pela figura 8.10. Cada padrão pode estar associado a uma fonte, que corresponde ao autor ou publicação a partir do qual o padrão foi extraído ou elaborado.

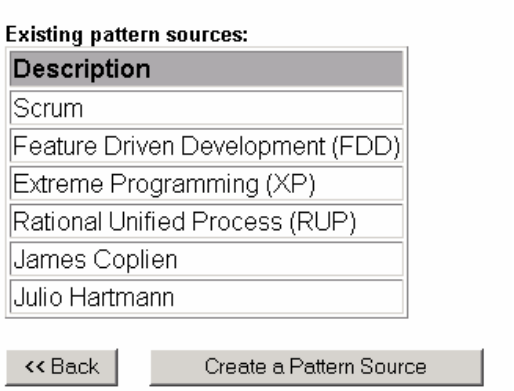


Figura 8.10: Exemplo de lista de fontes de padrões

O repositório da ferramenta PMT-Tool permite ainda o cadastro de riscos, que é exemplificado na figura 8.11. Os riscos são classificados pelo tipo de risco (cliente, requisitos, planejamento ou execução).

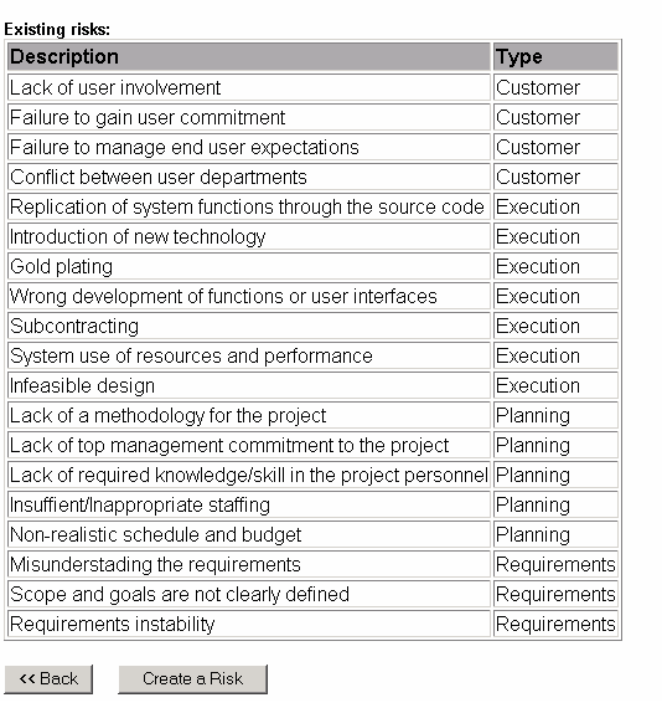


Figura 8.11: Cadastro de riscos

Outra informação importante no repositório são as regras de resolução de riscos. Conforme foi discutido no capítulo anterior, estas regras servem para associar os padrões aos riscos que pretendem resolver, minimizar ou prevenir. Cada regra está associada a um contexto de criticalidade de projetos (criticalidade dos defeitos, tamanho do projeto em termos de número de pessoas envolvidas, e habilidade da equipe). Cada regra especifica também um fator de eficácia (AF – *accuracy factor*), que serve para ajustar o mecanismo de seleção de padrões.

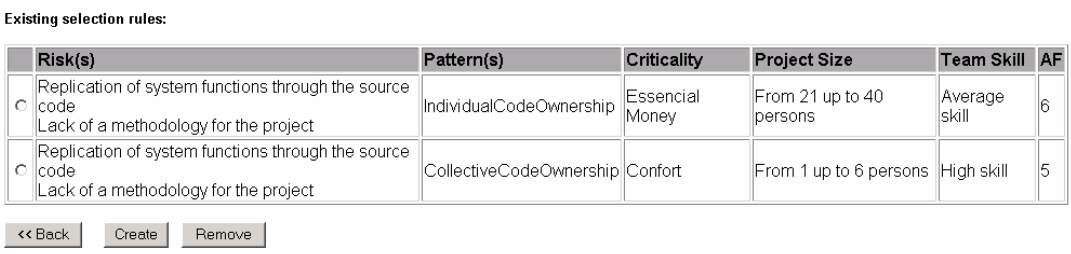


Figura 8.12: Listagem de regras de resolução de riscos

A figura 8.13, por outro lado, exemplifica o cadastro de uma nova regra de resolução de riscos. Ao risco de "replicação de funções do sistema pelo código-fonte", vincula-se a aplicação de padrões que representam técnicas de propriedade coletiva do código fonte (*CollectiveCodeOwnewship*) e redesenho do código (*Refactoring*), como uma fator de eficácia (AF) de valor igual a 5.

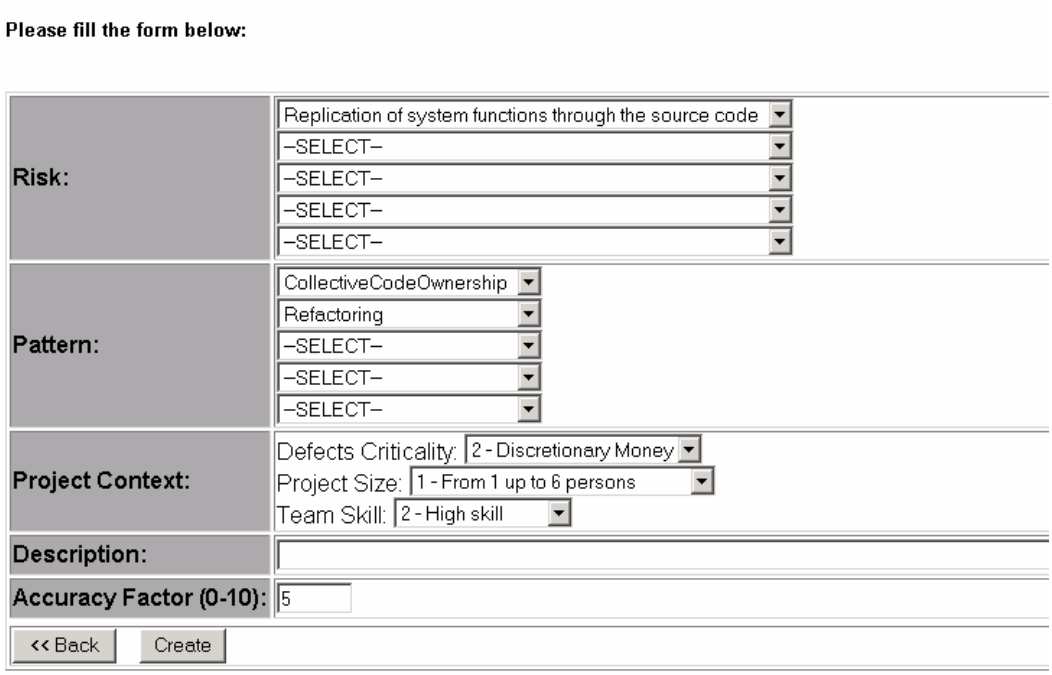


Figura 8.13: Cadastrando uma nova regra de resolução de riscos

As funções da ferramenta PMT-Tool que foram apresentadas nesta seção compreenderam principalmente o cadastro dos padrões da linguagem de padrões base, os riscos e as regras de resolução dos riscos. Estas informações formam um repositório de conhecimento sobre metodologias de desenvolvimento de *software* e sobre gerenciamento de riscos. O conhecimento relativo a metodologias (padrões) é utilizado de forma vinculada aos riscos, através das regras.

8.2 Cadastrando projetos

Esta seção apresenta as funcionalidades relativas ao cadastro de projetos e linguagens de padrões de projetos. A funcionalidade de identificação e priorização dos riscos de um projeto também será apresentada.

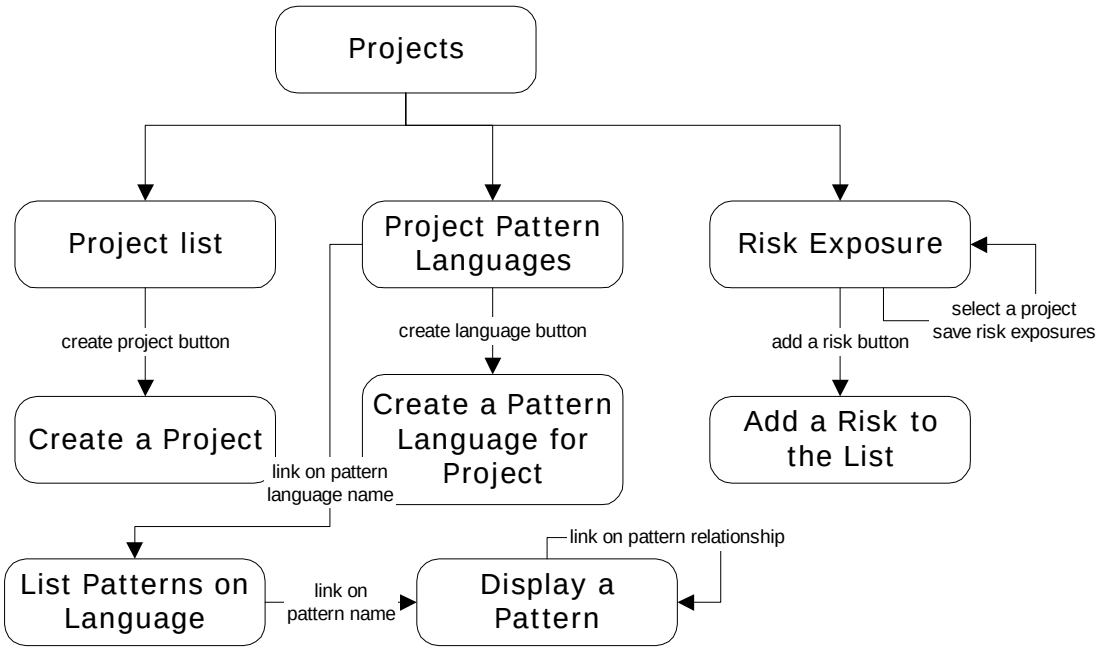


Figura 8.14: Navegação para o gerenciamento de projetos na ferramenta

A figura 8.14 mostra o fluxo de navegação que será detalhado nesta seção do trabalho, relativo ao cadastro de projetos, ao cadastro de linguagens de padrões associadas a estes projetos, e a análise de riscos destes projetos.

Para que um projeto de desenvolvimento de *software* possa se beneficiar da abordagem PMT através da ferramenta PMT-Tool, primeiramente este deve ser cadastrado. A figura 8.15 mostra a tela da ferramenta a qual lista os projetos já cadastrados. O cadastro de cada projeto inclui uma análise de seu contexto de criticalidade, em termos dos fatores de criticalidade dos defeitos, tamanho do projeto e habilidade da equipe.

Existing projects:

Name	Description	Defects Criticality	Project Size	Team Skill
Projeto Ágil	Este é um teste de projeto (contexto ágil)	1 - Confort	1 - From 1 up to 6 persons	1 - Very high skill
Projeto de Exemplo	Projeto de Exemplo	2 - Discretionary Money	1 - From 1 up to 6 persons	2 - High skill
Projeto Intermediário	Este é um teste de projeto (contexto intermediário)	3 - Essencial Money	3 - From 21 up to 40 persons	2 - High skill

<< Back Create a project

Figura 8.15: Listagem de projetos

A tela de cadastro de um novo projeto, por sua vez, é exemplificada na figura 8.16. Para que um projeto seja cadastrado, seu contexto de criticalidade deve ser analisado, o que é feito selecionando-se valores para os três critérios já discutidos. O

critério de habilidade da equipe é definido pela porcentagem de pessoas da equipe do projeto em cada nível de habilidade listado (-1, 1B, 1A, 2, e 3).

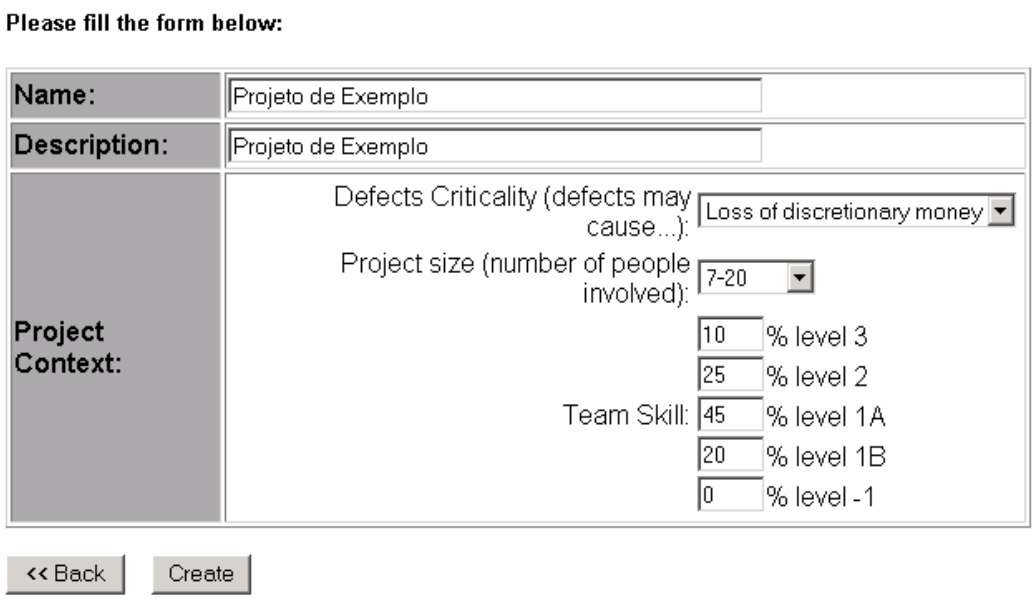


Figura 8.16: Creating a new project record on the tool

O cadastro de um novo projeto, conforme exemplificado pela figura 8.16, é feito junto com a determinação da criticalidade do contexto do projeto. Seleciona-se a criticalidade dos defeitos e o tamanho da equipe, a partir de listas de seleção. O tamanho do projeto, por sua vez, é determinado classificando-se a equipe do projeto em termos do nível de entendimento, como foi explicado no capítulo anterior. Entra-se com o percentual da equipe do projeto que se enquadra em cada uma dos níveis. A pontuação para a habilidade da equipe vaia de 1 (muito habilidosa) até 5 (muito pouco habilidosa), é calculada pela fórmula *skill = 6 - (n(3)*5 + n(2)*4 + n(1A)*3 + n(1B)*2 + n(-1)*1)/100*, na qual n(x) retorna o percentual de pessoas da equipe que foi informado pelo usuário para cada nível de habilidade. O resultado é arredondado para baixo.

Após o projeto ter sido cadastrado, deve-se criar uma linguagem de padrões do projeto. Esta linguagem de padrões corresponde a um ambiente de trabalho (*workspace*) em separado do repositório de padrões. Esta linguagem compreenderá um subconjunto dos padrões existentes no repositório completo (a linguagem base), e conterá os padrões que serão efetivamente aplicados ao projeto para minimizar seus riscos.

