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

LISANDRA MANZONI FONTOURA

PRiMA: *Project Risk Management Approach*

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Prof. Dr. Roberto Tom Price
Orientador

Porto Alegre, junho de 2006.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Fontoura, Lisandra Manzoni

PRiMA: Project Risk Management Approach / Lisandra Manzoni Fontoura – Porto Alegre: Programa de Pós-Graduação em Computação, 2006.

206 f.:il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2006. Orientador: Roberto Tom Price.

1.Gerência de Riscos. 2.Processos de Software 3.Padrões de Processo 4.Padrões Organizacionais 5.Goal/Question/Metric. I. Price, Roberto Tom. II. Título.

AGRADECIMENTOS

Agradeço a Deus, por sempre estar ao meio lado e me dar vontade de seguir em frente, mesmo nos momentos mais difíceis.

Adriano, meu muito obrigada por estar ao meu lado, me apoiando nessa difícil tarefa de ser esposa, professora e aluna ao mesmo tempo. Por ter entendido que eu precisava priorizar atividades e nos últimos tempos nosso tempo junto foi ficando cada vez mais raro. Mas, mesmo assim, você insistia para que eu o ignorasse e passasse horas sentada em frente ao computador pesquisando e escrevendo esse trabalho.

Agradeço aos meus pais, Volney e Irene, as minhas irmãs, cunhados, sobrinhos e amigos, que souberam entender os motivos pelos quais muitas vezes tive que estar ausente e sempre me incentivaram a continuar.

Agradeço ao meu orientador, professor Roberto Tom Price, que me acompanhou durante o mestrado e doutorado, por longos 7 anos, e que sempre confiou no trabalho que eu estava realizando, me ensinou a construir meu próprio conhecimento e me apoiou nos momentos decisivos para que essa tese fosse elaborada.

Agradeço aos colegas do Projeto Amadeus pelas sugestões. Em especial, gostaria de agradecer ao colega Júlio Hartmann, pela troca de idéias e pelo compartilhamento do conhecimento.

Agradeço aos professores Carlos Alberto Heuser, Marcelo Pimenta e Juliana Herbert por terem acompanhado meu trabalho desde o mestrado, sempre com sugestões sensatas e questionamentos importantes.

Agradeço aos colegas do SERPRO pelo incentivo recebido. Em especial, agradeço a Angélica e a Andréa pela amizade e pelo apoio.

Agradeço à Lara por sua amizade, desde os tempos em que éramos colegas de graduação.

Agradeço aos meus cunhados Fabiano e Tatiana, que sempre me receberam muito bem quando chegava à Porto Alegre às 6 horas da manhã, para conversar com meu orientador.

E, por fim, agradeço a todos que de alguma forma me ajudaram a tornar este trabalho uma realidade.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ASD	*Adaptative Software Design*
CMM	*Capability Maturity Model*
CMMI	*Capability Maturity Model Integrated*
CMM-SW	*Capability Maturity Model for Software*
DoD	*Department of Defense*
EF	*Experience Factory*
ESI	*European Software Institute*
FDD	*Feature Driven Development*
GQM	*Goal/Question/Metric*
GRS	Gerência de Riscos de Software
IPD-CMM	*Integrated Product Development CMM*
ISO	*Internacional Organizational for Standardization*
KM	*Knowledge Management*
KPA	*Key Process Area*
MOF	*MetaObject Facility*
OMG	*Object Management Group*
P-CMM	*People*-CMM
PE	Ponto de Extensão
PMI	*Project Management Institute*
PML	*Process Modeling Languages*
PMT	*Pattern-based Methodology Tailoring*
PRiMA	*Project Risk Management Approach*
PRiMA-F	*Project Risk Management Approach - Framework*
PRiMA-M	*Project Risk Management Approach - Metamodel*
PSEE	*Process-centered Software Engineering Environment*
PSPO	Processo de Software Padrão da Organização
QIP	*Quality Improvement Paradigm*

RUP	*Rational Unified Process*
SA-CMM	*Software Acquisition* - CMM
SE-CMM	*Systems Engineering* - CMM
SEI	*Software Engineering Institute*
SEL	*Software Engineering Laboratory*
SGW	Sistema de Gerência de *Workflow*
SPEM	*Software Process Engineering Metamodel Specification*
SRE	*Software Risk Evaluation*
SW-CMM	*Software*-CMM
UC	*Use Case*
UML	*Unified Modeling Language*
US	*User Story*
XML	*eXtensible Markup Language*
XP	*Extreme Programming*
XPDL	*XML Process Definition Language*
WfMC	*Workflow Management Coalition*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Esta tese propõe uma abordagem sistemática para gerenciar riscos em projetos de software, por meio da adaptação de processos. O objetivo da abordagem é permitir a elaboração de um processo específico para um dado projeto, visando minimizar a exposição do projeto aos riscos, identificados de acordo com o contexto do projeto.

As atividades, possíveis de serem executadas em processos de projetos de uma organização, são estruturadas em um *framework* de processo (PRiMA-F), que inclui também os padrões de processo e organizacionais usados para descrever ações preventivas e corretivas aos riscos. A estruturação do *framework* básico, construído pela organização, poderá permitir distintas instanciações, como por exemplo, processos de acordo com o paradigma ágil ou planejado, ou em conformidade com normas de qualidade, como CMM e outras; além dos padrões organizacionais e de processo para gestão de riscos de projeto.

PRiMA-F define o escopo maior do processo de software da organização e este é adaptado de acordo com os riscos identificados para o projeto e suas necessidades específicas, dando origem ao processo a ser usado no projeto.

Guias de adaptação e configurações de processo são propostos para facilitar a tarefa de adaptação. Os guias descrevem como adaptar elementos de processo de acordo com o tamanho e o formalismo do projeto. Configurações de processo são modelos pré-definidos, visando atender projetos típicos ou modelos de qualidade. Prima-F pode ser estendida para novos riscos, padrões e processos, de acordo com as necessidades da organização.

Utilizando o paradigma *Goal/Question/Metric*, no *framework* de processo (PRiMA-F), são definidas métricas do processo de software, associadas aos riscos, para serem usadas para acompanhar o progresso dos fatores de risco, possibilitando ao gerente de projeto tomar ações corretivas, quando necessário e no momento adequado. As ações corretivas são descritas usando padrões organizacionais e de processo.

Uma ferramenta de apoio à sistemática proposta (PRiMA-Tool) foi desenvolvida. Estudos de caso foram elaborados para validar a sistemática proposta.

Palavras-Chave: gerência de riscos, processos de software, adaptação de processos, RUP, XP, padrões organizacionais, padrões de processo, *Goal/Question/Metric*.

PRiMA: Project Risk Management Approach

ABSTRACT

The proposal of this thesis is a systematic approach to manage risks in software projects via process tailoring. The aim of the approach is to allow the elaborate of a specific process for a specific project, to minimize the exposition of this project to some risks, identified according to the project's context.

The activities possible to be performed in processes of projects in an organization, are structured in a process framework (PRiMA-F), which also include the process and organizational patterns used to describe preventive and corrective actions to the risks. The basic framework structure, built by the organization, may allow different instantiations, for example, processes according to the agile or planned paradigm, or according to the quality models or standards, like CMM and others; besides the organizational and process patterns for the project's risk management.

PRiMA–F defines the organization's software process biggest scope and this is tailored according to the risks identified for the project and its specific needs, originating the process to be used in the project.

Tailoring guidelines and process configurations are proposed to make the adaptation task easier. The guides describe how to tailor process elements according to the project's size and formalism. Process configurations are models pre-defined, aiming to help typical projects or quality models. PRiMA-F can be extended to new risks, patterns and processes, according to the organization's needs.

Using the paradigm Goal/Question/Metric, in the process framework (PRiMA-F) software process metrics are defined, associated to the risks, to be used to keep up with the risk factors progress, making possible to the project's manager take corrective actions, if necessary and on the right moment. The corrective actions are described using organizational and process patterns.

A tool to support the proposed systematic (PRiMA-Tool) was developed. Some case studies were elaborated to validate the systematic proposed.

Keywords: risk management, software process, process tailoring, RUP, XP, organizational pattern, process pattern, Goal/Question/Metric

1 INTRODUÇÃO

Pesquisadores e profissionais da indústria têm investido esforços em entender e melhorar a qualidade do software, por meio de uma melhoria do processo usado para desenvolver o software, pois é consenso na comunidade de software que a qualidade do software é diretamente proporcional à qualidade do processo, usado no seu desenvolvimento. Isto é a base dos modelos de avaliação de processos como: CMM, CMMI, ISO 15507, ISO 12207, entre outros.

Considerando que, projetos de software apresentam diferenças quanto à tecnologia, clientes, riscos, formalismo, equipe, tamanho, entre outros, definir um processo único para ser utilizado em todos os projetos da organização é inviável. É necessário que os processos de software sejam adequados ao domínio da aplicação e aos requisitos específicos de cada projeto.

O SEI, por meio do CMM-SW, propõe que seja definido um processo de software padrão da organização (PSPO), com os elementos de processo fundamentais que são esperados em todos os projetos da organização, e esse processo seja adaptado para cada projeto, de acordo com as características e restrições deste, dando origem ao processo definido de software para um projeto (PAULK, 1993). Segundo Ginsberg e Quinn (1995), a adaptação pode ser definida como o ato de ajustar uma definição particularizando os termos de uma descrição geral para derivar uma descrição aplicável a um ambiente alternativo (menos geral).

Para que o processo de software padrão da organização possa ser adaptado a projetos específicos é necessário que, exista uma clara definição do processo da organização, e que esse processo possa ser reutilizado por meio de adaptação. É importante que na adaptação sejam preservadas as propriedades do processo padrão da organização (YOON, 2001). Tendo um processo padrão é possível melhorar constantemente este processo por meio de avaliações de processo, e todos os processos da organização serão beneficiados, já que são obtidos a partir deste.

A adaptação de processos pode ocorrer com base na experiência adquirida em projetos anteriores, desde que o conhecimento organizacional seja gerenciado de forma que possa ser reutilizado em outros projetos da organização (KEENAN, 2004; RUS, 2002) (FALBO, 2004; FIORINI, 2001).

Atualmente, existem duas correntes de processos de desenvolvimentos, que são métodos ágeis (BECK, 2004; AMBLER, 1998) e métodos planejados (RATIONAL, 2003). A adaptação de processos pode balancear entre o uso de práticas preconizadas por ambos processos, dando origem a métodos híbridos, de acordo com as necessidades e peculiaridades de cada projeto e da organização (BOEHM, 2003; LYCETT, 2003).

Padrões são maneiras de descrever melhores práticas, bons projetos, e capturar experiências de uma forma que seja possível para outros reusarem essa experiência (HILLSIDE, 2003). Padrões de processo e organizacionais capturam práticas bem-sucedidas em desenvolvimento de software, e podem ser utilizados na elaboração de processos de desenvolvimento da organização, ou na melhoria de processos existentes. Padrões podem descrever ações preventivas a riscos. Para cada risco podem existir vários padrões recomendados para a sua prevenção.

Comparado com outros tipos de projetos, projetos de software apresentam altos percentuais de incerteza. Cada tarefa em um projeto de software tem probabilidade de desviar de seu plano, e os riscos podem ser transferidos e acumulados durante a execução de uma série de tarefas. A previsão e o controle de riscos são essenciais para que os objetivos definidos para o projeto de software sejam cumpridos.

1.1 Motivação/Problemas

A motivação para esta tese está relacionada a dois tópicos principais, que são: gerência de riscos e adaptação de processos de software.

1.1.1 Gerência de Riscos

Projetos de software envolvem riscos. Segundo Tom DeMarco e Timothy Lister (2003), se um projeto não possui riscos, não o faça (DEMARCO, 2003, p.9). Os benefícios e os riscos caminham lado a lado, o que significa que, se o projeto não tem riscos, também não o terá benefícios. Quanto mais riscos possuir o projeto, maiores são seus benefícios.

Considera-se um projeto bem-sucedido aquele que satisfaz as necessidades do cliente e é entregue dentro do prazo, custo e com a qualidade estimados (PMI, 2000). Segundo estudos do *Standish Group* (2004), em 2004, 29% dos projetos foram bem-sucedidos, 18% foram cancelados e 53% foram entregues fora do prazo, com custo acima do estimado, ou com menos funcionalidades que as inicialmente especificadas. O gerenciamento sistemático de riscos pode evitar esse caus. Por meio do acompanhamento dos fatores de riscos, o gerente de projeto pode tomar ações preventivas e corretivas durante o projeto; evitando comprometer prazo, qualidade, custo e funcionalidades.

Como poucas organizações de software aplicam métodos de gerência de riscos documentados e sistemáticos, os gerentes de software experientes usam sua intuição e sorte, para gerenciar riscos, ao invés de usar uma sistemática estabelecida e documentada. Considerando que, a maioria dos projetos de software são complexos e envolvem diferentes tipos de risco, utilizar intuição e iniciativa individual dos gerentes, é um substituto pobre para uma abordagem consistente e profissional para gerenciamento de riscos (KONTIO, 1998).

Gerência de riscos de software (GRS) é uma abordagem que busca organizar o enfoque dos riscos correlatos ao desenvolvimento bem-sucedido de sistemas de software em um conjunto de princípios e técnicas para analisar, preparar ações preventivas e controlar os riscos de projetos de software. GRS sugere medidas para prevenir que riscos afetem o projeto ou para reduzir seu impacto, e deve ser vista como um componente fundamental do processo de gerenciamento de projetos (ADDISON,

2002). Gerenciamento de riscos deve focar nas questões que podem por em risco o alcance dos objetivos críticos de um projeto (CMMI,2002).

Uma abordagem mais eficiente para gerência de riscos é uma abordagem pró-ativa onde os riscos são identificados mais cedo no desenvolvimento e medidas preventivas são tomadas para evitar a materialização dos riscos (APPULLUTTY, 2005).

1.1.2 Adaptação de Processos

Descrições de processos são importantes porque permitem que o conhecimento da organização seja reutilizado, o que é tema de pesquisas na área de *Learning Software Organizations* (HENNINGER, 1999), e que os processos sejam avaliados e melhorados (PAULK, 1993).

Segundo Prieto-Díaz (1991), a reutilização de processos é um meio de reutilizar experiências (práticas) e conhecimento, possibilitando a criação de coleções de processos reutilizáveis que podem ser conectados para instanciar processos novos e mais complexos. Prieto-Díaz diz ainda que esta área tem recebido pouca atenção, e possui um grande potencial.

Entretanto, criar um processo único que possa ser utilizado em diferentes projetos é uma tarefa difícil, e pouco eficiente, pois os projetos possuem características diferentes, envolvem clientes e equipes diferentes. Muitas organizações buscam definir um processo padrão, que seja comum a todos os projetos da organização e adaptar este processo de acordo com as características particulares de cada projeto.

Porém, o desenvolvimento de software é um mercado altamente dinâmico, os aspectos estruturais e organizacionais dos projetos e clientes alteram-se facilmente, os requisitos dos clientes mudam rapidamente e novas tecnologias são adotadas pelas organizações. Para produzir software de alta qualidade de forma competitiva é preciso gerenciar as mudanças. Isto implica em ser capaz de rapidamente adaptar os processos de software em relação às novas mudanças (WEINBERG apud GNATZ, 2003).

A definição do processo da organização pode ser considerada como generalização, a partir da qual diferentes processos são especializados. Essa definição generalizada pode estar em conformidade com alguma norma ou modelo, e pode ser avaliada e melhorada constantemente. Como os demais processos são gerados a partir dessa definição, os benefícios e melhorias no processo padrão serão refletidos nos processos dos projetos.

Processos de software podem ser construídos a partir de um metamodelo de processo, do qual vários processos podem ser construídos - por instanciação e montagem, isto é, são criadas instâncias de elementos do processo (metaclasses) e adicionadas regras para seqüenciamento entre essas instâncias para formar o processo; ou de um *framework*, no qual são representados elementos preconizados por diferentes processos, possibilitando que sejam instanciados diferentes processos a partir dele, para atender diferentes tipos de requisitos de projetos e situações.

Existem duas abordagens principais para processos de software, que são: processos ágeis e processos planejados. Processos planejados preconizam o uso de um extensivo planejamento e um rigoroso reuso como forma de tornar o desenvolvimento mais eficiente e previsível. Uma nova geração de desenvolvedores cita o peso da burocracia, as rápidas mudanças da tecnologia, e os efeitos da desumanização dos métodos planejados como as causas para revolução (BOEHM, 2002). Esses desenvolvedores

criaram o manifesto ágil, propondo uma nova forma de desenvolver software, chamada de métodos ágeis (HIGHSMITH, 2001).

Boehm (2002) utiliza gerência de riscos para balancear entre métodos ágeis e planejados, determinando quanto de planejamento é necessário em cada projeto de software. Ele conclui que abordagens híbridas que combinam métodos planejados e métodos ágeis são viáveis e necessárias para projetos que combinam características desses dois métodos.

No entanto, adaptar o processo da organização para cada projeto é uma tarefa complexa e trabalhosa, envolvendo a seleção, elaboração e combinação de técnicas e processos, além da compreensão das características e necessidades do projeto. Portanto, é desejável a pesquisa de abordagens sistemáticas e ferramentas que apóiem o projetista de processos a executar este trabalho.

1.1.3 Abordagem Escolhida para Solução

Esta tese propõe uma sistemática para gerência de riscos usando padrões organizacionais e de processo, chamada de *Project Management Risk Approach* (PRiMA). Um processo definido para determinado projeto é elaborado a partir do processo padrão da organização, inserindo neste as ações preventivas para os riscos analisados e priorizados para o projeto.

Considerando que, padrões organizacionais e de processo descrevem soluções para problemas recorrentes, podem ser utilizados na construção e na adaptação de processos de software, e existem inúmeros catálogos de padrões de processo e organizacionais (COPLIEN, 1995; CUNNINGHAM, 2004; SCHWABER, 2001; PAULK, 1993; YATCO, 1999) disponíveis, optou-se por utilizar padrões de processo para representar ações preventivas ou corretivas para riscos de software.

Padrões são associados aos riscos por meio de regras estabelecidas de acordo com o contexto do projeto. O contexto, utilizado neste trabalho, é similar ao descrito por Cockburn (2000) e Boehm (2003), onde são analisados os seguintes critérios: a criticidade dos defeitos (possibilidade de perda causada pelo risco), o número de pessoas envolvidas, e a habilidade da equipe de desenvolvimento (HARTMANN, 2005).

Inicialmente, optou-se por definir um metamodelo de processo que permita instanciar métodos ágeis, planejados e híbridos e reresente os padrões de processo. O metamodelo elaborado, chamado *Project Risk Management Approach Metamodel* (PRiMA-M), define um conjunto de elementos de processo que são usados para construir diferentes processos. Esse metamodelo foi elaborado a partir dos metamodelos do *Extreme Programming* (XP) e do *Rational Unified Process* (RUP). Usando os elementos descritos no metamodelo poder-se-á definir processos planejados, ágeis e híbridos; bem como customizações desses processos para uso em projetos específicos.

Métodos ágeis e planejados são abordagens extremas. Autores, como Boehm (2003), propõem que em muitos projetos é possível usar um processo de software que combine características desses dois métodos. A partir de uma análise dos riscos do projeto e do contexto do projeto serão selecionados diferentes padrões, resultando em processos híbridos.

Rational Unified Process (RUP) (RATIONAL, 2001) foi selecionado para representar métodos planejados por ser um dos mais utilizados processos de software

(CHARETTE, 2002), ser bastante completo, e já terem sido desenvolvidos trabalhos anteriores com este processo (MANZONI, 2003). O XP (BECK, 1999) foi escolhido por ser o método ágil mais conhecido e utilizado atualmente (CHARETTE, 2002).

PRiMA-M define os elementos de processo que podem existir nos processos instanciados a partir deste, mas não garante que esses elementos sejam compatíveis e consistentes entre si, não resolve questões relacionadas a conflitos provenientes dos diferentes enfoques de cada método e não possibilita descrever a seqüência de execução das atividades. Para solucionar essas deficiências, foi elaborado um *framework* de processo, chamado de PRiMA-F.

Project Risk Management Approach Framework (PRiMA-F) integra as atividades possíveis de serem executadas em processos de projetos de uma organização, incluindo atividades que visam implantar padrões de processo e organizacionais como ações preventivas ou corretivas aos riscos. Os elementos que compõem o *framework* são representados de acordo com os conceitos descritos no metamodelo PRiMA-M. A partir de PRiMA-F podem ser instanciados diferentes processos pela seleção de elementos de processo (previamente definidos no *framework)*. Essa adaptação é realizada no início do projeto e é chamada de adaptação estática.

Os elementos de processo, os padrões, os riscos e as regras de associação de padrões a riscos, definidos em PRiMA-F, formam a base de conhecimento da organização, que deve evoluir com o tempo e com base na experiência dos desenvolvedores, possibilitando o reuso e a melhoria dos bens de processo da organização.

São previstos outros dois tipos de adaptação que são: adaptação dinâmica, que ocorre ao final de cada iteração do processo em execução, quando o processo é avaliado e pode ser re-configurado; e alterações em PRiMA-F, visando à melhoria contínua dos processos da organização e dos projetos, com base na experiência da organização no uso do *framework*.

Para facilitar a tarefa de elaboração do processo da organização, e também como ilustrações de processos, na própria estrutura do *framework*, são propostas configurações "populares" de processo. Como exemplo, são propostas algumas configurações específicas, tais como: RUP essencial, RUP completo, XP; ou estendidas para satisfazer modelos de qualidade, como: RUP CMM Nível 2 e RUP CMM Nível 3. Cabe ao projetista definir as configurações para a organização.

Para acompanhar a efetividade das ações preventivas inseridas ao processo é necessário monitorar os fatores de risco, visando garantir que os riscos estão sob controle ou acionar o plano de resolução de riscos, caso algum risco se materialize.

Utilizando-se a abordagem GQM é elaborado um plano de gerência de riscos, chamado de Plano GQM. O Plano GQM descreve um conjunto de metas visando o monitoramento dos fatores de riscos. As metas são desdobradas em questões quantificáveis e posteriormente em métricas a serem obtidas para monitorar o progresso dos fatores de risco do projeto, indicando ao gerente quando ações corretivas devem ser tomadas.

Um ambiente experimental de apoio à abordagem sistemática de gerência de riscos foi elaborado. Considerando o contexto do projeto e os riscos associados, o processo padrão da organização será adaptado para atender as necessidades específicas de um projeto, com ações de prevenção aos riscos.

1.2 Resumo das Contribuições

Esta tese apresenta como contribuição a proposta de uma abordagem sistemática para gerenciar riscos em projetos de software, chamada de PRiMA - *Project Risk Management Approach*. A elaboração de um processo, para uso em projetos específicos de software, ocorre com base no processo padrão da organização e nos bens de processos, armazenados em uma base de conhecimento, conforme preconizam os modelos CMM e CMMI.

A principal contribuição do trabalho é experimentar o uso de adaptação de processos de software com base na análise dos riscos de um projeto. Outras contribuições do trabalho incluem:

- ⇨ Elaboração de uma lista com os riscos mais comuns em projetos de software, para auxiliar o gerente de projeto na identificação dos riscos de software;
- ⇨ Experimentação do uso de padrões organizacionais e de processos para descrever ações preventivas e corretivas aos riscos de software;
- ⇨ Apoio à reutilização de processos, uma vez que é proposta uma base de conhecimento da organização com um conjunto de bens de processos que podem ser reutilizados em diferentes projetos;
- ⇨ Construção de um metamodelo que permita instanciar elementos de processo descritos por processos planejados, ágeis e híbridos (PRiMA-M), e um *framework* de processo (PRiMA-F), do qual diferentes processos podem ser instanciados;
- ⇨ O estabelecimento de uma base de dados inicial e de um *framework* de processo, elaborado a partir dos processos RUP e XP e alguns padrões para prevenção e correção de riscos, que poderão ser alterados e adaptados, de acordo com as necessidades da organização;
- ⇨ Proposta de um processo para melhoria do *framework,* para que este seja adaptado a mudanças de tecnologia, clientes, riscos, etc.
- ⇨ A definição da estrutura de guias de adaptação, que são utilizadas para facilitar a tarefa de adaptação de processos, conforme preconizado pelo CMM;
- ⇨ Definição de modelos de processos pré-definidos, visando facilitar a elaboração de processos padrão para organização;
- ⇨ Experimentação do uso de GQM para elaboração de planos para monitoramento de riscos em projetos de software;
- ⇨ Elaboração de ambiente experimental de apoio à sistemática proposta, visando a adaptação de processos com base nos riscos, e possibilitando a evolução do *framework* e da base de conhecimento da organização;
- ⇨ Elaboração de um *website* com a descrição do processo definido para um projeto, facilitando a comunicação entre os membros do projeto, e o uso do processo definido do projeto por toda a equipe de projeto;
- ⇨ Descrição de dois estudos de caso realizados, relatando os resultados obtidos no uso da sistemática PRiMA.

1.3 Organização do Texto

O texto está organizado como segue. No capítulo 2 são apresentados os referenciais teóricos relacionados a processos de software, necessários ao entendimento da tese. Esses referenciais são: processos ágeis, planejados e híbridos, CMM, padrões organizacionais e de processo e medição em projetos de software. No capítulo 3 são descritas abordagens de gerência de riscos, propostas na literatura. No capítulo 4 é apresentada a abordagem sistemática para gerência de riscos (PRiMA). No capítulo 5 é descrito o ambiente experimental de apoio à adaptação de processos com base nos riscos (PRiMA-Tool). No capítulo 6 são descritos os estudos de caso realizados. O capítulo 7 descreve as considerações finais. Nos anexos são descritas informações complementares sobre o *framework* elaborado como exemplo e utilizado nos estudos de caso.

2 PROCESSOS DE SOFTWARE

Softwares são produtos complexos, difíceis de serem desenvolvidos e testados. Freqüentemente, o software apresenta um comportamento indesejado e inesperado, e pode causar vários problemas e danos, colocando em risco a vida de pessoas e a perda de muito dinheiro (FUGGETA, 2000).

A área de processos de software, como disciplina autônoma, iniciou-se em 1980, a partir de uma série de eventos e workshops sobre o assunto, dentre eles destaca-se o *International Software Process Workshop*. Vários jornais e eventos foram criados, após 1980, para divulgar pesquisas sobre Processo de Software. Foram criadas também instituições nos EUA e na Europa para estudar processos de software, tais como: *Software Engineering Institute* (SEI) e *European Software Institute* (ESI) (FUGGETA, 2000).

Segundo Humphrey (1990), processo de software é o conjunto completo e ordenado de atividades de engenharia de software necessárias para transformar os requisitos do usuário em software.

Fuggeta (2000) define processo de software como o conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos que são necessários para conceber, desenvolver, implantar e manter um produto de software.

O processo de software provê um amplo e compreensível conceito para organizar os diferentes fatores e assuntos relacionados às atividades de desenvolvimento de software tais como: tecnologias usadas no desenvolvimento de software, métodos e técnicas para desenvolvimento de software, comportamento organizacional (pessoas e estrutura organizacional), marketing e economia (FUGGETA, 2000).

Um processo de software definido possibilita que os desenvolvedores utilizem uma linguagem comum para se comunicarem, elaborem documentos similares, usem técnicas similares na resolução de problemas, adotem estilos de projeto e codificação que sejam prontamente entendidos e fáceis de transmitir entre membros da equipe e que facilite o gerenciamento dos projetos (HULL, 2002). Produtos de trabalho padronizados são mais fáceis de serem compreendidos e avaliados.

Processos de software são descritos por meio de modelos de processo. Um modelo de processo é uma representação abstrata de um processo de software (FEILER, 1993). Modelos de processo especificam as atividades que devem ser executadas, os papéis e os artefatos que devem ser elaborados e mantidos, e como usar e controlar ferramentas de desenvolvimento (ACUÑA, 1999). Um modelo de processo é então um meio de representar, entender, melhorar a comunicar o processo (AMBRIOLA, 1997).

Para representar os modelos de processo, são usadas Linguagens de Modelagem de Processos (*Process Modeling Languages* – PMLs), que são formalismos que tornam possível representar de uma maneira compreensível e precisa as características e facetas de processos de software. Segundo Cugola (1998), podem-se distinguir quatro principais paradigmas utilizados para representar modelos de processos, que são: baseados em linguagens de programação, baseados em regras, baseados em autômatos estendidos (como redes de petri ou máquinas de estado) e multiparadigma (combinam dois ou mais paradigmas).

Na literatura são descritos vários ambientes que suportam a definição e a execução de processos de software, descritos usando PMLs. Esses ambientes são chamados PSEEs (*Process-centered Software Engineering Environments*) e foram desenvolvidos na década de 80 e no início da década de 90 (FUGGETA, 2000; CUGOLA, 1998). Alguns exemplos incluem: Marvel (BARGHOUTI, 1996), Merlin (JUNKERMANN, 1994) SPADE (BANDINELLI, 1996), Arcádia (TAYLOR, 1988), IPSE 2.5 (WARBOYS, 1990), Adele/tempo (BELKHATIR, 1991), EPOS (CONRADI, 1994), etc.

Fuggeta (2000), conclui que PMLs, independente do paradigma usado, são complexas, extremamente sofisticadas, fortemente orientadas a modelagem de processos detalhados. As PMLs não são adotadas na prática devido a essa necessidade de definição extremamente detalhada dos processos, que dificulta o seu uso. Esses problemas se refletem nos PSEEs. O esforço inicial para configuração de processos em PSEEs é muito alto, e como nesses ambientes todos os detalhes devem ser modelados, os PSEEs são rígidos e inflexíveis.

Alguns autores citam os ambientes de gerência de configuração como os PSEEs reais. Ambientes de gerência de configuração visam automatizar somente os fragmentos de processos que parecem óbvios de serem automatizados (FUGGETA, 2000).

Outra possibilidade para representação dos modelos de processo é por meio da notação *Unified Modeling Language* (UML) (BOOCH, 2002), como proposto em OMG e pelo RUP (RATIONAL, 2003). UML possibilita uma maior flexibilidade na definição dos processos (OMG, 2002).

Uma organização pode definir sua própria maneira de desenvolver software. Porém, algumas atividades são comuns a todos os processos (ACUÑA, 1999). A norma ISO/IEC 12207 (1995) descreve um conjunto de processos para desenvolver e manter um software. Essa norma não prescreve um ciclo de vida específico. A organização pode instanciar um processo de software a partir dos processos preconizados pela norma.

Nos últimos anos tem aumentado o interesse em métodos, modelos e técnicas para suportar a avaliação e melhoria de processos de software (CONRADI, 2002). O objetivo da melhoria de processos de software é melhorar a capacidade de uma organização de desenvolver softwares melhores. Para melhorar seus processos de software, a organização precisa conhecer o estado atual de seus processos (LEPASAAR, 2002). Avaliações de processo ajudam as organizações de software a melhorar seus processos por identificar seus problemas críticos e estabelecer prioridades de melhoria (HUMPHREY, 1990).

Uma avaliação de processo é baseada em um modelo de processo. Várias organizações têm proposto e suportado modelos e normas para melhoria de processos,

como o *Capability Maturity Model for Software* (SW-CMM) (PAULK, 1993) e o *Capability Maturity Model Integration* (CMMI) (CMMI, 2002), propostos pelo *Software Engineering Institute* (SEI); as normas ISO/IEC 15504 (1995-a), ISO/IEC 12207 (1995) e a ISO 9000-3 (1991), propostas pela *International Organization for Standardization*; Trillium (HAASE, 1994), Bootstrap (BELL CANADÁ, 1994) e *Quality Improvement Paradigm* (BASILI, 1988), entre outros.

Os projetos de software apresentam diferenças quanto à tecnologia, clientes, riscos, formalismo, entre outros. Definir um processo de software que possa ser utilizado em todos os projetos da organização é inviável, devido às características específicas de cada projeto. É necessário definir um processo que possa ser adaptado de acordo com as características do projeto no qual será utilizado (BOEHM, 2002).

Para que o processo padrão possa ser adaptado a processos específicos é necessário que exista uma definição clara do processo da organização, e que esse processo possa ser reutilizado por meio de adaptação. É importante que na adaptação sejam preservadas as propriedades do processo padrão (CMMI, 2002; YOON, 2001). Tendo um processo padrão é possível melhorar constantemente este processo por meio de avaliações de processo, e todos os processos da organização serão beneficiados, já que são obtidos a partir deste.