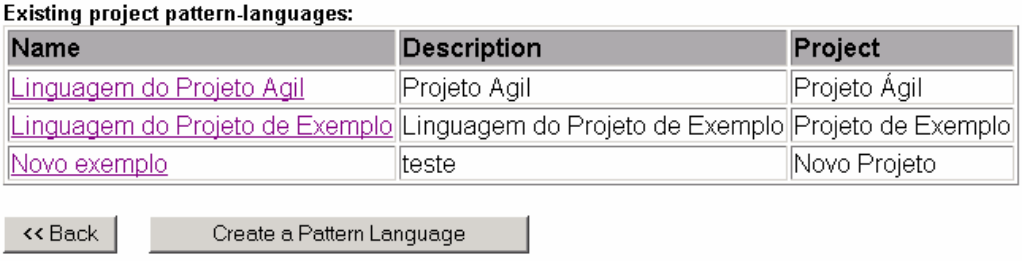


Figura 8.17: Lista de linguagens de padrões de projetos

Como mostrado no diagrama de fluxo de navegação da figura 8.14, a partir da tela de listagem de linguagens de padrões, pode-se criar uma nova linguagem, vinculando-a a um projeto ainda não vinculado, ou então pode-se clicar na referência de hipertexto sobre o nome da linguagem, que levará a tela de listagem dos padrões existentes na linguagem. A tela resultante é semelhante à exemplificada na figura 8.18, com a diferença de que, ao invés de mostrar todos os padrões cadastrados no repositório, lista apenas os padrões que estão associados com a linguagem de padrões selecionada. Da mesma forma que na listagem de padrões da linguagem base, é possível clicar sobre a referência de hipertexto no nome de um padrão, e assim navegar para sua descrição textual. A descrição textual do padrão contém referências de hipertexto para os padrões relacionados – neste caso, somente os relacionamentos com padrões que também pertencem à linguagem de padrões do projeto são apresentados.

Uma vez cadastrado o projeto junto com seu contexto de criticalidade, e de configurar-se uma linguagem de padrões para o mesmo, o próximo passo dentro da ferramenta é realizar uma análise de exposição de seus riscos. Primeiramente se identificam os riscos do projeto, o que é feito pela tela de adição de riscos (figura 8.18) à lista de riscos do projeto. Seleciona-se um dos riscos já previamente cadastrados, e se atribui uma nota de 0 a 10 para a probabilidade de o risco ocorrer no projeto, e uma nota de 0 a 10 para a perda que o risco pode causar caso se concretize.

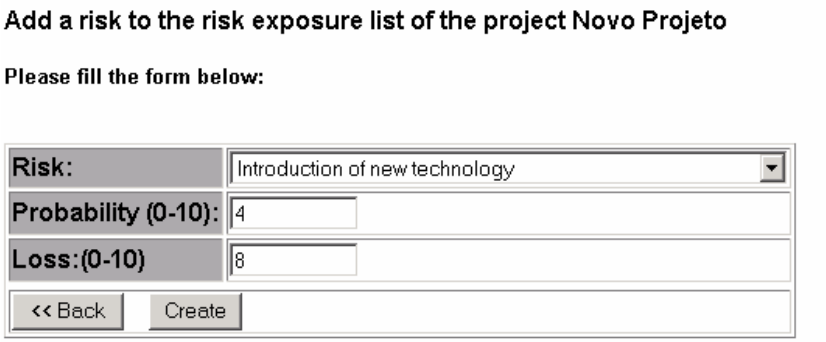


Figura 8.18: Tela de identificação de riscos

Após os riscos terem sido identificados, eles são listados como na figura 8.19. Os riscos são listados em ordem decrescente da exposição do risco (probabilidade x perda). Os riscos são priorizados pelo fator de priorização de riscos (RPF), o qual também é especificado nesta tela. Os riscos cuja exposição do risco é maior ou igual ao fator RPF são marcados em negrito, e serão utilizados no mecanismo sistemático de seleção de padrões. A tela de análise de exposição de riscos permite que o gerente de projeto ajuste as notas de probabilidade, perda, e fator RPF, até que acredite que a análise está terminada.

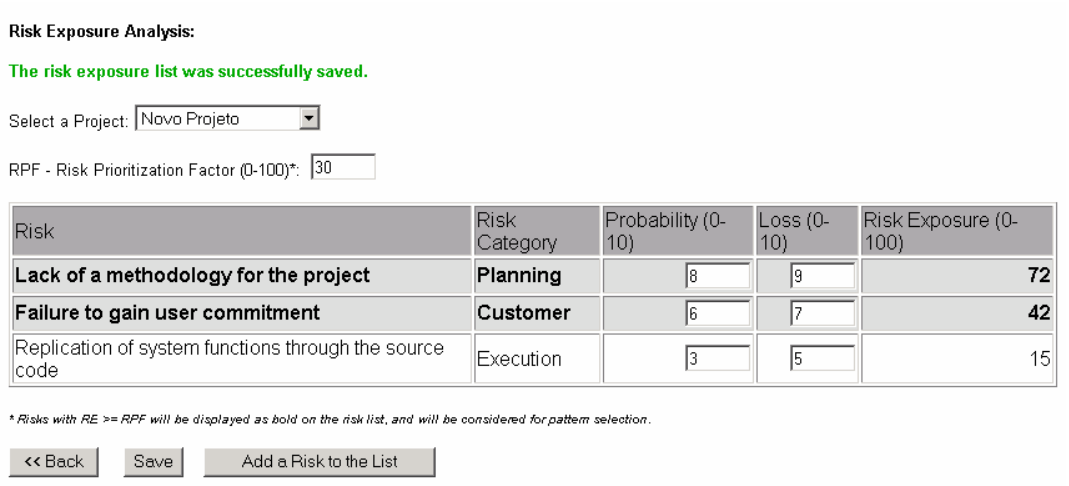


Figura 8.19: Análise de exposição de riscos de um projeto

8.3 O Mecanismo de Seleção

Esta seção apresenta as funcionalidades do mecanismo sistemático de seleção de padrões, conforme foi descrito na seção 6.5. A ferramenta PMT-Tool trabalha com o conceito de sessão de seleção. Uma sessão de seleção é criada e executada. Os resultados da sessão permanecem armazenados pelo servidor Web até que uma nova sessão de seleção seja executada para o mesmo projeto, ou até que o usuário feche seu navegador. Desta forma, o usuário da ferramenta pode executar uma sessão de seleção e listar os resultados na forma de uma lista de padrões sugeridos. O usuário pode então navegar nas descrições dos padrões, voltando ao resultado da sessão de seleção sempre que desejado.

O propósito do mecanismo de seleção da ferramenta é o de sugerir padrões que possam ser aplicados pelo gerente de projeto de forma a minimizar, resolver ou prevenir os riscos do projeto. O papel do gerente de projeto ou do projetista de processos, o qual utilizará a ferramenta com o propósito de selecionar os padrões e adaptar uma linguagem de padrões para o seu projeto, é bastante importante. A ferramenta não tem a intenção de eliminar o papel do projetista de processos, mas sim de auxiliá-lo na difícil tarefa de adaptar uma metodologia de desenvolvimento de *software* a um projeto específico.

Para que uma sessão de seleção possa ser realizada, é necessário que haja um repositório de padrões, riscos e regras de resolução dos riscos. O repositório deve ter sido preenchido utilizando as funcionalidades da ferramenta descritas na seção 7.1. Além disso, é necessário que haja um projeto-alvo, para o qual deseja-se adaptar uma metodologia na forma de uma linguagem de padrões. Este projeto-alvo deve ter sido registrado na ferramenta, e seu contexto de criticalidade deve ter sido definido. O projeto deve ter uma linguagem de padrões própria configurada, e, finalmente, os riscos do projeto precisam ter sido identificados e priorizados utilizando a exposição do risco. Estas últimas funcionalidades foram descritas na seção 7.2. A figura 8.20 mostra o fluxo de navegação com relação às funcionalidades de seleção de padrões.

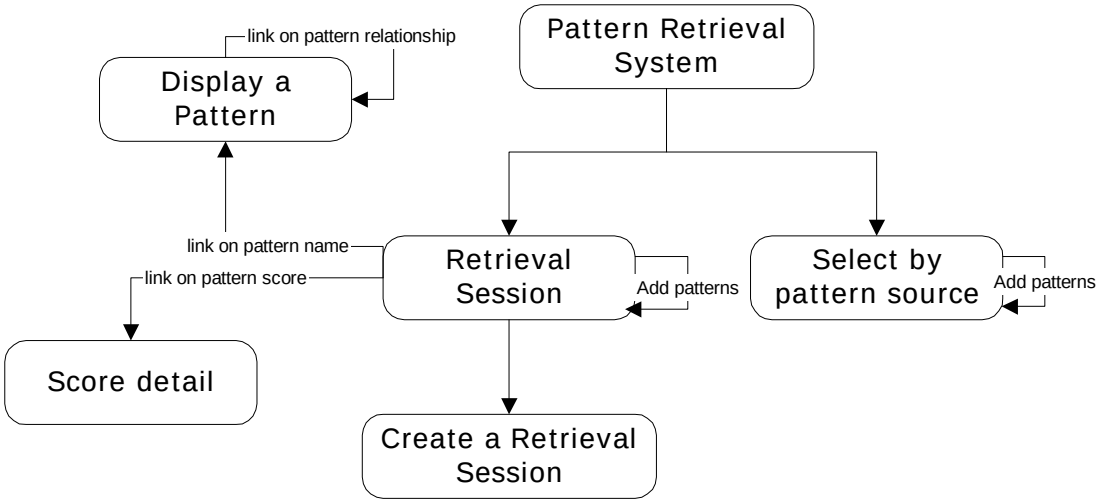


Figura 8.20: Fluxo de navegação para seleção de padrões

Uma seção de seleção é configurada com três parâmetros: a disciplina de processos, o mecanismo do padrão, e o tipo de riscos. Estes três parâmetros são opcionais, e podem ser utilizados para diminuir o escopo da pesquisa a somente um subconjunto dos padrões ou dos riscos do projeto no qual o usuário está concentrado. Há ainda um quarto parâmetro, que é o fator de priorização de riscos, o qual já é definido quando se faz a análise de riscos do projeto. A figura 8.21 exemplifica a tela de seleção dos parâmetros de seleção.

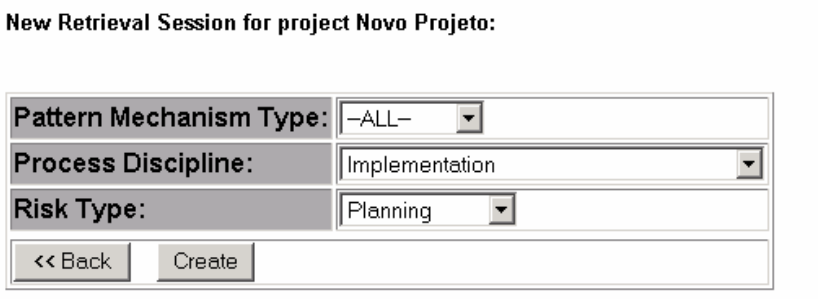


Figura 8.21: Tela de criação de uma sessão de seleção

Uma sessão de seleção é executada de acordo com o algoritmo descrito na seção 6.5, e o seu resultado é uma lista de padrões. Os padrões que já estão associados com a linguagem de padrões do projeto não são selecionados novamente. A lista é ordenada pela pontuação que cada padrão obteve. A pontuação de cada padrão é calculada aplicando-se as regras de resolução de riscos cadastradas no repositório aos riscos que foram priorizados para o projeto. A figura 8.22 mostra um exemplo de resultado para a seleção de padrões.

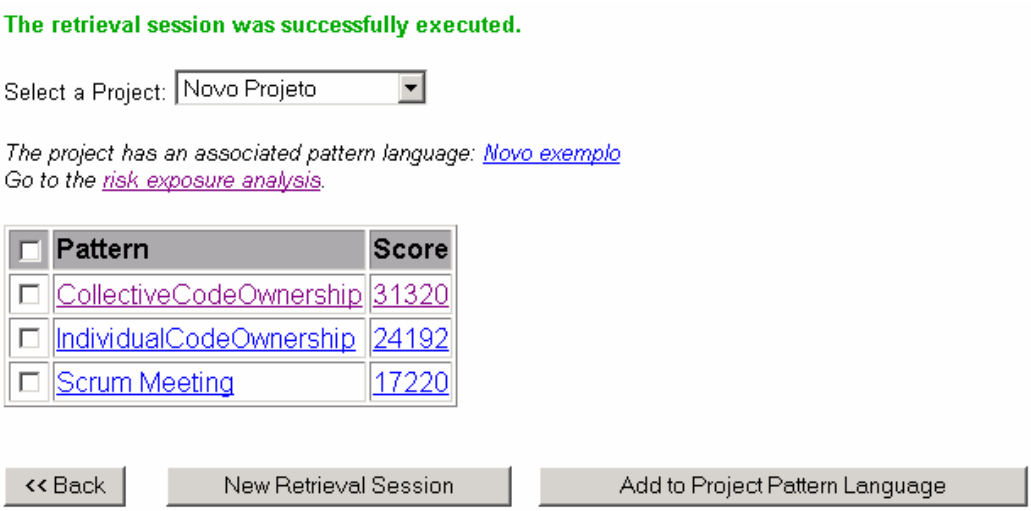


Figura 8.22: Resultado de uma sessão de seleção de padrões

A partir da tela de resultado, a qual lista os padrões que são sugeridos ao gerente de projetos para aplicação no projeto, há 3 opções principais. O usuário pode clicar sobre o nome de um padrão e navegar para sua descrição textual, podendo posteriormente retornar a esta mesma tela com a lista de padrões sugeridos. Outra opção é clicar sobre a pontuação de um padrão, e navegar para uma tela que detalha como que esta pontuação foi calculada. Quando o gerente de projeto decide quais os padrões que deseja adicionar à linguagem de padrões do projeto, este os seleciona e clica no botão de adição.

A tela de detalhamento da pontuação de um padrão é exemplificada pela figura 8.23. São apresentadas quais foram as regras de resolução de riscos que foram aplicadas, de acordo com o algoritmo da seção 6.5. Para cada regra, são apresentados os resultados intermediários utilizados para computar a sua pontuação (exposição de risco média, proximidade do contexto, e fator de eficácia). A pontuação total para um padrão é a soma das pontuações das regras que foram aplicadas, se houver mais de uma.

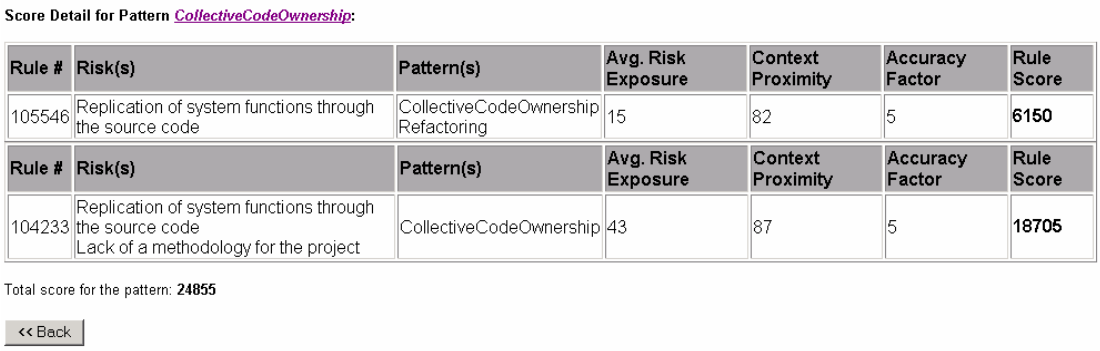


Figura 8.23: Exemplo de detalhamento da pontuação de um padrão

Outra possibilidade de seleção de padrões é a seleção de todos os padrões pertencentes a uma determinada fonte. Por exemplo, o gerente de projeto pode decidir utilizar todos os padrões de uma determinada metodologia (exemplo: *Scrum* [BEE 99]) como base. Uma vez selecionados todos os padrões da metodologia, o gerente de

projeto pode realizar novas sessões de seleção, avaliando os riscos e utilizando o mecanismo sistemático, para endereçar os riscos do projeto que ainda não estejam sendo tratados. A figura 8.27 mostra a tela de seleção de padrões a partir de uma determinada fonte (*pattern source*).

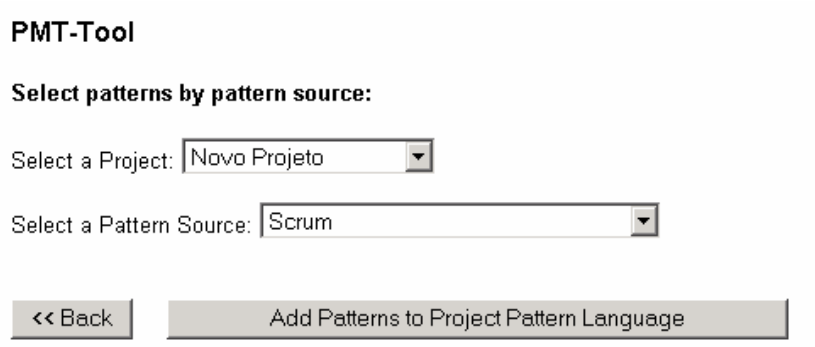


Figura 8.27: Exemplo de seleção de todos os padrões de uma mesma metodologia

8.4 A Arquitetura da Ferramenta

Esta seção descreve a arquitetura da ferramenta PMT-Tool. A ferramenta foi implementada em Java, utilizando-se de outras ferramentas e bibliotecas de software aberto (*open-source*). A tabela 8.1 lista as ferramentas e bibliotecas que foram utilizadas na construção da PMT-Tool, e qual a função de cada uma, bem como mostra o endereço de Internet onde a mesma pode ser obtida.

O desenvolvimento da ferramenta envolveu a realização de uma modelagem conceitual das principais classes envolvidas, resultando em um diagrama de classes. Este diagrama já foi descrito na seção 6.5 deste trabalho, a qual descreve o mecanismo sistemático de seleção de padrões. As classes do modelo conceitual orientado a objetos foram mapeadas para um modelo de banco de dados relacional, utilizando algumas regras para o mapeamento de herança e de associações, como as descritas em [HAR 99]. Para suportar a persistência das classes da ferramenta, foi desenvolvido um modelo físico de banco de dados relacional (figura 8.28).

Tabela 8.1: Ferramentas e bibliotecas de software aberto utilizadas na construção da ferramenta PMT-Tool

Ferramenta/Biblioteca	Função	Endereço de Internet
Java 2 SDK 1.4.2	Máquina Virtual Java	java.sun.com
Apache Tomcat 4.1	Servidor Web Java	jakarta.apache.org/tomcat
Hibernate 2.1.6	Mapeamento de objetos para banco de dados relacional	www.hibernate.org
MySQL 3.2	Banco de dados relacional	dev.mysql.com
Ant 1.5.1	Script de compilação (*build*)	ant.apache.org
JUnit 3.8.1	Arcabouço para testes unitários	www.junit.org
ArgoUML 0.16	Modelagem UML	www.argouml.org

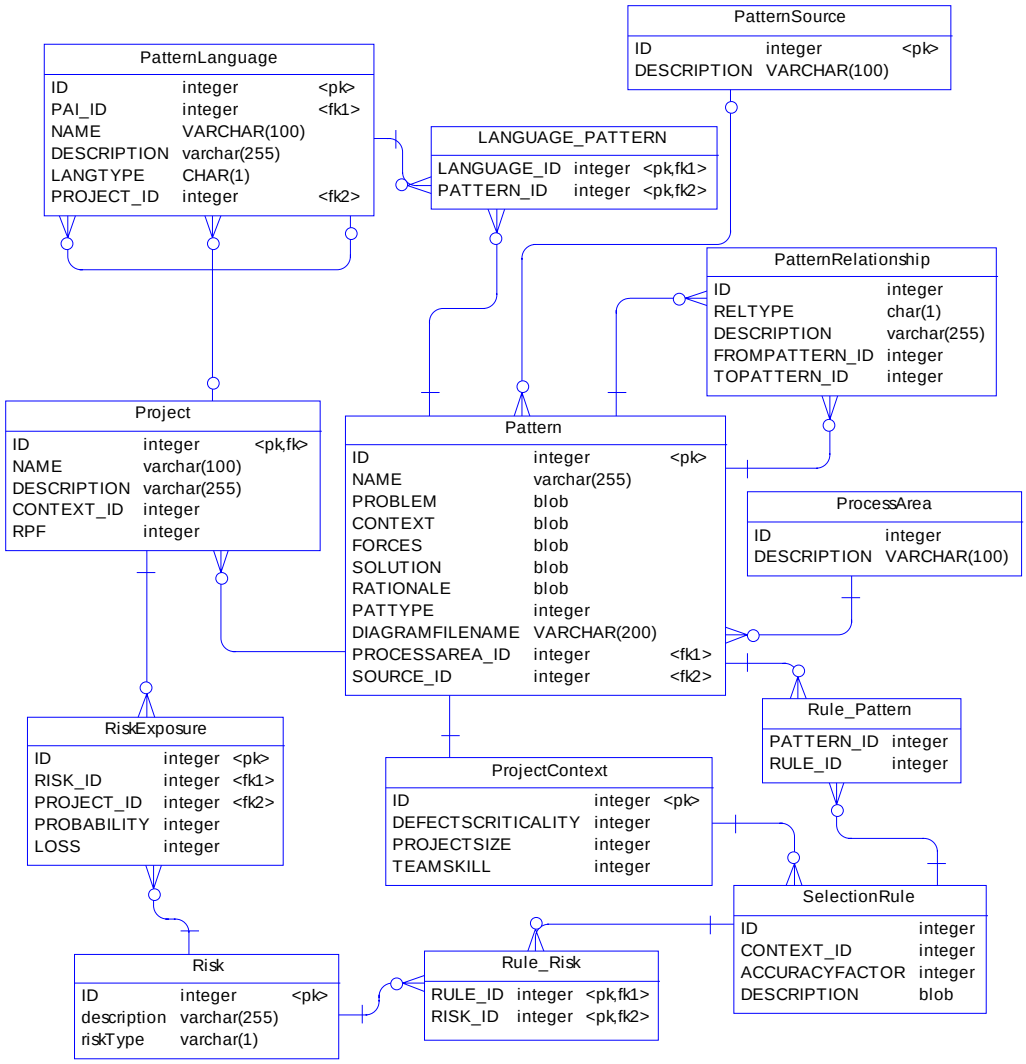


Figura 8.28: Modelo de banco de dados da ferramenta

As classes do modelo conceitual orientado a objetos da ferramenta foram implementadas em Java e sua persistência para o banco de dados foi implementada com o auxílio da biblioteca Hibernate [HIB 2004]. Esta biblioteca facilita o trabalho de interação com o banco de dados por parte do programador, que praticamente não precisa se preocupar com a estrutura do banco de dados relacional, mas somente com a estrutura das classes Java. A biblioteca se encarrega dos detalhes de persistência das classes, como materializar objetos carregando-os a partir do banco de dados, ou desmaterializar estes objetos persistindo-os. A biblioteca ainda inclui instrumentos para a formulação de consultas no nível do modelo de classes, as quais são convertidas para comandos em SQL, que são executados de forma transparente ao programador sobre o banco de dados relacional.

Há duas formas principais de formulação das consultas utilizando Hibernate, sendo que ambas foram utilizadas na realização deste trabalho: *a. classes de interface de programação (API). b. linguagem de consultas (HQL – Hibernate Query Language)*. A opção (a), que consiste em se utilizar classes da biblioteca do Hibernate, como *Criteria* e *Expression*, para formular as consultas, foi a mais

utilizada no trabalho por ser mais simples. Algumas consultas mais complexas foram formuladas utilizando HQL, que é uma linguagem de consultas semelhante a SQL (*Structured Query Language*), com extensões para orientação a objetos. Algumas consultas ainda mais complexas precisaram ser quebradas em partes, e o seu resultado final precisou ser obtido através de operações nas coleções de objetos Java resultantes. Isto ocorreu por limitações no banco de dados MySQL, que não suporta alguns tipos de construções da SQL, como a utilização de critérios de seleção "IN" combinados com sub-consultas.

Tabela 8.2: Principais decisões de mapeamento objeto-relacional que foram utilizadas.

Estrutura no modelos de classes	**Mapemento no banco de dados**
Classe *PatternLanguage* e suas subclasses.	Tabela PatternLanguage, com o campo LANGTYPE definindo qual a classe a qual a tupla pertence: P = *ProjectPatternLanguage* B = *BasePatternLanguage*
Classe *ProjectRisk*.	Tabela Risk.
Associação muitos-para-muitos entre as classes *Pattern* e *SelectionRule*.	Tabela Rule_Pattern.
Associação muitos-para-muitos entre as classes *ProjectRisk* e *SelectionRule*.	Tabela Rule_Risk.
Associação de composição muitos-para-muitos entre as classes *PatternLanguage* e *Pattern*.	Tabela Language_Pattern.
Associação um-para-um entre as classes *Project* e *ProjectPatternLanguage*.	Chave estrangeira na coluna PROJECT_ID na tabela PatternLanguage. Esta coluna somente é utilizada quando LANGTYPE=P.
Associação um-para-muitos entre as classes *BasePatternLanguage* e *ProjectPatternLanguage*.	Tabela Language_Pattern.

A tabela 8.2 descreve as principais regras de mapeamento que foram utilizadas neste trabalho. Estas regras foram especificadas em arquivos XML (*eXtensible Markup Language*) no formato que a biblioteca Hibernate define, de forma que esta possa automatizar as operações de persistência e consulta dos objetos. As demais classes, que não foram mapeadas na tabela 8.2, têm correspondência direta 1-1 (um para um) com as tabelas do banco de dados. As associações não consideradas na tabela 8.2, que são do tipo "um-para-muitos", foram todas implementadas utilizando-se uma chave estrangeira na tabela que representa o lado "muitos" da associação.

A ferramenta foi desenvolvida utilizando-se uma separação das classes do código-fonte em camadas. A camada de apresentação foi construída utilizando-se Java Server Pages (JSP), e não acessa o banco de dados diretamente, mas sim acessa a camada de serviços. Cada JSP é responsável por atender determinadas requisições que são recebidas pelo servidor Web, gerando páginas dinamicamente (em HTML e Javascript) que são enviadas para o navegador (*browser*) do usuário. A camada de

serviços manipula as classes de negócio persistentes, tais como *Project* e *RiskExposure*, e se utilizada da camada de persistência para interagir com o banco de dados. Como camada de persistência, utilizou-se a biblioteca Hibernate [HIB 2004], que é genérica e utiliza informações na forma de metadados para realizar a persistência das classes de negócio. Os metadados são especificados em arquivos XML.

Para testar-se a camada de serviços de forma automatizada e independente da camada de apresentação, foram implementados roteiros de testes unitários com o arcabouço (*framework*) JUnit. Cada roteiro de teste inicia com uma configuração inicial dos dados no banco de dados, e consiste em uma seqüência de casos de teste em que os métodos das classes de serviços são chamados e seus resultados são verificados.

Como processo de desenvolvimento para a ferramenta utilizou-se intensivamente da técnica de *Test-Driven Development* (TDD), ou desenvolvimento guiado a testes. Conforme foi estudado em [HAR 2003], TDD é uma técnica da metodologia Extreme Programming [BEC 99], e pode ser definido, segundo [GEO 2003], como: “uma prática na qual casos de teste unitários são incrementalmente escritos antes de o código estar completo”. Conforme a ferramenta foi incrementalmente implementada, os roteiros de testes foram sendo aprimorados com novos casos de testes, e os roteiros de testes eram freqüentemente executados. Desta forma verifica-se que não somente os novos casos de teste que foram implementados estão funcionando, como também se garante que as modificações que foram realizadas não causaram impacto negativo nas funções já previamente desenvolvidas.

9 EXEMPLO DE APLICAÇÃO DE PMT

Neste capítulo, será apresentado um exemplo de aplicação da abordagem PMT – *Pattern-based Methodology Tailoring*. Na primeira seção, será descrito o repositório de padrões que foi desenvolvido para o exemplo. A segunda seção explica o projeto que foi utilizado como exemplo para aplicação da abordagem, e o resultado da análise do contexto e dos riscos deste projeto. Na terceira seção, com base no repositório de padrões que foi montado, e na análise do projeto, será descrito como se utilizou o mecanismo de seleção sistemático da abordagem PMT para selecionar uma linguagem de padrões organizacionais para o projeto.

9.1 Exemplo de repositório

Para que fosse possível demonstrar a aplicação da abordagem PMT, primeiramente foi necessário criar um repositório de padrões organizacionais. Os padrões foram obtidos a partir de várias fontes, e alguns foram elaborados pelo autor do trabalho. A tabela 9.1 mostra os padrões que foram desenvolvidos.

Tabela 9.1: Padrões organizacionais no repositório

Org. Pattern	Process Discipline	Mechanism	Source
SizeTheSchedule	Proj. Mgmt	Process	J. Coplien
EarlyAndRegularDelivery	Proj. Mgmt	Process	J. Coplien
HolisticDiversity	Proj. Mgmt	Role	J. Coplien
EngageQA	Test	Technique	J. Coplien
NamedStableBases	Config. and Change Mgmt	Technique	J. Coplien
Firewall	Proj. Mgmt	Role	J. Coplien
ArchitectAlsoImplements	Implementation	Role	J. Coplien
DeveloperControlsProcess	Implementation	Role	J. Coplien
ArchitectControlsProduct	Analysis and Design	Role	J. Coplien
CodeOwnership	Implementation	Technique	J. Coplien
IncrementalIntegration	Change and Config. Mgmt	Process	J. Coplien
PrivateVersioning	Change and Config. Mgmt	Process	J. Coplien
ScenariosDefineProblem	Analysis and Design	Process	J. Coplien

Org. Pattern	Process Discipline	Mechanism	Source
SacrificeOnePerson	Proj. Mgmt	Role	J. Coplien
StandUpMeeting	Proj. Mgmt	Technique	J. Coplien
PublicCharacter	Environment	Team Values	J. Coplien
Sprint	Proj. Mgmt	Process	Scrum
Backlog	Proj. Mgmt	Technique	Scrum
DemoAfterSprint	Proj. Mgmt	Process	Scrum
ScrumMaster	Proj. Mgmt	Role	Scrum
ScrumMeetings	Proj. Mgmt	Technique	Scrum
ScrumTeam	Proj. Mgmt	Technique	Scrum
SprintPlanningMeeting	Proj. Mgmt	Technique	Scrum
ProductOwner	Proj. Mgmt	Role	Scrum
CustomerBillOfRights	Environment	Team Values	XP
DeveloperBillOfRights	Enviornment	Team Values	XP
SusteinablePace	Environment	Team Values	XP
TheCoach	Proj. Mgmt	Role	XP
TrackerRole	Proj. Mgmt	Role	XP
AllEngineersInOneRoom	Environment	Technique	XP
StandUpMeeting	Proj. Mgmt	Technique	XP
ProjectVelocity	Proj. Mgmt	Technique	XP
OnSiteCustomer	Requirements	Role	XP
PlanningGame	Proj. Mgmt	Process	XP
UserStories	Requirements	Technique	XP
AcceptanceTests	Test	Technique	XP
ReleasePlan	Proj. Mgmt	Technique	XP
WorstThingsFirst	Proj. Mgmt	Process	XP
SpikeSolutions	Implementation	Process	XP
TestDrivenDevelopment	Test	Process	XP
PairProgramming	Implementation	Technique	XP
CollectiveCodeOwnership	Implementation	Technique	XP
CodingConventions	Implementation	Technique	XP
UnitTests	Test	Technique	XP
ContinuosIntegration	Change and Config. Mgmt	Technique	XP
FrequentReleases	Proj. Mgmt	Process	XP
Refactoring	Implementation	Technique	XP
DoTheSimplestThing	Analysis and Design	Team Values	XP
ProjectManager	Proj. Mgmt	Role	FDD
ChiefArquitect	Analysis and Design	Role	FDD
DevelopmentManager	Proj. Mgmt	Role	FDD
ChiefProgrammer	Implementation	Role	FDD
ClassOwner	Implementation	Role	FDD
DomainExpert	Requirements	Role	FDD
ReleaseManager	Proj. Mgmt	Role	FDD
LanguageGuru	Implementation	Role	FDD
BuildEngineer	Config. And Change Mgmt	Role	FDD