A norma ISO/IEC 12207 descreve um processo de adaptação com as atividades que devem ser executadas para adaptar a norma para um projeto de software específico (ISO/IEC 12207, 1995). Os modelos de avaliação de processos *Capability Maturity Model* (CMM) (PAULK, 1993; PAULK, 1993-a) e *Capability Maturity Model Integration* (CMMI) (CMMI, 2001), e a norma ISO 15540 (ISO 15540, 1995-a) descrevem guias para auxiliar a tarefa de adaptação de processos da organização para atender necessidades de projetos específicos.

Porém, adaptar o processo padrão da organização para uso em um projeto específico não é uma tarefa trivial. Essa atividade requer conhecimento e experiências dos gerentes de projeto e projetistas de software. Vários trabalhos exploram o uso de gerência de conhecimento (*Knowledge Management – KM*) da organização para suportar as atividades de engenharia de software (FALBO, 2004; XU, 2002). Em uma edição especial da IEEE Software de May/June 2002 (RUS, 2004) foram publicados artigos sobre iniciativas práticas do uso de gerenciamento de conhecimento em engenharia de software. Os artigos reportam as necessidades, implementações, resultados, fatores de sucesso e lições aprendidas em aplicações de gerência de conhecimento. Em (SCHNEIDER, 2002) os autores reportam uma experiência prática do uso de KM na DaimlerChrysler, onde foi criado o *Software Experience Center* para investigar experiências de reuso e encorajar a melhoria do processo de software com base na experiência. Wei et al. (2002) reportam implicações de KM baseado em um sistema para uma empresa de montagem e teste de circuitos integrados. O sistema suporta a criação, atualização, compartilhamento e reuso do conhecimento. Liebowitz (2002) descreve uma série de iniciativas de KM na NASA *Goddard Space Flight Center*.

Os processos tradicionais focam no planejamento exaustivo das atividades que serão executadas no projeto, e uma grande quantidade de documentação é gerada como resultado da aplicação desse tipo de processo. Infelizmente, esse tipo de processo nem sempre funciona, na história de projetos de software encontramos grandes catástrofes, projetos cancelados e a perda de muito dinheiro, como o Aeroporto *Denver* (DEMARCO, 2003) e Ariane 5 (GLEICK, 2005).

Uma nova geração de desenvolvedores cita a quantidade de documentação, a rápida evolução das tecnologias e os efeitos da desumanização dos métodos planejados (BOEHM, 2002) como a causa para a mudança de paradigma. Esses desenvolvedores propõem o uso de métodos ágeis para desenvolver software.

Métodos ágeis focam na comunicação entre os membros da equipe, e com o cliente; e na agilidade como forma de atingir os objetivos do projeto (FOWLER, 2002). Métodos ágeis devem ser utilizados se a flexibilidade, adaptabilidade, e mudanças nos requisitos são as principais características do projeto e se controle do progresso e produtos são necessidades secundárias (KELTER, 2002).

Empresas com uma grande base de clientes não necessitam apenas de rapidez ou de alta qualidade - necessitam de ambas. Agilidade pura ou disciplina dirigida a planos pura não pode sozinha encontrar essas necessidades. É necessário combinar ambas (BOEHM, 2002).

Segundo Boehm (2002), análise de riscos das características do projeto versus as características de um dado método pode ajudar a determinar um melhor balanço de disciplinas dirigidas a planos ou ágeis.

Boehm propôs os modelos de processo Espiral e Win-Win que preconizam uma análise de riscos como critério para decisão de qual alternativa utilizar para desenvolver o software. O modelo de processo de software em espiral, proposto por Boehm em 1988, representa o desenvolvimento de software como uma espiral, ao invés de representar o processo de software como uma seqüência de atividades com retorno de uma atividade para outra (SOMMERVILLE, 2003). Em cada iteração, a análise de riscos pondera diferentes alternativas em face de requisitos e das restrições. A prototipação verifica a viabilidade e a adequação, antes que haja a decisão por alguma alternativa. Quando os riscos são identificados, o gerenciamento do projeto decide como eliminar ou minimizar cada risco (PFLEEGER, 1998). O Modelo Espiral Win-Win é uma extensão proposta por Boehm do Modelo Espiral. O Modelo Win-Win encoraja os participantes a convergir para um entendimento comum dos objetivos, das alternativas e limitações do próximo nível do sistema.

Métodos ágeis não trabalham bem com grandes equipes, essa é uma das maiores limitações desses métodos. Metodologias ágeis trabalham muito bem quando os requisitos são incertos e voláteis. Se os requisitos mudam constantemente, não será possível ter um projeto estável, então um processo planejado deve ser evitado. Nestes casos, um processo adaptável pode ser bem mais efetivo (FOWLER, 2002).

As atividades, preconizadas por métodos ágeis e planejados, podem ser mescladas dando origem a processos híbridos, que aproveitam as melhores práticas desses dois métodos de acordo com o contexto do projeto.

Em uma edição especial da revista *Computer* (junho/2003) foram publicados artigos sobre metodologias ágeis, descrevendo o estado atual das pesquisas realizadas sobre métodos ágeis, experiências na implantação de métodos ágeis e como combinar métodos ágeis e planejados, visando elaborar processos híbridos (WILLIAMS, 2003). Kent Beck e Barry Boehm (2003) sugerem que os processos planejados e ágeis obtêm sucesso se aplicados em projetos com características que se adaptem ao método, independente se é ágil ou planejado. Ambos os processos exigem disciplina, mesmo que "disciplina" tenha significado diferente em cada um dos processos. Boehm e Turner (2003) apresentam uma abordagem de como estruturar projetos para incorporar métodos

ágeis e planejados em proporção às necessidades do projeto. Lippert et al. (2003) propõem extensões a XP que focam em aspectos de controle e planejamento de projetos, demonstrando que um processo de desenvolvimento de software ágil adaptado pode ser aplicado em projetos grandes e longos. Cohm e Ford (2003) descrevem suas experiências com a implantação de *Scrum* em sete organizações de software. Em (LYCETT, 2003), os autores reportam suas experiências na introdução de métodos ágeis em organizações que utilizam ISO 9000 e CMMI.

Nas próximas seções são descritos os conceitos básicos, relacionados ao processo de software, necessários ao entendimento desta tese. A seção 2.1 descreve sobre processos de software tradicionais e, em especial, sobre o *Rational Unified Process*. A seção 2.2 apresenta os métodos ágeis e *Extreme Programming* (XP). A seção 2.3 descreve sobre *Capability Maturity Model* (CMM) e a adaptação de processos proposta por este modelo. A seção 2.4 apresenta padrões de processo e organizacionais, usados para descrever ações preventivas a riscos. A seção 2.5 descreve sobre medições em processos de software e, em especial, sobre a Abordagem *Goal/Question/Metric*.

2.1 Processos Tradicionais

Processos de software tradicionais, também conhecidos como projetos planejados, impõem um processo disciplinado para desenvolvimento de software com o objetivo de tornar o desenvolvimento de software previsível e eficiente, por meio de um processo detalhado. Esses processos de desenvolvimento estão sendo utilizados há anos, mas até hoje apresentam problemas como: estimativas imprecisas de prazos e custos, baixa qualidade dos produtos desenvolvidos, insatisfação dos clientes, etc.

Exemplos de processos de software tradicionais incluem: *Rational Unified Process* (RATIONAL, 2001), OPEN (, 2001), IBM *Global Services Method* (NICHOLS, 2005), DMR *Macroscope* (OMG, 2002), e processos elaborados para atender normas ou modelos de qualidade (SW-CMM, CMMI, ISO9000-3, ISO/IEC12207, ISO/IEC15504, etc.).

Rational Unified Process (RUP) será apresentado na próxima seção, pois foi o escolhido para representar métodos planejados no metamodelo elaborado.

2.1.1 *Rational Unified Process*

Rational Unified Process (RUP) (RATIONAL, 2001) é um processo de engenharia de software que provê um enfoque disciplinado para distribuição de tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a produção de produtos de alta qualidade que satisfaçam as necessidades de seus usuários finais, dentro de cronogramas e orçamentos previsíveis.

RUP descreve como implantar comercialmente abordagens comprovadas para desenvolvimento de software. Estas são chamadas “melhores práticas”, porque elas são comumente usadas por organizações bem-sucedidas. As melhores práticas são (RATIONAL, 2001; KRUCHTEN, 2000):

1. Desenvolvimento iterativo;
2. Gerência de requisitos;
3. Uso de arquitetura baseada em componentes;

4. Modelagem visual;
5. Verificação contínua da qualidade de software;
6. Gerência de alteração.

A Figura 2.1 mostra a arquitetura do RUP, a qual apresenta duas dimensões: o eixo horizontal representa como os aspectos dinâmicos do processo (tempo), expressos em termos de fases, iterações e marcos; o eixo vertical representa os aspectos estáticos do processo (disciplinas), isto é, como o processo é descrito por meio de: disciplinas, atividades, *workflows detail*, artefatos e papéis.

O gráfico, exibido na Figura 2.1, mostra como a ênfase varia ao longo do tempo. Por exemplo, nas primeiras iterações é gasto mais tempo em requisitos; nas últimas iterações é gasto mais tempo em implementação.

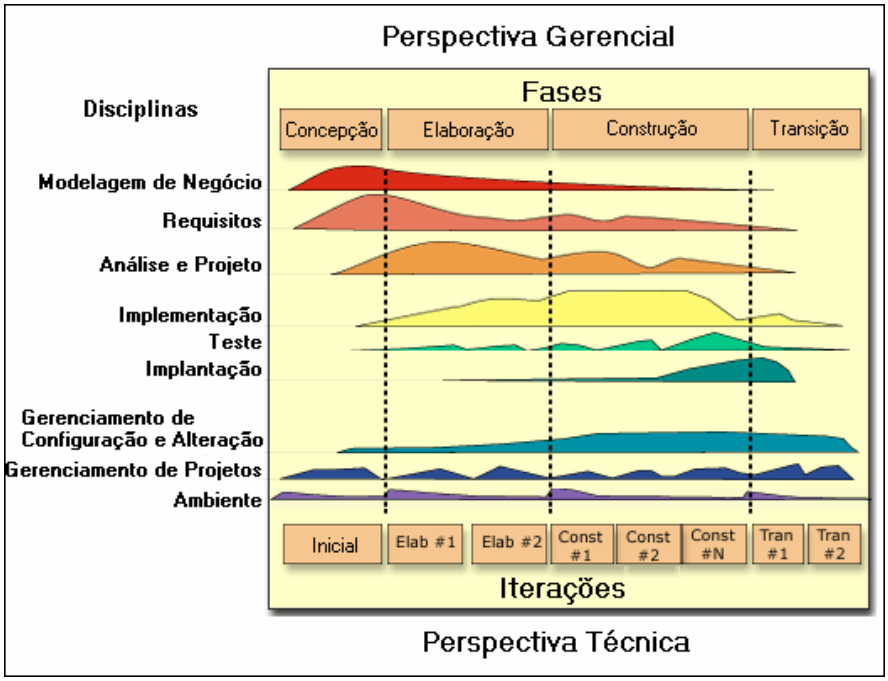


Figura 2.1: Estrutura do RUP – duas dimensões (RATIONAL, 2001)

O Processo Unificado é baseado em componentes, e utiliza a Linguagem de Modelagem Unificada (UML) como notação (KRUCHTEN, 2000). O Processo Unificado integra ciclos, fases, disciplinas, suavização de riscos, controle de qualidade, gerenciamento de projetos e controle de configuração.

2.1.1.1 Metamodelo Rational Unified Process

A Figura 2.2 mostra os conceitos de processo reconhecidos pelo RUP. Para cada conceito do processo é definido como ele é modelado (*operation* ou *class*) e suas associações com outros conceitos de processo (BENCOMO, 2005). Esse metamodelo foi encontrado na literatura referente ao RUP (BENCOMO, 2005). A seguir uma breve descrição de cada elemento descrito na Figura 2.2.

Lifecycle: do ponto de vista gerencial o ciclo de vida é composto por fases (*phases*) concluídas por um marco, e do ponto de vista técnico o ciclo de vida consiste de várias disciplinas (*disciplines*), como mostra a Figura 2.1.

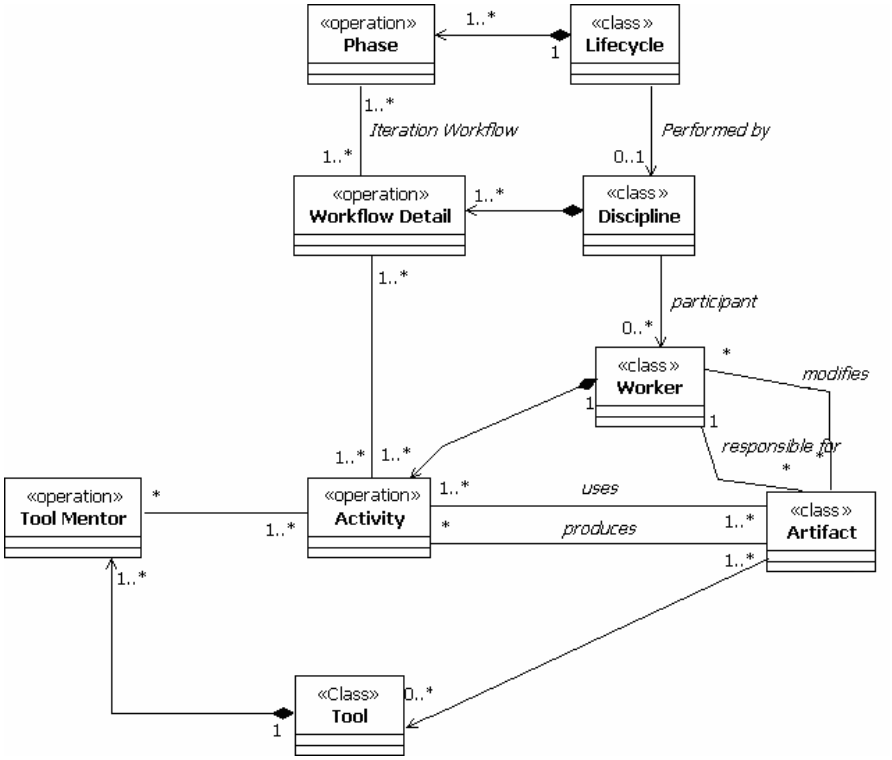


Figura 2.2: Elementos de processo do RUP (BENCOMO, 2005)

Phases: são associadas a *Workflow Details*.

Discipline: identifica um conjunto de trabalhadores (*workers)* que participa na disciplina (*discipline)* e define um conjunto de *workflows detail* da disciplina (*discipline)*.

Workflow Detail: especifica uma colaboração específica dentro de uma disciplina (*discipline*).

Activity: representa uma atividade do processo, que um papel (*worker)* deve executar.

Artifacts: são produtos gerados durante as atividades de desenvolvimento do software, tais como: modelos, planos, versões do software, relatórios, etc.

Worker: representa os cargos executados por indivíduos em um projeto.

Tool: representa as ferramentas de desenvolvimento usadas na organização.

Tool Mentor: descreve como executar uma atividade (*activity*), usando determinada ferramenta (*tool)*.

2.2 Métodos Ágeis

Em fevereiro de 2001, vários metodologistas se reuniram em Utah, nos Estados Unidos, para discutir as similaridades entre suas abordagens, que foram por eles denominadas de métodos ágeis (WILLIAMS, 2003). Esses novos métodos tentam balancear nenhum processo e muito processo, provendo apenas um processo suficiente para obter o resultado esperado (FOWLER, 2002). Métodos ágeis podem ser caracterizados por:

Desenvolvimento Iterativo

O objetivo do desenvolvimento iterativo é freqüentemente produzir versões do sistema final com um conjunto das funcionalidades requeridas. Esses sistemas são limitados nas funcionalidades, mas devem ser fiéis as demandas. Cada versão intermediária deve ser completamente integrada e cuidadosamente testada como se fosse a versão final (FOWLER, 2002). O desenvolvimento iterativo possibilita a entrega mais cedo e contínua de software operacional ao cliente.

Envolvimento do Cliente

O envolvimento do cliente é imprescindível para o sucesso de métodos ágeis. Em cada iteração, clientes e desenvolvedores verificam o progresso e mudam a direção do desenvolvimento de software. Isto conduz a uma relação muito mais restrita com o desenvolvimento de software, uma verdadeira relação de negócio (FOWLER, 2002).

Equipe com boa qualificação técnica

A equipe precisa ser efetiva tanto na qualidade dos indivíduos, como na maneira na qual se une para trabalhar junto. Bons desenvolvedores preferem processos adaptáveis (FOWLER, 2002). Em métodos ágeis, as pessoas da equipe não são tratadas como recursos, como acontece em métodos planejados. Os planos de projeto devem levar em consideração as diferenças entre as pessoas. Métodos ágeis consideram que a melhor forma da equipe se comunicar é pela conversação face a face.

Desenvolvedores devem tomar decisões técnicas

Métodos ágeis consideram que desenvolvedores são capazes de tomar decisões técnicas. Tal abordagem requer um compartilhamento de responsabilidades entre os desenvolvedores e o gerenciamento. O gerenciamento é responsável pela liderança do projeto, e os desenvolvedores pelas decisões técnicas (FOWLER, 2002).

Processo Adaptável

O processo de desenvolvimento precisa ser constantemente avaliado e melhorado. Ao final de cada iteração, são realizadas revisões para rever e melhorar o processo (HIGHSMITH, 2001).

Mudanças nos Requisitos

Mudanças nos requisitos são bem-vindas. Processos ágeis aproveitam as mudanças para obter vantagens competitivas para o cliente (HIGHSMITH, 2001).

Projetos Simples

Os projetos devem ser o mais simples possível, os desenvolvedores devem projetar somente o que é necessário para hoje.

Como exemplos de métodos ágeis citam-se: *Scrum* (SCHWABER, 2001), *Extreme Programming* (XP) (BECK, 2004), *Feature Driven D*evelopment (FDD) (PALMER, 2002), Adaptative Software Development (ASD), etc.

2.2.1 *Extreme Programming*

XP é uma metodologia leve (*lightweigh*) para times de tamanho pequeno a médio, que desenvolvem software em face a requisitos vagos que se modificam rapidamente. O termo "*extreme*", no nome XP, decorre do fato que XP leva princípios e práticas de

senso comum a níveis extremos, por exemplo: se testar é bom, todos vão testar o tempo todo; se revisar o código é bom, todos irão revisar o código o tempo todo (BECK, 2004).

Com XP, as atividades centradas em código estão em todas as etapas do ciclo de vida de desenvolvimento (HIGHTOWER, 2002). XP foi proposto por Kent Beck, com base em seus anos de prática como desenvolvedor de software orientado a objetos (BECK, 2004).

Em XP, o desenvolvimento é realizado de forma iterativa e incremental. No início de cada ciclo, o cliente identifica as funcionalidades mais importantes, isto é, as que trarão maior valor para seu negócio. Essas funcionalidades são construídas e constituem versões do software. A cada iteração, novas funcionalidades são agregadas e adicionam mais valor ao software (BECK, 2004).

XP é altamente disciplinado e foca nas práticas (BECK, 2004): revisões de código constante, testes freqüentes, envolvimento do cliente e rápido *feedback*, refatoração e refinamento da arquitetura, integração contínua para descobrir problemas mais cedo no processo de desenvolvimento, projeto e reprojeto à medida que novas funcionalidades vão sendo implementadas (*ongoing*) e planejamento constante.

2.2.1.1 Quatro Valores de XP:

Valores, em XP, são conceitos básicos que orientam a metodologia a ser seguida. Os valores resumem a filosofia de trabalho e a orientação a ser utilizada na tomada de decisão. Kent Beck definiu quatro valores-chave para XP, que são (HIGHTOWER, 2002):

- **Comunicação**: o objetivo da comunicação é prover um lugar onde as pessoas podem livremente discutir o projeto sem medo ou represália;
- **Simplicidade**: escolha o projeto, tecnologia, algoritmos e técnicas mais simples para satisfazer as necessidades do cliente para a iteração atual do projeto;
- **Retroalimentação (*Feedback*)**: obtido por meio de teste de código, *User Stories*, entregas freqüentes/pequenas iterações, programação por pares/revisões constantes de código e outros métodos;
- **Coragem**: seja corajoso o suficiente para fazer o que é certo. Teste de regressão é chave para atingir esse valor.

2.2.1.2 Cinco Princípios de XP

Com base nos quatro valores, Kent Beck (2004) definiu cinco princípios de conduta para XP. Esses princípios ajudam no momento de tomar decisões entre alternativas. A idéia é sempre escolher a alternativa que melhor preserva os princípios de XP. Os cinco princípios são:

- Retroalimentação rápida: quanto mais cedo se obtém o retorno sobre algo que foi feito, melhor se avalia se o resultado obtido foi satisfatório;
- Assuma simplicidade: XP prega a busca pela solução mais simples possível para cada problema. Projetos simples são mais fáceis de serem implementados e podem ser alterados, caso seja necessário;

- Mudanças incrementais: pequenas mudanças podem ser implementadas rapidamente e com baixo risco. Então, segundo XP, as mudanças devem ser realizadas aos poucos, em uma série de pequenos esforços;
- Mudanças são bem-vindas: durante o desenvolvimento de software várias mudanças vão ocorrer. A forma mais produtiva de encarar esse ambiente dinâmico é aceitar as mudanças e adicioná-las ao processo de desenvolvimento;
- Trabalho de qualidade: executar as atividades de desenvolvimento de software com qualidade. Quanto maior a qualidade do código produzido mais fácil é seu entendimento, teste e manutenção. Além disso, o trabalho de qualidade contribui para o bem estar do pessoal envolvido, melhorando sua produtividade e sua satisfação com o processo.

A partir desses princípios são definidas as práticas a serem seguidas durante o desenvolvimento de um sistema.

2.2.1.3 Doze Práticas de XP

As práticas que devem ser seguidas durante o desenvolvimento de um projeto XP são brevemente descritas abaixo (HIGHTOWER, 2002; BECK, 1999; BECK, 2004).

Jogo do Planejamento (*Planning Game*)

O objetivo dessa prática é o planejamento da iteração. No jogo do planejamento, o software é efetivamente planejado pela equipe do cliente e pela equipe de desenvolvimento. Cada equipe tem um papel bem-definido. A equipe do cliente decide o escopo e as prioridades do próximo release. A equipe técnica, por outro lado, determina a complexidade das funcionalidades idealizadas para o sistema, e estima seu custo (em termos de tempo de desenvolvimento).

Regras básicas:

1. O cliente apresenta uma lista de funcionalidades desejadas para o sistema. Cada funcionalidade é escrita como uma ***User Story*** **(US)** em um cartão. Cada história é descrita em um cartão indexado e possui um nome e as funcionalidades requeridas.

2. A equipe de desenvolvimento, com base em projetos anteriores, estima quanto de esforço cada US exige, e quanto de esforço a equipe pode produzir em um determinado intervalo de tempo (iteração). Se uma história for muito grande para ser desenvolvida em uma iteração ou não puder ser estimada adequadamente, significa que a mesma está muito complexa e deve ser segmentada em histórias menores. Por outro lado, se for muito pequena, deve ser agregada a outras para formar uma história de tamanho adequado. Entende-se por histórias adequadas, aquelas que possam ser desenvolvidas em dois ou três dias de programação. A relação entre as funcionalidades entregues e as funcionalidades solicitadas por um cliente para uma iteração é chamada de velocidade da equipe de desenvolvimento.

3. Com base nas estimativas realizadas, o cliente define as histórias que serão implementadas na próxima iteração. Os clientes devem levar em consideração o princípio de que devem receber antes o que vai gerar mais retorno para o seu negócio.

Pequenas versões (*Releases*)

A filosofia acima de pequenas versões é prover o maior valor para o negócio com a menor quantidade de esforço de codificação. Comece com um conjunto pequeno de

funcionalidades que seja bastante útil ao cliente. Crie versões mais cedo e freqüentemente, adicionando uma funcionalidade de cada vez.

Projeto (*design*) simples

A idéia de projeto simples é manter o código simples. Sempre faça o projeto mais simples possível para o trabalho que está sendo feito. Os requisitos podem mudar amanhã, mas faça somente o que é necessário para o projeto hoje.

Teste

Os programadores escrevem testes unitários para as funcionalidades, antes de implementá-las. Os testes automatizados são fundamentais para a XP, uma vez que dão suporte a outras técnicas da metodologia, como refatoração e propriedade coletiva de código. Sem testes automatizados, é muito difícil e trabalhoso implementar estes dois princípios, pois não se tem a confiança necessária para saber se o código alterado não vai apresentar problemas. Testes em XP podem ser:

Testes de unidade: são testes automatizados escritos pelos programadores para testar funcionalidades como eles as codificam.

Testes de aceitação: são especificados pelo cliente para testar se o sistema completo está funcionando como especificado. Quando são executados todos os testes de aceitação em uma determinada *User Story*, está é considerada concluída.

Integração Contínua

Geralmente, existe uma máquina dedicada ao processo de integração. O sistema é integrado várias vezes por dia, cada vez que uma tarefa é concluída. O desenvolvedor, após integrar seu código, executa todos os testes automáticos. Caso ocorra algum problema, este deve ser corrigido imediatamente. Para facilitar a integração contínua, os processos de *build*, distribuição e implantação devem ser automatizados.

Refatoração

A técnica de refatoração é utilizada sempre com o intuito de simplificar o código. Refatoração significa reescrever o código, melhorando e simplificando o mesmo, sempre que for possível. Sempre que uma nova funcionalidade for adicionada deve-se verificar como fazer a implementação de forma que o código continue simples. Testes ao nível de unidade automatizados dão aos desenvolvedores coragem de refatorar e ainda manter o código simples.

Programação em pares

Todo o código produzido é escrito por dois programadores sentados em uma mesma máquina. Dessa forma, todo o código é revisado enquanto é escrito.

Responsabilidade Coletiva

Ninguém é proprietário do módulo. Qualquer programador deve ser capaz de trabalhar em qualquer parte do código, em qualquer tempo. Qualquer programador pode fazer alterações ou refatorar código para implementar uma nova funcionalidade, em qualquer tempo. Por outro lado, é necessário que sejam feitos todos os testes para garantir que as alterações não geraram erros no sistema.

Semana de 40 horas

Se os desenvolvedores não conseguem realizar seus trabalhos em 40 horas, algo está errado no projeto. XP aconselha que trabalhar mais que 40 horas por semana não é produtivo. Se mais do que uma semana de hora-extra é necessário, o processo precisa ser revisto.

Cliente presente

Os clientes devem estar disponíveis quando os desenvolvedores precisam solucionar dúvidas, e se possível, estar localizado junto aos desenvolvedores.

Metáfora

Uma metáfora é uma linguagem comum e um conjunto de termos usados para visualizar as funcionalidades do projeto. A metáfora é uma história simples que deve descrever como o sistema funciona e serve para guiar o desenvolvimento. Clientes e desenvolvedores devem ter um entendimento comum sobre o sistema e devem falar usando um dialeto único (BECK, 2004).

Padrões de codificação

Os programadores devem utilizar um padrão único de codificação, facilitando o entendimento dos códigos entre os programadores.

XP trabalha bem com equipe acima de 12 programadores, mas acima de 24 programadores encontram-se algumas dificuldades. As opções para solucionar as dificuldades podem ser: incorporar práticas de metodologias planejadas ou dividir a equipe em duas ou mais equipes XP, com algumas equipes transformando-se em clientes para as outras equipes.

2.2.1.4 Metamodelo Extreme Programming

A partir da literatura sobre XP foi elaborado um metamodelo para representar os conceitos descritos por XP, já que este não foi encontrado na literatura. Utilizou-se na elaboração do metamodelo do XP, os mesmos conceitos do metamodelo do RUP, para facilitar a integração desses modelos.

O objetivo deste trabalho é elaborar um metamodelo que possibilite a instanciação de métodos ágeis. Não é objetivo deste trabalho propor um metamodelo definitivo e exaustivo para XP. A Figura 2.3 mostra o metamodelo elaborado.

***Activity*:** atividades são modeladas como operações (*operations*). Uma atividade é executada por um papel (*XP Role*) e é associada a um ou mais artefatos (*artifacts*).

***Artifacts*:** artefatos são modelados como classes e representam os produtos de trabalho criados ou alterados durante a execução de atividades (*activities*) por papéis (*XP Role*), tais como: *user stories*, relatórios de teste, modelos, código, etc.

***XP Role*:** papéis são modelados como classe. Papéis (*XP Roles*) executam atividades (*activities*) e criam e alteram artefatos (*artifacts*). São considerados *XP Roles* o cliente, programador (desenvolvedor), treinador (XP Manager), medidor (Tracker) e o chefão.

***Guidelines*:** prevê conselhos de como determinado artefato (*artifact*) deve ser elaborado para obter sucesso. Essas indicações são geradas a partir das práticas XP, tais como: padrões de codificação, propriedade coletiva de código e semana de 40h. São modeladas como operações.

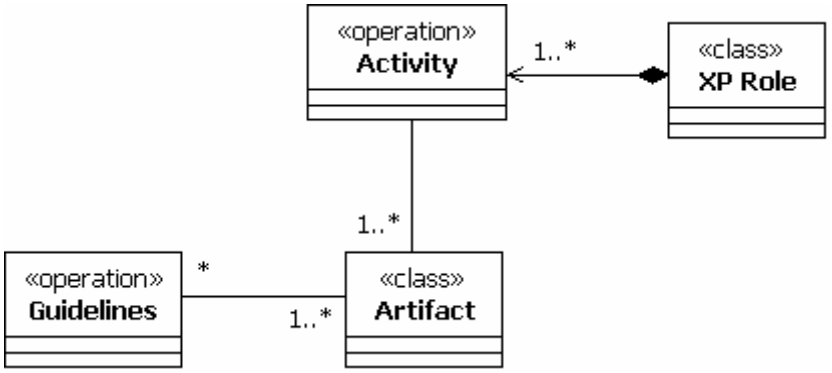


Figura 2.3: Elementos de processo do XP

Para demonstrar que é possível representar um modelo de processo XP usando o metamodelo definido, são elaborados um diagrama de objetos com elementos de processo XP instanciados a partir do metamodelo e um diagrama de atividades de um processo XP. A Figura 2.4 mostra os elementos de processos do XP instanciados a partir do metamodelo da Figura 2.3.

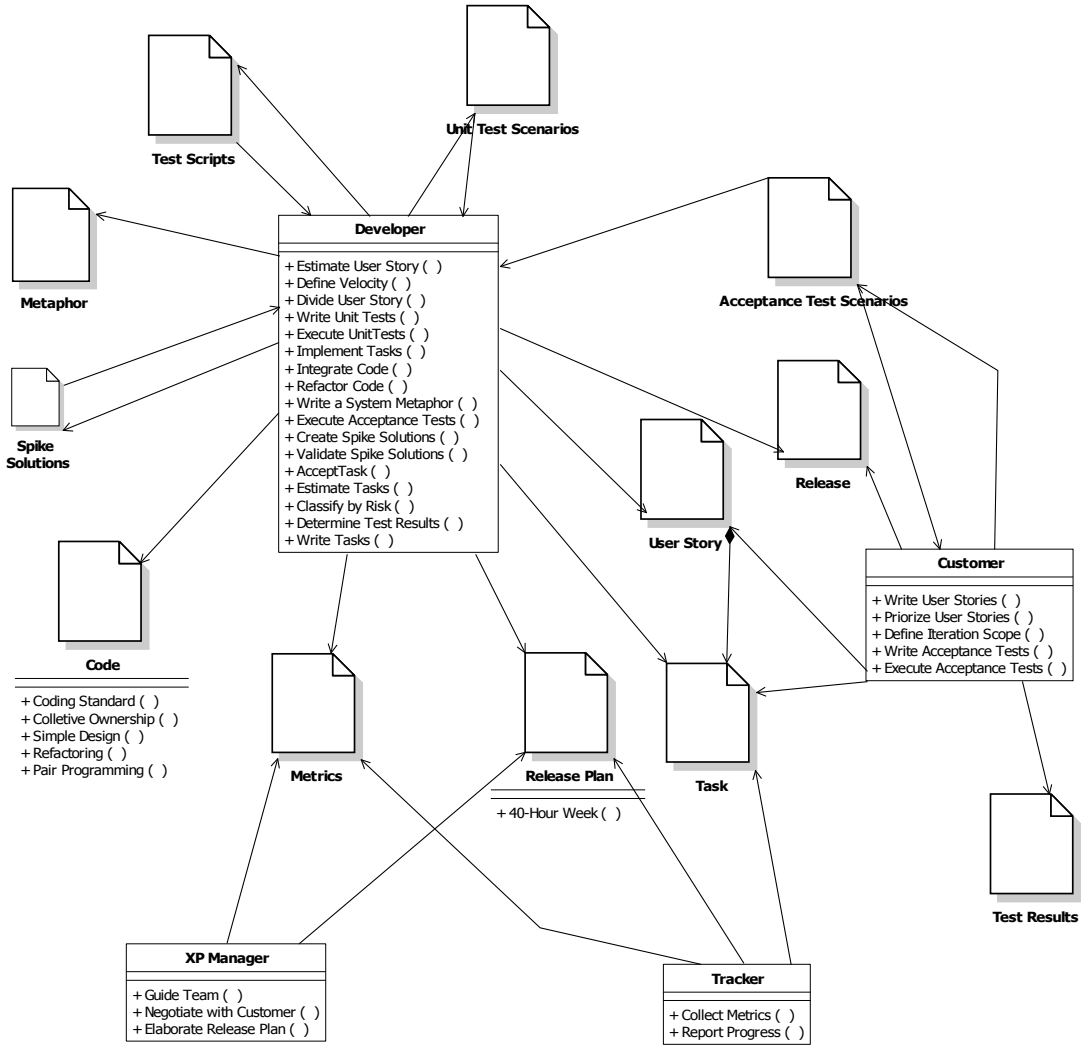


Figura 2.4: Diagrama de objetos de XP

A Figura 2.5 mostra a seqüência das atividades de um processo XP, usando um diagrama de atividades, como um exemplo.