Org. Pattern	Process Discipline	Mechanism	Source
Toolsmith	Environment	Role	FDD
DomainObjectModeling	Analysis and Design	Technique	FDD
DevelopingByFeature	Proj. Mgmt	Technique	FDD
ClassCodeOwnership	Implementation	Technique	FDD
FeatureTeam	Proj. Mgmt	Technique	FDD
Inspections	Tests	Technique	FDD
RegularBuildSchedule	Config. and Change Mgmt	Technique	FDD
VersionControl	Config. and Change Mgmt	Technique	FDD
VisibilityOfResults	Proj. Mgmt	Technique	FDD
DevelopAnOverallModel	Analysis and Design	Process	FDD
BuildAFeaturesList	Requirements	Process	FDD
PlanByFeature	Proj. Mgmt	Process	FDD
DesignByFeature	Analysis and Design	Process	FDD
BuildByFeature	Implementation	Process	FDD
RegressionTesting	Test	Technique	J. Hartmann
FrequentSafeDeployment	Config. and Change Mgmt	Process	J. Hartmann
MapTheKnownTerritory	Analysis and Design	Technique	J. Hartmann
SoftwareConfigMgmt	Config. and Change Mgmt	Process	CMM
DocumentedChangeMgmt	Config. and Change Mgmt	Process	CMM
ChangesToWorkProducts AreControlled	Config. And Change Mgmt	Process	CMM
AffectedGroupsAre InformedOfChanges	Config. And Change Mgmt	Process	CMM
StatusIsRecorded	Config. And Change Mgmt	Process	CMM
RequirementsAreValidated	Requirements	Process	CMM
StablishResponsabilityFor Requirements	Requirements	Process	CMM
AlocateTheRequirements	Requirements	Process	CMM
TrainingOnRequirements Management	Requirements	Process	CMM
ReviewAndAlocateChanges ToRequirements	Requirements	Process	CMM
DocumentSoftware Estimates	Proj. Mgmt	Process	CMM
NegociateTheProject Commitments	Proj. Mgmt	Process	CMM
SeniorManagementReview	Proj. Mgmt	Process	CMM
DocumentedAndApproved StatementOfWork	Proj. Mgmt	Process	CMM
SoftwareLifeCycleIsDefined	Proj. Mgmt	Process	CMM
SubContractManager	Proj. Mgmt	Role	CMM

Org. Pattern	Process Discipline	Mechanism	Source
DocumentedSubContractor Selection	Proj. Mgmt	Process	CMM
SQAGroup	Test	Role	CMM
PeerReviews	Test	Process	CMM

Os padrões listados na tabela 9.1 foram obtidos ou elaborados a partir de várias fontes. As principais fontes utilizadas foram o *web site* de James Coplien [COP 2004], o método Scrum publicado na forma de padrões organizacionais por Mike Beedle [BEE 99], as páginas Wiki que descrevem práticas de *Extreme Programming* (XP) [BEC 2004], e os papéis, práticas e processos do método *Feature-Driven Development* (FDD) [PAL 2002]. Alguns padrões foram elaborados pelo próprio autor do trabalho com base em sua experiência (J. Hartmann), e outros ainda foram elaborados a partir de práticas e processos sugeridos pelo CMM [SEI 95].

Considerando-se que o CMM diz às organizações o que fazer em termos gerais, mas não mostra exatamente como fazer isto [PAU 2001], acredita-se ser válida a elaboração de padrões de processos a partir do CMM. Com um padrão de processos, procura-se capturar os aspectos mais importantes de uma solução para um determinado problema de processos que é recorrente. As metas, os compromissos, as habilidades, as atividades, as medidas e as verificações que são definidas pelo CMM para cada área-chave de processo (KPA) podem ser capturadas em padrões de processos. Desta forma, podem ser adaptadas para resolver problemas nas situações particulares dos projetos onde forem aplicados. A vantagem desta elaboração é que proporciona que os processos do CMM e as técnicas e processos das demais metodologias sejam descritos em um formato comum, sendo possível assim combiná-los.

A tabela 9.1 mostra a lista dos padrões organizacionais que foram cadastrados no repositório da ferramenta PMT-Tool para exemplificar a utilização da abordagem PMT. Os padrões não devem ser vistos somente de forma isolada, no entanto, por isso é importante a captura dos seus relacionamentos. Desta forma, pode-se imaginar o repositório de padrões como uma linguagem de padrões, e não somente como um catálogo. Para relacionar os padrões, foram utilizados os tipos de associação descritos na seção 6.3 deste trabalho: *dependência, generalização / especialização, alternativa, similaridade, composição* e *trabalho em conjunto*. A tabela 9.2 mostra os relacionamentos que foram identificados entre os padrões. Alguns destes relacionamentos já existiam na fonte dos padrões, ou seja, já eram descritos na publicação original do padrão. Outros relacionamentos foram identificados pelo próprio autor deste trabalho. Os relacionamentos também foram cadastrados na ferramenta PMT-Tool e contribuíram para formar o repositório de padrões utilizado neste exemplo.

Tabela 9.2: Relacionamentos entre os padrões no repositório de exemplo

#	Source Pattern	Target Pattern	Relationship Type
1	*EngageQA*	*DemoAfterSprint*	Dependency
2	*NamedStableBases*	*Sprint*	Dependency
3	*Backlog*	*Sprint*	Dependency
4	*Sprint*	*DemoAfterSprint*	Dependency

#	Source Pattern	Target Pattern	Relationship Type
5	*Firewall*	*ScrumMaster*	Specialization
6	*ScrumMaster*	*ScrumMeetings*	Dependency
7	*ScrumTeam*	*ScrumMeetings*	Dependency
8	*Backlog*	*ScrumMeetings*	Dependency
9	*DeveloperControlsProcess*	*ScrumTeam*	Dependency
10	*HolisticDiversity*	*ScrumTeam*	Work Together
11	*EarlyAndRegularDelivery*	*RegressionTesting*	Dependency
12	*RegressionTesting*	*UnitTests*	Dependency
13	*RegressionTesting*	*AcceptanceTests*	Dependency
14	*StandUpMeeting*	*ScrumMeetings*	Alternative
15	*PlanningGame*	*SprintPlanningMeeting*	Alternative
16	*SprintPlanningMeeting*	*PlanByFeature*	Alternative
17	*ProjectVelocity*	*TrackerRole*	Dependency
18	*CodeOwnership*	*CollectiveCodeOwnership*	Specialization
19	*CodeOwnership*	*IndividualCodeOwnership*	Specialization
20	*CollectiveCodeOwnership*	*IndividualCodeOwnership*	Alternative
21	*IndividualCodeOwnership*	*ClassCodeOwnership*	Specialization
22	*ClassCodeOwnership*	*BuildByFeature*	Dependency
23	*ClassCodeOwnership*	*DesignByFeature*	Dependendy
24	*BuildByFeature*	*DesignByFeature*	Work Together
25	*BuildByFeature*	*FeatureTeam*	Dependency
26	*Inspections*	*PairProgramming*	Alternative
27	*ScenariosDefineProblem*	*BuildAFeaturesList*	Alternative
28	*Inspections*	*PeerReviews*	Alternative
29	*DocumentedSubContractor Selection*	*SubContractManager*	Dependency
30	*StatusIsRecorded*	*DocumentedChangeMgmt*	Dependency
31	*AffectedGroupsAreInformed OfChanges*	*ChangesToWorkProducts AreControlled*	Dependency
32	*AcceptanceTests*	*TestDrivenDevelopment*	Work Together
33	*EarlyAndRegularDelivery*	*Refactoring*	Dependency
34	*EarlyAndRegularDelivery*	*ScenariosDefineProblem*	Work Together
35	*TestDrivenDevelopment*	*Inspections*	Work Together
36	*Inspections*	*DomainObjectModelling*	Dependency
37	*DomainObjectModelling*	*ScenariosDefineProblem*	Work Together
38	*SoftwareConfigMgmt*	*DocumentedChangeMgmt*	Composition
39	*SoftwareConfigMgmt*	*ChangesToWorkProducts AreControlled*	Composition
40	*SoftwareConfigMgmt*	*AffectedGroupsAre InformedOfChanges*	Composition
41	*SoftwareConfigMgmt*	*StatusIsRecorded*	Composition

Os relacionamentos da tabela 9.2 de 1 a 9 foram retirados do método Scrum [BEE 99], enquanto que os demais foram identificados pelo autor do trabalho. Alguns destes relacionamentos merecem comentário. O padrão organizacional *Firewall* descreve um papel importante em times de desenvolvimento de *software*, no qual um

membro do time de desenvolvimento tem a missão de interagir com os clientes e "bloquear" requisições indesejadas, de forma a não perturbar a concentração do restante do time. Uma especialização mais completa deste papel é a descrição do padrão *ScrumMaster*, o qual utiliza-se de outras definições do método Scrum, como *Sprint* e *Backlog,* para realizar o seu trabalho.

Um time de Scrum (*ScrumTeam*) caracteriza-se por ser um time multidisciplinar, caracterizado por uma diversidade holística (*HolisticDiversity*) entre seus membros – estes dois padrões trabalham em conjunto. Por exemplo, um time deste tipo pode incluir uma pessoa especializada em testes, dois programadores, um analista, dois usuários e um projetista gráfico.

Alguns padrões descrevem soluções alternativas para um mesmo problema. Em alguns casos, a solução apresentada é semelhante. Os padrões *StandUpMeeting* (método XP) e *ScrumMeetings* (método Scrum) são alternativas para a organização de reuniões diárias de uma equipe de desenvolvimento. Os padrões *PlanningGame* (método XP) e *SprintPlanningMeeting* (método Scrum) são alternativas para reuniões de planejamento de iterações. Em ambos os casos, a solução proposta é bem semelhante. Em outros casos, as alternativas propostas pelos padrões são soluções diferentes. Este é o caso ao se decidir entre *CollectiveCodeOwnership* (propriedade coletiva do código-fonte) e *ClassCodeOwnership* (propriedade de classes do código-fonte). Também é o caso de, para melhorar a qualidade do código-fonte, decidir-se entre a realização de inspeções formais (*Inspections*) ou se fazer programação em pares (*PairProgramming*). É o caso ainda, ao se decidir entre a definição dos requisitos na forma de cenários (*ScenariosDefineProblem*) ou na forma de uma lista de características (*BuildAFeaturesList*).

Uma vez havendo-se preenchido a linguagem de padrões base, ou seja, o repositório de padrões, o próximo passo é o cadastro de uma lista de riscos que podem ser avaliados nos projetos e associados com os padrões através das regras de resolução de riscos. Para a elaboração do exemplo desta seção, utilizou-se como base o *checklist* de riscos descrito na seção 6.4, o qual foi obtido ao se compilar o trabalho de vários autores. A lista de riscos foi completada com alguns outros riscos, os quais são minimizados ou prevenidos por alguns dos padrões utilizados neste exemplo. Para vincular os riscos aos padrões, foram cadastradas regras de resolução de riscos. Cada regra de resolução de riscos associa um ou mais riscos a um ou mais padrões que os resolvem, e está associada a um contexto de projeto. A tabela 9.3 lista as regras de resolução de riscos que foram cadastradas para a realização do exemplo.

Tabela 9.3: Regras de resolução de riscos definidas para o exemplo. Cada regra é aplicada em um contexto de projeto (PC) e tem um fator de eficácia (AF).

#	Comentários	Criticalidade dos defeitos	Tamanho da equipe	Habilidade da equipe
A	Características ágeis	2 (Dinheiro discreto)	1 (1-6)	2 (Alta)
B	Características guiadas por planos	5 (Muitas vidas)	4 (41-500)	4 (Baixa)
C	Intermediário	3 (Dinheiro essencial)	3 (21-40)	3 (Média)

#	Risco(s)	PC	Padrão(ões)	AF

#	Risco(s)	PC	Padrão(ões)	AF
1	Falta de compromisso da gerência sênior com o projeto.	A	*SprintPlanningMeeting* *PlanningGame*	7
2	Falta de compromisso da gerência sênior com o projeto.	B	*SeniorManagementReview*	5
3	Instabilidade dos requisitos; Não entendimentos dos requisitos; Conflitos entre os departamentos dos usuários.	A	*EarlyAndRegularDelivery* *ScenariosDefineProblem*	8
4	Não entendimento dos requisitos.	B	*ScenariosDefineProblem* *RequirementsAreValidated* *DocumentedChangeMgmt*	6
5	Requisitos criados pelos desenvolvedores (*gold plating*).	C	*EarlyAndRegularDelivery* *ScenariosDefineProblem* *BuildAFeaturesList*	6
6	Falta de envolvimento do usuário	A	*OnSiteCustomer*	9
7	Falta de envolvimento do usuário	C	*ScrumTeam* *HolisticDiversity*	5
8	Requisitos criados pelos desenvolvedores (*gold plating*).	A	*DoTheSimplestThing*	8
9	Custo do redesenho ser muito alto	A	*DevelopAnOverallModel* *MapTheKnownTerritory*	4
10	Custo do redesenho ser muito alto	B	*DevelopAnOverallModel* *Inspections* *RequirementsAreValidated*	5
11	Dificuldade de comunicação/conflitos entre diferentes funções (usuários, analistas, programadores, testadores)	C	*HolisticDiversity* *ScrumTeam*	8
12	Falta de uma medida realista sobre o progresso do projeto	C	*Sprint* *DemoAfterSprint*	10
13	Conflito entre os departamentos do usuários	C	*ProductOwner*	6
14	Cronograma e orçamentos não realistas	C	*SizeTheSchedule*	2
15	Falha em gerenciar a expectativa do usuário	A	*SprintPlanningMeeting* *DemoAfterSprint*	7

#	Risco(s)	PC	Padrão(ões)	AF
16	Falha em gerenciar a expectativa do usuário	B	*RequirementsAreValidated*	5
17	Introdução de nova tecnologia	A	*SpikeSolutions* *DoTheSimplestThing* *PairProgramming*	6
18	Utilização de recursos e performance do sistema	A	*WorstThingsFirst* *SpikeSolutions*	4
19	Desenvolvimento errado das funções ou interfaces	C	*AcceptanceTests* *TestDrivenDevelopment* *Refactoring*	8
20	Saída de pessoal da equipe (*turn-over*)	C	*CompensateSuccess* *PairProgramming* *ScrumTeam* *HolisticDiversity*	5
21	Projeto (desenho) inviável	C	*EarlyAndRegularDelivery* *Inspections* *Refactoring*	6
22	Falta de conhecimento/ habilidade pela equipe	C	*ChiefProgrammer* *PairProgramming*	7
23	Cronograma e orçamentos não realistas	C	*DocumentSoftwareEstimates* *NegociateTheProjectCommitments* *DocumentedAndApprovedStatementOf Work*	6
24	Falta de uma metodologia para o projeto	C	*SoftwareLifeCycleIsDefined*	3
25	Sub-contratação	C	*SubContractManager* *DocumentedSubContractorSelection*	5
26	Projeto (desenho) inviável	B	*SQAGroup* *PeerReviews*	6
27	Defeitos que reaparecem	C	*ChangesToWorkProductsAreControlled* *StatusIsRecorded* *RegressionTesting*	8

9.2 Projeto X

O projeto X é um projeto de desenvolvimento de *software* típico dos dias atuais. Seu objetivo é a construção de um novo sistema de informação para uma grande empresa de Internet. A equipe de desenvolvimento tem um tamanho médio (15 integrantes), e a criticalidade do sistema a ser desenvolvido também é média: um defeito em algumas partes do sistema pode causar a perda de dinheiro essencial para a companhia; outras partes deste sistema serão menos críticas. A equipe de desenvolvimento já está montada, e a sua habilidade foi medida como sendo médio-alta. A empresa cliente precisa de algumas funcionalidades do sistema com alguma pressa, pois estas são consideradas vantagem competitiva sobre os seus concorrentes. No entanto, o gerente de tecnologia e o gerente de operações divergem quanto ao que esperam destas funcionalidades. O mercado em que o cliente atua é dinâmico; várias

novas necessidades surgem a cada dia e precisam ser incorporadas aos requisitos do sistema.

Uma tentativa de implementação do sistema com uma equipe interna apoiada por alguns consultores não foi bem sucedida. A implementação foi cancelada após seis meses, deixando como resultado várias páginas de documentação com modelos de bancos de dados – considerada incompleta em algumas partes e muito complexa em outras partes pelo gerente de tecnologia - e descrições detalhadas dos processos da empresa – consideradas já ultrapassadas pelo gerente de operações, mas nenhum *software* rodando. Os conflitos entre os departamentos dos usuários, o projeto do banco de dados sendo inviável, a instabilidade dos requisitos e a pressa em ter-se o sistema rodando, aliados ao alto custo que estava sendo previsto para o projeto, foram considerados os principais motivos para o cancelamento.

A gerência sênior decidiu apostar em uma nova abordagem inovadora, chamada PMT – *Pattern-based Methodology Tailoring*, a qual consegue combinar diversas metodologias de uma forma customizada para um projeto, e promete bons resultados. O projeto foi reiniciado.

Tabela 9.4: Lista de riscos identificados e priorizados para o projeto X. P é a probabilidade do risco ocorrer, L é a perda que o risco pode causar, e RE é a exposição do risco resultante.

Risco	**Tipo**	**P**	**L**	**RE**
Projeto (desenho) inviável	Execução	8	7	56
Conflito entre os departamentos dos usuários	Cliente	8	6	48
Instabilidade de requisitos	Requisitos	8	5	40
Não-entendimento dos requisitos	Requisitos	8	4	32
Falha em obter compromisso dos usuários	Cliente	6	5	30
Desenvolvimento errado das funções ou interfaces	Execução	7	4	28

Segundo a abordagem PMT (ver seção 6.4), para definir-se a metodologia do projeto, é necessário inicialmente que se realize uma análise de riscos. Primeiramente, os riscos foram identificados, o que foi feito de três formas: *1. consultando-se o checklist de riscos da abordagem PMT. 2. realizando-se reuniões com os envolvidos no projeto, de forma a se levantar quais os riscos envolvidos. 3. analisando-se os principais problemas que causaram o cancelamento do projeto anterior*. O segundo passo foi, para cada um dos riscos identificados, determinar o valor da exposição do risco (RE), que é o produto da probabilidade do risco ocorrer (P) e da perda que pode causar (L). O terceiro passo foi, ainda, determinar o fator de priorização de riscos (RPF – *Risk Prioritization Factor*), de forma a considerar na análise, ao menos no primeiro momento, os riscos mais relevantes. O resultado é apresentado na tabela 9.4.

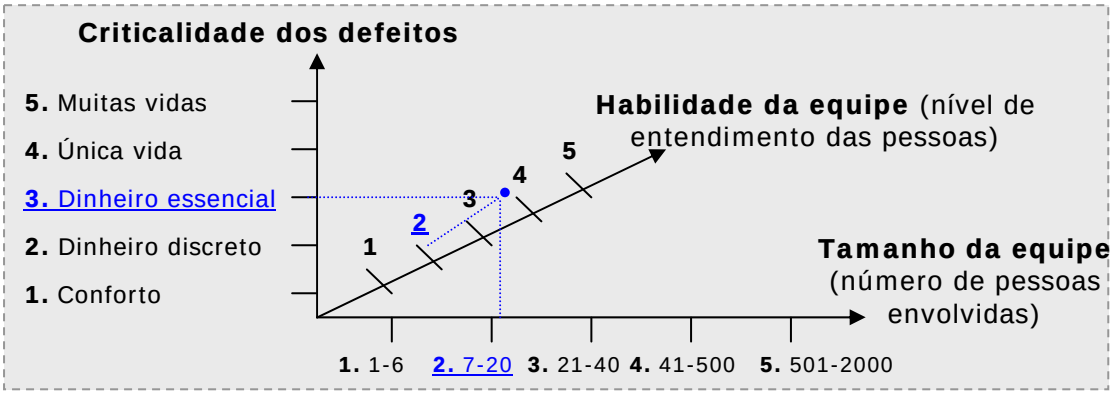


Figura 9.1: Determinando o contexto de criticalidade do projeto X

Foi considerado como fator de priorização de riscos (RPF) para o projeto X o valor 25. Assim, somente os riscos com valor de exposição maior ou igual a este valor foram selecionados. Os riscos cujo valor de exposição é menor do que o valor RPF não foram considerados relevantes para a análise, e por isso não serão considerados pelo mecanismo de seleção de padrões neste momento.

Outro ponto é a determinação do contexto de criticalidade do projeto X. Conforme já mencionado, o projeto X tem uma equipe composta por 15 integrantes, portanto seu tamanho é nível 2 (7 a 20 integantes), na escala de 1 a 5, conforme mostra a figura 9.1. A habilidade desta equipe é médio-alta (nível 2), e a criticalidade dos defeitos é media (nível 3), o que significa que defeitos no *software* podem causar a perda de dinheiro essencial para o cliente. A figura 9.1 mostra como que o contexto de criticalidade do projeto X se situa em relação aos três eixos no espaço tridimensional.

9.3 Adaptando a linguagem de padrões para minimizar os riscos do projeto X

Os dados descritos nas seções anteriores – repositório de padrões e análise do projeto X – foram cadastrados na ferramenta PMT-Tool, de forma que se possa tirar proveito do seu mecanismo de seleção de padrões organizacionais.

☐	Pattern	Score
☑	EarlyAndRegularDelivery	53792
☑	Refactoring	40096
☑	ScenariosDefineProblem	36992
☑	Inspections	27552
☑	ProductOwner	23616
☐	PeerReviews	18816
☐	SQAGroup	18816
☑	TestDrivenDevelopment	12544
☑	AcceptanceTests	12544
☐	RequirementsAreValidated	10752
☐	DocumentedChangeManagement	10752

Figura 9.2: Resultado da seleção de padrões

Inicialmente a linguagem de padrões do projeto X se encontrava vazia, sem nenhum padrão. Criou-se uma seção de pesquisa sistemática na ferramenta PMT-Tool, sem restringir a pesquisa por nenhum parâmetro. O resultado é apresentado na figura 9.2. Após o projetista de processos ter navegado pela descrição textual dos padrões e tê-los examinado cuidadosamente, este selecionou um subconjunto dos padrões sugeridos pela ferramenta para adição na linguagem de padrões do projeto X. Os padrões selecionados são os marcados na figura 9.2.

O padrão que recebeu a maior pontuação foi *EarlyAndRegularDelivery*, que estabelece um ciclo de via iterativo e incremental para o desenvolvimento e a entrega das funcionalidades desejadas no projeto X. A boa pontuação se deve à sua presença em duas regras de resolução de riscos, as quais foram bem avaliadas durante o processo de seleção. Como mostra a figura 9.3, ambas as regras tiveram boa pontuação nos quesitos exposição do risco média, proximidade do contexto, e fator de eficácia. Isto significa que este padrão contribui significativamente para resolver mais de um risco importante do projeto, e é tipicamente aplicado em projetos com contexto de criticalidade semelhantes ao do projeto X.

Score Detail for Pattern *EarlyAndRegularDelivery*:

Rule #	Risk(s)	Pattern(s)	Avg. Risk Exposure	Context Proximity	Accuracy Factor	Rule Score
106374	Infeasible design	Inspections Refactoring EarlyAndRegularDelivery	56	82	6	**27552**

Rule #	Risk(s)	Pattern(s)	Avg. Risk Exposure	Context Proximity	Accuracy Factor	Rule Score
106356	Misunderstading the requirements Conflict between user departments Requirements instability	ScenariosDefineProblem EarlyAndRegularDelivery	40	82	8	**26240**

Total score for the pattern: **53792**

Figura 9.3: Detalhamento da pontuação do padrão EarlyAndRegularDelivery

Os padrões selecionados pelo usuário foram adicionados à linguagem de padrões do projeto X. De forma a completar a metodologia inicial do projeto, o projetista de processos selecionou ainda o padrão *RegressionTesting*, pois este padrão auxilia a implementar o desenvolvimento iterativo e incremental descrito em *EarlyAndRegularDelivery*. O padrão *DomainObjectModeling* também foi acrescentado, após o projetista de processos ter navegado na linguagem de padrões base (repositório). A linguagem de padrões resultante é apresentada na figura 9.4.

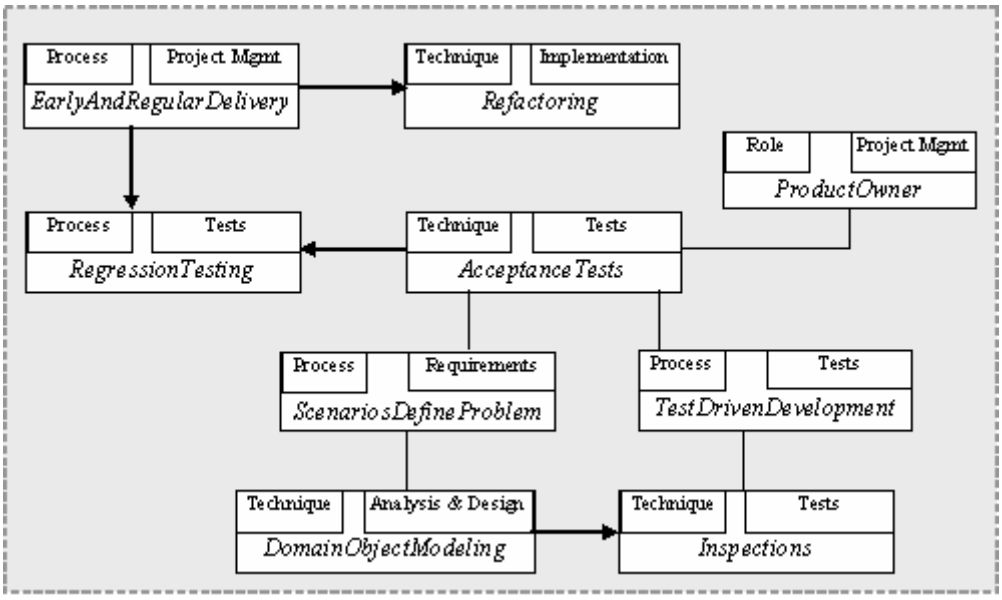


Figura 9.4: Linguagem de padrões resultante

A linguagem de padrões foi utilizada pelo time do projeto X como metodologia de desenvolvimento. Após 6 meses o projeto foi considerado um sucesso. Neste período, foram realizadas 3 grandes liberações de funcionalidades, e os usuários a partir da primeira liberação já foram beneficiados. O sucesso foi creditado ao fato de a metodologia de desenvolvimento ter conseguido suavizar os principais

riscos que poderiam afetar o projeto. O papel do *ProductOwner* (gerente de produto) foi decisivo para resolver os conflitos entre os departamentos dos usuários. A instabilidade dos requisitos foi endereçado através do desenvolvimento iterativo e incremental (*EarlyAndRegularDelivery*), que permitiu aos usuários visualizar o *software* funcionando cedo no ciclo de vida do projeto, e assim sugerir mudanças que foram incorporadas nas versões seguintes.

O modelo de ciclo de vida iterativo e incremental foi viabilizado com um foco grande em testes de regressão (*RegressionTesting*) pela equipe de desenvolvimento, e a técnica de redesenho (*Refactoring*) foi importante para permitir acomodar as mudanças que foram requisitadas pelos usuários.

Já o risco de o projeto ser inviável foi suavizado através da realização de inspeções nos modelos de classes e no código-fonte, e também pelas liberações freqüentes. Como foi completada e liberada uma versão do *software* cedo no ciclo de vida do projeto, os problemas de projeto existentes foram detectados e corrigidos nas versões seguintes, o que também foi viabilizado pela técnica de redesenho.

Mesmo tendo sido considerada a primeira fase do projeto um sucesso, alguns problemas foram identificados. Defeitos que haviam sido detectados e corrigidos reapareceram em versões posteriores. Foi detectado que a causa destes problemas era a falta de processos de gerência de configuração, o que causou algum desgaste com os usuários do sistema. Considerando que este problema seria um risco para a fase seguinte do projeto, o gerente de projetos realizou uma nova sessão de seleção utilizando a ferramenta PMT-Tool.

Antes de realizar a nova seleção de padrões, a análise de riscos do projeto X foi revisada, sendo acrescentado o risco de “defeitos que reaparecem” com probabilidade 8, perda 6, sendo sua exposição de risco resultante 48. A sessão de seleção desta vez foi parametrizada, como mostra a figura 9.5, procurando-se especificamente por processos de gerência de configuração. O resultado também é apresentado na figura 9.5.

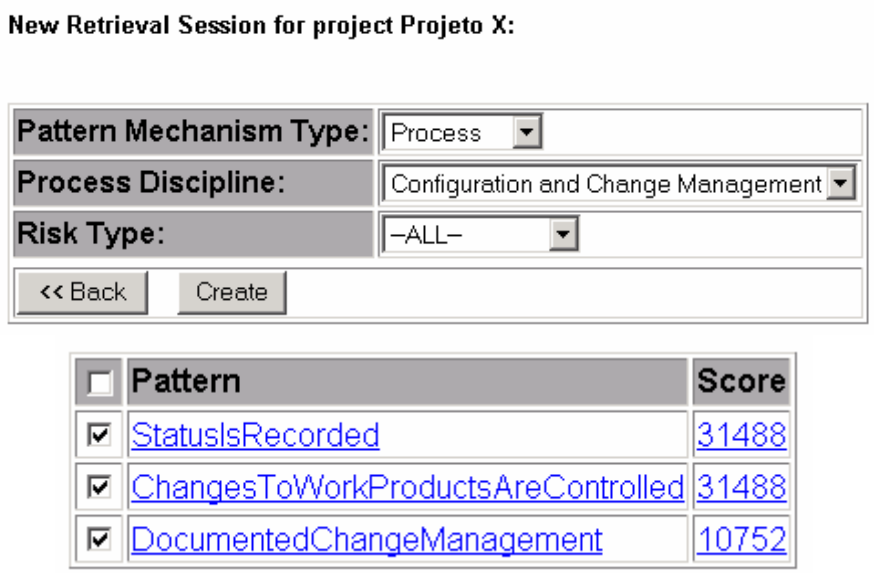


Figura 9.5: Realizando uma nova seleção por padrões de processos de gerência de configuração

O resultado da nova seleção realizada foram três padrões de processos os quais foram elaborados a partir do CMM (área-chave de processo de SCM – *Software Configuration Management*). A figura 9.6 mostra o detalhamento do cálculo da pontuação de um dos padrões selecionados (*StatusIsRecorded*).