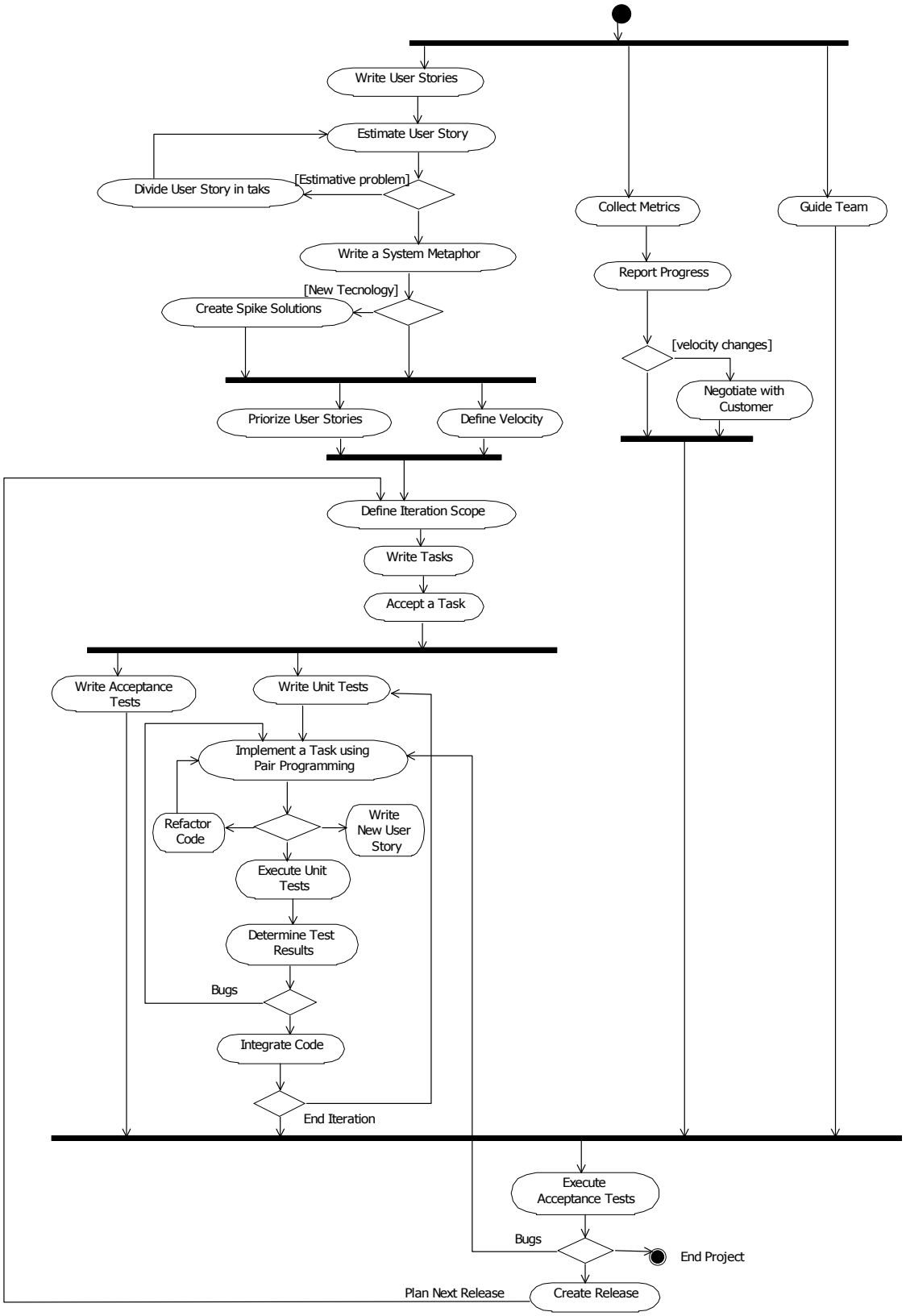


Figura 2.5: Diagrama de atividades de XP

2.3 *Capability Maturity Model* e Adaptação de Processo

Capability Maturity Model for Software (CMM ou SW-CMM) foi desenvolvido em 1991, com o objetivo de avaliar a capacidade e a maturidade de uma organização. O CMM foi desenvolvido pelo *Software Engineering Institute* (SEI), da *Carnegie Mellon University*, e foi apoiado pelo *Department of Defense* (DoD) dos EUA, que é um grande consumidor de software e precisava de um modelo formal que o permitisse selecionar seus fornecedores de software de maneira adequada (PAULK, 1993).

O CMM é um *framework* que se caracteriza por uma melhoria de processo, voltado a organizações mais maduras (PAULK, 1996). Uma organização pode usar o CMM para determinar sua classificação de maturidade do processo de software e então estabelecer prioridades para melhoria (PAULK, 1999).

No SW-CMM, os processos de software são organizados em cinco níveis de maturidade. Essa estrutura em níveis do CMM está baseada nos princípios de qualidade propostos por Walter Shewart, W. Edwards Deming, Joseph Juran e Philip Crosby (PAULK, 1993-a).

Tabela 2.1: Níveis de Maturidade do CMM

Nível	Foco	Área-chave de Processo (KPA)
5 Otimizado	Melhoria contínua do processo	Prevenção de Defeitos Gerência de Mudanças de Tecnologia Gerência de Mudanças de Processo
4 Gerenciado	Qualidade do processo e produto	Gerência Quantitativa do Processo Gerência Qualitativa de Software
3 Definido	Processos de Engenharia e suporte organizacional	Foco no Processo da Organização Definição do Processo da Organização Programa de Treinamento Gerência de Software Integrada Engenharia de Produto de Software Coordenação Inter-grupos Revisão por Pares
2 Repetível	Processos de Gerenciamento de Projetos	Planejamento de Projetos de Software Acompanhamento e Supervisão de Projetos Garantia da qualidade de Software Gerência de Configuração de Software Gerência de Subcontratação de Software
1 Inicial	Pessoas competentes e heróicas	

Fonte: PAULK, 1999. p. 2.

Além do SW-CMM, o SEI criou outros modelos de avaliação para melhoria de outros tópicos relacionados ao desenvolvimento de sistemas, tais como: *Software*

Acquisition CMM (SA-CMM), *Systems Engineering CMM* (SE-CMM), *Integrated Product Development* CMM (IPD-CMM) e *People* CMM (P-CMM). Existiam diferenças estruturais e de conceitos entre esses modelos, dificultando a implantação de mais de um desses modelos nas organizações (CMMI, 2002). Então foi elaborado o *Capability Maturity Model Integration* (CMMI) para integrar esses diferentes modelos, numa estrutura única, terminologia e processos de avaliação, facilitando a sua adoção pelas empresas.

Neste trabalho, a adaptação de processos é realizada de acordo com o CMM. O relatório técnico *"Process Tailoring and the Software Capability Maturity Model"* (GINSBERG, 1995) descreve como adaptar o processo padrão da organização para uso em projetos específicos.

2.3.1 Adaptação de Processos segundo CMM

Segundo o CMM, adaptar um processo é alterar a descrição de um processo para um fim particular. Por exemplo, um processo definido para um projeto é definido por adaptar o conjunto de processos padrão da organização para encontrar os objetivos, restrições e ambiente do projeto específico. É importante definir dois conceitos utilizados pelo SW-CMM que são: o processo de software padrão da organização e o processo definido para um projeto.

O processo de software padrão da organização descreve os elementos de processo fundamentais que são esperados que cada projeto incorpore em seu processo de software definido, e as relações entre esses elementos de processo de software.

O processo de software definido para um projeto é a definição operacional do processo de software usado no projeto. Ele é desenvolvido pela adaptação do processo de software padrão da organização de acordo com as características do projeto.

SW-CMM preconiza algumas áreas-chave, que objetivam a definição do processo de software padrão da organização e a customização deste processo para projetos específicos, são elas (PAULK, 1993-a):

- **Definição dos Processos da Organização** tem como finalidade desenvolver e manter o processo de software padrão da organização, assim como os bens relacionados ao processo, tais como: descrição do ciclo de vida do software, guias e critérios de adaptação para o processo, base de dados de processos de software da organização e uma biblioteca de documentação relacionada ao processo de software;
- **Foco nos Processos da Organização** tem como finalidade estabelecer a responsabilidade organizacional para as atividades do processo de software que melhoram a capacidade geral do processo de software da organização. Foco nos Processos da Organização envolve desenvolver e manter um entendimento dos processos de software dos projetos e da organização, e coordenar as atividades para avaliar, desenvolver, manter e melhorar esses processos;
- **Gerenciamento Integrado de Software** tem como finalidade integrar as atividades de engenharia e gerência de software em um processo de software coerente e definido, que é adaptado do processo de software padrão da organização e bens de processo relacionados, os quais são descritos na Definição do Processo da Organização. Gerenciamento Integrado de Software envolve

desenvolver o processo de software definido para um projeto e gerenciar o projeto de software, usando este processo de software definido. O processo de software definido do projeto é adaptado do processo de software padrão da organização para enfocar as características específicas do projeto.

2.4 Padrões de Software

Padrões são maneiras de descrever melhores práticas, bons projetos, e capturar experiências de uma forma que seja possível para outros reusarem essa experiência. O objetivo dos padrões para a comunidade de software é documentar experiências para ajudar desenvolvedores de software a resolver problemas recorrentes encontrados durante o desenvolvimento de software. Padrões são identificados a partir da experiência coletiva de desenvolvedores de software (HILLSIDE, 2003).

A proposta original de padrões tem origem no trabalho de Christopher Alexander, na área de arquitetura. Segundo Alexander, cada padrão é uma regra de três partes que expressa uma relação entre um certo contexto, um problema, e uma solução (CUNNINGHAM, 2004).

Padrões de software tornaram-se populares a partir da ampla aceitação do livro *Design patterns: elements of reusable object-oriented software*, de Eric Gamma et al. (GAMMA, 1994). Os autores do livro são conhecidos como *Gang of Four*, ou simplesmente GoF. Padrões de projeto são os padrões mais populares.

Existem também anti-padrões, que descrevem práticas ruins, que não funcionam em determinado contexto de projeto. Anti-padrões têm como objetivo documentar lições aprendidas.

Padrões apresentam as seguintes características:

- Endereçam um problema recorrente em uma situação específica, e apresentam uma solução para isto (HILLSIDE, 2003);
- Não são inventados ou criados artificialmente (BUSCHMANN, 1996). São soluções que já foram utilizadas em projetos anteriores, e que provavelmente serão utilizadas no futuro;
- Provêem vocabulário e entendimento comum para os princípios de projeto (BUSCHMANN, 1996). Uma vez que um padrão abstrai uma solução, não existe a necessidade de uma longa e detalhada explicação do problema e o que foi feito para sua solução. Ao citar o nome do padrão utilizado, automaticamente serão conhecidos o problema, o contexto e a solução adotada. Ao longo do tempo, estes nomes vão se tornando parte do vocabulário comum de uma equipe de desenvolvimento;
- Auxiliam na construção de arquiteturas de software complexo e heterogêneo, no caso de padrões de projeto (HILLSIDE, 2003) ou de processos de software, no caso de padrões de processo e organizacionais;
- Auxiliam no gerenciamento da complexidade do software ou de processos (HILLSIDE, 2003), abstraindo detalhes da solução.

Segundo Coplien (1996) um padrão pode ser descrito pelos seguintes itens:

Nome: o padrão deve ter um nome significativo. Esse nome identifica o padrão de forma a facilitar sua escolha, e, rapidamente se torna parte do vocabulário da equipe de projeto.

Problema: descreve o problema que o padrão se propõe a resolver.

Contexto: as pré-condições sob as quais o problema e sob solução parecem recorrer, e para os quais a solução é desejável. O contexto descreve a situação na qual existe um problema.

Forças: descrevem os requisitos que caracterizam o problema, as restrições que devem ser aplicadas e as propriedades desejáveis à solução.

Solução: descreve a solução conhecida para o problema.

Contexto Resultante: o estado do sistema após o padrão ter sido aplicado.

Raciocínio: aprofunda o entendimento do raciocínio por trás do padrão e sobre o problema e a solução.

Uma linguagem de padrões é uma coleção de padrões inter-relacionados, que funcionam de forma integrada, visando transformar requisitos e restrições em uma arquitetura (COPLIEN, 2004). Exemplos de metodologias de desenvolvimento, descritas na forma de uma linguagem de padrões são: a linguagem original de Coplien (COPLIEN, 1995) e a metodologia *Scrum* (SCHWABER, 2001).

2.4.1 Padrões Organizacionais e de Processo

Os padrões que interessam ao escopo deste trabalho são os padrões organizacionais e de processo. Coplien (1995) foi um dos primeiros pesquisadores a estudar padrões de processo e organizacionais. Ele propôs uma linguagem de padrões para descrever e construir novas organizações.

Segundo Coplien, as organizações de software bem-sucedidas apresentam os mesmos padrões organizacionais e de processo. Esses padrões não são encontrados em organizações menos produtivas ou não tão bem-sucedidas. Os padrões de organizações bem-sucedidas podem ser capturados e usados para estabelecer estruturas organizacionais e práticas que podem melhorar a probabilidade de sucesso em novas organizações.

Padrões organizacionais e de processos capturam práticas de sucesso no gerenciamento do desenvolvimento de software (DEVEDZIC, 2002) e podem ser utilizados para modelar uma nova organização e um novo processo de desenvolvimento para um projeto, ou então melhorar um processo já existente.

Scott Ambler (1998) considera que padrões de processo descrevem uma coleção de técnicas, ações e/ou tarefas (atividades) gerais para desenvolver software orientado a objetos. Uma importante característica de um padrão de processo é que ele descreve o que deve ser feito, mas não exatamente os detalhes de como fazer. Quando aplicados juntos de uma maneira organizada, padrões de processo podem ser usados para construir processos de software para a organização. Padrões de processo são considerados blocos de processo reutilizáveis, que podem ser usados para adaptar o processo de software para encontrar as necessidades específicas da organização.

Padrões organizacionais descrevem técnicas de gerenciamento ou estruturas organizacionais. Segundo Ambler (1998), padrões de processo e padrões organizacionais caminham lado-a-lado.

Coplien e Harrison (2004) propõem uma divisão dos padrões em quatro linguagens de padrões inter-relacionadas. Essas linguagens devem ser usadas em conjunto para solucionar problemas de desenvolvimento de software de uma organização, são elas:

- *Project Management Pattern Language*: descreve aspectos organizacionais de gerenciamento de projetos (cronograma, progresso, controle, etc.);
- *Piecemeal Growth Pattern Language*: descreve padrões para fortalecer e coordenar uma organização, usando *feedback* e percepção. É um processo de reparação;
- *Organizational Style Pattern Language*: descreve o relacionamento entre os papéis de uma organização;
- *People and Code Pattern Language*: explica o relacionamento entre a estrutura de uma organização e os artefatos que são construídos.

Padrões podem ser descritos usando extensões da UML como proposto por Hagen (2004) e Störrle (2001).

Padrões de processo podem ser utilizados para adaptar e melhorar o processo de desenvolvimento de software, como proposto por Huang e Zhang (2003), ou para processos específicos, como o uso de padrões para melhoria do processo de inspeção de software proposto por Harjumaa et al. (2004). May e Taylor (2003) propõem o uso de padrões para converter informações em conhecimento.

2.5 Medição em Processos de Software

Embora o primeiro livro dedicado a métricas de software tenha sido publicado somente em 1975 (GILB apud FENTON, 2000, p. 360), a história de métricas de software data de meados de 1960, quando a métrica de linhas de código foi utilizada como base para medir produtividade e esforço em programação. Apesar de, métricas de software antecederem a própria engenharia de software (reconhecida em 1968), há ainda uma pequena consciência do que são as métricas de software e de sua importância (FENTON, 2000).

Métricas de software é um termo coletivo usado para descrever um conjunto de atividades para medição em engenharia de software. Uma métrica de software pode ser definida como um método de determinar quantitativamente uma medida para determinado atributo do processo, produto, projeto ou recurso. Isto inclui não apenas a fórmula usada para determinar a medida e a forma de coletar o valor, mas também o gráfico para apresentar os valores da métrica, e os guias que serão usadas para interpretar o gráfico (e a métrica) no contexto de projetos específicos (OMAN, 1997).

A coleta e a análise de métricas, durante a execução do processo de software, possibilitam o monitoramento dos projetos de software, a melhoria contínua do processo de desenvolvimento e a avaliação da qualidade do produto (NASA, 1994).

Com base nos dados coletados, os gerentes de projeto podem fundamentar suas decisões, produtos podem ser comparados entre si, o desempenho de profissionais pode ser comparado com seu histórico, avaliando impactos de treinamento e ferramentas,

processos podem ser avaliados e melhorados, projetos podem ser acompanhados e estimativas traçadas, etc.

Geralmente é aceito que a medição não é um fim em si, mas um meio para o fim. Assim, os objetivos finais para o estabelecimento de um programa de medição em uma organização de software são (NASA, 1994):

- Compreensão dos processos de software e avaliação dos produtos,
- Gerência de projetos de software;
- Melhoria contínua;
- Melhoria pessoal (HUMPHREY, 1997).

O propósito subjacente de qualquer programa de mensuração deve ser o de alcançar os resultados específicos da coleta e a interpretação dos dados de mensuração. Sem tais objetivos, nenhum benefício será alcançado através do esforço da medição (OMAN, 1997).

Medição para compreender produtos e processos de software

Fundamental para a melhoria sistemática de processos e produtos de software é a compreensão e o estabelecimento de linhas-base quantitativas em relação aos processos atuais da organização. Esta compreensão dos processos e das práticas da organização é crucial para planejar, controlar e melhorar o desenvolvimento e a manutenção do software (WANGENHEIM, 2000).

Um primeiro passo para abordar estes assuntos é estabelecer uma linha-base dos produtos e dos processos de software da organização. Com base na informação quantitativa, obtida por meio de um programa de medição, uma organização pode construir seus modelos de qualidade específicos e pode examinar fatores de contexto que têm impacto nos modelos.

Fenton et al. (1994) descrevem sobre a importância da medição para avaliar a efetividade de qualquer tecnologia proposta. Eles explicam que a experimentação cuidadosa é necessária para determinar os efeitos de novas técnicas ou ferramentas na qualidade de software resultante. Os autores apresentam guias para avaliação de experimentos e estudos de casos.

Medição para gerenciar projetos de software

Os modelos organizacionais, baseados na compreensão dos produtos e processos de software e nos fatores de influência destes, podem ser usados para prover informações (quantitativas) para decisões gerenciais. O conhecimento ganho pela análise e interpretação dos dados medidos pode ser usado para: planejamento de novos projetos de software com base em estimativas (esforço requerido, cronograma, alocação da equipe, etc.), controle de um projeto por meio da comparação dos valores estimados com os valores atuais coletados e validação dos modelos organizacionais, desenvolvidos em programas de medição passados (NASA, 1994).

Boehm (1991) propõe um modelo para gerenciar os riscos que podem afetar o projeto de software. Usando este modelo, uma organização de desenvolvimento pode reduzir a probabilidade e a exposição a vários riscos.

Weller (1994) mostra como métricas coletadas de projetos passados podem ser usadas no planejamento de novos projetos.

Medição para guiar melhoramento

O objetivo principal de uma organização é desenvolver um produto de alta qualidade, dentro de um cronograma e orçamento previsíveis (FITZGERALD, 1999). A melhoria do produto é obtida por meio da melhoria dos processos usados para produzir o produto. A melhoria dos processos pode ser atingida pela modificação dos processos gerenciais ou técnicos, ou pela introdução de novas tecnologias. Medição de software é chave para qualquer programa de melhoria.

Focando na melhoria da qualidade, os pontos fortes e fracos podem ser identificados pelo programa de medição inicial. Com base nessa compreensão inicial, áreas de melhoria potenciais com alto impacto na qualidade do produto podem ser identificadas e ações de melhoria podem ser selecionadas, executadas e avaliadas. O impacto dessas mudanças precisa ser avaliado, quantitativamente, pela medição, monitorando as modificações. Ele só pode ser determinado, se uma linha-base quantitativa está disponível, contra a qual as comparações podem ser feitas. De outra maneira é impossível determinar se uma mudança obteve um efeito positivo ou não (WANGENHEIM, 2000).

Daskalantonakis (1994) descreve como a Motorola conduz e monitora avaliações incrementais para ajudar os projetos a alcançar suas metas de melhoria a longo prazo. As técnicas descritas por Daskalantonakis são úteis para prover *feedback* junto a programas de melhoria de longo prazo.

Grady (1994) compartilha sua experiência na aplicação de um programa de medição bem-sucedido na Hewlett-Packard.

Medição para Melhoria Pessoal

PSP é um *framework* de técnicas que visam ajudar o desenvolvedor a melhorar seu desempenho pessoal e a qualidade de seu trabalho, por meio de uma disciplina pessoal de controle e análise quantitativa de seu trabalho (HUMPHREY, 1997).

Por ser tão voltado às pessoas, este processo exige total empenho e dedicação dos envolvidos, de maneira que o sucesso da implantação depende, quase que exclusivamente, das pessoas.

O PSP exige a coleta de diversos dados durante o desenvolvimento. É preciso apurar o tempo consumido em diversas atividades, obter a evolução do tamanho da aplicação, contabilizar erros, enfim, este método depende muito da "qualidade" dos dados coletados.

Zhong (2000) descreve os fatores críticos que afetam os processos de software pessoais. O artigo discute os resultados de um experimento, realizado pelos autores na Universidade de McGill, para estudar os fatores críticos que afetam os processos pessoais. Em (SILLITTI, 2003), os autores propõem uma ferramenta automatizada, chamada PROM (PRO *Metrics*) para coletar e analisar métricas de software e dados do PSP. A ferramenta coleta, automaticamente, dados de outras ferramentas, por meio de *plug-ins*.

Após definir claramente os objetivos da medição, é necessário definir o que será medido, isto é, escolher as métricas. A escolha das métricas não é uma tarefa fácil, e pode envolver recursos e tempo, nem sempre disponíveis. Um programa de medição deve iniciar com um pequeno conjunto de métricas, que será aumentado à medida que a organização sentir necessidade, de acordo com as suas metas. Para cada métrica é

necessário definir como está será utilizada (OMAN, 1997). Um projeto de software pode ser visualizado de pelo menos duas perspectivas: *bottom-up* e *top-down*.

A visão *bottom-up* inicia com os produtos e tarefas de desenvolvimento do software, analisando como estes estão relacionados, e a partir desses elementos se chega as metas do projeto. Hetzel (1997) defende a abordagem *bottom-up*. Ele propõe que organizações de desenvolvimento de software iniciem o processo de desenvolvimento coletando registros de dados em todos os produtos de trabalho desenvolvidos e usados.

A visão *top-down* inicia com as metas do projeto, analisando como as metas se desdobram em atividades e produtos para enfocá-las. Cada uma dessas perspectivas pode ser usada na seleção de métricas apropriadas.

Basili e Rombach (1988) propõem o uso de uma abordagem *top-down* para medição de software, chamada de *Goal/Question/Metric*. Essa abordagem é discutida em detalhes na seção 2.5.1.

Outra forma de definir quais métricas serão coletas é por meio de modelos de qualidade, como CMM ou CMMI; ou normas de qualidade, como as normas ISO.

Pfleeger (1990) sugere um conjunto de métricas a serem coletadas e analisadas, com base no CMM. Clark utiliza métricas para quantificar os efeitos de melhoria de processos, usando o modelo CMM, com base no esforço (CLARK, 2000).

É importante salientar que o objetivo principal dos desenvolvedores é desenvolver o produto, não coletar métricas (OMAN, 1997). Então, é importante que as atividades de coleta e análise de métricas sejam o mais simples possível, não impactando nas atividades de desenvolvimento de software, e se possível, sejam utilizadas ferramentas, modelos e padrões para facilitar a coleta.

Segundo Lavazza (2000), ferramentas automatizadas podem transformar projetos de desenvolvimento de software caótico em processos gerenciáveis e controlados no futuro, por meio da medição e melhoria dos processos.

Pflegeer e Fitzgerald (1991) provêem um *framework* para auxiliar os desenvolvedores a avaliar ferramentas automatizadas para a coleta, armazenamento e análise de métricas. Usando um esquema de classificação, geralmente utilizado para repositórios de reuso, eles sugerem um método para organizar palavras-chave que descrevem as ferramentas e seus usos.

Michael Daskalantonakis (1992) descreve uma visão multidimensional das métricas que formam a base do programa de métricas na Motorola. Nesse estudo de caso, são descritas lições aprendidas em medições de produtos e processos na Motorola.

Neste trabalho, a medição é utilizada para monitorar os fatores de riscos de projetos de software. A abordagem *Goal/Question/Metric* foi escolhida porque apresenta várias vantagens, tais como: suporta a identificação das métricas úteis e relevantes, suporta a análise e interpretação dos dados coletados, permite uma validação das conclusões obtidas e por ser baseada em metas diminui a resistência contra programas de medição (BASILI, 1988; WANGENHEIM, 2000).

2.5.1 Abordagem *Goal/Question/Metric*

O paradigma *Goal/Question/Metric* (GQM) (BASILI, 1988) é uma abordagem orientada a metas e utilizada em engenharia de software para a medição de produtos e

processos de software. GQM é baseado no requisito de que toda a coleta dos dados deve ser baseada num fundamento lógico, em um objetivo ou em uma meta, que é documentado explicitamente.

O primeiro passo nessa abordagem é definir as metas a serem alcançadas no programa de medição. Após a identificação das metas, um plano GQM é elaborado para cada meta identificada. 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 (BASILI, 1988). A Figura 2.6 mostra essa estrutura.

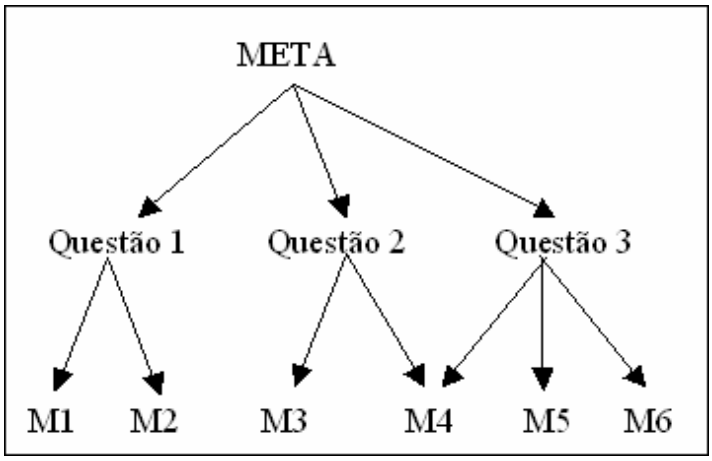


Figura 2.6: Estrutura da abordagem *Goal/Question/Metric* (BRIAN, 1996)

2.5.1.1 Processo de Medição GQM

As fases do processo GQM são orientadas pelo Paradigma de Melhoria de Qualidade (*Quality Improvement Paradigm* – QIP). QIP foi desenvolvido na The *University of Maryland* em cooperação com o *Software Engineering Laboratory* (SEL), da NASA.

QIP baseia-se no conceito de que a melhoria do processo e do produto ocorre com base nas experiências acumuladas ao longo dos projetos de software. Essas experiências devem ser acumuladas (aprendizado) em um formato que possa ser efetivamente entendido e modificado (modelos de experiência), e em um repositório integrado (base de experiências). Esse repositório deve ser acessado e modificado de acordo com as necessidades do projeto (reuso). Para apoiar o QIP foi desenvolvida uma estrutura distinta de desenvolvimento de software, denominada de Fábrica de Experiências (*Experience Factory* - EF) (OIVO, 1992).

QIP propõe seis passos principais para um processo de medição, baseado em GQM, como mostra a Figura 2.7 (BASILI, 1988; BRIAN, 1996; OIVO, 1992).

Passo 1: Caracterizar o ambiente

Identificar as características relevantes do projeto e da organização onde está sendo implantado o programa de medição. Nessa fase são utilizadas questões como:

- Que tipo de produto está sendo desenvolvido?
- Qual o processo que está sendo usado?

- Quais são os principais problemas encontrados nos projetos anteriores?

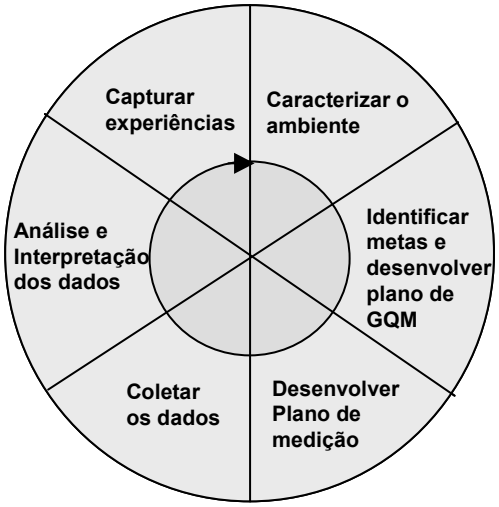


Figura 2.7: Processo de medição GQM (BRIAN, 1996)

Passo 2: Identificar as metas e desenvolver o plano de GQM

Com base nas metas organizacionais e do projeto e nas características do ambiente, determinar as metas a serem alcançadas pelo programa de medição. Essas metas servem como base para o desenvolvimento de um programa de medição efetivo. As metas GQM devem ser formuladas da seguinte forma: "Analisar o <*objeto de estudo*> com a finalidade de <*objetivo*> com respeito ao <*enfoque*> do ponto de vista de <*ponto de vista*> no seguinte contexto <*contexto*>".

Sendo que os atributos, em itálico na sentença acima, podem ser definidos como:

- ⇨ Objeto de estudo: identifica o que será analisado. Exemplo: processo de software, teste, documento de projeto, projeto, sistema, etc.
- ⇨ Objetivo: porque o objeto será analisado. Exemplo: avaliar, melhorar, monitorar, controlar, modificar, etc.
- ⇨ Enfoque: identifica o atributo que será analisado. Exemplo: confiabilidade, custos, correção, remoção de defeitos, modificações, manutenibilidade, etc.
- ⇨ Ponto de vista: identifica quem utilizará as métricas coletadas. Exemplo: equipe de desenvolvimento, gerente de projeto, desenvolvedor, equipe de garantia de qualidade, usuário, cliente, etc.
- ⇨ Contexto: identifica o ambiente onde o programa de medição está localizado. Exemplo: projeto A, departamento X.

Passo 3: Desenvolver o plano de medição

Para todas as metas identificadas durante o passo 2, procedimentos de coleta precisam ser definidos, isto é, quando e como os dados serão coletados e quem os coletará. Para otimizar os procedimentos de coleta de dados e limitar o esforço, o processo de desenvolvimento é o principal elemento que deve ser considerado quando se define os procedimentos de coleta.