Score Detail for Pattern *StatusIsRecorded*:

Rule #	Risk(s)	Pattern(s)	Avg. Risk Exposure	Context Proximity	Accuracy Factor	Rule Score
107368	Defects coming back	StatusIsRecorded ChangesToWorkProductsAreControlled RegressionTesting	48	82	8	**31488**

Total score for the pattern: **31488**

Figura 9.6: Detalhamento da pontuação do padrão *StatusIsRecorded*

Os três padrões selecionados foram acrescentados à linguagem de padrões do projeto X. A metodologia resultante tornou-se uma combinação eficiente, empregando processos rigorosos para controlar as mudanças sobre os produtos de trabalho, e utilizando técnicas de metodologias ágeis para lidar com a instabilidade dos requisitos do *software*. A segunda fase do projeto X foi considerada um sucesso ainda maior do que a primeira.

10 CONCLUSÕES

Este trabalho descreveu PMT – *Pattern-based Methodology Tailoring*, uma abordagem para adaptar metodologias de desenvolvimento de *software* às necessidades de um determinado projeto, utilizando padrões (*patterns*) e critérios de risco. A abordagem estrutura um repositório de padrões como uma base de conhecimento sobre práticas de sucesso para gerenciar o trabalho em projetos de *software*. Padrões organizacionais são eficientes para a documentação de metodologias e de processos, por descreverem a parte essencial de uma solução para um problema recorrente, o qual pode ser adaptado para resolver o problema específico de um projeto de *software*. Podem evitar que documentos de processos se tornem burocráticos e ineficientes, que não refletem os processos que são realmente realizados, tem descrições que são ambíguas ou incompreensíveis, são muito alto nível para serem utilizados na prática, que é uma situação encontrada em muitas organizações [VAS 98].

PMT estrutura o repositório de padrões como sendo uma grande linguagem ou "sistema" de padrões. Os padrões são classificados por seu mecanismo de solução do problema – processo, papel, técnica ou valores do time – e pela disciplina de processos em que atuam - Modelagem de Negócios, Requisitos, Análise e Projeto, Implementação, Testes, Liberação, Gerenciamento de Configuração e de Mudanças, Gerenciamento de Projetos, e Ambiente. Esta classificação é útil para que o projetista de processos possa mais facilmente encontrar um padrão que resolva um problema em particular, quando pesquisando no repositório. Os padrões também são relacionados uns aos outros, através da utilização de referências (*links*) tipados. Os tipos de referências permitidos são: dependência, alternativa, generalização/especialização, trabalho em conjunto, e composição.

O repositório de padrões deve ser elaborado a partir de fontes já existentes de padrões organizacionais, como os padrões de Coplien [COP 95] [COP 2004] e de Harrison [HAR 96b], os padrões da metodologia Scrum [BEE 99], a linguagem de padrões para teste de DeLano e Rising [DEL 96], e os padrões para reengenharia de sistemas de Stevens e Poley [STE 98]. Também deve considerar o conhecimento documentado em metodologias publicadas, como Extreme Programming [BEC 99], Rational Unified Process [JAC 2000], e o CMM [SEI 95]. Padrões também podem ser obtidos ao se documentar a experiência de gerentes de projetos de *software* ou projetistas de processos ao resolver problemas recorrentes. No entanto, é necessária uma consideração cuidadosa ao se adicionar um novo padrão ao repositório, de forma a manter a sua qualidade como uma fonte de informação eficiente e significativa.

Consideração especial também é necessária ao se adicionarem novas regras de resolução de riscos ao repositório, as quais são utilizadas pelo mecanismo de seleção

para escolher os padrões que melhor resolvem os riscos de um projeto. As regras devem ser elaboradas através da experiência dos gerentes de projetos ao resolver riscos de projetos de *software*, ou a partir do conhecimento existente publicado na literatura.

PMT também enfatiza a necessidade de se avaliar os requisitos metodológicos de um projeto antes de definir a metodologia a ser utilizada com o mesmo. Isto é feito através de uma abordagem sistemática para a identificação e análise dos riscos do projeto, e de um arcabouço para definir o contexto de criticalidade do mesmo. Os riscos são identificados a partir de um *checklist* também armazenado no repositório, que serve como base para o processo de identificação. A lista de riscos foi preenchida inicialmente a partir da compilação do trabalho de vários autores [FON 2004]. A identificação dos riscos não deve se restringir ao *checklist*, mas sim deve procurar identificar todos os riscos que podem vir a afetar o projeto. Os problemas identificados em projetos passados da mesma organização, tanto de sucesso quanto de falha, através de análises *post-mortem*, são uma boa fonte de riscos potencias para um novo projeto [DEM 2003].

Ao se utilizar uma análise objetiva dos riscos de um projeto como critério para a montagem da sua metodologia, evita-se que as decisões neste processo se baseiem somente na experiência e no bom senso do projetista de processos, ou nos seus medos subjetivos e intangíveis. Desta forma, evita-se que a metodologia inclua processos e práticas desnecessárias, que aumentariam os custos do projeto, mas contribuiriam pouco para o seu sucesso. A abordagem não substitui o papel do projetista de processos, no entanto, mas sim o auxilia. O projetista de processos continua tendo uma função muito importante, e a sua experiência e consideração cuidadosa na adaptação e montagem de uma metodologia são muito bem-vindas. A vantagem é que se pode contar com uma base de conhecimento – montada a partir de experiências de sucesso – e com um mecanismo sistemático de seleção, o qual sugere as melhores práticas mais adequadas.

O mecanismo sistemático de seleção, implementado na ferramenta PMT-Tool, utiliza a base de conhecimento armazenada no repositório para selecionar padrões que sejam adequados a um determinado projeto. Utiliza como principais critérios o cálculo de exposição do risco e a proximidade do contexto de criticalidade. A exposição do risco é calculada para cada um dos riscos que foram identificados para o projeto, sendo o produto da quantificação da probabilidade de o risco ocorrer, com a perda que este pode causar caso ocorra. O contexto de criticalidade de um projeto é determinado em três dimensões: criticalidade dos defeitos, tamanho do projeto em número de pessoas, e habilidade da equipe. Ao mesmo tempo, cada regra de resolução de risco está associada ao contexto em que funciona melhor. Por isso, calcula-se o fator proximidade do contexto para cada regra durante o processo de seleção sistemático. Este fator é combinado com a exposição média dos riscos presentes na regra, contribuindo para a pontuação que é calculada para cada padrão associado à regra.

Espera-se que o conhecimento existente no repositório, capturado na forma de padrões, riscos e regras, evolua ao longo do tempo. Neste sentido, a abordagem PMT busca estruturar e gerenciar o conhecimento, mais do que defini-lo. Através de casos de experimentação em projetos, espera-se que a base de conhecimento seja aprimorada. Novos padrões podem ser capturados, na medida em que a ocorrência de soluções recorrentes for observada. O conhecimento a respeito de metodologias também deve evoluir, e novos padrões organizacionais e novas metodologias devem

ser publicadas. Portanto, o objetivo de PMT não é o de substituir outras metodologias, mas sim capturar o seu conhecimento. A lista de riscos também deve crescer bastante, na medida em que os muitos riscos existentes em projetos de *software* forem levantados e associados aos padrões por intermédio das regras de resolução de riscos. Portanto espera-se que, ao aplicar-se PMT em projetos, não somente os projetos se beneficiem do conhecimento capturado pela abordagem, mas também o repositório de PMT se aproveite da experiência do projeto para aprimorar o seu conhecimento.

No tempo em que este trabalho foi escrito, há duas correntes principais relacionadas a metodologias de desenvolvimento de *software*. Uma corrente defende a utilização de processos rigorosos, baseados na produção de muita documentação e artefatos intermediários, como instrumento para melhorar a qualidade do *software* produzido e a previsibilidade dos projetos. Estas metodologias são normalmente guiadas pelo modelo CMM (*Capability Maturity Model*) e são classificadas como guiadas por planos [MAN 2003]. Por outro lado, há a corrente das metodologias ágeis, que valorizam a habilidade das pessoas e das equipes, a criatividade e a inventividade, a incorporação de retroalimentação (*feedback*) durante o projeto, o desenvolvimento iterativo e incremental, e a facilidade em se realizar e absorver mudanças. [ORR 2002] descreve as discussões como “guerras religiosas”. Beck e Boehm debatem sobre o tema criatividade *versus* disciplina [BEC 2003], embora o próprio conceito de disciplina também esteja em discussão. Felizmente, há trabalhos buscando combinar ambas as abordagens [BOE 2003a] [BOE 2003b].

Este trabalho busca colaborar para que as metodologias ágeis e guiadas por planos possam ser combinadas. Dependendo do contexto de criticalidade de um projeto, e de sua análise de riscos, são sugeridos padrões típicos de metodologias ágeis, ou de metodologias guiadas por planos. Ou ainda uma combinação de ambas. No capítulo 8, foi descrito um exemplo no qual utilizou-se de processos rigorosos de gerência de configuração, elaborados a partir do CMM, de forma conjunta com técnicas ágeis, como desenvolvimento incremental e redesenho, para compor a metodologia de um projeto.

Há várias oportunidades para aprimoramento da abordagem que foi desenvolvida neste trabalho, as quais podem ser endereçadas em trabalhos futuros. Um problema ainda em aberto é a questão de validação dos padrões. Existem muitos padrões publicados, que podem servir de fonte para o repositório. No entanto, como comprovar que suas soluções funcionam e que estes são recorrentes? A mesma questão vale para novos padrões que forem elaborados e acrescentados ao repositório. Da mesma forma, as regras de resolução de riscos, que também representam conhecimento armazenado no repositório, precisam ser validadas.

A sugestão para tratar-se a validação de padrões e de regras é utilizar-se de mecanismos de aquisição de conhecimento, como os utilizados pela ferramenta de Girardi [PRI 90], desenvolvida para suporte ao reuso de software em ambientes orientados a objeto. Um mecanismo de busca de classes que integra a procura sistemática (sobre descritores das classes) com a exploração através de navegação é sugerido. São utilizados fatores auto-adaptáveis que refletem a experiência dos usuários na reutilização das classes existentes. Os fatores são obtidos por mecanismos de aquisição de conhecimento, e são incorporados no mecanismo sistemático de busca. Da mesma forma, a ferramenta PMT-Tool poderia ser estendida com mecanismos de aquisição de conhecimento, na forma de retroalimentação (*feedback*) por parte do usuário. Desta forma, a experiência do usuário na utilização dos padrões e das regras poderia ser computada por fatores auto-adaptáveis, os quais seriam

aprimorados ao longo do tempo, conforme as experiências fossem reportadas. Estes fatores poderiam ser considerados pelo mecanismo sistemático de seleção, de forma a valorizar os padrões e as regras mais comprovadas em seus resultados.

A ferramenta PMT-Tool pode ser estendida também com uma funcionalidade de representação gráfica, de forma a montar diagramas na notação da abordagem PMT conforme definido neste trabalho. Isto permitiria aos usuários uma visualização mais global de uma linguagem de padrões. Ao clicar-se sobre o símbolo de um padrão, poderia se navegar para a descrição textual do mesmo, como no *web site* de Coplien [COP 2004].

Outra oportunidade para aprimoramento do trabalho está relacionada à gerência de riscos. Este trabalho se baseou no enfoque sistemático para o tratamento de riscos desenvolvido por Fontoura [FON 2004], o qual resumidamente propõe as seguintes atividades: *a. elaborar lista de riscos, analisar e priorizar os riscos; b. selecionar e elaborar ações preventivas; c. identificar um conjunto de métricas e monitorar os riscos; d. desenvolver mecanismos de capturar métricas associadas ao processo; e. monitorar as métricas continuamente, para tomar ações corretivas caso o progresso desvie do esperado*. Somente as atividades a e b foram endereçadas, não foi considerada a necessidade de monitoramento dos riscos durante a execução do projeto. Este monitoramento poderia ser feito através da coleta de métricas, utilizando o paradigma *Goal/Question/Metric* (GQM) para monitorar os riscos.

Ainda, há arcabouços para processos de desenvolvimento de *software* orientado a objetos, como OPEN [HEN 99] e RUP [JAC 2000], os quais definem a atividade de adaptação de um processo de desenvolvimento a um determinado projeto como uma "instanciação". A partir de um meta-modelo de processos, é criada uma instância de processo específica para o projeto desejado. As decisões envolvidas neste processo de instanciação, no entanto, são deixadas a cargo do projetista de processos. Assim, entende-se que uma abordagem semelhante a PMT poderia ser desenvolvida para auxiliar o projetista de processos nas decisões de instanciação, as quais também seriam baseadas em uma análise dos riscos do projeto.

Uma outra possibilidade a ser explorada é que os padrões de processos utilizados em PMT poderiam descrever sua solução utilizando o meta-modelo de RUP ou OPEN. Desta forma, imagina-se que um trabalho semelhante a abordagem de Dantas [DAN 2002], estudada em [HAR 2004], seria possível. Dantas desenvolveu um ambiente de suporte à utilização de padrões para o projeto orientado a objetos. Este ambiente suporta principalmente a instanciação de padrões (em que um padrão é adicionado a um modelo), e a detecção de padrões e anti-padrões em um projeto já existente. Um ambiente semelhante poderia ser desenvolvido, mas para o suporte a utilização de padrões em modelos de processos de desenvolvimento de *software*. Neste ambiente, um modelo de processos (instância do meta-modelo) poderia ser mantido, e padrões poderiam ser adicionados, combinando suas estruturas com as já existentes no modelo, ou então criando novas estruturas. Poderiam também ser detectados padrões e anti-padrões neste modelo de processos.

Finalmente, PMT poderia ser estendida para suportar esforços de melhoria de processos de *software*, como os guiados pelo CMM (*Capability Maturity Model*) [SEI 95]. Os padrões existentes no repositório poderiam estar associados às áreas-chave de processos (KPAs) definidas por este modelo. Assim, a ferramenta PMT-Tool poderia ser estendida com uma funcionalidade de seleção de padrões por nível de maturidade CMM. Ou então, sobre uma linguagem de padrões já existente, representando a metodologia de um determinado projeto, a ferramenta poderia sugerir padrões

organizacionais e de processos que completassem as necessidades de atendimento de um determinado nível de maturidade. E isto sempre considerando o contexto de criticalidade e os riscos do projeto.

REFERÊNCIAS

[ABR 2002] ABRAHAMSSON, P. et al. **Agile software development methods. Review and analysis**. 2002. Disponível em <http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf>. Acesso em: 23 mar 2003.

[ADD 2002] ADDISON, T.; VALLABH, S. Controlling Software Project Risks – An Empirical Study of Methods used by Experienced Project Managers. In: ANNUAL CONFERENCE OF THE SOUTH AFRICAN INSTITUTE OF COMPUTER SCIENTISTS & INFORMATION TECHNOLOGIES, 2002, Port Elizabeth. **Proceedings…** Port Elizabeth: ACM International Conference Proceeding Series, 2002. p. 128-140.

[ALE 77] ALEXANDER, C.A. et al. **A Pattern Language.** New York: Oxford Univ. Press, 1977.

[APP 97a] APPLETON, B. Patterns and Software: Essential Concepts and Terminology. **Object Magazine Online**, [S.l.], v.3, n.5, May 1997. Disponível em <http://www.enteract.com/-bradapp/docs>. Acesso em: 7 ago 2004.

[APP 97b] APPLETON, B. **Patterns for Conducting Process Improvement**. 1997. Disponível em: <http://www.cmcrossroads.com/bradapp/docs/i-spi/plop97.html>. Acesso em: 7 ago 1997.

[BAS 75] BASILI, V. R.; TURNER, A. J., Iterative Enhancement: A Practical Technique for Software Development. **IEEE Transactions on Software Engineering**, Los Alamitos, v. 1, n. 4, December 1975.

[BAS 88] BASILI, V. R; ROMBACH, D. The TAME Project: towards improvement-oriented software environments. **IEEE Transactions on Software Engineering**, Los Alamitos, v.14, n.6, p. 758-773, June 1988.

[BEC 94] BECK, K.; JOHNSON, R. Patterns Generate Architectures. In: EUROPEAN CONFERENCE ON OBJECT ORIENTED

PROGRAMMING, ECOOP, 8., 1994, Bologna, It. **Object-oriented Programming**. Berlin: Springer-Verlag, 1994.

[BEC 99] BECK, K. Embracing change with Extreme Programming. **IEEE Computer**, Los Alamitos, v. 32, n.10, p. 70-77, October 1999.

[BEC 2000] BECK, K.; FOWLER, M. **Planning Extreme Programming**. [S.l.]: Addison-Wesley, 1977.

[BEC 2001] BECK, K. et al. **The Agile Manifesto**. 2001. Disponível em <http://agilemanifesto.org>. Acesso em 7 ago. 2004.

[BEC 2003] BECK, K.; BOEHM, B. Agility through Discipline: a debate. **IEEE Computer**, Los Alamitos, v. 36, n. 6, p. 44 - 46, June 2003.

[BEC 2004] BECK, K.; CUNNINGHAM, W. **Wiki pages about Extreme Programming**. 2004. Disponível em: <http://c2.com/cgi/wiki?ExtremeProgramming>. Acesso em: 1 nov. 2004.

[BEE 97] BEEDLE, M. cOOherentBPR – A pattern language to build agile organizations. In: COPLIEN, J.; SCHIMIDT, D. (Ed.) **Pattern Languages of Program Design**. New York: Addison-Wesley, 1997.

.

[BEE 99] BEEDLE, M. et al. SCRUM: An extension pattern language for hyperproductive software development. In: VLISSIDES, J. M.; COPLIEN, J. O.; KERTH, N. L. (Ed.). **Pattern Languages of Program Design 4**. Reading: Addison-Wesley, 1999.

[BOE 91] BOEHM, B. Software Risk Management: Principles and Practices. **IEEE Software**, Los Alamitos, v. 8, n.1, January/February 1991.

[BOE 2002] BOEHM, B. Get Ready for Agile Methods, with Care. **IEEE Computer**, Los Alamitos, v.35, n.1, p. 64-69, January 2002.

[BOE 2003a] BOEHM, B.; TURNER, R. Using Risk to Balance Agile and Plan-Driven Methods. **IEEE Computer**, Los Alamitos, v. 36, n. 6, p. 57-66, June 2003.

[BOE 2003b] BOEHM, B.; TURNER, R. **Balancing Agility and Discipline:** a guide for the perplexed. New York: Addison-Wesley, 2003.

[C3T 98] C3 TEAM. **Chrysler goes to "Extremes"**. 1998. Disponível em <http://www.xprogramming.com/publications/dc9810cs.pdf>. Acesso em: 14 mar 2004.

[COA 92] COAD, P. Object-oriented patterns. **Communications of the ACM**, New York, v. 35, n. 9, p. 152-159, September 1992.

[COA 99] COAD, P. et al. **Java Modeling in Color with UML**. [S.l.]: Prentice Hall, 1999.

[COC 2000] COCKBURN, A. Selecting a Project's Methodology. **IEEE Software**, Los Alamitos, v. 17, n. 4, p. 64-71, July/August 2000.

[COC 2001a] COCKBURN, A.; HIGHSMITH, J. Agile Software Development: the people factor. **IEEE Computer**, Los Alamitos, v. 34, n.11, p.131-133, November 2001.

[COC 2001b] COCKBURN, A. **Agile Software Development**. [S.l.]: Addison-Wesley, 2001.

[CON 2002] CONRADI, R.; FUGGETTA, A. Improving Software Process Improvement. **IEEE Software**, Los Alamitos, v. 19, n. 4, p. 92-99, July/August 2002.

[COE 95] COPPENDALE, J. Managing Risk in Product and Process Development and Avoid Unpleasant Surprises. **Engineering Management Journal**, [S. l.], v.5, n.1, p. 35-38, February 1995.

[COP 95] COPLIEN, J. A Development Process Generative Pattern Language. In: COPLIEN, J.; SCHIMIDT, D. (Ed.) **Pattern Languages of Program Design**. New York: Addison-Wesley, 1995.

[COP 96] COPLIEN, J. **Sofware Patterns**. 1996. Disponível em <http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/>. Acesso em: 6 set 2003.

[COP 2004] COPLIEN, J. **Organizational Patterns Web Site**. 2004. Disponível em: <http://www1.bell-labs.com/user/cope/Patterns/Process/OrgPatternsMap.html>. Acesso em: 7 ago 2004.

[CUN 2004] CUNINGHAM, W. **Wiki pages about what are patterns**. Disponível em <http://c2.com/ppr/wiki/WikiPagesAboutWhatArePatterns/html.zip>. Acesso em: 7 ago 2004.

[CPE 95] COPPENDALE, J. Managing Risk in Product and Process Development and Avoid Unpleasant Surprises. **Engineering Management Journal**, [S.l.], v.5, n.1, p. 35-38, February 1995.

[CUN 2004] CUNINGHAM, W. **The WikiWikiWeb**. Disponível em <http://www.c2.com/cgi/wiki>. Acesso em: 2 oct 2004.

[CUN 2005] CUNINGHAM, W. **Wiki pages on Extreme Programming**. Disponível em <http://c2.com/cgi/wiki?ExtremeProgramming>. Acesso em: 23 feb 2005.

[DAN 2002] DANTAS, A. et al. Suporte a Padrões no Projeto de Software. In: SIMPOSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 16., 2002, Gramado. **Anais…** Porto Alegre: Instituto de Informática da UFRGS, 2002.

[DEA 2002] DEARDEN, A. et al. Using Pattern Languages in Participatory Design. In: PATTERNS IN PRACTICE: A WORKSHOP FOR UI DESIGNERS, 2002, [S.l.]. Disponível em <http://www.welie.com/patterns/chi2002-workshop/Dearden-CHIWorkshopPaper.pdf>. Acesso em 7 ago 2004.

[DEL 96] DELANO, D.; RISING, L. **System Test Pattern Language**. Disponível em: <http://www.agcs.com/supportv2/techpapers/patterns/papers/systestp.htm>. Acesso em: 7 ago 2004.

[DEM 2003] DEMARCO, T.; LISTER, T. **Waltzing with Bears:** managing risk on software projects. New York: Dorset House Publishing Company, 2003.

[DEV 2002] DEVEDZIC, V. Software Patterns. In: CHANG, S.K. (Ed.). **Handbook of Software Engineering and Knowledge Engineering**. Singapore: World Scientific Publishing Co., 2002. v. 2.

[FON 2004] FONTOURA, L. M.; PRICE, R. T. Usando GQM para Gerenciar Riscos em Projetos de Software. In: SIMPOSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 18., 2004, Brasília. **Anais...** Recife, Centro de Informática da Universidade Federal de Pernambuco, 2004.

[FOX 97] FOX, M. S.; GRUNINGER, M. **Enterprise Modelling**. Disponível em <http://www.eil.utoronto.ca/enterprise-modelling/index.html>. Acesso em: 10 oct 2003.

[FOW 2001] FOWLER, M.; HIGHSMITH, J. The Agile Manifesto. **Software Development Online**, [S.l.], August of 2001. Disponível em <http://www.sdmagazine.com/documents/s=844/sdm0108a/0108a.htm>. Acesso em: 7 ago 2004.

[FOW 2003] FOWLER, M. **The New Methodology**. Disponível em <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em: 5 jun 2004.

[GAM 95] GAMMA, E. et al. **Design Patterns - Elements of Reusable Object Oriented Software**. Reading: Addison Wesley, 1995.

[GEO 2003] GEORGE, B.; WILLIAMS, L. An Initial Investigation of Test-Driven Development in Industry. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2003, Melbourne. **Proceedings…** New York: ACM Press, 2003. p. 1135-1139.

[GOL 95] GOLDENSON, D. R.; HERBSLEB, J. D. **After the Appraisal:** a systematic survey of process improvement, its benefits, and factors that influence success. Disponível em <http://www.sei.cmu.edu/pub/documents/95.reports/pdf/tr009.95.pdf>. Acesso em: 20 abr 2004.

[GRE 2001] GREENING, J. Launching Extreme Programming at a Process-Intensive Company. **IEEE Software**, Los Alamitos, v. 18, n. 6, p. 27-33, November/December 2001.

[HAL 98] HALL, E. **Managing Risk:** methods for software systems development. New York: Addison-Wesley, 1998.

[HAR 96a] HARRISON, N. B. Organizational Patterns for Teams. In: VLISSIDES, J. M.; COPLIEN, J. O.; KERTH, N. L. (Ed.). **Pattern Languages of Program Design 2**. Reading: Addison-Wesley, 1996.

[HAR 96b] HARRISON, N. B.; COPLIEN, J. O. Patterns of productive software organizations. **Bell Labs Technical Journal**, [S. l.], v.1, n.1, p. 138-145, Summer 1996.

[HAR 99] HARTMANN, J. **Um Framework para a Construção de Aplicações Orientadas a Objeto Robustas sobre Banco de Dados Relacional**. 1999. Projeto de Diplomação (Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[HAR 2003] HARTMANN, J. **Estudo Sobre a Aplicação de Métodos Ágeis (Agile) no Desenvolvimento e Gerenciamento de Projetos de Software**. 2003. Trabalho Individual (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[HAR 2004] HARTMANN, J. **Viabilidade de Uma Abordagem Baseada em Padrões (Patterns) e Critérios de Risco para a Melhoria de Processos de Software**. 2004. Trabalho Individual (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[HEN 95] HENDERSON-SELLERS, B. Who needs an object-oriented methodology anyway? **Journal of Object-Oriented Programming**, [S. l.], v. 8, n. 6, p. 6-8, 1995.

[HEN 99] HENDERSON-SELLERS, B.; MELLOR, S. Tailoring process-focussed OO methods. **Journal of Object Oriented Programming**, Santa Barbara, n. 12, v. 4, p. 40-44, August 1999.

[HIB 2004] HIBERNATE TEAM. **The Hibernate project web site**. Disponível em <http://www.hibernate.org>. Acesso em: 10 nov 2003.

[HIG 2001] HIGHSMITH, J.; COCKBURN, A. Agile Software Development: The Business of Innovation. **IEEE Computer**, Los Alamitos, v. 34, n.12, p. 120-122, September 2001.

[HOH 2002] HOHPE, G. **Enterprise Integration Patterns**. Disponível em: <http://www.eaipatterns.com/eaipatterns.html>. Acesso em: 1 nov 2003.

[HUL 2002] HULL, M. et al., Software development processes – an assessment. **Information and Software Technology**, [S.l.], v. 44, n. 1, p. 1-12, 2002.

[JAC 94] JACOBSON, I. **Object-Oriented Software Engineering**. New York: Addison-Wesley, 1994.

[JAC 99] JACOBSON, I. et al. **The Unified Software Development Process**. New York: Addyson Wesley, 1999.

[JAC 2000] JACOBSON, I. **The Road to the Rational Unified Process**. [S. l.]: Cambridge University Press, 2000.

[KEI 98] KEIL, M. et al. A Framework for Identifying Software Project Risks. **Communications of the ACM**, New York, v. 41, n. 11, p. 76-83, November 1998.

[LAR 98] LARMAN, C. **Applying UML and Patterns**. [S. l.]: Prentice-Hall, 1998.

[LAR 2003] LARMAN, C.; BASILI, V. A History of Iterative and Incremental Development. **IEEE Computer**, Los Alamitos, v. 36, n. 6, p. 47-56, June 2003.

[LIN 2002] LINDVALL, M. et al. Empirical Findings in Agile Methods. In: XP/AGILE UNIVERSE, 2002, Chicago. **Proceedings…** [S. l. : s. n.], 2002. p. 197-207.

[MAN 2000] MANNS, M. L. **Introducing Patterns (or any new idea) into Organizations.** Disponível em <http://www.cs.unca.edu/~manns/Intropat.pdf>. Acesso em: 3 ago 2004.

[MAN 2003a] MANZONI, L. V. **Processos de Software**. 2003. Exame de Qualificação em Abrangência (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[MAN 2003b] MANZONI, L. V.; PRICE, R. T. Identifying Extensions Required by RUP (Rational Unified Process) to Comply with CMM (Capability Maturity Model) Levels 2 and 3. **IEEE Transactions on Software Engineering**, Los Alamitos, v. 29, n. 2, p. 181-192, February 2003.

[MAR 87] MARTIN, J. **Recommended Diagramming Standards for Analysts and Programmers**. [S.l.]: Prentice-Hall, 1987.

[MES 95] MESZAROS, G.; DOBLE, J. Metapatterns: A pattern language for pattern writing. In: COPLIEN, J.; SCHIMIDT, D. (Ed.). **Pattern Languages of Program Design**. New York: Addison-Wesley, 1995.

[OLL 83] OLLE, T. W. et al. Information Systems Design Methodologies: A Feature Analysis. In: COMPARATIVE REVIEW OF INFORMATION SYSTEMS DESIGN METHODOLOGIES, CRIS, 1983, North Holland. **Proceedings…** [S. l. : s. n.], 1983.

[ORR 2002] ORR, K. **CMM versus Agile Development:** religious wars and software development. Disponível em <http://www.cutter.com/freestuff/apmreport.html>. Acesso em: 5 oct 2004.

[PAL 2002] PALMER, S.; FELSING, J. **A Practical Guide to Feature-Driven Development**. [S. l.]: Prentice Hall, 2002.

[PAU 2001] PAULK, M. C. Extreme Programming From a CMM Perspective. **IEEE Software**, Los Alamitos, v. 18, n. 6, p. 19-26, November/December 2001.

[PRI 90] PRICE, R. T.; GIRARDI, R. A Class Retrieval Tool for an Object Oriented Environment. In: INTERNATIONAL CONFERENCE ON THE TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS, TOOLS, 3., 1990, Sidney. **Proceedings…** [S. l. : s. n], 1990.

[RAS 2003] RASMUSSON, J. Introducing XP into Greenfield Projects: Lessons Learned. **IEEE Software**, Los Alamitos, v. 20, n. 3, p. 21-28, May-June 2003.

[RIE 96] RIEHLE, D.; ZULLIGHOVEN, H. Understanding and Using Patterns in Software Development. **Theory and Practice of Object Systems**, New York, v. 2, n. 1, p. 3-13, 1996.

[RIS 2000] RISING, L. et al. The Scrum Software Development Process for Small Teams. **IEEE Software**, Los Alamitos, v. 17, n. 4, p. 26-32, July/August 2000.

[RUM 91] RUMBAUGH, J. et al **Object-Oriented Modeling and Design**. [S. l.]: Prentice-Hall, 1991.

[RUM 95] RUMBAUGH, J. What Is a Method? **Journal of Object Oriented Programming**, [S. l.], v. 8, n. 6, p. 10-16, October 1995.

[SCH 95] SCHWABER, K. **Scrum Development Process**. Disponível em <http://www.controlchaos.com/old-site/scrumwp.htm>. Acesso em: 10 jan 2005.

[SCH 2002] SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. Upper Saddle River: Prentice Hall, 2002.

[SCH 2003] SCHWABER, K. **Advanced Development Methods web site**. Disponível em <http://www.controlchaos.com>. Acesso em: 5 abr 2003.

[SEI 95] SOFTWARE ENGINEERING INSTITUTE. **The Capability Maturity Model:** guidelines for improving the software process. [S. l.]: Addison-Wesley, 1995.

[SEI 2002] SOFTWARE ENGINEERING INSTITUTE. **Capability Maturity Model Integration (CMMI), Version 1.1**. Disponível em <http://www.sei.cmu.edu/cmmi/general/>. Acesso em: 10 jan 2005.

[SON 91] SONG, X.; OSTERWEIL, L. J. Comparing Design Methodologies through Process Modeling. In: INTERNATIONAL CONFERENCE ON SOFTWARE PROCESS, 1., 1991, Redondo Beach. **Proceedings…** Los Alamitos: IEEE CS Press, 1991. p. 29-44.