Planos GQM contêm a informação necessária para planejar a medição, analisar e interpretar os dados coletados. De acordo com as metas GQM, um plano GQM é desenvolvido para cada meta GQM identificada. Um plano GQM consiste de uma meta

GQM e um conjunto de questões, modelos e medidas. O plano define porque as medidas foram definidas e como elas serão usadas. As questões identificam a informação necessária para atingir a meta. As medidas definem os dados a serem coletados para responder as questões. O modelo usa os dados coletados como entrada para gerar respostas às questões.

Passo 4: Coletar os dados

Coletar os dados do projeto, analisar esses dados e interpretar os resultados da análise junto ao pessoal técnico e de gerenciamento.

Durante a fase de execução do programa de medição, os dados são coletados de acordo com o plano de mensuração.

Após a coleta, os dados passam por um processo de garantia de qualidade antes de serem armazenados ou analisados. O processo de garantia de qualidade dos dados coletados aborda os seguintes assuntos:

- Completude: garantir que todos os questionários foram respondidos e estão completos;
- Validação: verificar se os dados são do tipo especificado e se estão dentro do intervalo de valores possíveis.

Caso sejam encontrados problemas no processo de garantia de qualidade, os problemas devem ser discutidos com os coletores. Quando possível, busca-se corrigir os dados. Se alguns dados apresentam baixa confiabilidade regularmente, os procedimentos de coleta de dados e o treinamento dos coletores devem ser reconsiderados, avaliados e melhorados. Os dados válidos são armazenados no banco de dados de medição e disponibilizados para análise.

Passo 5: Análise e interpretação dos dados

A análise dos dados serve para avaliar as práticas atuais, determinar problemas, registrar resultados, e sugerir melhorias. O objetivo da análise é identificar padrões e relações entre atributos para permitir o estabelecimento de linhas-base (situações típicas ou usuais) e a identificação de áreas com problemas.

A análise de dados é realizada com base no plano GQM. Os dados atuais são comparados com a linha-base (dados históricos). Com base na comparação, hipóteses sobre as relações entre os fatores de qualidade e os fatores de variação são formuladas. Essas hipóteses são validadas e quantificadas.

Os dados coletados e analisados são interpretados no contexto da meta de medição. A interpretação dos dados coletados é feita em *feedback session*s, e reuniões dos representantes. O objetivo dessas sessões é interpretar os dados coletados com os especialistas no assunto. Assim, os dados analisados são apresentados aos participantes da reunião, possíveis interpretações são discutidas e melhorias planejadas.

Passo 6: Capturar experiência

Essa etapa tem como objetivo estruturar e armazenar documentos, resultados da análise dos dados, lições aprendidas no projeto e no programa de medição de uma forma reutilizável.

Este paradigma tem como objetivo fornecer uma base de aprendizado e melhoria organizacional. Melhoria é possível somente se:

- Houver um entendimento do status atual do ambiente;
- Determinar as metas de melhoria para um projeto particular e quantificá-las com a finalidade de controle;
- Escolher o modelo de execução de processo, métodos e ferramentas, apropriadamente, para alcançar as metas de melhoria;
- Executar e monitorar o desempenho do projeto minuciosamente, e avaliá-lo.
- Com base nos resultados de avaliação, forem executadas sessões de *feedback* para os projetos em andamento ou no planejamento de projetos futuros.

Essas experiências de projetos passados são armazenadas na Fábrica de Experiências (*Experience Factory* - EF). As experiências são armazenadas em termos de soluções adotadas, para que as equipes possam fazer reuso das soluções e dos códigos já desenvolvidos.

Fábrica de Experiências é uma estrutura física e/ou lógica que apóia o desenvolvimento de sistemas por meio de uma análise e síntese de experiências, semelhante em um repositório de experiências, que são disponibilizadas aos outros projetos (OIVO, 1992).

Vários artigos reportam experiências práticas do uso de GQM na indústria, tais como os trabalhos reportados por Wang e Qing He (2003), Solingen e Berghout (2001), e Latum et al. (1998).

Olsson e Runeson (2001) propõem uma abordagem chamada de V-GQM (*Validating Goal/Question/Metric*) para analisar um estudo GQM, depois que os dados tenham sido coletados, como um meio de facilitar o estudo. Briand et al. (2002) propõem uma abordagem para definir medidas de atributos de produtos em engenharia de software. A abordagem é dirigida por metas experimentais de medição, expressas via o paradigma GQM, e um conjunto de hipóteses empíricas.

3 GERÊNCIA DE RISCOS EM PROJETOS DE SOFTWARE

Segundo Tom DeMarco e Timothy Lister, "risco é um evento futuro possível que conduzirá a um resultado indesejado" (2003, p. 16). Como uma alternativa, os autores sugerem uma definição circular para risco: "risco é um problema que ainda não ocorreu, e um problema é um risco que já se materializou" (2003, p. 17).

Antes de ocorrer, o risco é apenas uma abstração. É algo que pode ou não afetar o projeto. Gerenciamento de riscos é o processo de pensar nas ações corretivas antes que os riscos se materializem, enquanto eles são apenas abstrações. O oposto de gerenciamento de riscos é gerenciamento de crise. No caso de gerenciamento de crise, as ações a serem tomadas são decididas após os problemas já terem ocorrido.

Gerenciamento de riscos deve focar nas questões que podem pôr em risco o alcance dos objetivos críticos. Uma abordagem de gerenciamento de riscos contínua é aplicada para efetivamente antecipar e mitigar os riscos que têm impacto no projeto (CMMI, 2002).

Gerenciamento de riscos é baseado em teorias que provêem estratégias para a tomada de decisão, sob condições probabilísticas (HALL, 1998). Todas as estratégias tentam melhorar a qualidade da decisão na avaliação de duas ou mais alternativas de ação.

Segundo uma pesquisa, realizada com 21 gerentes seniores, que teve como objetivo determinar os fatores que conduzem a projetos de sucesso, gerência de riscos é citada como uma das práticas de gerência de projetos que conduz a um projeto de software bem-sucedido (VERNER, 2005). Outros fatores de sucesso, citados na pesquisa, são: visão clara do produto final, requisitos bem-definidos e revisões *post-mortem*.

Boehm descreve um conjunto de atividades para gerenciamento de riscos de software. Ele considera que gerenciamento de riscos envolve dois passos primários principais, que são: avaliação de risco e controle de risco (BOEHM, 1991).

A avaliação de risco define o risco. Avaliação de risco é o processo de descoberta das fontes de riscos e avaliação de seus efeitos potenciais (HALL, 1998). A avaliação de risco envolve as atividades: identificação de risco, análise de risco e priorização de risco.

- **Identificação de risco:** resulta em uma lista dos possíveis riscos de um projeto específico. Os riscos podem ser identificados por meio de listas de verificação (*checklists*), reuniões entre os membros da equipe, entrevistas, *brainstorms, etc.*

- **Análise de risco:** avalia a probabilidade e a magnitude da perda associadas a cada risco identificado.
- **Priorização de risco:** produz uma lista dos riscos analisados, ordenada por importância.

Controle de risco identifica como resolver o risco. Controle de riscos é o processo de desenvolver planos de resolução de riscos, monitorar o status do risco, implementar planos de resolução de riscos, e corrigir desvios do plano (HALL, 1998).

O controle do risco envolve os subpassos: planejamento de gerenciamento de riscos, resolução do risco e monitoramento do risco (BOEHM, 1991).

- **Planejamento do gerenciamento do risco:** ajuda a se preparar para mitigar cada item de risco (por exemplo, via compra de informação, evitando o risco, transferindo o risco ou reduzindo o risco). O plano de gerenciamento de riscos do projeto é composto pelos planos dos itens de risco individuais.
- **Resolução de riscos:** consiste em implementar as técnicas de resolução de riscos descritas no plano de gerência de riscos, visando eliminar ou resolver os itens de risco.
- **Monitoramento de riscos:** envolve acompanhar o progresso do projeto para resolver os itens de risco e tomar ações corretivas quando apropriado. Técnicas típicas incluem: acompanhamento de marcos e da lista dos 10 principais itens de risco. A lista de riscos é acompanhada semanalmente, mensalmente, ou nas revisões dos marcos do projeto.

A seguir uma breve descrição sobre essas atividades.

3.1 Identificação de Riscos

A identificação de riscos consiste em listar os riscos que podem afetar o projeto, e documentar as características desses riscos. O objetivo não é identificar todos os possíveis riscos, mas todo risco com alto impacto, considerado possível de se materializar, deve ser incluído na análise (HALL, 1998).

Diferentes técnicas são usadas para identificação de risco nas organizações, as mais comuns são: entrevistas, listas de verificação (*checklists*), e *brainstorms*. É bastante comum o uso de taxonomia de riscos para guiar a identificação de riscos do projeto (HALL, 1998) como por exemplo, a taxonomia de riscos, proposta pelo SEI (CARR, 1993) e por Webster (WEBSTER, 2005).

Entrevistas podem ser realizadas com os membros da equipe, individualmente ou com toda a equipe. O método *Software Risk Evaluation* (SRE), desenvolvido pelo *Software Engineering Institute* (SEI), propõe um processo de entrevistas sistemático para identificar riscos de projetos de software (HIGUERA, 1996).

Boehm (1991) propõe o uso de *checklist* na identificação dos riscos de projeto, e sugere um *checklist* baseado nos dez principais riscos de projeto.

Coppendale (1995) sugere uma abordagem estruturada para gerenciamento de riscos, elaborada com base na experiência prática de projetos. Nessa abordagem, a identificação de riscos é realizada por meio de entrevistas individuais e em grupo e

seções de *brainstorms*. Após a identificação, são avaliados a probabilidade e o impacto de cada risco, e elaborados planos de gerenciamento de riscos para os riscos potenciais.

Yau (1994) propõe definir categorias de fontes de riscos e categorias de conseqüências de riscos para os projetos de software. Cada categoria de fontes de riscos deve ser dividida em um número de fatores de risco que representam os elementos de riscos relacionados. Para cada categoria de riscos é estabelecida uma tabela com as implicações dos diferentes níveis de riscos. É construído um questionário para avaliar os valores dos fatores de riscos. O questionário deve associar um peso e um intervalo de valores para cada fator de risco.

Alguns autores, tais como Lynne Nix (2002) e Tom DeMarco e Timothy Lister (2003), consideram que para iniciar uma análise de riscos deve-se considerar a experiência de projetos passados similares, para entender onde e como os projetos já concluídos deram errados e porque; também devem ser consideradas as causas de sucesso, para serem repetidas. Se uma organização regularmente avalia projetos passados e disponibiliza os resultados, estes são uma ferramenta valiosa para avaliação de risco, mesmo que, os projetos tenham diferenças (NIX, 2002).

Segundo Hall (1998), o processo de identificação de riscos é suficiente quando ele satisfaz os seguintes objetivos:

- Encoraja a identificação de riscos percebidos pela equipe;
- Identifica riscos, enquanto a tempo para tomar ações;
- Descobre riscos e fontes de riscos;
- Captura riscos em um formato compreensível;
- Comunica os riscos para aqueles que podem resolvê-los;
- Previne surpresas durante o projeto.

As atividades do processo de identificação de riscos são as tarefas necessárias para transformar incerteza em enunciados de riscos, e são (HALL, 1998):

Realizar uma avaliação de riscos

A avaliação de riscos identifica e avalia os riscos com base em critérios pré-estabelecidos (probabilidade de ocorrência, conseqüência, e tempo para ação). Provê uma linha-base de riscos avaliados que serão gerenciados pelo projeto e então deve ser executada no início do projeto. Avaliações de riscos subseqüentes são recomendadas nos marcos principais ou quando ocorrem mudanças significativas no projeto, em áreas como custo, cronograma, escopo, ou equipe.

Identificar riscos sistematicamente

Existem muitas maneiras de identificar riscos, normalmente os métodos de identificação se enquadram em uma das seguintes categorias: listas de verificação, entrevistas, reuniões periódicas, revisões, pesquisas, modelos de formulários, e grupos de trabalho.

Definir os atributos dos riscos

Após a identificação dos riscos, é necessário definir a probabilidade e a conseqüência para cada risco.

Documentar riscos identificados

A maneira mais concisa para especificar um risco é por meio de uma breve descrição do risco, da probabilidade, e da conseqüência em termos subjetivos. O uso de um formato padrão aumenta a legibilidade de um risco.

Comunicar riscos identificados

Comunicação de riscos é melhor quando realizada tanto escrita como verbal. A comunicação verbal provê o dialogo que favorece o entendimento, e a comunicação escrita provê um documento para propósitos históricos. Essas atividades podem ser executadas em paralelo ou de forma iterativa.

3.2 Análise e Priorização de Riscos

Análise de riscos é uma atividade que consiste em avaliar o nível dos riscos e reduzir as maiores perdas. Uma abordagem consistente para gerenciamento de riscos facilita a condução de uma análise de riscos continuamente, durante o projeto (BOEHM, 1991).

As técnicas de análise de riscos são ferramentas poderosas para ajudar as pessoas a gerenciar incertezas. A análise de riscos pode ser usada na tomada de decisão. Usando ferramentas automatizadas para análise de riscos, é possível armazenar, organizar e processar dados transformando-os em conhecimento significativo. Ferramentas de análise de riscos incorporam incerteza em suas estimativas para gerar resultados que testem todos os possíveis resultados (HALL, 1998).

Uma análise de riscos eficiente, pode ser seguida de uma abordagem proativa, onde os riscos potenciais são identificados mais cedo e medidas preventivas são tomadas para evitar os riscos de ocorrerem (APPULLUTTY, 2005).

Murthi (2002), assim como Appullutty et al. (2005), propõe uma abordagem de gerência de riscos preventiva, baseada em práticas flexíveis. Na abordagem de gerência de riscos preventiva, a equipe identifica e avalia os riscos e desenvolve estratégias de mitigação durante o desenvolvimento.

Segundo (HALL, 1998) o processo de análise de riscos é suficiente quando satisfaz os seguintes objetivos:

- Refina o contexto dos riscos;
- Determina as fontes de risco;
- Determina a exposição ao risco;
- Determina a janela de tempo para a ação;
- Determina os riscos de alta severidade.

As atividades do processo de análise de riscos são as tarefas necessárias para transformar declarações de riscos em uma lista de riscos priorizada. As atividades do processo de análise de riscos são:

Agrupar riscos similares e relacionados

Riscos similares podem ser agrupados com base em categorias de riscos. Riscos redundantes devem ser eliminados. Riscos relacionados podem ser combinados quando faz sentido trabalhar os riscos em conjunto. Riscos dependentes podem ser tratados em conjunto quando o resultado de um deve ser considerado, quando priorizado o outro.

Determinar variáveis de riscos

Variáveis de riscos podem causar variações significativas na conseqüência e probabilidade de riscos de software. Variáveis de performance são encontradas nas especificações técnicas. Variáveis de custo adicional são fatores encontrados em modelos de estimativa de custos de software. Variáveis de cronograma incluem os itens do caminho crítico. As variáveis de risco servem para determinar o contexto dos riscos e ajudam a avaliar os riscos.

Determinar as fontes de riscos

Fontes de risco são as causas principais dos riscos, determinadas por perguntar – Por que? – cinco vezes para cada risco. Embora a questão permaneça a mesma, a resposta difere até que a causa raiz seja determinada.

Usar técnicas e ferramentas para análise de riscos

Técnicas de análise de riscos tratam com metas de custos e desempenho conflitantes, incerteza, e preferência do risco. Técnicas de análise de riscos são usadas para estruturar, analisar, avaliar e comunicar problemas difíceis:

Estruturar: determinar a decisão, alternativas, incertezas, e valor de resultados e especificar suas relações. Diagramas de influência e árvores de decisão são úteis para estruturar um modelo de decisão e descrever possíveis cenários.

Analisar: o modelo de decisão para determinar as variáveis importantes e definir suas relações probabilísticas.

Avaliar: o modelo de decisão por calcular possíveis resultados para determinar perfis de riscos e a melhor política de decisão.

Comunicar: resultados de análise de riscos por compartilhar entendimentos e percepção para facilitar a tomada de decisão. Resultados de análise de riscos podem ser disponibilizados em uma base de riscos centralizada.

Existem ferramentas que automatizam a análise de riscos, usando diferentes algoritmos de decisão e podendo chegar a diferentes conclusões. Como exemplos de ferramentas de análise de riscos cita-se: quantificação da exposição ao risco (BOEHM, 1991; COPLIEN, 1995), análise de redes (MCCABE, 2001; FENTON, 2000), modelos de custos (FAIRLEY, 94), riscos baseado em performance (CORTELLESSA, 2005), análise de confiabilidade de risco (YACOUB, 2002), análise de regressão (MIZUNO, 2001), árvores de decisão (KLERK, 2001), simulação (MADACHY, 1996; HILGERS, 2000), etc.

Estimar a exposição ao risco

Exposição ao risco é calculada por multiplicar a probabilidade e a conseqüência do risco ocorrer. Probabilidade é definida como maior que 0 e menor que 100. Conseqüência é tipicamente determinada em relação ao custo, cronograma, e metas técnicas. Quantificar os riscos por exposição provê uma ordem de prioridade relativa para todos os riscos identificados (HALL, 1998).

Avaliar os riscos contra critérios

Critérios pré-definidos para avaliações asseguram que todos os riscos serão julgados contra o mesmo padrão. Severidade de riscos determina a prioridade relativa, por

mapear categorias de exposição de riscos contra os critérios de tempo. *Time frame* é o intervalo de tempo para a ação requerida ser tomada para prevenir o risco de ocorrer.

Ordenar os riscos relativos com outros riscos

Riscos podem ser agrupados por critérios de avaliação, então os riscos com exposição ao risco mais alta e com *time frame* menor são os primeiros revisados. Ordene os riscos para focar em recursos efetiva e eficientemente, e considere o *time frame* para obter uma lista priorizada dos riscos avaliados (HALL, 1998).

Na próxima seção será descrita a técnica de Quantificação da Exposição ao Risco, que é utilizada nesta tese para priorizar os riscos do projeto.

3.2.1 Quantificação da Exposição ao Risco

A exposição do risco, algumas vezes chamada de impacto do risco ou fator de risco, é definida como o produto entre a probabilidade de um resultado não satisfatório ocorrer e a perda associada a esse resultado não-satisfatório. Com base no resultado obtido na exposição do risco, os riscos identificados podem ser priorizados para serem enfocados.

ER = Probabilidade (Resultado NS) * Perda (Resultado NS)

Uma dificuldade na obtenção da exposição de riscos é fazer estimativas de entrada com acurácia da probabilidade e da perda associada ao resultado não satisfatório.

Em (COPPENDALE, 1995) é proposta uma escala para quantificação dos riscos de software. A probabilidade de um risco ocorrer e seu impacto podem ser avaliados em uma escala de 0 a 10 aplicando as seguintes guias:

Probabilidade:	0 representa a probabilidade de ocorrência de menos de 5%. 5 representa a probabilidade de aproximadamente 50% 10 representa a probabilidade maior que 95%
Impacto:	0 representa nenhum aumento no custo ou tempo do projeto 10 representa um aumento significativo no tempo ou no custo

Embora seja importante se chegar a um consenso sobre os valores, não é válido ficar muito tempo discutindo se, por exemplo, um escore de 6, 7, ou 8 é adequado. Em muitos casos, é suficiente avaliar a probabilidade apenas como alto, médio ou baixo (HALL, 1998). Quando é obtido um acordo quanto aos valores de probabilidade e impacto, estes são plotados em uma matriz, como mostra a Figura 3.1.

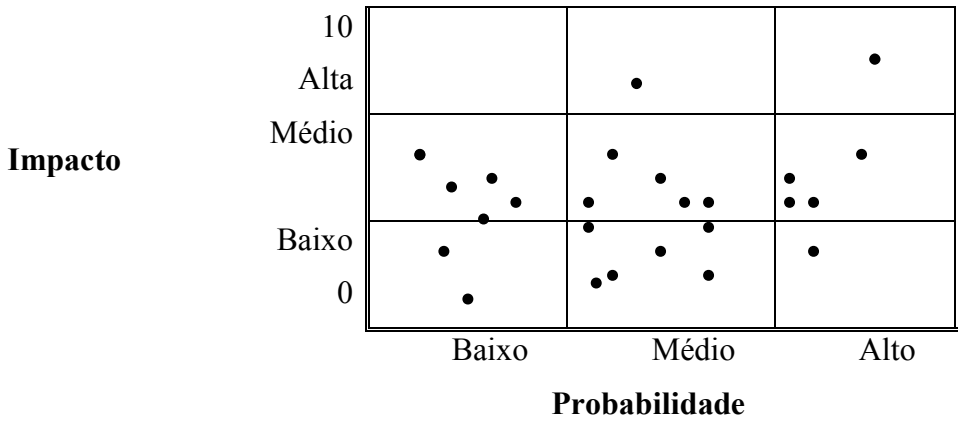


Figura 3.1: Matriz probabilidade x impacto (COPPENDALE, 1995)

Nem todos os riscos requerem ação imediata. Restrições práticas (recursos e custos) impossibilitam tratar todos os riscos, neste caso devem ser tratados os riscos com alta probabilidade de ocorrência e alto impacto.

Cada risco identificado para o projeto deve ser quantificado conforme a técnica. Como resultado da aplicação da técnica tem-se uma lista de riscos ordenada por exposição ao risco.

O gerente de projeto define se todos os riscos identificados serão enfocados, ou somente os riscos com ER maior que determinado valor. O custo para enfocar todos os riscos pode ser muito alto, e riscos com pouca probabilidade de ocorrer ou baixo impacto, talvez não justifiquem os custos para tratá-los (HALL, 1998).

3.3 Planejamento do Gerenciamento de Riscos

Uma vez que os principais riscos do projeto e suas prioridades relativas são determinados, é necessário estabelecer um conjunto de funções para manter os riscos sob controle. O primeiro passo nesse processo é desenvolver um conjunto de planos de gerenciamento de riscos que definam as atividades necessárias para manter os riscos sob controle (BOEHM, 1991).

O plano de gerenciamento de riscos detalha como gerenciar os riscos associados ao projeto de software. Ele detalha as atividades de gerenciamento de riscos que devem ser executadas, os responsáveis pela atividade, e qualquer recurso adicional necessário para executar as atividades de gerência de riscos (RATIONAL, 2001).

Em geral, o risco torna-se um problema quando o valor de uma métrica quantitativa cruza um limite pré-determinado. Por esta razão, é necessário determinar acima de quais limites são necessárias ações corretivas e quais são as ações que devem ser tomadas para gerenciar os riscos (FAIRLEY, 1994).

O processo de planejamento de riscos é suficiente quando ele satisfaz os objetivos (HALL, 1998):

- Provê visibilidade para eventos e condições importantes;
- Reusa estratégias de resolução de riscos com sucesso;
- Otimiza critérios de seleção;
- Entende as próximas ações para riscos de alta severidade;
- Estabelece mecanismos de disparos automáticos.

As atividades do processo de planejamento de riscos são as tarefas necessárias para transformar uma lista de riscos priorizada em um plano de resolução de riscos. As atividades do processo de planejamento de riscos são os seguintes (HALL, 1998):

Desenvolver cenários de riscos para os riscos de alta severidade

Um cenário de riscos é uma projeção de eventos e condições que podem levar a ocorrência do risco. Cenários de riscos devem ser desenvolvidos para todos os riscos que são críticos para o sucesso do projeto. Para desenvolver um cenário:

- Pense nos riscos como se eles tivessem ocorrido;

- Determine o cenário do risco como se ele já tivesse ocorrido;
- Liste os eventos e condições que poderiam preceder a ocorrência do risco.

Appukkutty et al. (2005) apresentam uma metodologia para avaliar riscos de software em nível de requisitos, usando especificações UML. Cada requisito é mapeado para um cenário operacional específico em UML. São determinados os modos de falhas do cenário e a complexidade do cenário. O fator de risco do cenário é obtido por combinar a complexidade da possível falha do cenário com a severidade da falha.

Desenvolver alternativas de resolução de riscos

Alternativas de resolução de riscos são: o conjunto de opções, que se implantadas, podem resolver riscos. Alternativas de estratégias de resolução de riscos incluem: aceitar, evitar, reduzir ou transferir o risco. Cada estratégia deve conter objetivos, restrições e alternativas (HALL, 1998).

Selecionar a abordagem de resolução de riscos

A abordagem de resolução de riscos reduz o conjunto de opções por focar nas melhores alternativas para reduzir riscos. Critérios de seleção ajudam a determinar a melhor alternativa para resolver um risco (HALL, 1998).

Desenvolver um plano de ação

O plano de ação de risco detalha a abordagem selecionada para resolução de risco. Ele documenta a abordagem, os recursos requeridos, aprovação e contém os seguintes elementos: pessoa responsável, recursos requeridos, data de início, atividades, prazo, ações tomadas e resultados alcançados (HALL, 1998).

Estabelecer limites para prevenção mais cedo

O plano de riscos não precisa ser implementado imediatamente. No início do projeto, a avaliação de riscos tende a identificar riscos importantes que ainda não são urgentes. Como a implantação não é imediata, esses itens de risco podem ser esquecidos de serem acompanhados. Torna-se necessário o uso de dispositivos de disparo, que mostrem quando um risco está próximo a se materializar, baseado em alvos quantitativos e limites (HALL, 1998).

Alvos quantitativos são as metas expressas quantitativamente, que provêem objetivos que podem ser determinados por métricas e comparações. Toda a medição deve ter uma meta quantitativa, ou alvo; e limites pré-definidos. Valores abaixo ou acima deste limite, dependendo do risco, definem o início de ocorrência do risco (HALL, 1998).

3.4 Monitorar Risco

Atividades de monitoramento de riscos incluem: medir o risco e observar os indicadores do projeto para obter informações com relação ao momento de executar o plano de ações de riscos. Um indicador implica em um valor sem especificar a quantidade diretamente (por exemplo, o número de pontos por função é uma medida de tamanho que indica a complexidade do software). Grupos de indicadores provêem visibilidade da situação do projeto (custo planejado versus custos atual). Um indicador de direção tem a capacidade preditiva. Indicadores ajudam o gerente a decidir o momento de tomar ações para evitar as conseqüências dos riscos (HALL, 1998).

O processo de monitoramento de riscos é suficiente quando satisfaz os seguintes objetivos (HALL, 1998):

- Monitora os eventos e condições dos cenários de riscos;
- Acompanha os indicadores de riscos para notificação dos riscos antecipadamente;
- Captura resultados de esforços de resolução de riscos;
- Reporta métricas e medidas de riscos regularmente;
- Provê visibilidade do status dos riscos.

As atividades do processo de monitoramento de riscos são as tarefas necessárias para monitorar o status dos riscos e prover notificação para disparar ações para resolução de riscos. As atividades do processo de acompanhamento de riscos são (HALL, 1998):

Monitorar os cenários de riscos

O tempo gasto para monitorar os cenários de riscos é um investimento justificável para riscos de alta severidade. Cenários de riscos são como cordas que amarram um risco a um problema. Cenários de riscos são monitorados para determinar se a probabilidade de ocorrência do risco está aumentando. Eventos e condições de um cenário de risco são acompanhados para decidir se, o aumento na exposição do risco justifica ação imediata. O acompanhamento de cenários de riscos pode aumentar o nível de confiança que a probabilidade está diminuindo, indicando o progresso já feito.

Comparar limites com status

Atividades de trabalho possuem status que são capturados em ferramentas de acompanhamento de projetos. Valores de indicadores fora de seus limites aceitáveis detectam condições inaceitáveis e servem como um sistema de aviso. Um gatilho (*trigger*) é um dispositivo para controlar a implementação de um plano de ações de riscos. Ele pode ser disparado por meio do monitoramento de indicadores de status, limites planejados, alvos quantitativos, e o cronograma do projeto. A variação entre limites planejados e o status atual provê informações em quando disparar o gatilho (*trigger*). Para riscos que estão dentro de limites planejados, *triggers* podem sinalizar o sucesso de uma atividade de resolução de riscos.

Prover notificação para disparos

Quando um *trigger* é disparado, a notificação é enviada para a pessoa apropriada através de canais de comunicação estabelecidos. Quatro tipos de *triggers* podem prover notificação de riscos inaceitáveis:

- Eventos periódicos: notificação para atividades programadas do projeto;
- Tempo decorrido: notificação para datas, baseada em um calendário;
- Variação relativa: notificação para valores fora de um intervalo de valores aceitáveis;
- Valor limite: notificação para valores que cruzaram um limite pré-determinado.

Reportar medidas e métricas de riscos

Uma medida é uma unidade padrão de medição para determinar dimensões, quantidades, ou capacidade. A exposição ao risco, é uma medida complexa que determina a magnitude do risco, a qual é calculada por multiplicar a probabilidade do risco e a conseqüência. Uma métrica usa medidas históricas para fornecer guias que podem ajudar a gerenciar no futuro, e é determinada pela composição de dados medidos ao longo do tempo.

3.5 Resolução do Risco

Uma vez estabelecido um bom conjunto de planos de gerenciamento de riscos, o processo de resolução de riscos consiste em implementar as técnicas de resolução de riscos descritas nos planos, tais como: implementar protótipos, simulações, etc (FAIRLEY, 1994).

O plano de contingência de riscos é invocado quando um indicador de risco quantitativo cruza um limite predeterminado. Caso as ações descritas no plano de contingência não tenham sido suficiente para gerenciar os riscos, o projeto entra em estado de crise (FAIRLEY, 1994).

O processo de resolução de risco é suficiente quando satisfaz os seguintes objetivos:

- Assinala responsabilidades e autoridade para os níveis mais baixos possíveis;
- Segue um plano de ação de riscos documentado;
- Reporta resultados de esforços de resolução de riscos;
- Toma ações corretivas quando necessário;
- Determina o custo-eficácia do gerenciamento de riscos;
- É preparado para se adaptar a circunstâncias que estão mudando;
- Melhora a comunicação na equipe;
- Controla os riscos de software sistematicamente.

As atividades do processo de resolução de riscos são as tarefas necessárias para executar o plano de ação de riscos para reduzir os riscos a níveis aceitáveis (HALL, 1998):

Responder a notificação de eventos de disparos

Triggers provêem notificação para a pessoa apropriada. Um indivíduo com autoridade deve responder ao evento disparado. Não significa que ele deva abandonar outro trabalho importante. *Triggers* devem prover um tempo razoável para resposta e não deve constituir uma situação de crise. Uma resposta apropriada inclui uma revisão da realidade atual e uma determinação atualizada do tempo para ação.

Uma pessoa responsável deve ser identificada se o plano de ação é designado para um grupo. Fatores como disponibilidade e nível de habilidade ajudam a determinar quem deve ser o responsável para executar o plano de ação. O responsável por identificar e analisar o risco não pode ser o responsável por resolvê-lo.

Executar o plano de ação de riscos

Executar um plano de ação documentado para resolver o risco. Se o plano não é documentado, é possível que existam problemas de comunicação. Um plano

documentado provê um guia de alto nível, permitindo entender o que é necessário fazer. O executor do plano é responsável por mapear os objetivos do plano para ações específicas que reduzirão a incerteza e aumentarão o controle.

Reportar progresso contra o plano

É necessário reportar os resultados de esforços na resolução de planos. Determinar e comunicar o progresso feito contra o plano. Reportar o status dos riscos em bases regulares para melhorar a comunicação dentro da equipe.

Simplesmente reportar o status dos riscos não é suficiente para gerenciar riscos. A equipe deve revisar o status dos riscos, medidas, e métricas regularmente para prover uma tomada de decisão ciente para os riscos.

Corrigir desvios do plano

Quando os resultados obtidos não são satisfatórios é necessário usar outra abordagem. Devem ser tomadas ações corretivas quando necessário. Um procedimento de ações corretivas tem os seguintes passos:

- Identificar o problema: encontre o problema no processo ou no produto.
- Avalie o problema: execute uma análise para entender e avaliar o problema documentado.
- Planeje ação: aprove um plano de ações para resolver o problema.
- Monitore o progresso: acompanhe o progresso até que o problema seja resolvido. Registre lições aprendidas para referências futuras.

4 SISTEMÁTICA PARA GERENCIAR RISCOS EM PROJETOS DE SOFTWARE

Este capítulo descreve a abordagem sistemática para a gerência de riscos em projetos de software (*Project Risk Management Approach* - PRiMA).

O objetivo da abordagem é permitir a elaboração de um processo específico para um dado projeto, visando minimizar a exposição do projeto aos riscos, identificados e mensurados de acordo com o contexto do projeto. A adaptação tem como base um metamodelo de processo (PRiMA-M) e um *framework* de processo (PRiMA-F)

PRiMA-M representa os conceitos usados na definição de processos de software, isto é, a partir dos conceitos descritos no metamodelo são instanciados elementos de processo (atividades, papéis, artefatos, ferramentas, etc.) que são usados na construção de modelos de processo. O metamodelo define a linguagem para expressar processos de software; descreve os conceitos e suas relações com a finalidade de construir e interpretar modelos de processo.

PRiMA-F organiza os elementos de processo, instanciados a partir de PRiMA-M, possíveis de serem executadas em processos de projetos de uma organização, incluindo atividades que visam implantar padrões de processo e organizacionais associados a riscos. Esses elementos são armazenados em uma base de conhecimento da organização, são descritos diagramas de atividades para descrever os aspectos dinâmicos do processo, e configurações de processo e guias de adaptação para facilitar a tarefa de instanciação de diferentes processos de software a partir de PRiMA-F.

Durante a execução do projeto, os riscos precisam ser acompanhados para identificar se estão dentro de limites aceitáveis ou se é necessário que seja tomada alguma ação para resolução de riscos. O monitoramento é realizado por meio de um plano de medição, elaborado usando o paradigma *Goal/Question/Metric* (GQM).

A Figura 4.1 mostra, usando um diagrama de atividades da UML (BOOCH, 2002), a seqüência de atividades proposta na abordagem sistemática para gerenciar riscos em projetos de software. A seguir uma breve descrição de cada atividade descrita na Figura 4.1.

A atividade *Identificar Riscos do Projeto* descreve como identificar os riscos que podem afetar o projeto. Para auxiliar o gerente de projeto nesta atividade, é elaborada uma lista contendo os riscos mais comuns em projetos de software a partir dos trabalhos dos autores Keil et al. (1998), Addison e Vallabh (2002), Boehm (1991) e os riscos sugeridos pelo modelo CMMI (CMMI, 2002). A atividade *Identificar Riscos do Projeto* é descrita na seção 4.1 desta tese.

Considerando que os riscos do projeto podem ser muitos, de várias naturezas e de impactos distintos, na atividade *Priorizar Riscos* é utilizada a técnica de quantificação da exposição ao risco para priorizar os riscos que têm maior impacto sobre o projeto e

alta ou média probabilidade de ocorrer. Segundo Boehm (1991), a quantificação da exposição ao risco é a técnica mais efetiva para a priorização dos riscos. A atividade *Priorizar Riscos* é descrita na seção 4.2 desta tese.

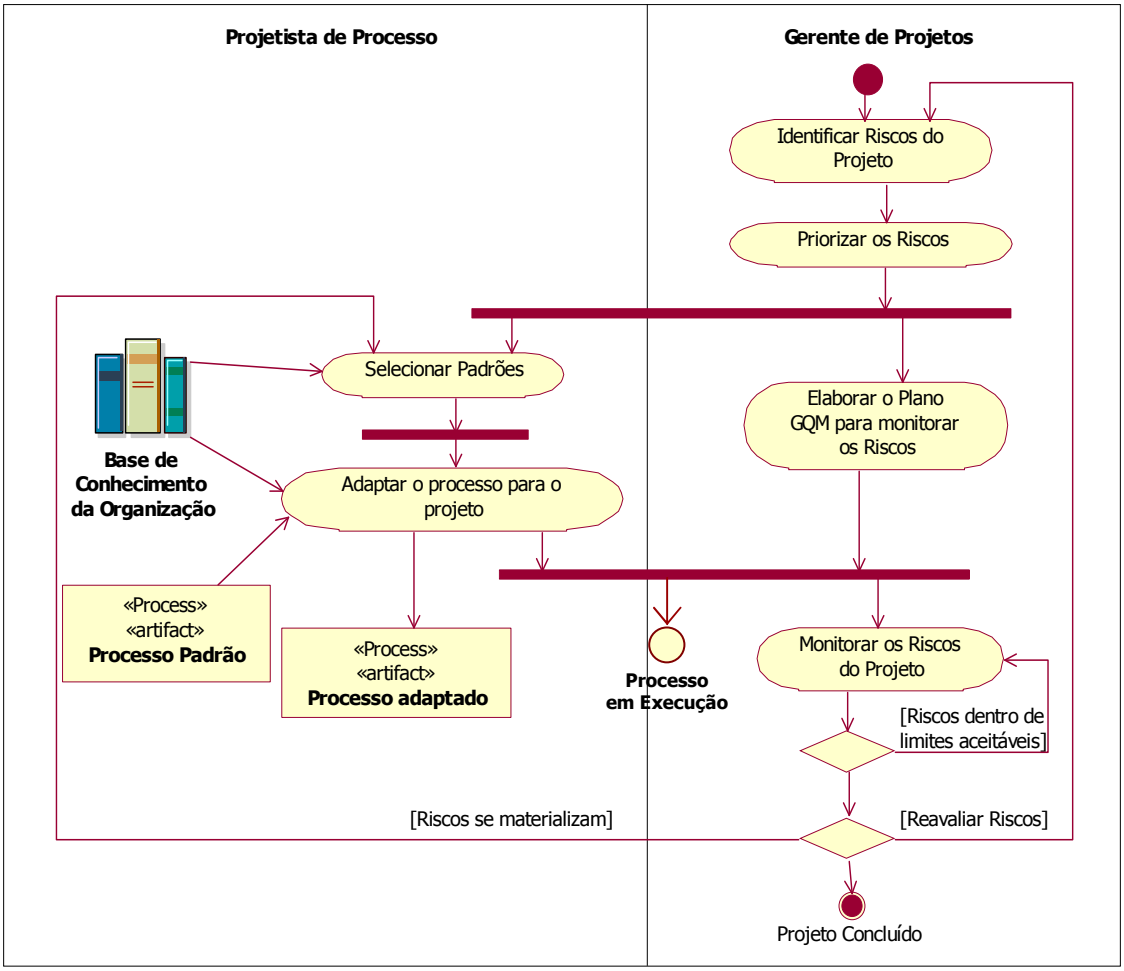


Figura 4.1: *Project Risk Management Approach* (PRiMA)

Após a priorização dos riscos é necessário definir ações preventivas para riscos de alto ou médio impacto. As ações preventivas são descritas como padrões organizacionais e de processo, facilitando dessa forma a identificação de qual padrão adotar na customização do processo, já que existem inúmeros padrões catalogados (CUNNINGHAM, 2004; OLDFIELD, 2002; COPLIEN, 2004; SCHWABER, 2001).

A atividade *Selecionar Padrões* define como recuperar da base de conhecimento, os padrões que visam prevenir os riscos priorizados para um dado projeto (HARTMANN, 2005). A seleção de padrões considera o contexto do projeto. O contexto é caracterizado pelos critérios: criticidade dos defeitos, número de pessoas envolvidas e habilidade da equipe de desenvolvimento. A atividade *Selecionar Padrões* é descrita na seção 4.3 desta tese.

A atividade *Adaptar o Processo para o Projeto* descreve como o processo padrão da organização pode ser adaptado para englobar os padrões selecionados para prevenir os riscos, gerando como resultado o processo definido para um projeto. O processo definido para um projeto é a definição operacional do processo de software usado no projeto. A adaptação ocorre com base em um *framework* de processo. O *framework* é um arcabouço de processo, a partir do qual podem ser instanciados diferentes processos

pela seleção de elementos de processo (previamente definidos pela organização). O *framework* é formado por uma base de conhecimento da organização, um conjunto de diagramas de atividades e guias de adaptação. A atividade *Adaptar o Processo para o Projeto* é descrita na seção 4.4 desta tese.

Durante o projeto, é necessário monitorar os riscos para garantir que estes estão sob controle, e tomar ações corretivas para os riscos que estejam fora de limites aceitáveis. Na atividade *Elaborar o Plano GQM para Monitorar os Riscos* é definido o Plano GQM, que descreve as metas a serem obtidas no monitoramento dos riscos. O plano GQM detalha as medidas que devem ser coletadas, para responder as questões associadas às metas do plano. A atividade *Elaborar o Plano GQM para Monitorar os Riscos* é descrita na seção 4.5 desta tese.

O monitoramento dos riscos é realizado com base nas métricas definidas no Plano GQM. O ideal é que as medidas sejam inseridas em uma base histórica, para serem usadas em estimativas futuras e no acompanhamento do projeto. As métricas auxiliam o gerente de projeto a *Monitorar os Riscos do Projeto*, dando visibilidade quanto ao progresso destes. Cabe ao gerente de projeto identificar o momento em que as ações corretivas devem ser tomadas porque a probabilidade de ocorrência do risco está aumentando. A probabilidade de o risco ocorrer é definida comparando as medições realizadas com limites pré-estabelecidos.

Durante o monitoramento dos riscos pode-se detectar que alguma medida ultrapassou limites pré-estabelecidos. Neste caso é necessário que sejam tomadas ações para resolver o risco, para isto o projetista deve executar novamente a atividade *Selecionar Padrões*, para selecionar outros padrões para resolução do risco.

Durante o projeto novos riscos podem ser identificados, periodicamente, o gerente de projeto deve executar a atividade *Identificar Riscos*, visando reavaliar os riscos do projeto de acordo com a situação atual. A atividade *Monitorar os Riscos do Projeto* é descrita na seção 4.6.

4.1 Identificar os Riscos do Projeto de software

Vários estudos foram realizados com o objetivo de identificar os principais riscos de projetos de software. Alguns desses estudos consideram apenas os riscos sobre os quais os gerentes de projeto têm controle, outros consideram todos os tipos de riscos.

Neste trabalho, os riscos de projeto citados pelos autores Keil et al. (1998), Boehm (1991), Addison e Vallabh (2002) e pelo modelo CMMI (CMMI, 2002) são comparados, com o objetivo de elaborar uma lista com os mais significativos riscos que ocorrem em projetos de software.

A lista de riscos elaborada pode ser utilizada pelo gerente de projeto (*checklist*) para auxiliá-lo a identificar os riscos que podem afetar o projeto de software. Novos riscos devem ser adicionados à lista, de acordo com a experiência da equipe e de análises *post-mortem* de projetos (FONTOURA, 2004).

Como resultado da execução da atividade *Identificar os riscos do projeto de software*, tem-se uma lista com os possíveis riscos de ocorrerem em determinado projeto de software.

Outras técnicas podem ser utilizadas em conjunto com a lista de risco, para identificação dos riscos do projeto, tais como: entrevistas, reuniões, revisões de artefatos, grupos de trabalho, etc. (HALL, 1998).

Na seção 4.1.1 é descrito o trabalho de Keil et al, na seção 4.1.2 é descrita a pesquisa realizada por Addison e Vallabh, na seção 4.1.3 são descritos os riscos propostos por Boehm, na seção 4.1.4 são descritos os riscos propostos pelo CMMI e na seção 4.1.5 é apresentada a comparação realizada.

4.1.1 Keil, Cule, Lyytinen e Schmidt

Keil et al. (1998) descrevem um estudo envolvendo a identificação dos principais fatores de riscos e a importância relativa de cada um destes. Com base nos dados coletados, é elaborado um *framework* para classificação dos riscos de projetos de software.

Os autores montaram painéis com gerentes de projetos de software experientes em Hong Kong, Finlândia e USA e solicitaram que eles identificassem os principais riscos e classificassem esses riscos por ordem de importância.

A pesquisa é baseada na experiência de um grupo de gerentes de projeto. A técnica Delphi, para consenso em grupos, foi utilizada para classificação dos riscos. Como resultado da pesquisa tem-se os riscos listados abaixo, por ordem decrescente de importância (KEIL, 1998).

1. Falta de compromisso da gerência sênior com o projeto
2. Falha em obter compromisso dos usuários
3. Não-entendimento dos requisitos
4. Falta de envolvimento adequado dos usuários
5. Falha na gerência de expectativas dos usuários
6. Mudança de escopo / objetivos
7. Falta de conhecimento / habilidade da equipe do projeto
8. Requisitos instáveis
9. Introdução de novas tecnologias
10. Equipe insuficiente / inadequada
11. Conflito entre os departamentos do usuário

Muitos painelistas consideraram como riscos mais importantes aqueles que eles não possuem controle. O nível de controle e a importância do risco possibilitaram a elaboração de um *framework* para mapear os diferentes tipos de riscos. A Figura 4.2 mostra o *framework* elaborado pelos autores da pesquisa.

Os riscos que se enquadram no quadrante 1 são: falta de compromisso da gerência sênior com o projeto, falha em obter compromisso dos usuários, não-entendimento dos requisitos e falta de envolvimento adequado dos usuários. Esses riscos requerem ações que criem e mantenham boas relações com clientes e promovam compromisso do cliente com o projeto.

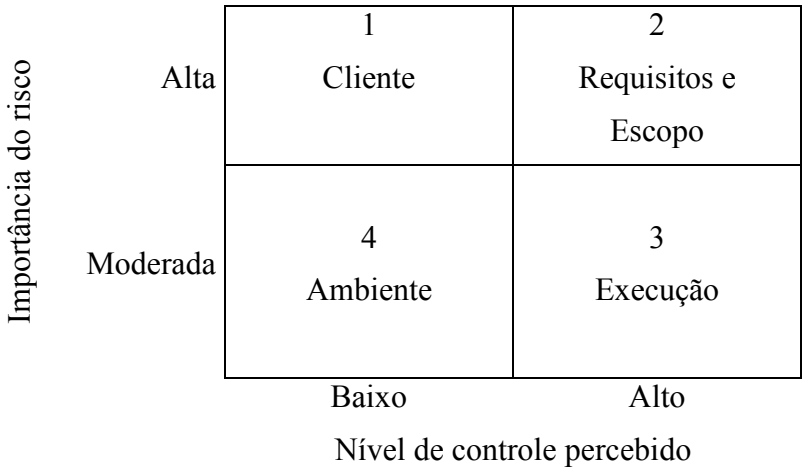


Figura 4.2: *Framework* para categorização de riscos (KEIL, 1998)

Os riscos que se enquadram no quadrante 2 são: não-entendimento dos requisitos e falha no gerenciamento de mudanças. As estratégias de mitigação de riscos devem enfatizar o gerenciamento de ambigüidade e as mudanças de requisitos.

Os riscos que se enquadram no quadrante 3 são: equipe insuficiente / inadequada, falta de uma metodologia de processo de desenvolvimento efetiva, estimativa pobre e definição imprópria de cargos e responsabilidades. As estratégias de mitigação de riscos enfatizam avaliações internas acopladas a revisões externas para manter o projeto sob controle. Pode-se utilizar um processo de desenvolvimento disciplinado e metodologias para dividir o projeto em partes gerenciáveis, definindo claramente cargos e responsabilidades, e desenvolvendo planos de contingência para tratar com diminuição da equipe e novas tecnologias.

Os riscos que se enquadram no quadrante 4 são: mudança de escopo / objetivos, conflitos entre os departamentos do usuário. São considerados riscos de ambiente, os riscos relacionados ao ambiente externo (desastres naturais, mudanças no ambiente competitivo,...). Planos de contingência, incluindo conceitos e táticas associadas com planejamento de desastres, é a estratégia mais sensível para tratar com esses riscos.

4.1.2 Addison e Vallabh

Addison e Vallabh (2002) descrevem um estudo empírico realizado com gerentes de projetos de software sobre a percepção destes quanto aos riscos de projetos de software e os controles mais efetivos para reduzir os fatores de risco, ou minimizar os efeitos dos riscos.

Os objetivos do estudo são: identificar os riscos envolvidos em projetos de software, ordenar os riscos por importância e freqüência de ocorrência e identificar as atividades executadas por gerentes de projeto para controlar riscos.

Os autores utilizaram um questionário para coletar dados para a pesquisa. A pesquisa abrangeu 70 gerentes de projeto. A análise dos questionários foi conduzida usando uma combinação de técnicas de regressão e de ordenação.

Como resultado da pesquisa tem-se os riscos listados abaixo, ordenados por importância (ADDISON, 2002).

1. Escopo/objetivos não claros ou não entendidos

2. Cronograma e orçamentos não realistas
3. Falta de compromisso da gerência sênior com o projeto
4. Falha em obter o envolvimento do usuário
5. Conhecimento/habilidades da equipe não adequadas
6. Falta de uma metodologia de gerência de projeto efetiva
7. Não entendimento dos requisitos
8. *Gold Plating* (adicionar capacidades por conta da equipe)
9. Mudanças contínuas nos requisitos
10. Desenvolvimento errado das funções de software
11. Subcontratação (falha na gerência de tarefas de subcontratadas)
12. Avaliação incorreta do uso e desempenho de recursos
13. Introdução de novas tecnologias
14. Falha na gerência de expectativas dos usuários finais.

Os controles de projetos de software listados abaixo são considerados importantes para reduzir os riscos identificados. Os controles estão listados por freqüência de uso (do mais usado para o menos usado) (ADDISON, 2002).

1. Assinalar responsabilidades aos membros da equipe
2. Desenvolver e ater-se ao plano de projeto de software
3. Envolver o gerenciamento durante todo o ciclo de vida do projeto
4. Dividir o projeto em porções controláveis
5. Estabilizar requisitos e especificações o mais cedo possível
6. Envolver os usuários durante todo o ciclo de vida do projeto
7. Avaliar impacto de custo e de cronograma de cada mudança
8. Revisar o progresso por datas e definir os objetivos para a próxima fase.
9. Educar os usuários no impacto das mudanças durante o projeto
10. Assegurar que existe um comitê gerencial
11. Combinar avaliações internas com revisões externas
12. Incluir uma avaliação de riscos periódica e formal.
13. Evitar incorporar muitas mudanças simultaneamente em projetos de software
14. Desenvolver planos de contingência para tratar com os problemas da equipe

4.1.3 Dez Principais Riscos de Boehm

Boehm (1991) cita os 10 principais riscos de software, com base em uma pesquisa com gerentes de projeto experientes. Os dez principais riscos, segundo Boehm, são:

1. Falta de pessoal
2. Orçamento e cronograma não realistas

3. Desenvolvimento de funções erradas
4. Desenvolvimento errado da interface do usuário
5. Acréscimo de funcionalidades idealizadas pela equipe sem o aval do cliente (*gold plating*)
6. Mudança contínua nos requisitos
7. Déficit em componentes fornecidos externamente
8. Déficit em tarefas executadas externamente
9. Falta de desempenho em tempo real
10. Distorcer as capacidades da ciência da computação

4.1.4 Riscos Típicos Segundo o CMMI

O modelo *Capability Maturity Model Integration* (CMMI) cita alguns riscos externos e internos típicos em projetos de software, que são (CMMI, 2002):

1. Instabilidade nos requisitos;
2. Estimativas incorretas;
3. Projeto inviável;
4. Tecnologia não disponível;
5. Estimativas de cronograma e alocação não-realistas;
6. Equipe e habilidades inadequadas;
7. Estimativas de custos e fundos incorretas;
8. Capacidade do subcontratado inadequada ou incerta;
9. Capacidade do vendedor inadequada ou incerta

O risco “Capacidade do vendedor inadequada ou incerta” não será considerado na comparação, pois não se aplica ao processo de desenvolvimento.

4.1.5 Comparação entre Riscos

A Tabela 4.1 mostra a relação de riscos citados por estes autores, agrupados por similaridade. Na coluna 1 são listados todos os riscos. Para as colunas 2, 3, 4 e 5, utilizou-se a seguinte convenção: a letra ‘S’ na coluna correspondente ao nome do autor indica que o mesmo cita o risco em seu trabalho e a letra ‘N’ indica que o autor não cita o risco no seu trabalho.

A lista de riscos, exibida na Tabela 4.1, pode ser utilizada pelo gerente de projeto como uma *checklist* para auxiliá-lo a identificar os riscos que podem afetar o projeto de software. Para realizar a identificação dos riscos, os membros da equipe devem ser entrevistados e sessões de grupo devem ser realizadas com a equipe e com outras pessoas envolvidas com o projeto (ADDISON, 2002). O processo de identificação dos riscos não deve ficar limitado a esta lista de riscos, pode-se, por exemplo, utilizar os principais problemas que ocorreram em projetos passados da organização, como sugerido por DeMarco e Lister (2003). Segundo esses autores: “Os problemas de ontem

são os riscos de amanhã" (DEMARCO, 2003, p. 61). Portanto, uma forma de popular a lista de riscos do projeto é através da utilização dos resultados de análises *postmortem* de projetos.

Tabela 4.1: Comparação entre os principais riscos de software

Riscos	**Keil**	**Boehm**	**Addison**	**CMMI**
Falta de compromisso da gerência sênior com o projeto	S	N	S	N
Falha em obter compromisso dos usuários e falta de envolvimento adequado dos usuários	S	N	S	N
Não-entendimento dos requisitos, falha na gerência de expectativas dos usuários e instabilidade dos requisitos	S	S	S	S
Escopo/objetivos não-claros e diferentes expectativas sobre o sistema entre distintos usuários.	S	N	S	N
Falta de conhecimento/habilidade requerida do pessoal do projeto e equipe insuficiente ou inadequada	S	S	S	S
Introdução de novas tecnologias	S	N	S	S
Cronograma e orçamento não-realistas	N	S	S	S
Falta de uma metodologia de gerência de projeto efetiva	N	N	S	N
Acréscimo de funcionalidades idealizadas pela equipe sem o aval do cliente (*gold plating*)	N	S	S	N
Desenvolvimento errado das funções (implementação não atende à especificação) ou interface	N	S	S	N
Subcontratação (gerência de tarefas ou componentes desenvolvidos externamente)	N	S	S	S
Uso de recursos e desempenho do sistema inadequados	N	S	S	N
Projeto (desenho) inviável	N	N	N	S

Revisões *postmortem* são importantes para a melhoria do processo, mas as companhias raramente as executam. Como resultado, as organizações tendem a repetir os mesmos erros de projetos passados (GLASS, 2001).

Os riscos listados na Tabela 4.1 podem ser classificados utilizando um critério adaptado do *framework* de classificação de riscos, proposto por Keil et al. (1998). Segundo essa classificação, os riscos podem ser:

Riscos de cliente: são classificados como risco de clientes, os que se originam no grau de envolvimento dos clientes com o sistema que está sendo desenvolvido.

Riscos de Requisitos: a definição dos requisitos de software é um dos pontos mais importantes do processo de desenvolvimento, pois se o software entregue ao cliente não atende aos requisitos definidos, não satisfazerá as necessidades do cliente. Os riscos de requisitos descrevem preocupações quanto ao entendimento das necessidades do cliente, tais como: definição inadequada de escopo, definições ambíguas ou inexistentes dos requisitos funcionais e não funcionais, entre outras.

Riscos de Planejamento: são classificados como riscos de planejamento, os riscos que monitoram o planejamento inicial do projeto de software, tais como: recursos

disponíveis, qualificação da equipe, estimativas de cronograma e custos, apoio da gerência sênior ao projeto, etc.

Riscos de Execução: os riscos de execução descrevem situações que podem ocorrer afetando a qualidade do software desenvolvido, incluindo: desempenho inadequado do sistema, desenvolvimento errado das funções de software, erros, etc.

Os riscos de gerência de projetos foram separados em riscos de planejamento e execução, para permitir separar os riscos que devem ser mitigados no início do projeto (planejamento) e os demais riscos de gerência (execução). A Tabela 4.2 mostra os riscos agrupados por tipo de risco.

Tabela 4.2: Riscos classificados por tipo de risco

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

4.2 Priorizar os Riscos

Considerando que, os riscos identificados para o projeto podem ser muitos, na atividade *Priorizar Riscos* são descritas técnicas para priorizar os riscos que tem maior impacto sobre o projeto e alta ou média probabilidade de ocorrer (FONTOURA, 2004).

A priorização de riscos pode ser realizada por meio das seguintes técnicas: Quantificação da Exposição ao Risco (HALL, 1998), Análise Causal (FENTON, 2001), Análise de Redes (MCCABE, 2001), Modelos de Custos (FAIRLEY, 1994), Simulação Monte Carlo (SCHUYLER, 2001), entre outras.

Neste trabalho, é utilizada a técnica de Quantificação da Exposição ao Risco (ER), descrita na seção 3.2.1 deste trabalho, para priorizar os riscos do projeto. Segundo Boehm (BOEHM, 1991), essa é a técnica mais efetiva para priorizar e quantificar os riscos do projeto.

Após a quantificação de todos os riscos identificados na atividade *Identificar Riscos do Projeto*, tem-se uma lista de riscos ordenada por exposição ao risco. O gerente de projeto define se todos os riscos identificados serão tratados, ou somente os riscos com ER maior que determinado valor. O custo para enfocar todos os riscos pode ser muito alto, e riscos com pouca probabilidade de ocorrer ou baixo impacto, talvez não justifiquem os custos para tratá-los (HALL, 1998).

4.3 Selecionar Padrões

Na proposta inicial da sistemática para gerenciar riscos, a adaptação do processo é realizada por meio da inserção de ações preventivas para cada risco priorizado no processo de software. Foram sugeridas ações preventivas para mitigar cada risco descrito na Tabela 4.1. As ações sugeridas resultam em mudanças no processo de modo a torná-lo mais ágil ou planejado.

As ações foram obtidas a partir de modelos como o CMM (PAULK, 1993); processos como RUP (RATIONAL, 2001), XP (BECK, 2004; BECK, 2003), Scrum (SCHWABER, 2001), PMBok (PMI, 2000); ou por autores como Pressman (PRESSMAN, 2002), Boehm (BOEHM, 2002), Sommerville (SOMMERVILLE, 2003), entre outros. As ações sugeridas foram organizadas em tabelas identificando para cada risco, quais as ações que podem ser executadas em cada etapa do desenvolvimento onde o risco de origina como forma de evitá-lo. Essa proposta deu origem ao artigo "Usando GQM para Gerenciar Riscos em Projetos de Software", publicado no Simpósio Brasileiro de Engenharia de Software 2004 (FONTOURA, 2004).

As ações preventivas podem ser vistas como padrões organizacionais e de processo, facilitando dessa forma a identificação de qual padrão adotar na customização do processo, já que existem inúmeros catálogos de padrões (CUNNINGHAM, 2004; OLDFIELD, 2002; COPLIEN, 2004; SCHWABER, 2001).

Padrões são maneiras de descrever melhores práticas, bons projetos, e capturar experiências de uma forma que seja possível para outros reusarem essa experiência (HILLSIDE, 2003). Padrões de processo e organizacionais capturam práticas bem-sucedidas em gerenciamento de software, e podem ser utilizados na elaboração de processos de desenvolvimento da organização, ou na melhoria de processos existentes.

Padrões são descritos utilizando-se uma descrição textual, em um formato semelhante ao de James Coplien (1996), que emprega os seguintes campos: *Nome, Classificação, Problema, Contexto, Forças, Solução, Relações, Justificativa* e *Fonte.*

A atividade *Selecionar Padrões* define como recuperar da base de conhecimento da organização, os padrões que visam prevenir os riscos identificados para um dado projeto (HARTMANN, 2005). O mecanismo de seleção de padrões foi proposto por Júlio Hartmann, como dissertação de mestrado, no mesmo projeto que este trabalho está inserido – o Projeto Amadeus do Prof. Dr. Roberto Tom Price.

Os padrões são associados aos riscos por meio de regras. Cada regra associa um ou mais riscos a um ou mais padrões que se propõem a resolver, prevenir ou minimizar os riscos. As regras são associadas a contextos de projeto, onde funcionam melhor (HARTMANN, 2005). Regras possuem um fator de acurácia, o qual mede a eficácia dos padrões associados no controle do risco. Um fator 0 (zero) significa que o padrão não ajuda em nada no controle do risco, um fator 10 (dez) significa que o padrão pode ser aplicado para eliminar os riscos associados de forma efetiva.

O contexto de projeto é estabelecido usando uma abordagem similar a Cockburn (2001) e Boehm e Turner (2003), por analisar três critérios (HARTMANN, 2005):

- Criticidade dos defeitos: a possibilidade da perda associada à ocorrência de um defeito (risco). O intervalo de valores permitidos varia desde perda de conforto (1) a perda de muitas vidas humanas (5). De acordo com Cockburn (2000),

quando mais crítico um sistema, mais densa deve ser sua metodologia. Artefatos intermediários devem ser elaborados e cuidadosamente revisados e verificados;

- Número de pessoas envolvidas: Cockburn (2001) considera que quanto mais pessoas envolvidas em um projeto, mais documentos intermediários devem ser gerados para coordenar o trabalho;
- Nível de habilidade da equipe de desenvolvimento: Boehm e Turner (2003) estenderam a classificação de pessoas proposta por Cockburn (2000), e a utilizam como um importante fator para balancear entre metodologias ágeis e planejadas. A classificação varia de: pessoas que são incapazes ou não desejam colaborar (nível -1) a pessoas que podem revisar um método, e adaptá-lo a uma nova situação (nível 3). O nível de habilidade da equipe é calculado de acordo com o percentual informado para cada classificação de habilidade.

A associação de padrões a riscos foi realizada com base na literatura, nas metodologias publicada, na experiência dos autores e nos catálogos de padrões existentes (COPLIEN, 1995; CUNNINGHAM, 2004; SCHWABER, 2001; PAULK, 1993; YATCO, 1999). Alguns padrões foram elaborados pelos autores a partir de práticas descritas na literatura, como os padrões: *SeniorManagementReview*, *DocumentedConfigurationManagementPlan*, *DocumentSoftwareEstimates*, entre outros, propostos pelo CMM como práticas-chave, mas não documentados como padrões.

A Figura 4.3 mostra exemplos de regras de associação do risco *Misunderstanding the Requirements* aos padrões *BuildPrototype*, *RequirementsAreValidated*, *ScenariosDefineProblem* e *PlanningGame*. No *framework* PRiMA-F são sugeridas algumas regras de associação de riscos a padrões, cabe a organização adequá-las a sua realidade e criar novas regras de acordo com as suas necessidades.

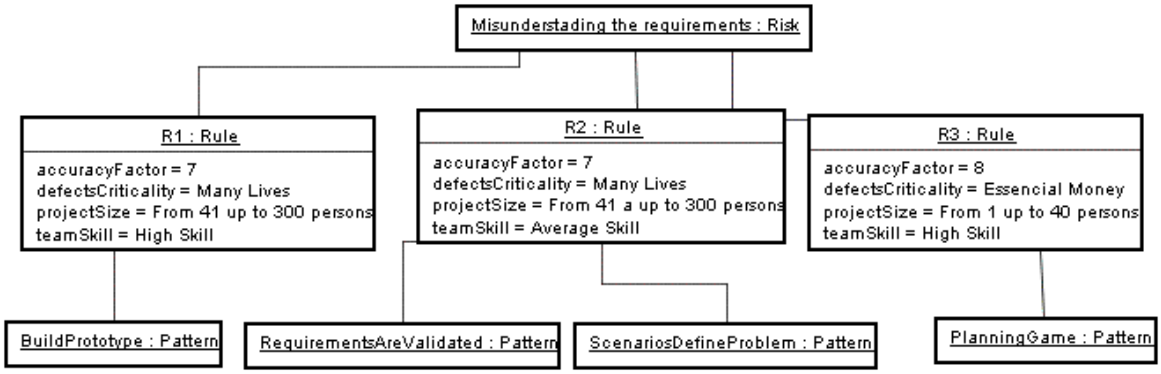


Figura 4.3: Regras de associação entre riscos e padrões

A Tabela 4.3 mostra uma relação de alguns padrões, sugeridos em PRiMA-F, para minimizar os riscos descritos na Tabela 4.1, para uso em contexto de projetos tradicionais e ágeis. O contexto do projeto é utilizado para balancear entre métodos ágeis e planejados. Na Figura 4.3, a regra R3 descreve um contexto de uma equipe pequena com alta habilidade e de baixa criticidade, sendo sugerido o padrão *PlanningGame*, já no caso nas regras R1 e R2, o contexto descrito é de equipes maiores de média e alta habilidade, com perda de vidas, os padrões sugeridos são provenientes de processos planejados.

Como resultado da execução da atividade *Selecionar Padrões* tem-se uma lista de padrões organizacionais e de processo sugeridos para serem usados como ações para

prevenir ou minimizar os riscos identificados para o projeto. O projetista de processo escolhe alguns padrões sugeridos da lista e adiciona-os a linguagem de padrões. A linguagem de padrões do projeto é utilizada na adaptação do processo da organização para um projeto específico.