[STE 98] STEVENS, P.; POOLEY, R. Systems Reengineering Patterns. In: INTERNATIONAL SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING, 6., 1998, Lake Buena Vista. **Proceedings…** New York: ACM Press, 1998. p. 17-23.

[STU 2003] STUCKENSCHMIDT, H. **A Pattern-Driven Approach to Ontology Language Customization**. Disponível em: <www.informatik.uni-bremen.de/~heiner/Ontology-Patterns.pdf>. Acesso em: 5 oct 2003.

[SUT 2001] SUTHERLAND, J. **Inventing and Reinventing SCRUM in Five Companies**. Disponível em: <http://www.agilealliance.org/articles/articles/InventingSCRUM.pdf>. Acesso em: 4 abr 2003.

[TAK 86] TAKEUCHI, H.; NONAKA, I. The New New Product Development Game. **Harvard Business Review**, [S. l.], p. 137-146, January-February 1986.

[VAS 98] VASCONCELOS, F. M. de; WERNER, C. M. L. Organizing the Software Development Process Knowledge: An Approach Based on Patterns. **International Journal of Software Engineering & Knowledge Engineering**, [S. l.], v. 8, n. 4, p. 461-482, December 1988.

[XAV 2002] XAVIER, J. R. et al. Uma Abordagem para a Seleção de Padrões Arquiteturais Baseada em Características de Qualidade. In: SIMPOSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 16., 2002, Gramado. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 2002.

[WAR 2001] WARD. P. et al. Thinking Objectively: software process improvement in the small. **Communications of the ACM**, New York, v. 44, n. 4, p. 105-107, April 2001.

[WEL 2005] WELLS, D. **The Rules and Practices of Extreme Programming.** Disponível em: <http://www.extremeprogramming.org/rules.html>. Acesso em: 23 fev 2005.

[WIL 2000] WILLIAMS, L. et al. Strengthening the Case for Pair-Programming. **IEEE Software**, Los Alamitos, v. 17, n. 4, p. 19-25, July/Aug 2000.

[WIL 2002] WILLIAMS, L. et al. In Support of Pair Programming in the Introductory Computer Science Course. **Journal of Computer Science Education**, [S. l.], September 2002. Disponível em: <http://collaboration.csc.ncsu.edu/laurie/Papers/PP%20in%20Introductory_CSED.pdf>. Acesso em: 6 set 2003.

[WIL 2003] WILLIAMS, L.; COCKBURN, A. Agile Software Development: it's about feedback and change. **IEEE Computer**, Los Alamitos, v. 36, n. 6, p. 39-43, June 2003.

ANEXO EXEMPLOS DE PADRÕES ORGANIZACIONAIS

Nesta seção de anexo do trabalho são apresentados alguns padrões organizacionais selecionados a partir da literatura.

Pattern Name: ***Size the Schedule***
Source: James Coplien [COP 95]
Problem: How long should the project take?
Context:
The product is understood and the project size has been estimated.
Forces:
If you make the schedule too generous, developers become complacent, and you miss market windows.
If the schedule is too ambitious, developers become burned out, and you miss market windows.
If the schedule is too ambitious, product quality suffers, and compromised architectural principles establish a poor foundation for future maintenance. Projects without schedule motivation tend to go on forever, or spend too much time polishing details that are either irrelevant or don't serve customer needs.
Solution:
Reward developers for meeting the schedule, with financial bonuses (or at-risk compensation), or with extra time off. Keep two sets of schedules: one for the market, and one for the developers. The external schedule is negotiated with the customer; the internal schedule, with development staff. The internal schedule should be shorter than the external schedule by two or three weeks for a moderate project (this figure comes from a senior staff member at a well-known software consulting firm). If the two schedules can't be reconciled, customer needs or the organization's resources—or the schedule itself—must be re-negotiated.
Resulting Context:
A project with a flexible target date. Dates are always difficult to estimate; DeMarco notes that one of the most serious signs of a project in trouble is a schedule worked backward from an end date. Other mechanisms are necessary to ensure that a hurried development doesn't compromise quality.

Pattern Name: ***Architect Controls Product***
Source: James Coplien [COP 95]
Problem: A product designed by many individuals lacks elegance and cohesiveness.
Context:

An organization of *Developers* that needs strategic technical direction.
Forces:
Totalitarian control is viewed by most development teams as a draconian measure. The right information must flow through the right roles.
Solution:
Create an *Architect* role. The *Architect* role should advise and control *Developer* roles, and should communicate closely with them. The *Architect* should also be in close touch with *Customer*.
Resulting Context:
This does for the architecture what the *Patron* does for the organization: it provides technical focus, and a rallying point for technical work as well as market-related work.
Design Rationale:
We have no role called *Designer* because design is really the whole task. Managers fill a supporting role; empirically, they are rarely seen to control a process except during crises. While the *Developer* controls the process, the *Architect* controls the product. The *Architect* is a "chief *Developer*" (see pattern *Architect Controls Product* on page 13). Their responsibilities include understanding requirements, framing the major system structure, and controlling the long-term evolution of that structure. The *Architect* controls the product in the visualization accompanying the pattern *Engage QA*. "Les oeuvres d'un seul architect sont plus belles...que ceux d'ont plusiers ont taché de faire." – Pascal, *Pensées*.

Pattern Name: ***Architect Also Implements***
Source: James Coplien [COP 95]
Problem: Preserving the architectural vision through to implementation
Context:
An organization of *Developers* that needs strategic technical direction.
Forces:
Totalitarian control is viewed by most development teams as a draconian measure. The right information must flow through the right roles.
Solution:
Beyond advising and communicating with *Developers, Architects* should also participate in implementation.
Resulting Context:
A development organization that perceives buy-in from the guiding architects, and that can directly avail itself of architectural expertise.
Design Rationale:
The importance of making this pattern explicit arose recently in a project I work with. The architecture team was being assembled across wide geographic boundaries with narrow communication bandwidth between them.
Though general architectural responsibilities were identified and the roles were staffed, one group had expectations that architects would also implement code; the other did not.

Pattern Name: ***Application Design is Bounded By Test Design***
Source: James Coplien [COP 95]
Problem: When do you design and implement test plans and scripts?
Context:

A system with mechanisms to document and enforce the software architecture, and developers to write the code. A *Testing* role is being defined.

Forces:

Test development takes time, and cannot be started just when the system is done ("when we know what we have to test").

Scenarios are known when requirements are known, and many of these are known early.

Test implementation needs to know the details of message formats, interfaces, and other architectural properties in great details (to support test scripts and test jigs).

Implementation changes daily; there should be no need for test designs to track ephemeral changes in software implementation.

Solution:

Scenario-driven test design starts when scenario requirements are first agreed to by the customer. Test design evolves along with software design, but only in response to customer scenario changes: the source software is inaccessible to the tester. When development decides that architectural interfaces have stabilized, low-level test design and implementation can proceed.

Resulting Context:

This provides a context for *Engage QA* on page 16 and for *Scenarios Define Problem* on page 18.

Design Rationale:

Making the software accessible to the tester causes them to see the developer view rather than the customer view, and leads to the chance they may test the wrong things, or at the wrong level of detail. Furthermore, the software will continue to evolve from requirements until the architecture gels, and there is no sense in causing test design to fishtail until interfaces settle down. In short, test design kicks off at the end of the first major influx of requirements, and touches base with design again when the architecture is stable. This is related to the (yet unspecified) pattern, "Testing first in last out," to the pattern *Engage QA* on page 16, and to the pattern *Scenarios Define Problem* on page 18.

Pattern Name: ***Engage Customer***

Source: James Coplien [COP 95]

Problem: Maintaining customer satisfaction

Context:

A quality assurance function exists, and needs input to drive its work.

Forces:

Developers used to be called "loose cannons on deck."

Requirements changes occur even after design reviews are complete and coding as started.

Missing customer requirements is a serious problem.

Customers are traditionally not part of the mainstream development, which makes it difficult to discover and incorporate their insights.

Trust relationship between managers and coders.

Solution:

Make *Customer* a role that is closely coupled to the *Developer* and *Architect*, not just to *QA*.

Resulting Context:

The new context supports requirements discovery from the customer, as required by the pattern *Scenarios Define Problem* on page 18, and the pattern *3 to 7 Helpers per Role* on page 26. Other patterns like *Firewalls* on page 20 and *Firewalls* on page 20 also build on this pattern.

Pattern Name: *Scenarios Define Problem*
Source: James Coplien [COP 95]
Problem: Design documents are often ineffective as vehicles to communicate the customer vision of how the system should work.
Context:
You want to engage the customer and need a mechanism to support other organizational alliances between customer and developers.
Forces:
There is a natural business distancing and mistrust between customers and developers. Communication between developers and customers is crucial to the success of a system.
Solution:
Capture system functional requirements as use cases, a la Jacobson.
Resulting Context:
The problem is now defined, and the architecture can proceed in earnest.

Pattern Name: *Sprint*
Source: Scrum patterns [BEE 99]
Problem
We want to balance the need of developers to work undisturbed and the need for management and the customer to see real progress.
Forces
Developers need time to work undisturbed, but they need support for logistics.
Management and users need to be convinced that real progress is made.
Often, by the time systems are delivered, it is obsolete or it requires major changes. The problem is that input from the environment is mostly collected at the start of the project, while the user learns most using the system or intermediate releases.
Some problems are “wicked”, that is it difficult to even describe the problem without a notion of the solution. It is wrong to expect developers do to a clean design and commit to it at the start of this kind of problems. Experimentation, feedback, creativity are needed.
Solution
Each *Sprint* takes a pre-allocated amount of work from the Backlog. The team commits to it. As a rule nothing is added externally during a sprint. External additions are added to the global backlog. Blocks resulting from the *Sprint* can also be added to the Backlog. A Sprint ends with a Demonstration of new functionality.
Give the developers the space to be creative, and to learn by exploring the design space, doing actual work, undisturbed by outside interruptions, free to adapt their way of working using opportunities and insights. At the same time keep the management and stakeholders confident by showing real progress instead of documents and reports produced as proof. Do this in short cycles, *Sprints*, where part of the *Backlog* is allocated to a small team. In a *Sprint*, during a period of approximately 30 days, an agreed amount of work will be performed, to create a deliverable. *Backlog* is assigned to *Sprints* by priority and by approximation of what can be accomplished during a

month. Chunks of high cohesion and low coupling are selected. The focus is on enabling, rather than micro-management.
During the *Sprint,* outside chaos is not allowed in the increment. The team, as they proceed, may change course and their way of working. By buffering them from the outside, we allow them to focus on the work at hand and on delivering the best they can and the best way they can, using their skill, experience and creativity.
Each *Sprint* produces a visible and usable deliverable. This is demonstrated in *Demo*. An increment can be either intermediate or shippable, but it should stand on its own. The goal of a *Sprint* is to complete as much quality software as possible and to ensure real progress, not paper milestones as alibi.

Rationale

Developing systems is unpredictable and chaotic. Development is an empirical process that requires significant thought during the process. A method can only supply a framework for the real work and indicate the places where creativity is needed. Yet we tread black-box processes often as fully defined processes. Unpredictable results occur.
We lack the controls to measure and respond to the unpredictable.
While building a system many artifacts come into existence, many new insights are gained. These new artifacts can guide future thinking. Increased productivity through good tools or uncovered components may open the opportunity for adding more *Backlog* and more functionality to our system, or for releasing a product early.
Therefore, during a *Sprint*, we optimize communications and maximize information sharing in daily *Scrum Meetings*.
Sprints set up a safe environment and time slots where developers can work undisturbed by outside requests or opportunities. They also offer a pre-allocated piece of work that the customer, management and the user can trust the *Scrum Team* to produce as a useful deliverable, such as a working piece of code at the end of the *Sprint*. The team focuses on the right things to do, management working on eliminating what stands in this way of doing in better.

Pattern Name: *Backlog*

Source: Scrum patterns [BEE 99]

Problem

What is the best way to organize the work to be done next at any stage of the project?

Forces

Project plans captured in Pert charts or Gannt charts often try to capture tasks to be done a priori, but they often fail in their implementations, because they lack flexibility. Tasks are pre-allocated time in Pert or Gant charts but their priorities and number grow or dimish as required in *real* projects**,** and therefore they are not good tools to use where the number of tasks changes drastically over time.
Not having a repository of tasks in any shape or form simply translates into project failure. There must be some sort of project control.

Solution

Use a *Backlog* to organize the work of a SCRUM team**.**
The *Backlog* is a prioritized list. The highest priority backlog will be worked on first, the lowest priority backlog will be worked on last. No feature, addition, enhancement to a product is worth fighting over; it is simply either more important or less important at any time to the success and relevance of the product.
Backlog is the work to be performed on a product. Completion of the work will

transform the product from its current form into its vision. But in SCRUM, the *Backlog* evolves as the product and the environment in which it will be used evolves. The backlog is dynamic, constantly changed by management to ensure that the product defined by completing the *Backlog* is the most appropriate, competitive, useful product possible.
There are many sources for the backlog list. Product marketing adds work that will fulfill their vision of the product. Sales add work that will add new sales or extend the usefulness to the installed base. Technology adds work that will ensure the product uses the most innovative and productive technology. Development adds work to enhance product functions. Customer support adds work to correct underlying product defects. Only one person prioritizes work. This person is responsible for meeting the product vision. The title usually is product manager or product marketing manager. If anyone wants the priority of work changed, they have to convince this person to change that priority. The highest priority backlog has the most definition. It is also prioritized with an eye toward dependencies.
Resulting Context
Project work is identified dynamically and prioritized according to:
1) the customer's needs, and
2) what the team can do.

Pattern: *Get Involved Early*
Source System Test Pattern Language [DEL 96]
Problem How can System Test maximize the support from the Designers?
Forces
During early development phases, System Testers have test plans to write.
System Testers have a broader view of the context of the system.
Impacts of social relationships aren't measured by the project.
Solution Establish a good working relationship with the Designers early in the project. Don't wait until you need to interact with a Designer to develop a working relationship. By that time it is too late. Trust must be built over time. One way to accomplish this is to learn the system and the features at the same time the Designer is learning them. Attend reviews of the requirements and design documentation. Invite the Designer to test plan reviews.
Resulting Context When a good working relationship is built over time, it is easier to resolve problems when they are discovered. Be sure that the relationship with the Designer does not affect your judgment as an effective System Tester. There can be a tendency to avoid areas of conflict when friends are involved, to look the other way when problems are found in a certain area where the Designer is a close friend. It is also dangerous to have too much information about an area. This can lead to certain kinds of testing that depends on this knowledge instead of an objective black box view.
Rationale We are all more willing to work with people we know well. Waiting until the heat of battle to get to know the people who can help you resolve problems leads to delays in solutions. If a trusting relationship has already been established, the problem solving process is smoother.

Pattern: *Old Problem Reports*
Source System Test Pattern Language [DEL 96]

Problem What areas of the system should be tested first so that the most problems can be found in the least amount of time?
Context Testing of existing features is being considered. Problem reports from previous releases are available.
Solution Use problem reports from previous releases to help select regression tests. Since it would be inefficient to retest for all old problems, look at problems reported after the last valid "snapshot" of the system. Categorize problem reports to see if a pattern is determined that could be used for additional testing.
Resulting Context Problems that still exist will be found and corrected.
Rationale Since a problem report represents something that escaped in a previous release, this could be a good indicator of a problem in the current load. Problem reports tend to point to areas where problems always occur. Problem reports often represent a symptom that is difficult to connect with an underlying problem. Additionally, fixes of a previous release are often done in parallel with new development on the current releases. These fixes don't always find their way into the current load.