Tabela 4.3: Padrões sugeridos para prevenir riscos

Riscos	**Padrões**	
	Métodos Tradicionais	**Métodos Ágeis**
Falta de compromisso da gerência sênior com o projeto	SeniorManagementReview (PAULK, 1993a)	IterationPlanning (CUNNINGHAM, 2004) SprintPlanningMeeting (SCHWABER, 2001)
Falha em obter compromisso dos usuários e falta de envolvimento adequado dos usuários	EngageCustomer (COPLIEN, 2004) WorkshopOnCustomerInvolvement (COPLIEN, 2004)	OnSiteCustomer (CUNNINGHAM, 2004) SprintPlanningMeeting (SCHWABER, 2001) PlanningGame (CUNNINGHAM, 2004)
Não-entendimento dos requisitos, falha na gerência de expectativas dos usuários e requisitos instáveis	EarlyAndRegularDeliveryRUP (COPLIEN, 2004) ImpliedRequirements (COPLIEN, 2004) BuildPrototype (COPLIEN, 2004) ScenariosDefineProblem (COPLIEN, 2004) DocumentedConfigurationManagementPlan (PAULK, 1993a)	EarlyAndRegularDeliveryXP (COPLIEN, 2004) PlanningGame (CUNNINGHAM, 2004) OnSiteCustomer(CUNNINGHAM, 2004) SimpleDesign (CUNNINGHAM, 2004) ConstantRefactoring (CUNNINGHAM, 2004)
Escopo/objetivos não-claros e conflito entre os departamentos do usuário	EarlyAndRegularDeliveryRUP (COPLIEN, 2004) JointApplicationDesign (YATCO, 1999)	EarlyAndRegularDeliveryXP (CUNNINGHAM, 2004) ConstantRefactoring (CUNNINGHAM, 2004) ModelTheDomain (OLDFIELD, 2002)
Falta de conhecimento/ habilidade requerida do pessoal do projeto e equipe insuficiente ou inadequada	GenericsAndSpecifics (COPLIEN, 2004) DayCare (COPLIEN, 2004) ApprenticeShip (COPLIEN, 2004)	DevelopingInPairs (CUNNINGHAM, 2004)
Introdução de novas tecnologias	SkunkWorks (COPLIEN, 2004) ArchitectureTeam (COPLIEN, 2004) EarlyAndRegularDeliveryRUP (COPLIEN, 2004) BuildPrototype (COPLIEN, 2004)	SimpleDesign (CUNNINGHAM, 2004) ConstantRefactoring (CUNNINGHAM, 2004) SpikeSolutions (CUNNINGHAM, 2004) EarlyAndRegularDeliveryXP (CUNNINGHAM, 2004)

Cronograma e orçamento não-realistas	SizeTheSchedule (COPLIEN, 2004) WorkQueue (COPLIEN, 2004) DocumentSoftwareEstimates (PAULK, 1993a)	PlanningGame (CUNNINGHAM, 2004) IterationPlanning (CUNNINGHAM, 2004)
Falta de uma metodologia de projeto efetiva	SoftwareLifecycleIsDefined (PAULK, 1993a) ProjectProcessIsDefined (PAULK, 1993a)	SoftwareLifecycleIsDefined (PAULK, 1993a) DefinedProcess (CUNNINGHAM, 2004)
Gold Plating Acréscimo de funcionalidades idealizadas pela equipe sem o aval do cliente.	EarlyAndRegularDeliveryRUP (COPLIEN, 2004) PeerReviews (PAULK, 1993a)	EarlyAndRegularDeliveryXP (CUNNINGHAM, 2004) SimpleDesign (CUNNINGHAM, 2004) DevelopingInPairs (CUNNINGHAM, 2004)
Desenvolvimento errado das funções ou interface	EarlyAndRegularDeliveryRUP (COPLIEN, 2004) AcceptanceTests (CUNNINGHAM, 2004) BuildPrototype (COPLIEN, 2004) ScenariosDefineProblem (COPLIEN, 2004) PeerReviews (PAULK, 1993a)	EarlyAndRegularDeliveryXP (CUNNINGHAM, 2004) DevelopingInPairs (CUNNINGHAM, 2004) AcceptanceTests (CUNNINGHAM, 2004) PlanningGame (CUNNINGHAM, 2004)
Subcontratação (tarefas ou componentes desenvolvidos externamente)	DocumentedSubcontract-ManagementPlan (PAULK, 1993a) SubcontractManager (PAULK, 1993a)	DocumentedSubcontract-ManagementPlan (PAULK, 1993a) SubcontractManager (PAULK, 1993a)
Uso de recursos e desempenho do sistema (ou desempenho em tempo real)	ArquitectureTeams (COPLIEN, 2004) GroupValidation (COPLIEN, 2004) PeerReviews (PAULK, 1993a)	ConstantRefactoring (CUNNINGHAM, 2004) SimpleDesign (CUNNINGHAM, 2004) DevelopingInPairs (CUNNINGHAM, 2004) SpikeSolutions (CUNNINGHAM, 2004)
Projeto (desenho) inviável	ArchitectureTeam (COPLIEN, 2004) PeerReviews (PAULK, 1993a)	SystemMetaphor(CUNNINGHAM, 2004) SimpleDesign (CUNNINGHAM, 2004) ConstantRefactoring (CUNNINGHAM, 2004) SpikeSolutions (CUNNINGHAM, 2004)

4.4 Adaptar o Processo para o Projeto

Existem muitos componentes envolvidos nos projetos de desenvolvimento de software, tais como: ferramentas, linguagens de programação, pessoas, processos, objetivos de qualidade, tamanho, entre outros. Quando uma organização adota um

processo de software, busca um que possa ser adaptado as suas próprias necessidades (HENDERSON-SELLERS, 2001).

O SEI, por meio do SW-CMM (PAULK, 1993) propõe que seja definido um processo de software padrão da organização (PSPO), com os elementos de processo fundamentais que são esperados em todos os projetos, e esse processo seja adaptado para cada projeto, de acordo com as características e restrições deste, dando origem ao processo de software para um projeto.

SW-CMM nível 3 preconiza algumas áreas-chave (*Key Process Areas* - KPAs) relacionadas à definição e gerência de processos de software, são estas: *Definição dos Processos da Organização*, *Foco nos Processos da Organização* e *Gerenciamento Integrado de Software*, descritas na seção 2.3.1 deste trabalho. Essas KPAs descrevem práticas relacionadas à definição e à melhoria dos processos de software da organização e à adaptação de processos de software da organização para enfocar as características específicas de projetos de software (PAULK, 1993).

Na literatura são descritas duas formas principais para elaboração de processos, descritas a seguir.

Um processo pode ser construído de forma que se inicie com uma instância de um *framework* pré-configurado, talvez vendido como um produto por um vendedor que tenha feito a seleção de itens com os quais o *framework* é populado (HENDERSON-SELLERS, 2001).

O processo já é pré-configurado no sentido de ter um ciclo de vida e modelo de arquitetura já definidos. Então, a customização deste processo é essencialmente uma subtração, pela qual, elementos do processo são removidos ou modificados. Isto é chamado, normalmente, de adaptação (*tailoring*). Modificações são restritas, e é mais difícil migrar ou alcançar compatibilidade com *releases* futuros do processo. Os custos de adaptação do processo são baixos, uma vez que não existe o custo de construção do processo (HENDERSON-SELLERS, 2000).

Outra opção é partir de um metamodelo, do qual vários processos podem ser construídos - por instanciação e montagem, isto é, são criadas instâncias de elementos do processo (metaclasses) e adicionadas regras para seqüenciamento entre essas instâncias para formar o processo. O custo de criação do processo a partir do metamodelo é alto, pois todos os elementos precisam ser instanciados e o processo elaborado, mas tem-se um ganho em flexibilidade (HENDERSON-SELLERS, 2000).

Como exemplo de um processo pré-configurado pode-se citar o *Rational Unified Process* (RUP). No RUP, algumas decisões arquiteturais já foram feitas, tais como: o processo é dividido em fases, seis disciplinas de processo através dessas fases e três disciplinas de suporte, junto com uma abordagem elaboracional e dirigida a casos de uso. As possibilidades de customização no RUP incluem: excluir ou modificar atividades ou passos de atividades, modificar modelos e adicionar novos pontos de verificação em atividades de revisão.

Como exemplo de um metamodelo pode-se citar o *Software Process Engineering Metamodel Specification* (SPEM) (OMG, 2002). SPEM descreve um processo de desenvolvimento de software e seus componentes. Ele utiliza uma abordagem orientada a objetos para modelar uma família de processo e usa a UML como notação. SPEM foi elaborado pelo Object Management Group (OMG) (OMG, 2002). *Rational Unified Process* (RATIONAL, 2003), IBM Global Services Method (NICHOLS, 2005), DMR

Macroscope (OMG, 2002), OPEN (HENDERSON-SELLERS, 2000), entre outros, são compatíveis com SPEM. A Figura 4.4 mostra a arquitetura de 4 camadas de modelagem definida pela OMG (OMG, 2002).

MetaObject Facility (MOF), nível M3, é uma tecnologia adotada pela OMG para definir metadados e representá-los como objetos CORBA. O metamodelo MOF define a sintaxe abstrata dos metadados, na representação de um modelo MOF. MOF é uma linguagem para especificar metamodelos.

No nível M2, chamado de metamodelo de processo, é definido a linguagem para especificar modelos. SPEM foca na definição do metamodelo, e serve como um modelo para o nível M1.

Um processo em execução em um projeto está no nível M0. A definição deste processo está no nível M1. Tanto um processo genérico, como o RUP; como uma adaptação do RUP para determinado projeto, estão no nível M1.

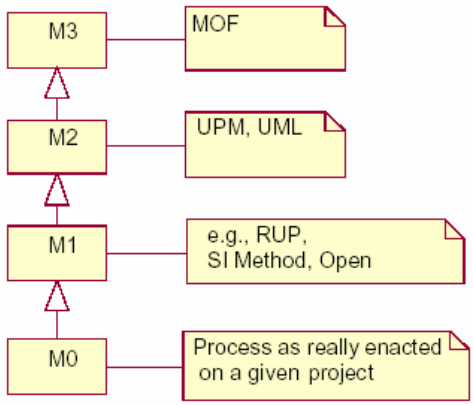


Figura 4.4: Níveis de modelagem (OMG, 2002)

Nesta tese optou-se por elaborar um metamodelo para representar os elementos de processo necessários a definição do processo padrão da organização, do processo definido para o projeto e para representar padrões, provendo flexibilidade e um *framework* para organização dos elementos de processo instanciados a partir do metamodelo, facilitando a construção de processos.

O metamodelo PRiMA-M visa descrever os conceitos que serão usados na modelagem dos processos, e foi inicialmente criado pela integração dos metamodelos do RUP e do XP. Esse metamodelo está no nível M2.

O *framework* PRiMA-F visa facilitar a atividade de instanciação dos processos, e serve para organizar os elementos de processo utilizados em determinada organização, definindo a seqüência de execução das atividades, guias de adaptação e configurações de processo típicas. Novos elementos instanciados a partir do metamodelo devem ser incorporados no *framework* de processo. O *framework* de processo está no nível M1, assim como o processo padrão da organização e o processo definido para um projeto está no nível M0.

Essa seção está organizada da seguinte forma: na subseção 4.4.1 é proposto o metamodelo PRiMA-M e na subseção 4.4.2 é detalhado o *framework* PRiMA-F.

4.4.1 Elaboração do Metamodelo PRiMA-M

PRiMA-M foi elaborado a partir da integração dos metamodelos do Processo Unificado da Rational (RUP) e do *Extreme Programming* (XP). RUP (RATIONAL, 2001) foi selecionado para representar métodos planejados por ser um dos mais utilizados processos de software e ser bastante completo (MANZONI, 2003). O XP (BECK, 1999) foi escolhido por ser o método ágil mais conhecido e mais utilizado atualmente (CHARETTE, 2002).

O metamodelo do RUP (RATIONAL, 2002) é descrito na seção 2.1.1. O metamodelo do XP não foi encontrado na literatura, então foi realizada uma engenharia reversa de processos XP para obter o metamodelo. Esse metamodelo está descrito na subseção 2.2.1.

Os metamodelos são integrados por meio das regras de integração de esquemas conceituais, propostas por Batini (1992), descritas a seguir.

4.4.1.1 Integração de Esquemas Conceituais

Integração de visão é o processo de juntar vários esquemas conceituais em um esquema conceitual global que represente todos os requisitos da aplicação (BATINI, 1992). Visando elaborar um metamodelo capaz de representar elementos de processos necessários à modelagem de métodos ágeis (XP) e de métodos planejados (RUP), os metamodelos do RUP e XP são integrados.

O principal objetivo da visão de integração é encontrar todas as partes dos esquemas conceituais globais que se referem à mesma porção da realidade e unificar sua representação (BATINI, 1992). A principal dificuldade da integração de modelos é descobrir as diferenças entre os esquemas que serão unidos, essas diferenças podem ser:

a) Diferentes perspectivas: no processo de projeto, os projetistas modelam os mesmos objetos de seus pontos de vista. Conceitos podem ser vistos de diferentes níveis de abstração, ou representados usando diferentes propriedades.

b) Equivalência entre construtores do modelo: os modelos conceituais têm uma rica variedade de estruturas de representação, então, eles permitem representações equivalentes da mesma realidade.

c) Especificações de projeto incompatíveis: erros durante a visão de projeto com relação a nomes, estruturas e restrições de integridade podem produzir modelos de entrada com erros para integração. Durante a integração esses erros devem ser detectados e corrigidos.

Antes de unir os esquemas é necessário analisar e resolver conflitos entre os modelos a serem unidos. Duas tarefas devem ser distinguidas: (1) análise de conflitos de nomes, na qual nomes de conceitos nos esquemas são comparados e unificados; (2) análise de conflitos estruturais, na qual representações de conceitos nos esquemas são comparados e unificados.

Análise de Conflitos de Nomes

Existem duas fontes de conflitos de nomes: sinônimos e homônimos. Sinônimos ocorrem quando os mesmos objetos do domínio da aplicação são representados por nomes diferentes nos dois esquemas; homônimos ocorrem quando objetos diferentes são representados com o mesmo nome nos dois esquemas.

Como resultado da detecção de conceitos com erro ou similares, várias modificações podem ser executadas nos esquemas (modificações de cenários). A modificação de cenário para conflitos de nome envolve renomear os conceitos, também pode envolver adicionar alguma propriedade interesquema.

Considerando os metamodelos do RUP e XP existe conflito de nome quanto ao *XP Role* e *Worker*, pois ambos representam os papéis executados por trabalhadores do processo. Para solucionar esse problema a classe <*XP Role*> e a classe <*Worker*> foram renomeadas para *Worker*.

Análise de Conflitos Arquiteturais

Após a análise de conflitos de nomes e resolução desses conflitos, assume-se que dois conceitos com o mesmo nome representam o mesmo conceito do mundo real. Na análise de conflitos estruturais, conceitos com o mesmo nome nos esquemas de entrada são comparados para ver se eles podem ser unidos. Neste caso existem as seguintes categorias:

1) Conceitos idênticos: tem exatamente a mesma representação e conceitos vizinhos;
2) Conceitos compatíveis: têm estrutura de representação diferente ou propriedades vizinhas que não são contraditórias, por exemplo, o uso de uma entidade e um atributo para representar o mesmo conceito. Conflitos de compatibilidade são facilmente resolvidos por mudar uma das duas representações de conceitos.
3) Conceitos incompatíveis: têm conceitos contraditórios. Fontes de incompatibilidade devem ser eliminadas antes de juntar os esquemas. Algumas incompatibilidades possíveis:
    a) Diferentes cardinalidades para o mesmo atributo ou entidade;
    b) Diferentes identificadores: um identificador em um esquema não é um identificador em outro;
    c) Relacionamentos de subconjuntos reversos: Entidade A é um subconjunto da entidade B em um esquema e a entidade B é um subconjunto da entidade A em outro.

Soluções possíveis para incompatibilidade incluem: selecionar uma das representações ou construir uma representação comum tal que todas as restrições dos dois esquemas sejam suportadas no esquema integrado.

Após a análise de cada uma dessas regras, os metamodelos do XP e do RUP foram integrados, dando origem ao metamodelo PRiMA-M mostrado na Figura 4.5.

Os conflitos de nomes resolvidos foram os seguintes:

- *XP Role* e *Worker* foram unificados para *Worker*.

Não foram detectados conflitos arquiteturais na integração das classes.

O metamodelo PRiMA-M representa um conjunto de conceitos que são usados na modelagem de processos de software. A partir do metamodelo são instanciados elementos de processo, que são usados na definição de processos planejados, ágeis ou híbridos; com a inserção de subprocessos específicos para padrões de tratamento de riscos, bem como customizações desses processos para uso em projetos específicos. As

instâncias de elementos de processo são armazenadas em uma base e combinadas para elaborar diferentes modelos de processos de software.

A utilização do metamodelo mostrou algumas deficiências, tais como:

- A definição dos elementos de processo é de responsabilidade do projetista de software, então o esforço inicial para descrever o processo de software da organização e os padrões, por meio de elementos de processo, é muito grande;
- Não garante que as atividades que compõem o processo definido do projeto sejam consistentes entre si, já que a definição do processo padrão e dos padrões depende do projetista;
- A seqüência de execução das atividades não é definida.

Para solucionar essas deficiências, foi elaborado um *framework* de processo e uma base de conhecimento da organização.

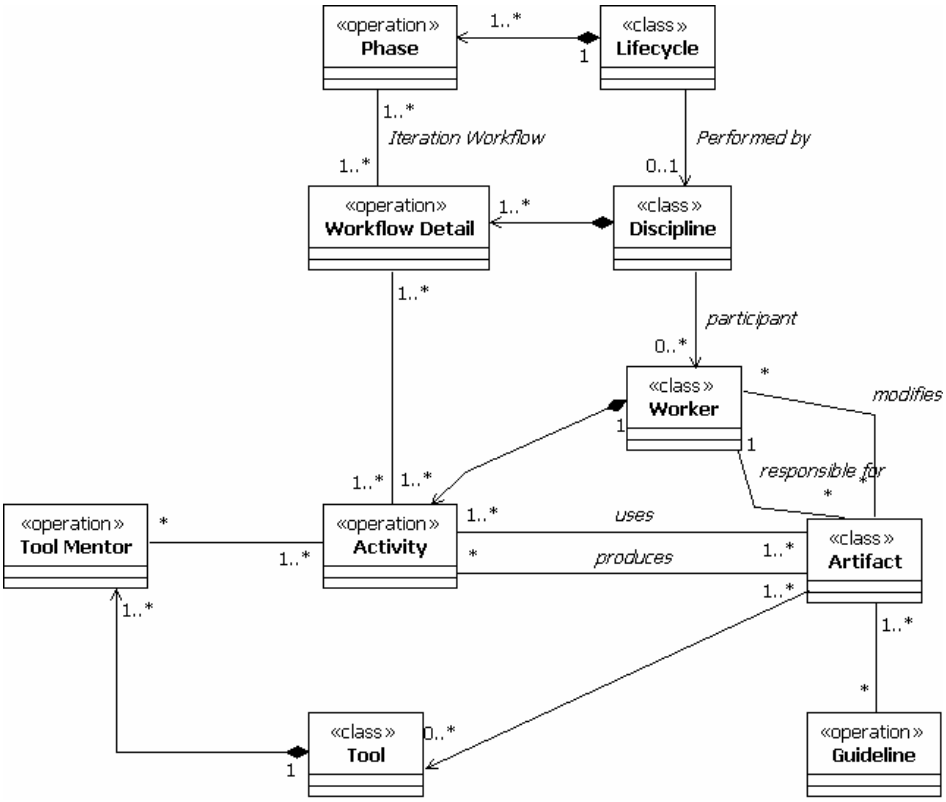


Figura 4.5: Metamodelo PRiMA-M

4.4.2 *Framework* de Processo

O *framework* é um arcabouço de processo a partir do qual podem ser instanciados diferentes processos pela seleção de elementos de processo (previamente definidos pela organização). O *framework* organiza as atividades possíveis de serem executadas em processos de software de uma organização e de seus projetos. O *framework* é composto por uma base de conhecimento da organização, um conjunto de diagramas de atividades, configurações de processo e guias de adaptação, que serão descritos a seguir:

4.4.2.1 Base de Conhecimento da Organização

A base de conhecimento da organização é formada por um conjunto de elementos de processo, usados para definir modelos de processo; uma lista de possíveis riscos de projetos, padrões de processo e organizacionais usados para representar ações preventivas e corretivas a riscos; regras de associação de padrões a riscos e metas, questões e métricas usadas para monitoramento de riscos.

A Figura 4.6 mostra um diagrama de classes, usado para representar os principais conceitos presentes na base de conhecimento.

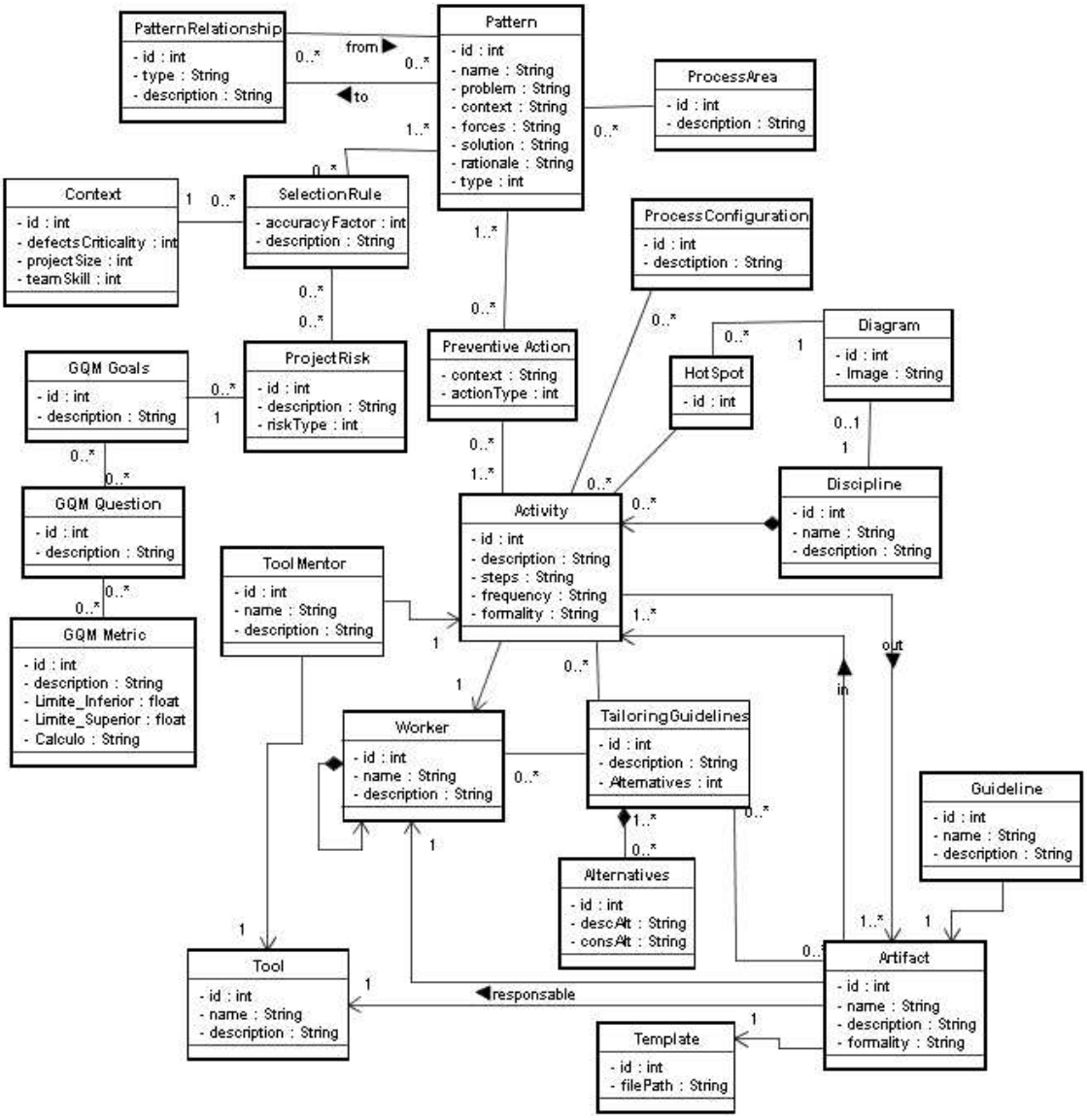


Figura 4.6: Modelo conceitual da base de conhecimento

As classes *Activity*, *Worker*, *Artifact*, *Guideline*, *Tool*, *ToolMentor*, *Discipline* e *Template* são usadas para representar os elementos de processo necessários para modelar processos de software. Essas classes fazem parte do metamodelo PRiMA-M.

No exemplo elaborado, a base de conhecimento armazena os elementos necessários para descrever customizações do *Rational Unified Process* (RUP) (RATIONAL, 2003) e do *Extreme Programming* (XP) (BECK, 2004), como uma sugestão.

Como um exemplo, na Figura 4.7 é exibida uma instância de uma atividade - "*Find Actors and Use Cases*", conforme definição do RUP (RATIONAL, 2003), instanciado a partir da base de conhecimento. A atividade é o elemento de processo principal, sendo que os demais elementos de processo estão associados a ela. Por questões de legibilidade do diagrama, os atributos foram omitidos nos diagramas de objetos. O Anexo A inclui uma descrição das atividades que fazem parte do estudo de caso realizado.

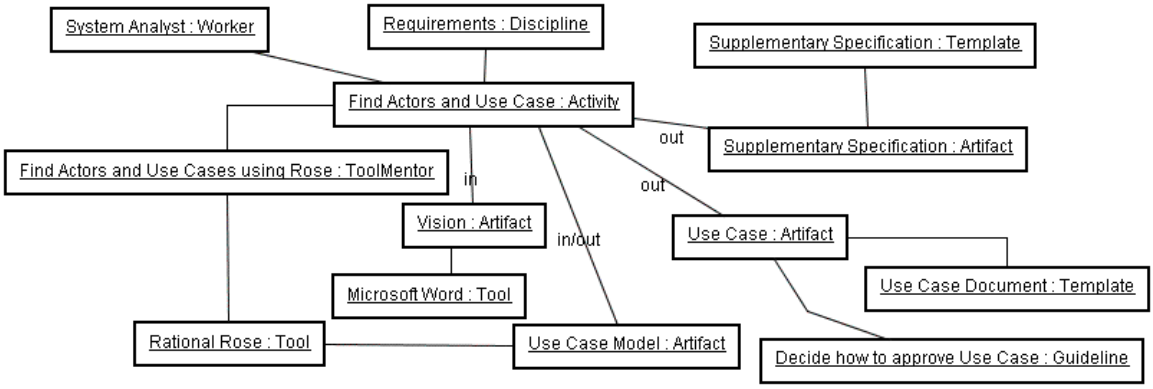


Figura 4.7: Exemplo de definição da atividade *Find Actors and Use Case*

A classe *ProjectRisk* representa instâncias de riscos que podem ocorrer em projetos de software. Na base de conhecimento de PRiMA-F são inseridos os riscos descritos na Tabela 4.2. Os riscos são associados aos padrões (classe *Pattern*) por meio de regras de seleção (classe *SelectionRule*) de acordo com o contexto (classe *Context*). A classe *PatternRelationship* descreve associações entre padrões.

A associação de padrões aos elementos de processo necessários à sua definição foi elaborada com base na descrição do próprio padrão e nas práticas descritas por modelos como o CMM (PAULK, 1993) por processos como RUP (RATIONAL, 2003) XP (BECK, 2004), Scrum (SCHWABER, 2001), FDD (PALMER, 2002), PMBok (PMI, 2000) ou por autores como Pressman (2002), Sommerville (2003), entre outros.

A classe *PreventiveAction* implementa a associação dos padrões de processo às atividades necessárias à sua implantação. Ao selecionar a atividade, os demais elementos de processo associados à atividade serão automaticamente selecionados.

A Figura 4.8, mostra como exemplo, alguns padrões associados ao risco "*Misunderstanding the Requirements"*. As regras *R1*, *R2* e *R3* associam esse risco aos padrões preventivos. Os padrões são associados às atividades necessárias a sua implantação por meio de ações preventivas. Por exemplo, a atividade *Find Actors and Use Cases* é associada como uma ação preventiva ao padrão *ScenariosDefineProblem*. As atividades precisam estar definidas na base da organização. A Figura 4.7 mostra a definição da atividade *Find Actors and Use Cases*.

Um diagrama de atividade (classe *Diagram*) é elaborado para cada disciplina para organizar as atividades possíveis de serem executadas nos processos de software de uma organização. As atividades são associadas ao diagrama de atividades por meio de pontos de extensão (classe *HotSpot*), previamente definidos para o diagrama.

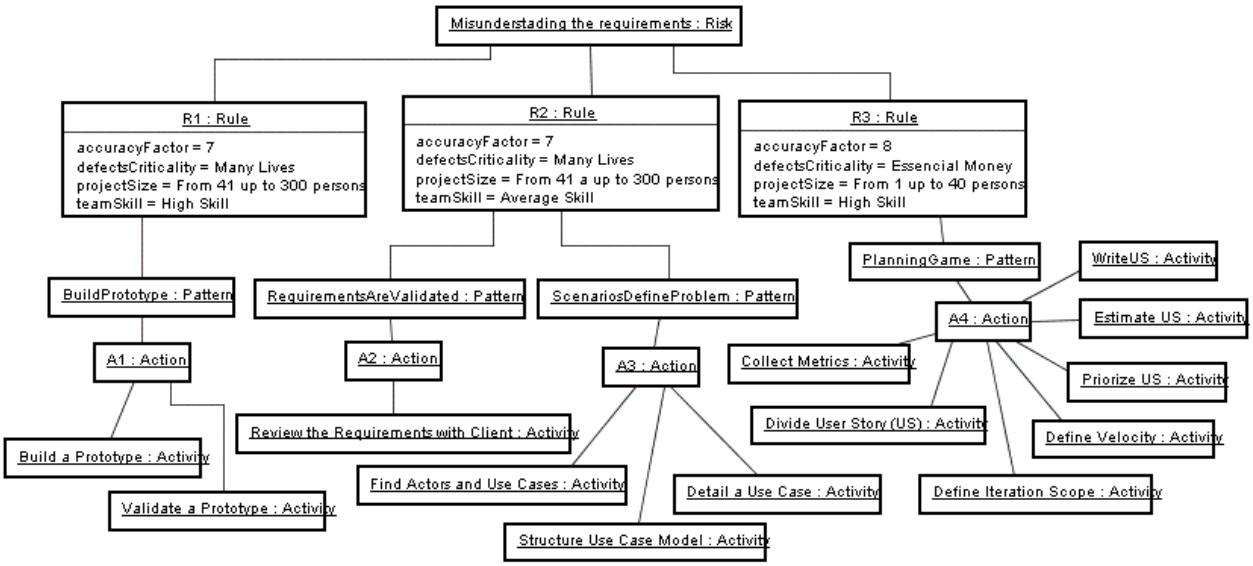


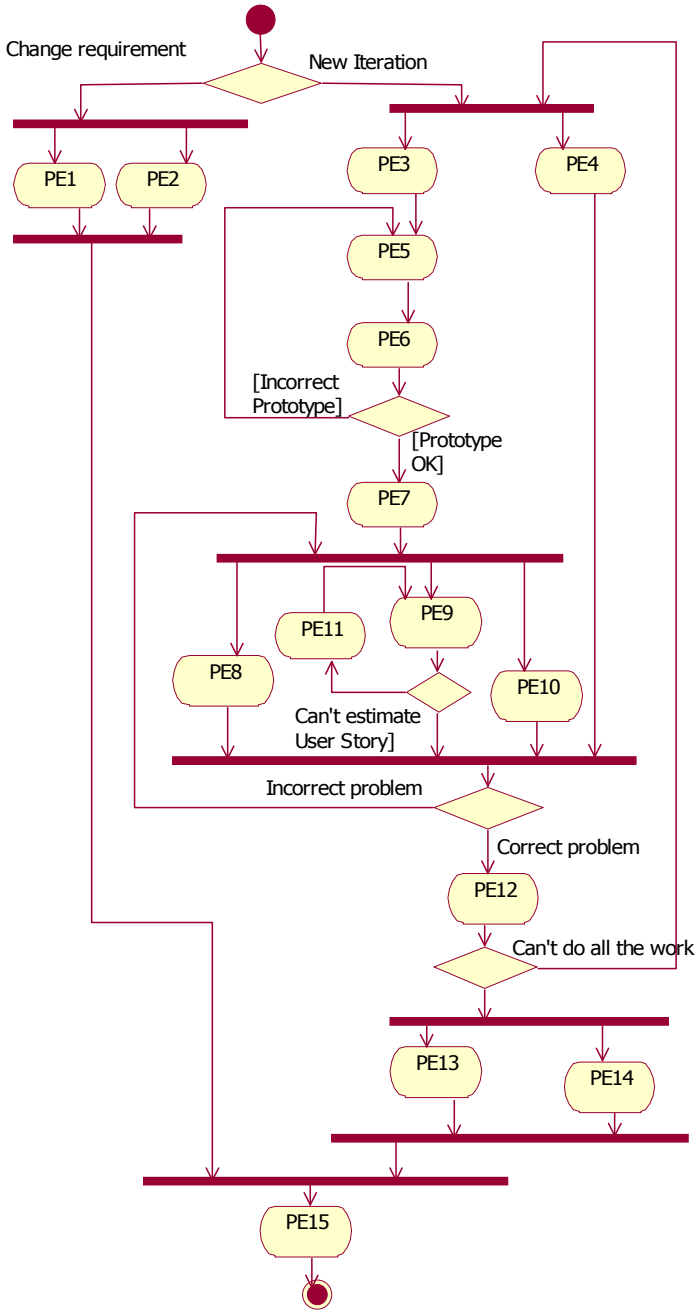
Figura 4.8: Exemplo de ações preventivas a riscos

Os pontos de flexibilização (*hot spots*) podem ser: fixos ou variáveis. Pontos de flexibilização fixos são as atividades que compõem o processo padrão da organização, e que devem existir em todos os processos de projetos de uma organização; e os pontos de flexibilização variáveis são as atividades que não fazem parte do processo da organização, e que podem ser instanciadas no processo específico para um projeto, de acordo com as características deste ou para prevenir riscos associados ao projeto. *Hot spots* possibilitam que o *framework* de processo seja flexível e tenha a capacidade de instanciar processos para diferentes projetos.

Em RUP, os diagramas de atividades descrevem a seqüência de execução de *workflow details.* Os *workflow details* descrevem grupos de atividades, mas não especificam em qual seqüência essas atividades são executadas. Neste trabalho optou-se por descrever a seqüência de execução de atividades, possibilitando a instanciação deste processo em um sistema de gerência de *workflow*, futuramente.

Cada *workflows detail*, proposto pelo RUP, foi analisado para definir a seqüência de execução das atividades que o compõem. Essa seqüência foi definida a partir de uma análise de artefatos necessários à execução de atividade e de artefatos gerados por atividades. As atividades seqüenciadas foram organizadas em diagramas de atividades, por disciplina.

O diagrama de atividades foi estendido para incluir atividades propostas por outros processos. Cada atividade do diagrama é considerada um ponto de flexibilização, pois neste ponto podem ser instanciadas diferentes atividades de acordo com as necessidades dos projetos. Os pontos de flexibilização foram definidos por comparar os objetivos das atividades, visando identificar atividades com objetivos comuns, em diferentes processos. A Figura 4.9 mostra como um exemplo, o diagrama de atividades elaborado para a disciplina de requisitos. No Anexo B são descritos diagramas de atividades elaborados para PRiMA-F, como exemplo.



Legend	
Hot Spots	**Activities**
PE1	Structure the Use-case Model (RUP)
PE2	Manage Dependencies (RUP)
PE3	Elicit Stakeholder Requests (RUP) Write User Story (XP)
PE4	Develop Requirements Management Plan (RUP)
PE5	Develop a System prototype
PE6	Validate a system prototype
PE7	Develop Vision (RUP)
PE8	Capture a Common Vocabulary (RUP)
PE9	Find Actors and Use Case (RUP) Write User Story (XP) Name chunks of functionality
PE10	Manage Dependencies (RUP)
PE11	Divide User Story (XP)
PE12	Priorize Use Cases (RUP) Priorize User Story (XP) Priorize Product Backlog Priorize the Software Requirements
PE13	Detail a Use Case (RUP)
PE14	Detail the Software Requirements (RUP)
PE15	Review Requirements (RUP) Review the Requirements with Customer

Figura 4.9: Diagrama de atividades da disciplina de Requisitos

As atividades comuns entre o RUP e o XP foram definidas com base nos trabalhos de Smith (2001) Pollice (2003) e Martin (1998), e na literatura referente ao RUP (RATIONAL, 2003) e ao XP (BECK, 2004; BECK, 1999). São consideradas atividades comuns aquelas que possuem os mesmos objetivos e seus artefatos de entrada e saída são equivalentes. Essas associações são sugestões e podem ser adaptadas as necessidades da organização.

Por exemplo, a atividade *"Write user story"* (XP) tem como objetivo definir as funcionalidades do sistema; em RUP a definição das funcionalidades do sistema é realizada por meio das atividades: *"Find actors and use case"* e *"Elicit Stakeholders Request"*. A legenda da Figura 4.9 associa os pontos de extensão PE3 e PE9 a essas atividades. Por serem atividades com objetivos comuns são instanciadas no mesmo ponto de flexibilização. No PE12, podem ser instanciadas as atividades *Priorize Use*

Cases (RUP), *Priorize User Story* (XP), *Priorize Backlog Product* (SCRUM) ou *Priorize the Software Requirements* (SOMMERVILLE, 2003). Essas atividades são propostas por diferentes processos, mas tem o mesmo objetivo, priorizar os requisitos do sistema, por isso são instanciadas no mesmo ponto de extensão.

O diagrama de atividades foi estendido para incluir atividades descritas por outros processos que não apresentam equivalentes em RUP. O local de inserção da atividade foi definido a partir de uma análise do objetivo, artefatos de entrada e saída da atividade. Por exemplo, os pontos de extensão PE6 e PE7 estão associados, respectivamente, às atividades *"Build a Sytem Prototype"* e *"Validate a System Prototype"*, que não apresentam equivalentes em RUP ou XP.

O projetista de processo é responsável por compatibilizar as atividades, papéis e artefatos entre os processos no momento de definição dos diagramas e pontos de extensão, garantindo que os processos instanciados a partir do *framework* sejam consistentes.

4.4.2.2 Configurações de Processo e Guias de Adaptação

Guias de adaptação e configurações de processo são propostos para facilitar a tarefa de adaptação.

Configurações de processo são modelos de processo pré-definidos, visando atender projetos típicos ou modelos de qualidade. Configurações de processo (classe *ProcessConfiguration*) descrevem modelos de processo que podem ser usados na definição do processo da organização. Estes modelos foram definidos a partir de configurações descritas na literatura. Exemplos de configurações incluem processos simplificados, como RUP essencial; completos, como RUP e XP; ou estendidos para satisfazer modelos de qualidade, como XP CMM Nível 2, RUP CMM Nível 2 e RUP CMM Nível 3. No *framework* de exemplo, as configurações pré-definidas são:

- RUP Essencial: descreve um conjunto mínimo de elementos para utilizar o RUP em um projeto. Definido a partir dos trabalhos de Probasco (2000), que descreve as dez atividades essenciais em processos RUP, e de Hirsch (2002), que descreve a experiência do uso de RUP com um conjunto reduzido de atividades;
- RUP Completo: inclui todas as atividades propostas pelo RUP (RATIONAL, 2003);
- RUP/CMM 2 e RUP/CMM 3: descreve os elementos de processo necessários para implantar o CMM nível 2 e nível 3, respectivamente; em um processo RUP. O conjunto e a definição das atividades necessárias para satisfazer as áreas-chave do CMM níveis 2 e 3 foram extraídos de (MANZONI, 2003);
- XP: inclui todas as atividades propostas por XP (BECK, 2004);

Essas configurações de processo são exemplos, a organização pode definir as configurações de processo de acordo com as suas necessidades. No Anexo C são descritas as atividades que compõem cada configuração sugerida como exemplo.

O SW-CMM (GINSBERG, 1995) e o CMMI (CMMI, 2002) preconizam que sejam elaborados guias descrevendo como adaptar o processo padrão da organização para encontrar as necessidades específicas de um projeto. Esses guias de adaptação de processo são associados aos elementos de processo e descrevem uma lista de alternativas para adaptação de elementos de processo (CMMI, 2002). Guias limitam as

possibilidades de adaptação, mas diminuem consideravelmente a carga de trabalho necessária para a adaptação.

Na base de conhecimento, guias (classe *TailoringGuidelines*) são associados a atividades, artefatos e papéis e descrevem alternativas para adaptação do elemento de processo (classe *Alternatives*). São propostos dois tipos de guias que são: guias para adaptação de atributos e guias para elaboração de processos híbridos.

Guias para adaptação de atributos de elementos de processo descrevem como atributos de elementos de processo podem ser adaptados de acordo com o tamanho e formalismo do projeto.

Seguindo recomendações de (GINSBERG, 1995) a Tabela 4.4 mostra alguns exemplos de adaptações possíveis. A tabela é um ponto inicial para a elaboração dos guias.

Tabela 4.4: Exemplos de elementos adaptáveis

Elemento de Processo	**Exemplo Elemento Processo**	**Atributo a ser adaptado**	**Alternativas**	**Considerações**
Worker	- Software Quality Assurance Group - Software Engineering Group - Software Engineering Process Group - Software Configuration Management Group	Performer	Individual Team Department	Small Projects Medium Projects Large Projects
Activity	Review Requirements	Frequency	Once/week Once/month At major milestones	Highly critical Medium critical Little critical
		Formality	Documented Review Informal Review Self Review	Large Project Medium Project Small Project
Artifact	Software Development Plan	Template	Informal Formal	Small Project Medium or Large Projects
		Formality	Configuration control board controlled document Configuration Management controlled document Data/version control	Large Projects Medium Projects Small Projects

Os guias são descrições textuais que descrevem detalhes de como a adaptação deve ser realizada para cada elemento de processo. No guia são identificados: o nome do elemento de processo, o tipo do elemento (atividade, artefato ou papel), seguidos de

uma lista de possíveis alternativas de adaptação. A Figura 4.10 mostra um guia de adaptação, elaborado como exemplo.

Guia de Adaptação de Processos Quanto ao Formalismo/Tamanho

Nome: Software Development Plan

Tipo de Elemento de Processo
- ☐ Atividade
- ☒ Artefato
- ☐ Papel

Alternativas de Adaptação	Considerações
Projetos Grandes e Médios	Utilizar o template formal para desenvolvimento do Software Development Plan.
Projetos Pequenos	Utilizar o template informal para desenvolvimento do Software Development Plan. O template informal é uma versão reduzida do SDP para projetos pequenos, apenas com itens necessários para esse tipo de projeto.

Figura 4.10: Guia de adaptação de processos quanto ao formalismo/tamanho

Guias também podem ser usados para auxiliar na elaboração de processos híbridos, descrevendo como adaptar elementos de processos ágeis para serem usados em métodos planejados e vice-versa.

Por exemplo: o artefato *User Story* proposto pelo método ágil XP, pode ser substituído por *Use Case*, em processos RUP. Um papel proposto por um método ágil pode ser mapeado para papéis já existentes na organização. Esse mapeamento é de responsabilidade da organização.

A Figura 4.11 mostra, como um exemplo, um guia de adaptação para elaboração de processos híbridos.

Guia de Adaptação de Processos Híbridos

Nome: Project Manager

Processo Origem: RUP

Tipo de Elemento de Processo
- ☐ Atividade
- ☐ Artefato
- ☒ Papel

Mapear para:	Processos	Considerações
Coach	XP	Em algumas situações pode ser mapeado também para o Tracker
Scrum Master	SCRUM	

Figura 4.11: Guia de adaptação de processos híbridos

4.4.2.3 Extensão do Framework

Com o uso do *framework* novos riscos e padrões podem ser identificados. Neste caso, é necessário estender o *framework* para esses novos elementos. Após identificar um novo risco é necessário associá-lo a ações preventivas, descritas como padrões. Os

padrões devem ser associados às atividades necessárias à sua implantação. Novas guias de adaptação devem ser elaboradas para os novos elementos de processo.

Após adaptar o *framework* para novos padrões e riscos, sugere-se seu uso em alguns projetos da organização, como processos experimentais e somente depois de avaliar e corrigir desvios, promover as extensões para o *framework* organizacional. A Figura 4.12 descreve, usando um diagrama de atividade, o processo proposto para extensão do *framework*.

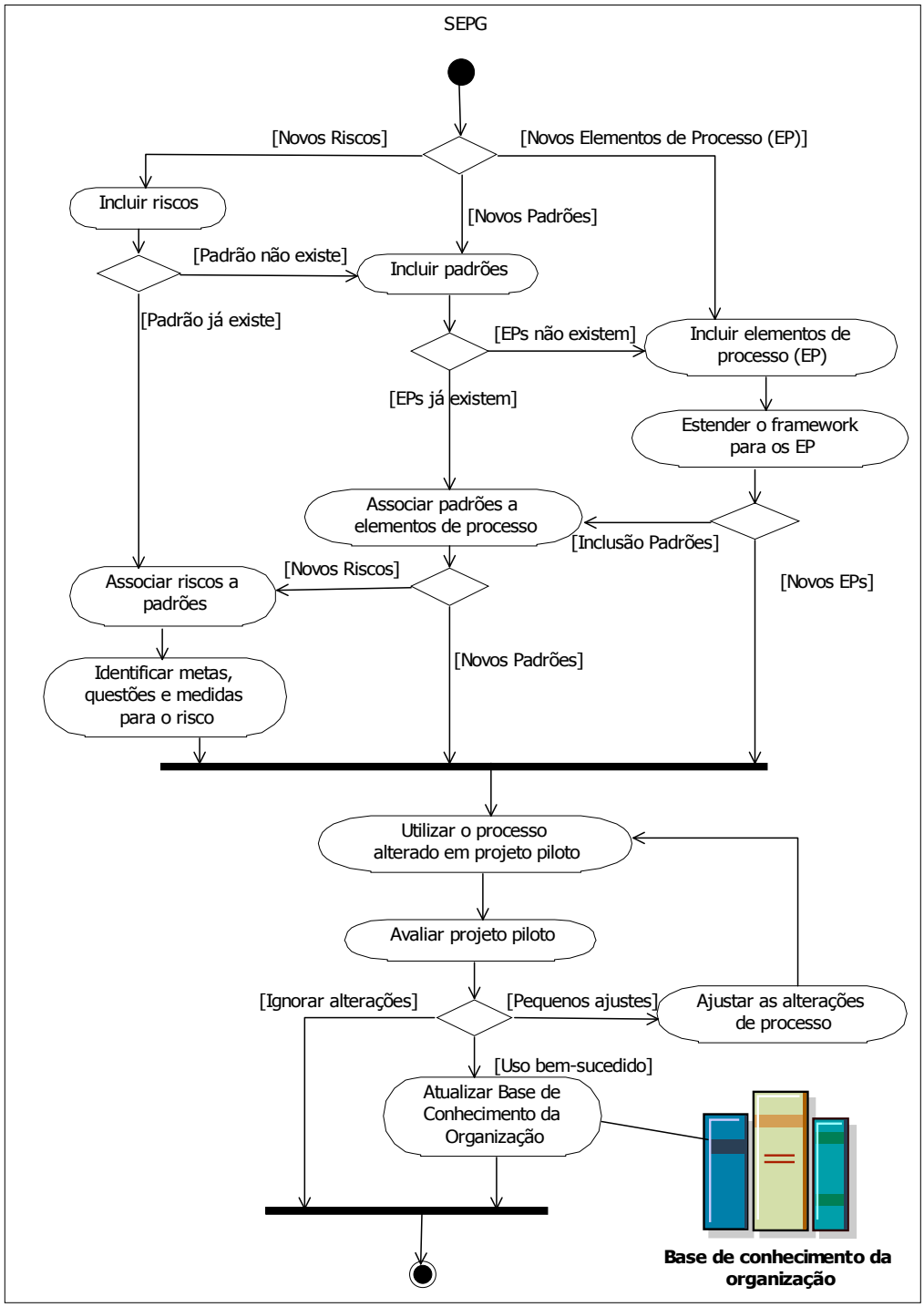


Figura 4.12: Extensão do *framework* PRiMA-F

As classes *GQMGoals*, *GQMQuestion* e *QMMetric* são usadas na definição do plano GQM descrito na próxima seção.

4.5 Elaborar o Plano GQM para Monitorar os Riscos

A abordagem *Goal/Question/Metric* (GQM) (BASILI, 1988) descrita na seção 2.5.1 deste trabalho, é utilizada para definir as métricas a serem usadas para acompanhar o progresso dos fatores de risco, possibilitando ao gerente de projeto tomar ações corretivas, quando necessário e no momento adequado.

As metas GQM devem ser formuladas da seguinte forma: "Analisar o <*objeto de estudo*> com a finalidade de <*objetivo*> com respeito ao <*enfoque*> do ponto de vista de <*ponto de vista*> no seguinte contexto <*contexto*>".

Sendo que os atributos, em itálico na sentença acima, podem ser definidos como (WANGENHEIM, 2000):

- Objeto de estudo: identifica o que será analisado. Exemplo: processo de software, projeto, documento, sistema, etc.
- Objetivo: porque o objeto será analisado. Exemplo: avaliar, melhorar, monitorar, controlar, etc.
- Enfoque: identifica o atributo que será analisado. Exemplo: confiabilidade, custos, correção, etc.
- Ponto de vista: identifica quem utilizará as métricas coletadas. Exemplo: equipe de desenvolvimento, gerente de projeto, etc.
- Contexto: identifica o ambiente onde o programa de medição está localizado. Exemplo: projeto A, departamento X.

4.5.1 Usando GQM para Monitorar Riscos

A Tabela 4.5 mostra algumas metas GQM, formuladas a partir dos riscos de projeto exibidos na Tabela 4.1; questões e métricas derivadas a partir da meta, apenas como sugestão. Cada projeto deverá identificar as métricas mais adequadas.

Neste trabalho, omitiu-se o contexto na definição da meta, pois este vai depender do ambiente em que a sistemática será aplicada.

Tabela 4.5: Metas e questões GQM por risco

Risco 1: Falta de compromisso da gerência sênior com o projeto	
Meta 1: Meta 1: Analisar o projeto com a finalidade de monitorar com respeito ao comprometimento da gerência do ponto de vista da equipe de desenvolvimento.	
Questão 1.1: A gerência sênior (Scrum Master, XP Manager) está participando das reuniões de acompanhamento do projeto?	*Métrica 1.1.1:* Percentual de participação do gerente sênior PPGS = (número de reuniões que o gerente sênior participou / número de reuniões gerenciais realizadas) * 100

Questão 1.2: A gerência sênior (Scrum Master, XP Manager) está solucionando os problemas da equipe a ela repassados?	*Métrica 1.2.1:* Percentual de solução de problemas PPGS = (número de problemas solucionados / número de problemas em aberto) * 100 *Métrica 1.2.2:* Percentual de solução de problemas no tempo apropriado PPGS = (número de problemas solucionados no prazo/ número de problemas repassados a gerência sênior) * 100
Risco 2: Falha em obter compromisso dos usuários e falta de envolvimento adequado dos usuários	
Meta 2: Analisar o projeto com a finalidade de monitorar com respeito ao comprometimento dos usuários do ponto de vista da equipe de desenvolvimento.	
Questão 2.1: Os usuários estão participando das reuniões do projeto?	*Métrica 2.1a:* Percentual de participação do usuário em reuniões (Planning Game, SprintPlanningMeeting, Elicitação Requisitos, revisões, workshops) PPU = (número de reuniões que usuários participaram / número de reuniões realizadas) * 100 *Métrica 2.1b:* Interações do usuário (comparar histórico) NIU = número de interações (telefone, e-mail, pessoalmente) entre o usuário e a equipe
Questão 2.2: Os usuários estão cumprindo os prazos acordados com o gerente de projeto para validação de protótipos, critérios de aceite de produtos, documentos?	*Métricas 2.2*: Cumprimento de tarefas pelo usuário (comparar histórico) TMCT = Tempo médio de cumprimento de tarefas pelo usuário por tipo de tarefa
Risco 3: Não-entendimento dos requisitos, falha na gerência de expectativas dos usuários e instabilidade dos requisitos	
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:* 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
Risco 4: Escopo/objetivos não-claros e diferentes expectativas sobre o sistema entre distintos usuários	
Meta 4: Analisar o projeto com a finalidade de controlar com respeito às solicitações de mudança do ponto de vista da equipe de desenvolvimento.	
Questão 4.1: Qual o número de solicitações de alteração de escopo/objetivos do projeto?	*Métrica 4.1:* Número de solicitações de mudança (comparar com dados históricos) NSM = Número de solicitações de mudança de escopo/objetivos
Questão 4.2: O tempo para incorporar as mudanças está adequado?	*Métrica 4.2:* 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
Risco 5: Falta de conhecimento/habilidade requerida do pessoal do projeto e equipe insuficiente ou inadequada	
Meta 5: Analisar o projeto com a finalidade de avaliar com respeito à habilidade da equipe do ponto de vista do gerente de projeto.	
Questão 5.1: Qual o percentual de pessoas habilitadas para desempenhar seu papel na equipe?	*Métrica 5.1:* Percentual de pessoas habilitadas PPH = (número de pessoas habilitadas / número de pessoas da equipe) * 100
Questão 5.2: Existem recursos humanos suficientes para o projeto?	*Métrica 5.2:* Percentual de Recursos Humanos PRH = (número de RH / número de RH estimados pelo gerente de projeto[1])*100

[1] O número de recursos estimados é baseado no esforço necessário para desenvolver o sistema. O esforço pode ser estimado com base no tamanho do software, usando-se técnicas como: Análise por Pontos de Função ou Análise por Casos de Uso.

Questão 5.3: Qual o percentual de treinamentos atendidos?	*Métrica 5.3:* Percentual de Treinamentos Atendidos PRH = (número de treinamentos atendidos / número de treinamentos solicitados)*100
Risco 6: Introdução de novas tecnologias	
Meta 6: Analisar o projeto com a finalidade de avaliar com respeito à nova tecnologia do ponto de vista da equipe de desenvolvimento.	
Questão 6.1: Qual o índice de retrabalho por problemas de tecnologia?	*Métrica 6.1:* Percentual de Retrabalho por tecnologia PRT = (quantidade de horas de retrabalho por defeitos ou falhas devido à tecnologia / quantidade total de horas de retrabalho) * 100
Questão 6.2: Qual o percentual de defeitos por tecnologia?	*Métrica 6.2:* Percentual de defeitos por tecnologia PDT = (quantidade de defeitos por tecnologia / quantidade total de defeitos)*100
Questão 6.3: Qual o percentual de pessoas na equipe que dominam a tecnologia?	*Métrica 6.3:* Percentual de pessoas que dominam a tecnologia PPDT = (número de pessoas que dominam a tecnologia / número total de pessoas)*100
Risco 7: Cronograma e orçamento não-realistas	
Meta 7: Analisar o projeto com a finalidade de monitorar com respeito as estimativas do ponto de vista do gerente de projeto.	
Questão 7.1: Qual a precisão das estimativas?	*Métrica 7.1a:* Percentual de casos de uso desenvolvidos iteração CUDI = (numero de casos de uso assinalados no inicio da iteração/casos de uso concluídos na interação) * 100 *Métrica 7.1b:* Precisão das estimativas de esforço PEE = (esforço real da iteração/ esforço estimado da iteração) *100 *Métrica 7.1c:* Precisão das estimativas de custo PEC = (custo real da iteração/ custo estimado da iteração)*100
Risco 8: Falta de uma metodologia de gerência de projeto efetiva	
Meta 8: Analisar o projeto com a finalidade de monitorar com respeito ao uso da metodologia do ponto de vista da equipe de desenvolvimento.	
Questão 8.1: A equipe está utilizando a metodologia definida para o projeto?	*Métrica 8.1:* Número de ocorrência geradas em revisões (comparar dados históricos) NOR = Número de ocorrência geradas nas revisões de qualidade

Questão 8.2: A equipe está produzindo todos os documentos previstos na metodologia?	*Métrica 8.2:* Percentual de tipos de documentos produzidos PDP = (tipos de documentos produzidos/tipos de documentos previstos na metodologia) * 100
Risco 9: Acréscimo de funcionalidades idealizadas pela equipe sem o aval do cliente (*gold plating*)	
Meta 9: Analisar o projeto com a finalidade de monitorar com respeito ao acréscimo de funcionalidades do ponto de vista do gerente de projeto.	
Questão 9.1: As funcionalidades entregues foram aceitas pelo cliente?	*Métrica 9.1:* Percentual de aceitação de funcionalidades PAF = (número de funcionalidades aceitas pelo cliente/ número de funcionalidades entregues) * 100
Risco 10: Desenvolvimento errado das funções ou interface	
Meta 10: Analisar o projeto com a finalidade de avaliar com respeito à comunicação do ponto de vista da equipe de desenvolvimento.	
Questão 10.1: As solicitações de mudanças encaminhadas ao gerente de projeto são comunicadas a equipe?	*Métrica 10.1:* Percentual de solicitações de mudanças encaminhadas PSME = (Quantidade de solicitações de mudanças encaminhadas à equipe / Quantidade de solicitações de mudanças do cliente)*100
Risco 11: Subcontratação	
Meta 11: Analisar o projeto com a finalidade de monitorar com respeito à subcontratação do ponto de vista do gerente de projeto.	
Questão 11.1: Os prazos acordados com a subcontratada estão sendo cumpridos?	*Métrica 11.1a:* Cumprimento de prazos pela subcontratada CPS = duração real da atividade / duração estimada da atividade *Métrica 11.1b:* Percentual de casos de uso entregues PCUE = (número de casos de uso entregues/número de casos de uso acordados) * 100
Questão 11.2: Os produtos entregues pela subcontratada satisfazem os critérios de aceite?	*Métrica 11.2:* Percentual de requisitos aceitos da subcontratada (inclui testes) PRAS = (número de requisitos aceitos / número de requisitos entregues)*100
Risco 12: Uso de recursos e desempenho do sistema inadequados	
Meta 12: Analisar o sistema com a finalidade de avaliar com respeito ao desempenho do ponto de vista do usuário.	
Questão 12.1: O desempenho atual do sistema satisfaz o usuário?	*Métrica 12.1:* Desempenho do sistema DS = (desempenho atual do sistema / desempenho desejado do sistema)*100

Risco 13: Projeto (desenho) inviável	
Meta 13: Analisar o projeto com a finalidade de avaliar com respeito ao projeto (desenho) do ponto de vista da equipe de desenvolvimento.	
Questão 13.1: A arquitetura definida é viável?	*Métrica 13.1a:* Viabilidade da arquitetura EA = (número de atividades de desenho concluídas / número de atividades de desenho total)
Observação: o desempenho do sistema envolve os requisitos não-funcionais da aplicação. Para cada projeto devem ser definidas métricas de desempenho para avaliar itens específicos estabelecidos pelo usuário, tais como: tamanho da base de dados, crescimento da base de dados, necessidades de reorganização da base de dados, tempo de resposta, processos concorrentes, etc.	

Para cada métrica são definidos dois atributos: o alvo e o limite. O alvo expressa quantitativamente a meta a ser atingida e o limite define o nível de alerta associado a meta, isto é, quando o limite é atingido tem-se um indicativo que determinado risco, associado a métrica, está próximo a se materializar.

As métricas, descritas neste trabalho, são apenas sugestões, e precisam ser avaliadas de acordo com as características de cada projeto. Usando a abordagem GQM, a equipe define as métricas a partir das metas de sua organização.

4.6 Monitorar os Riscos do Projeto

O objetivo dessa atividade é coletar as medidas a serem usadas para acompanhar os riscos de software. O ideal é que as medidas sejam inseridas em uma base histórica, para serem usadas em estimativas futuras e no acompanhamento do projeto, bem como servirem para extrair e comparar resultados entre projetos.

Durante o projeto, o gerente deve monitorar continuamente os riscos do projeto, para identificar quando algum risco está próximo a se materializar, para tomar ações corretivas. O monitoramento é realizado com base no Plano *Goal/Question/Metric*, elaborado para o projeto.

Como resultado do monitoramento dos riscos do projeto, três situações podem ocorrer: as métricas estão dentro de limites aceitáveis, as métricas ultrapassam limites permitidos ou novos riscos são identificados.

Quando as métricas estão dentro de limites aceitáveis, não é necessária nenhuma ação adicional, pois os riscos estão sob controle.

Quando os dados coletados indicam que as métricas estão próximas aos limites definidos, significa que a risco pode se materializar. Neste caso é necessário selecionar novos padrões para a resolução dos riscos. A atividade *Selecionar padrões* é executada e como conseqüência o processo definido para o projeto é alterado.

Considerando um processo de desenvolvimento iterativo, ao final de cada iteração, ou periodicamente, os riscos identificados e priorizados para o projeto devem ser revistos. Nessa revisão novos riscos podem ser identificados e novas ações preventivas podem ser incluídas no processo de software para o projeto. Outras atividades podem ser adicionadas ao processo definido para o projeto, devido a mudanças no ambiente do projeto.

Tanto a seleção de novos padrões quanto a identificação de novos riscos, gera redefinições no processo do projeto, modificando-o dinamicamente. *Dynamic Tailoring* suporta a redefinição do processo durante o projeto, não somente no início do mesmo.

4.7 Trabalhos Relacionados

Este trabalho diferencia-se dos demais trabalhos encontrados na literatura especializada por propor a adaptação de processos e a gerência de riscos como uma abordagem única. Nesta seção, os trabalhos relacionados estão organizados pelos temas adaptação de processos e gerência de projetos, visto que esses processos são tratados de forma independente nos trabalhos pesquisados.

4.7.1 Adaptação de Processos

Metamodelos e *frameworks* de processo são utilizados como base para a adaptação de processos. Outra forma de adaptação é a adequação de determinados processos a modelos de qualidade, tais como: CMM, CMMI, SPICE, entre outros. Nessa seção são descritos trabalhos que propõem a elaboração de metamodelos de processo e *frameworks* ou abordagens para adaptação de processos.

Em (LEPASAAR, 2002) é proposta a integração dos metamodelos do SPICE e CMMI para criar um modelo de avaliação de processos que permita avaliar processos com base em ambos os modelos de avaliação de processos. A integração dos modelos é realizada por meio da classificação dos elementos de processo dos modelos de avaliação, realizada de forma subjetiva. Não é descrito como o metamodelo pode ser utilizado, não é apresentada ferramenta para apoiar o uso do metamodelo.

Em (RAUTIAINEN, 2002) é descrito um *framework* para gerenciar desenvolvimento de produtos de software em pequenas companhias. O *framework* combina quatro ciclos de controle que são: gerenciamento estratégico de release, gerenciamento de release do projeto, gerenciamento de iteração e *mini-milestones*. Para cada ciclo são descritos componentes que podem ser usados para gerenciar o desenvolvimento e podem ser adaptados de acordo com as necessidades do projeto, mas esses componentes não são detalhados. Não descreve métricas que podem ser usadas no monitoramento do desenvolvimento.

Harjumaa (2004) propõe o uso de padrões para melhorar o processo de inspeção de software. Após uma análise do processo de inspeção atual são identificados padrões para melhorar o processo. A seleção dos padrões é realizada a partir da seção <*Symptoms*> da descrição do padrão. Não são descritas ferramentas para apoiar a seleção dos padrões.

Em (FALBO, 2004) é descrito uma abordagem de gerenciamento de conhecimento para adaptação de processos baseada em métricas coletadas de projetos passados. O conhecimento organizacional é descrito por meio de ontologias. A ferramenta ProKnowHow apóia o uso da abordagem proposta. De forma semelhante a esse trabalho, o paradigma GQM é usado para descrever as métricas.

Em (YOON, 2001) é sugerida uma sistemática para definição e adaptação de processos usando um Grafo Artefato Atividade (AAG). O modelo de processo consiste de módulos de processo inter-relacionados. Cada módulo é descrito por um AAG, que

representa as atividades e artefatos do processo. São definidos algoritmos para inclusão, exclusão, divisão e união de atividades.

Fiorini (2001) propõe uma arquitetura para descrever e organizar processos de software, visando o reuso destes. Esta arquitetura é baseada em tipos de processos (*standard*, *pattern* e usual), em *frameworks* de processo, e em diferentes tipos de linguagens de modelagem de processos, especificadas em XML – *Extensible Markup Language*, para descrever cada um dos tipos de processos. Com a finalidade de facilitar a reutilização e recuperação de informações se faz uso de facetas, guias de reutilização e utiliza-se *process patterns* organizados e classificados em comunidade, família e indivíduo. A arquitetura elaborada descreve processos relacionados à gerência de requisitos de software. Fiorini não descreve como os processos podem ser evoluídos.

Machado et al. (2000) propõem uma abordagem para a adaptação de processos de software para projetos específicos. A partir do processo adaptado é possível gerar um Ambiente de Desenvolvimento de Software Orientado à Organização para o projeto (ADSOrg). ADSOrg são ambientes que apóiam a atividade de engenharia de software em uma organização, fornecendo o conhecimento acumulado e relevante para esta atividade, e dando apoio ao aprendizado organizacional em Engenharia de Software. Dentro desde mesmo projeto, é proposta uma ferramenta para gerenciar riscos chamada RiscPlan (FARIAS, 2003). O processo criado é instanciado na estação Taba, limitando dessa forma a sua utilização.

Gnatz et al. (1998) propõem um *framework* para descrever processos de software. Esse *framework* suporta a adaptação estática e dinâmica dos processos, e assim como este trabalho utiliza padrões de processo para a adaptação de processos. No *framework* não são considerados modelos ou normas de qualidade.

Alistair Cockburn (2000) propõe um *framework* para escolha da metodologia apropriada para um dado projeto. Segundo Cockburn considerar múltiplas metodologias é apropriado e necessário. A metodologia adequada será escolhida considerando duas dimensões: o tamanho da equipe e a criticidade do sistema. Boehm e Turner (2003) estendem o *framework* de Cockburn e usam critérios de riscos para balancear entre métodos ágeis e planejados.

4.7.2 Gerência de Riscos

Em (KIPER, 2005), é descrito um modelo probabilístico de prevenção e detecção de defeitos (DDP), baseado em riscos e uma ferramenta de software. O modelo DDP envolve três conceitos: requisitos, riscos e mitigação. As fórmulas, usadas para calcular o impacto de múltiplos riscos em um mesmo requisito, são pré-definidas e não necessariamente se aplicam a todas as situações. O sucesso depende de especialistas no domínio da aplicação para elaborar as fórmulas de impacto dos riscos.

Keshlaf e Hashim (2005) propõem um processo de gerenciamento de riscos e uma ferramenta, chamada SoftRisk, para apoiar o processo. Não define como os riscos são monitorados e controlados durante o projeto de software.

Em (DILLON, 2005) é descrito um modelo de decisão projetado para suportar o gerenciamento de reservas para minimizar riscos técnicos e gerenciais.

Roy (2004) propõe um *framework*, chamado de ProRisk para gerenciar riscos em projetos de software. Esse *framework* requer uma análise detalhada da organização e

domínio do projeto para desenvolver um conjunto de fatores de riscos e organizá-los de forma que possa refletir as diferentes perspectivas de riscos. Diferentemente deste trabalho as ações a serem tomadas para minimizar ou evitar os riscos não são apoiadas pelo *framework* e nem a questão de identificação das métricas para monitorar os riscos.

Kontio et al. (1998) propõem uma sistemática para gerenciamento de riscos, chamada de Riskit. O método Riskit utiliza cenários de riscos e grafos de análise para definir os diferentes aspectos envolvidos na definição do risco. O grafo de análise de Riskit pode ser visto como um modelo conceitual para definição de riscos.

5 AMBIENTE EXPERIMENTAL PARA GERÊNCIA DE RISCOS EM PROJETOS

O uso da sistemática para gerenciar riscos em projetos de riscos pode ser uma tarefa complexa e trabalhosa em qualquer organização se executada de maneira manual. Por isso, é desejável a implementação de um ambiente que apóie a adaptação do processo da organização e a gerência de riscos. Tal implementação tem como finalidade possibilitar uma experimentação prática do uso da abordagem proposta.

O ambiente experimental foi desenvolvido utilizando-se a Linguagem Java para especificação das regras de negócio e acesso a dados; Java Server Pages (JSP), HTML e Javascript na camada de apresentação, banco de dados MySQL e o servidor de aplicação Tomcat.

A seção 5.1 descreve uma visão geral do ambiente experimental. A seção 5.2 descreve PRiMA-Tool, a ferramenta proposta por esta tese para adaptação de processos. Na seção 5.2.1 são descritas as atividadades necessárias para configuração do ambiente experimental para atender as necessidades específicas de uma organização. Na seção 5.2.2 são descritas as atividades para adaptação de processos da organização para prevenir riscos de projetos.

5.1 Visão Geral

O ambiente experimental elaborado é composto por dois módulos principais, que são: *Pattern-based Methodology Tailoring* (PMT-Tool) e *Project Risk Management Approach* (PRiMA-Tool).

O módulo *Pattern-based Methodology Tailoring* (PMT-Tool) (HARTMANN, 2005) foi desenvolvido por Júlio Hartmann. PMT é responsável por catalogar os padrões de processo e associá-los aos riscos de software por meio de regras, bem como selecionar os padrões para prevenir riscos priorizados para determinado projeto. Esta tese limita-se a descrever as funcionalidades de PMT que interagem com PRiMA-Tool. Maiores informações sobre PMT podem ser obtidas em (HARTMANN, 2005).

O módulo *Project Risk Management Approach* (PRiMA-Tool) foi desenvolvido como parte dessa tese. Esse módulo é responsável pela elaboração do processo de software do projeto, a partir da adaptação do processo da organização, inserindo neste os padrões selecionados para prevenir os riscos de projeto e pela definição do Plano *Goal/Question/Metric* para gerência dos riscos do projeto.

5.2 PRiMA – Tool

A Figura 5.1 mostra o formulário principal de PRiMA-Tool. No menu à esquerda são exibidas as funcionalidades da ferramenta. As funcionalidades de PRiMA-Tool são organizadas em três grupos, que são:

- *Organization*: descreve as funcionalidades relacionadas à organização, tais como: cadastrar uma nova organização, criar um processo padrão para a organização e modificar o processo padrão da organização, definido previamente;
- *Project*: descreve as funcionalidades relacionadas ao projeto de software, tais como: cadastrar um novo projeto, definir os riscos para o projeto, criar uma linguagem de padrões, selecionar padrões, elaborar o processo para o projeto, modificar o processo elaborado, criar o Plano GQM, visualizar o plano GQM e criar um *website*, com a descrição do processo para o projeto;
- *Knowledge Base*: armazena o conhecimento organizacional utilizado na adaptação dos processos e na gerência de riscos. A base de conhecimento é subdividida em quatro grupos, que são:
  - *Process Elements*: elementos de processo (atividades, artefatos, ferramentas, guias, papéis, etc.) usados na elaboração de processos de software, os diagramas de atividades, elaborados por disciplina e os pontos de flexibilização (*hot spots*);
  - *Process Configuration*: configurações de processo pré-definidas e as guias de adaptação de processos;
  - *Goal/Question/Metric*: metas, questões e métricas e associação de métricas a questões, de questões a metas e de metas a riscos;
  - *Pattern*: cadastro de padrões e associação de padrões a elementos de processo necessários à sua implantação.

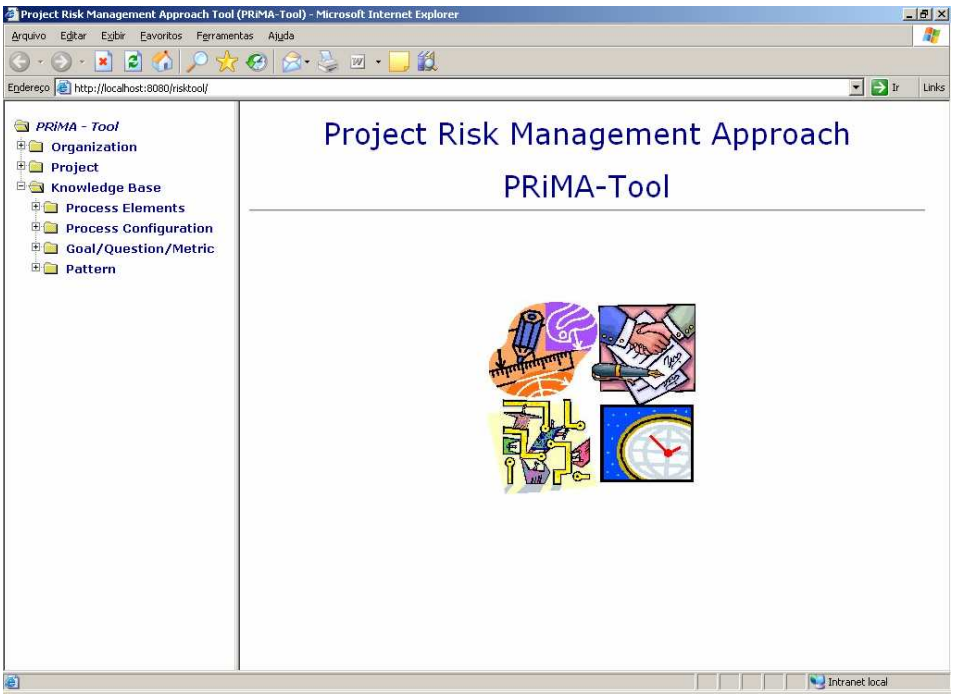


Figura 5.1: Formulário Principal de PRiMA - Tool

A Figura 5.2 mostra, em destaque, as classes atualizadas por PMT-Tool e acessadas para consulta por PRiMA-Tool, que são: *PatternRelationship*, *Pattern*, *Context*, *SelectionRule* e *ProcessArea*. As demais classes são atualizadas e acessadas somente por PRiMA-Tool. Esse modelo conceitual da base de conhecimento já foi discutido na seção 4.4.

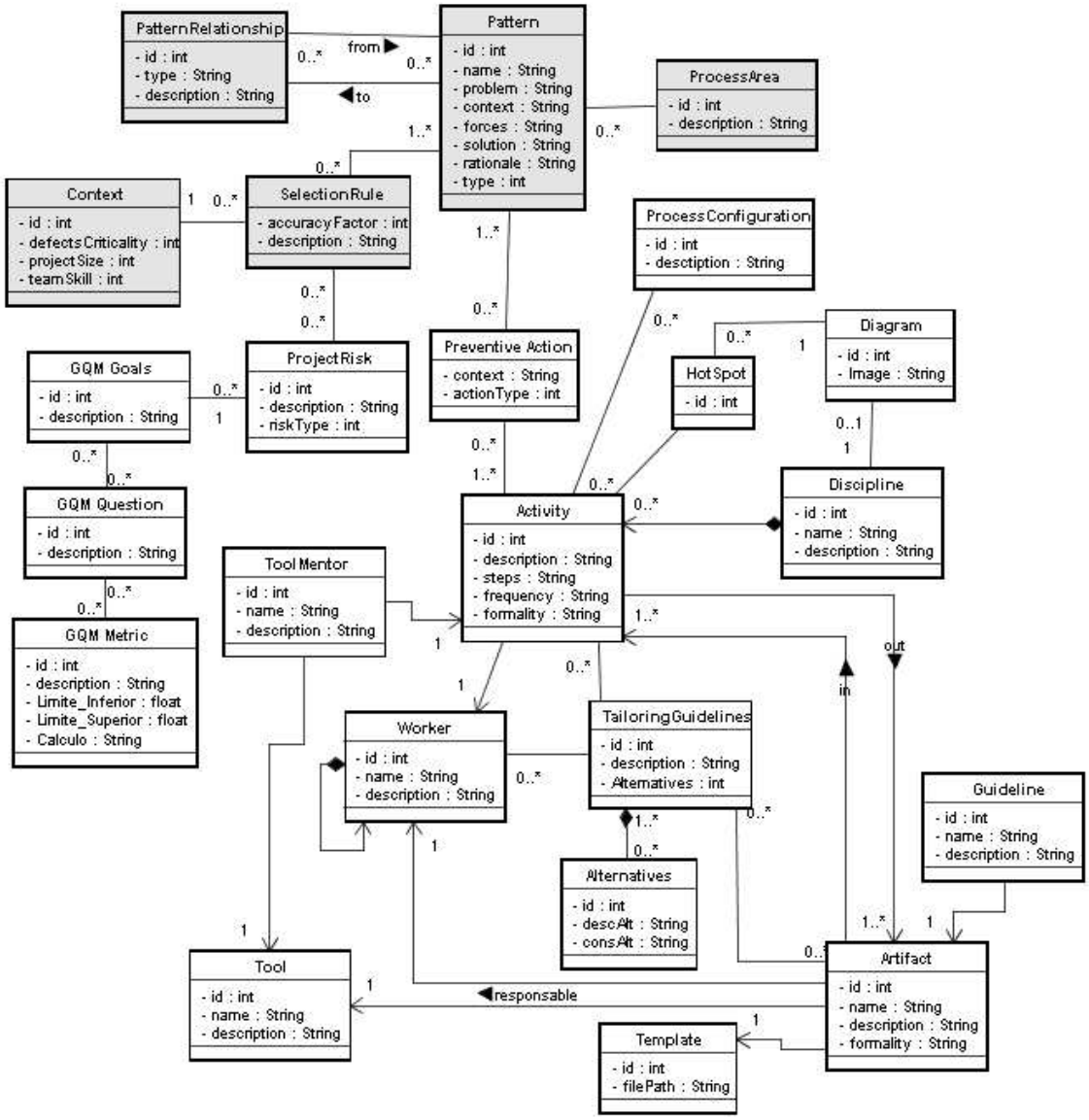


Figura 5.2: Modelo conceitual da base de conhecimento

A adaptação de processos ocorre com base em um *framework* de processo (PRiMA-F), conforme descrito na seção 4.4.2 desta tese. Inicialmente, é necessário configurar o ambiente experimental, de acordo com o *framework* definido pela organização para atender às suas necessidades. Na seção 5.2.1 são descritas as atividades necessárias à configuração do ambiente. Após a configuração do ambiente, o processo definido para a organização pode ser configurado para atender as necessidades específicas de cada projeto e prevenir os riscos identificados para esse. A adaptação do processo é descrita na seção 5.2.2 deste capítulo.

5.2.1 Configurando o *Framework* da Organização

A Figura 5.3 mostra, usando um diagrama de atividades, as atividades necessárias para a configuração do *framework* no ambiente experimental. O SEPG (*Software Engineering Process Group*) é o responsável pela configuração do *framework*, caso não SEPG exista na organização, a responsabilidade será do projetista de processo. No estudo de caso é descrita a elaboração de um *framework* de processo com as atividades do RUP e do XP.

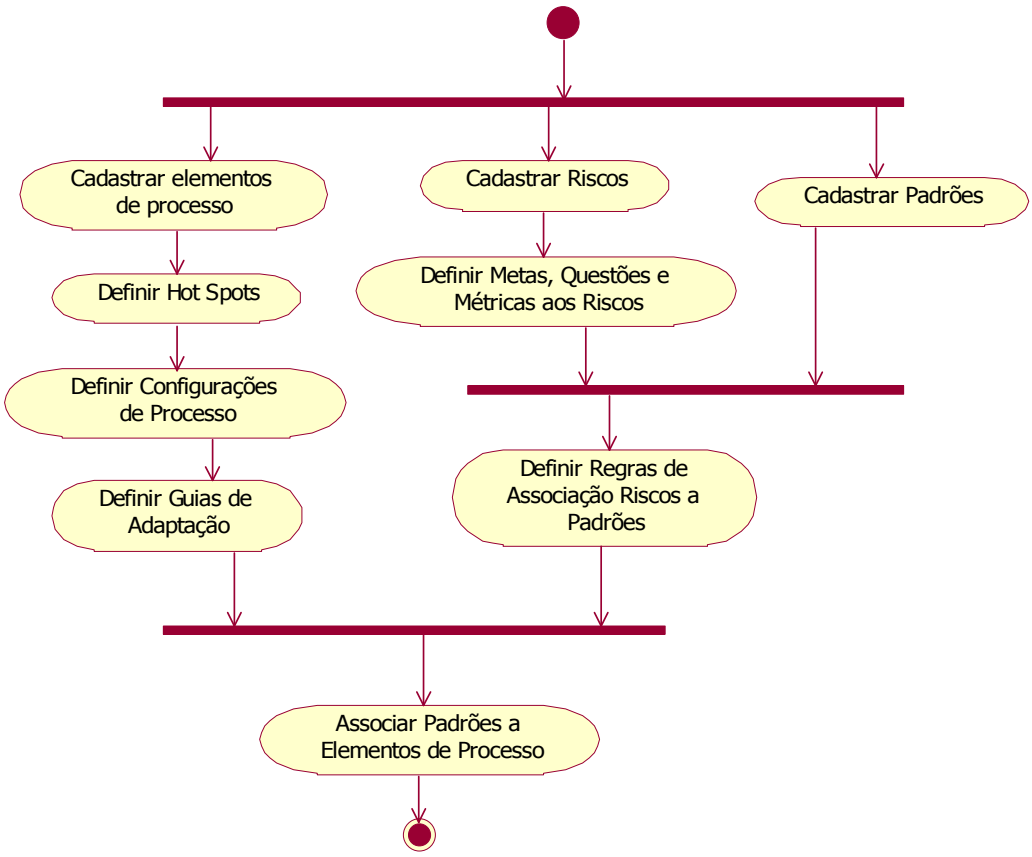


Figura 5.3: Atividades para configuração do *framework* no ambiente experimental

A primeira atividade a ser realizada é o cadastro dos elementos de processo (atividades, artefatos, papéis, modelos, guias, disciplinas, etc.) na base de conhecimento de PRiMA-Tool. Os elementos de processo são representados segundo os conceitos descritos no metamodelo PRiMA-M. Foram definidos formulários *Web* específicos para cada elemento de processo. A Figura 5.4 (esquerda) mostra o formulário para cadastro de disciplinas. Ao cadastrar a disciplina é necessário informar o diagrama de atividades e os pontos de extensão desse diagrama, como mostra a Figura 5.4 (direita). No anexo A são descritas as atividades que fazem parte do *framework* PRiMA-F, bem como os elementos associados a cada atividade.

A opção *Hot Spots* possibilita ao projetista associar os pontos de extensão às atividades possíveis de serem instanciadas em cada *hot spot*, inseridas previamente na base de conhecimento, como mostra a Figura 5.5. No Anexo B são descritos os diagramas de atividades e os pontos de extensão, definidos para cada disciplina, em PRiMA-F.

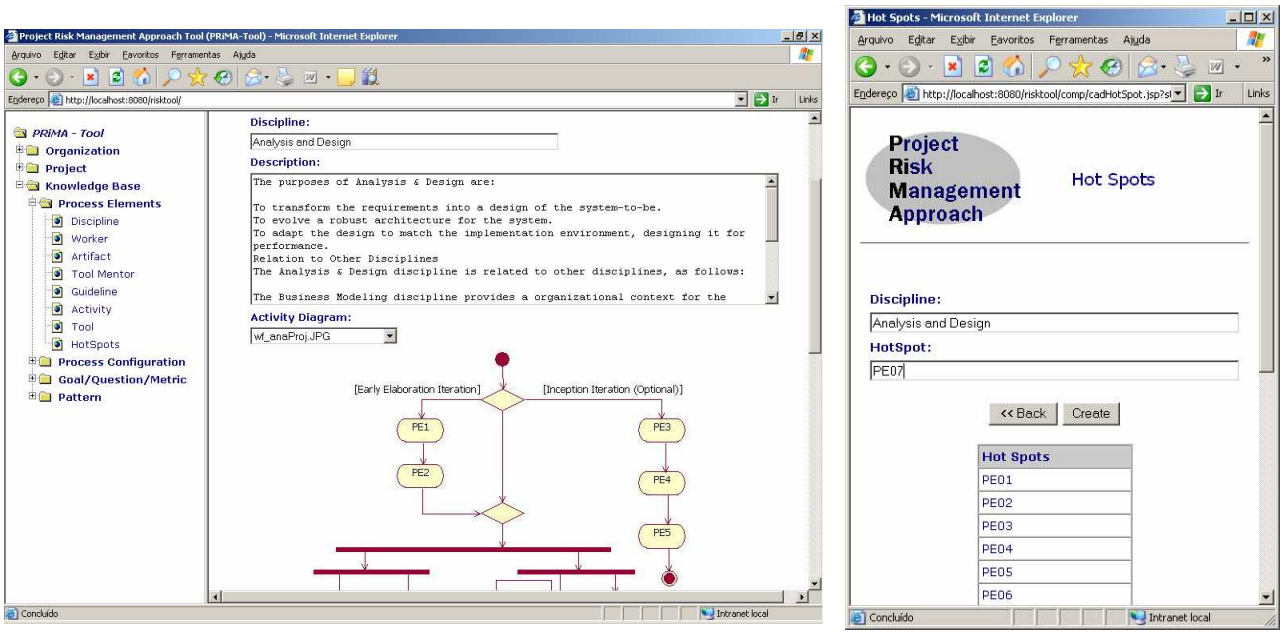


Figura 5.4: Formulário para cadastro de disciplina (esquerda) e inclusão de *hot spots* (direita)

Figura 5.5: Associar atividades aos *hot spots*

Para facilitar a tarefa de definição do processo da organização, no *framework* são definidas configurações de processo típicas. Essas configurações de processo são definidas por meio da opção *Process Configuration*. Utilizando o formulário, exibido na Figura 5.6, o SEPG ou o projetista seleciona as atividades que fazem parte da configuração. As atividades são organizadas por disciplina e dentro de cada disciplina por processo de origem da atividade. No Anexo C são descritas as configurações de processo: RUP Essencial, RUP Completo, XP, RUP CMM Nível 2 e RUP CMM Nível 3.

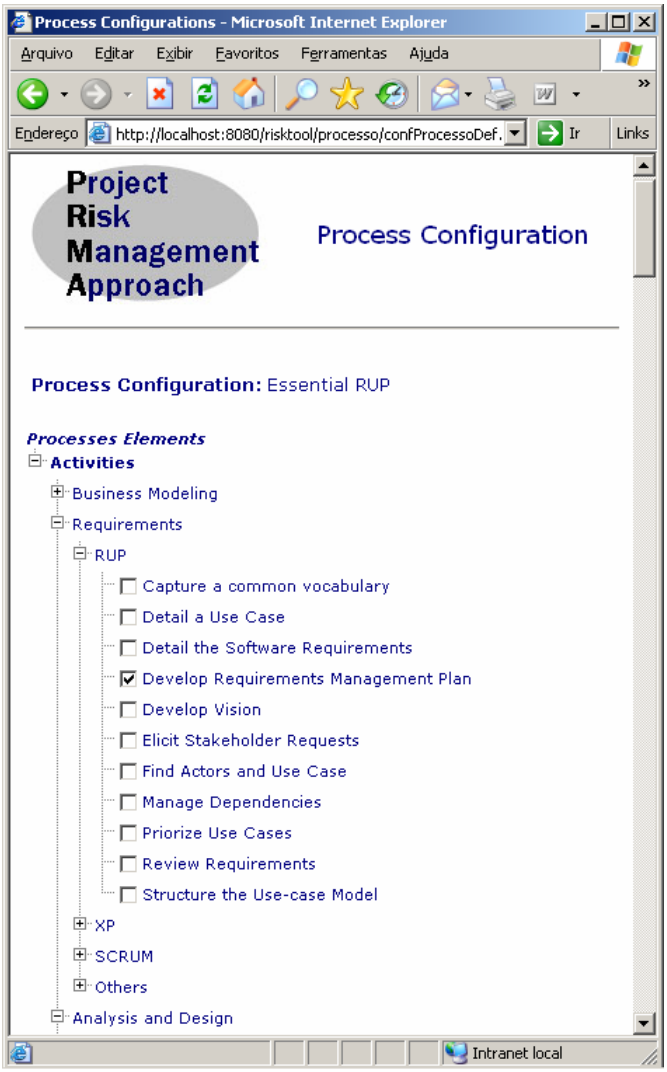


Figura 5.6: Formulário para definir configurações de processo

Guias de auxílio à adaptação do *framework* PRiMA-F, visando atender necessidades específicas da organização ou de projetos, são definidos por meio da opção *Tailoring Guidelines*, como mostra o formulário exibido na Figura 5.7.

Riscos são cadastrados no módulo PMT-Tool. A Figura 5.8 mostra o formulário usado para cadastrar novos riscos. Para cada risco é necessário definir um Plano GQM contendo as metas, as questões e as métricas associadas ao risco.

As métricas são definidas usando o formulário exibido na Figura 5.9. Ao cadastrar a métrica é necessário informar a fórmula usada para calcular o valor da métrica e os valores alvo (*target*) e limite (*threshold*).

As métricas, na abordagem *Goal/Question/Metric* são utilizadas para responder as questões. As questões são cadastradas usando o formulário exibido na Figura 5.10 (esquerda). Ao cadastrar a questão, o usuário deve associar à questão, as métricas usadas para respondê-la, como mostra a Figura 5.10 (direita).

As metas são definidas por meio do formulário exibido na Figura 5.11 (esquerda). As metas são associadas aos riscos e às questões, por meio do formulário exibido na Figura 5.11 (direita).

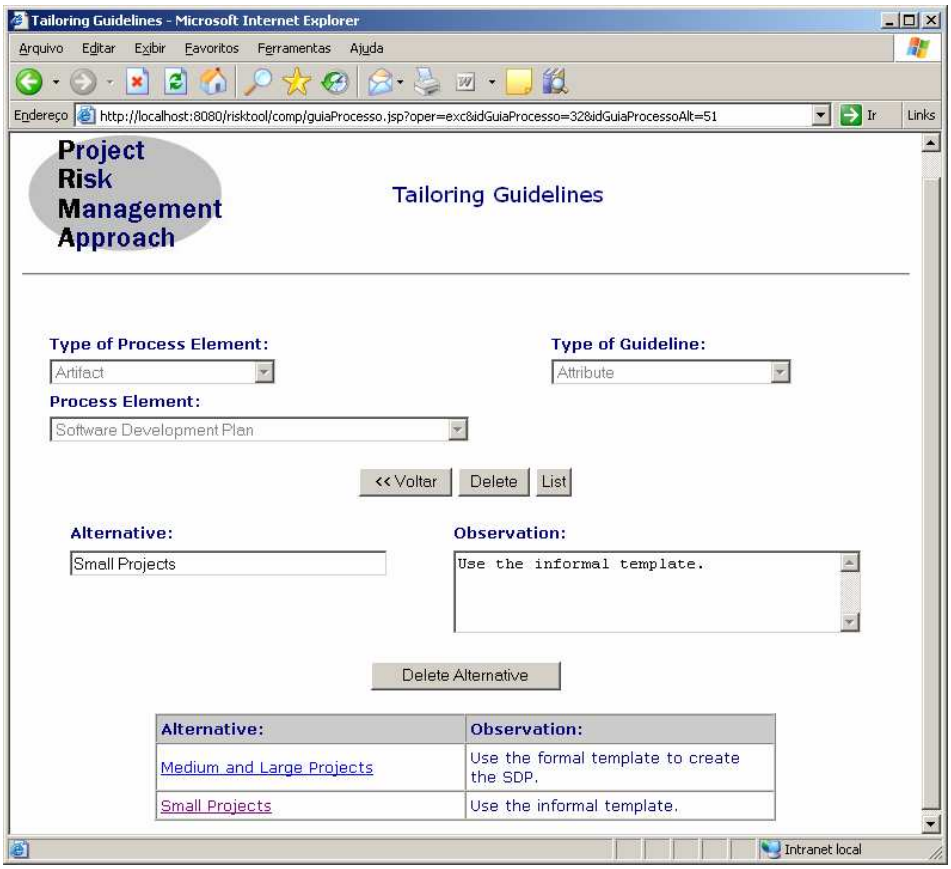


Figura 5.7: Formulário para definir guias de adaptação de processo

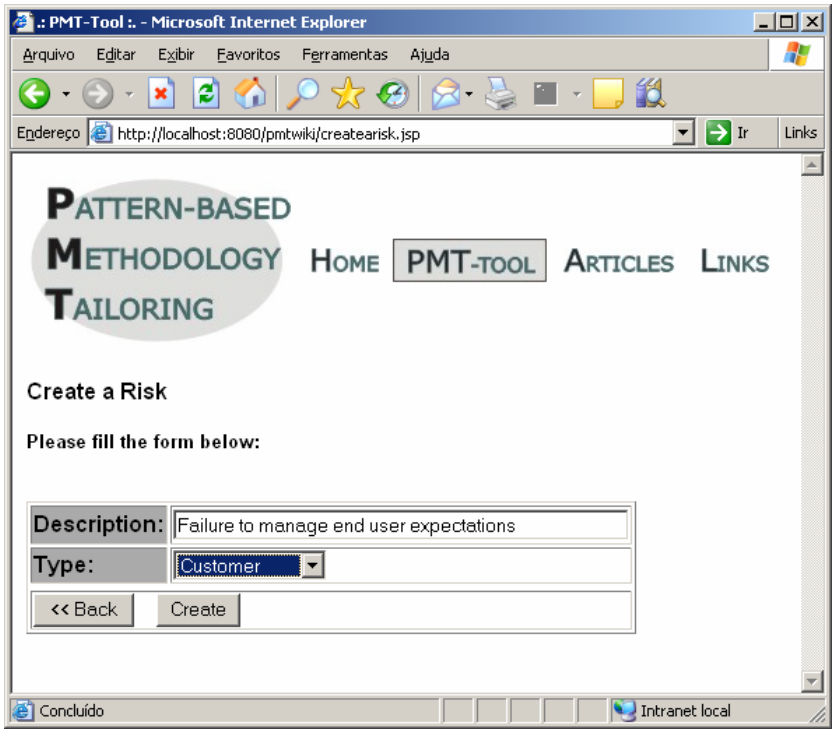


Figura 5.8: Formulário para cadastrar riscos

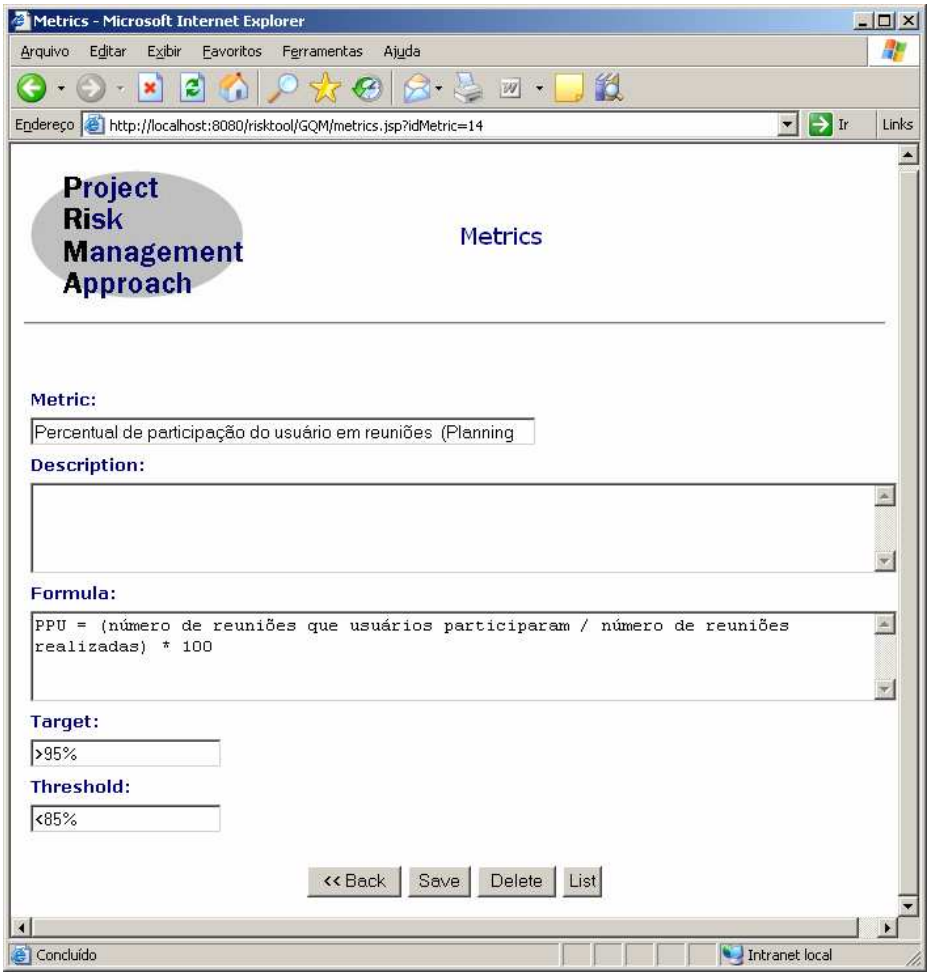


Figura 5.9: Formulário para definir métricas

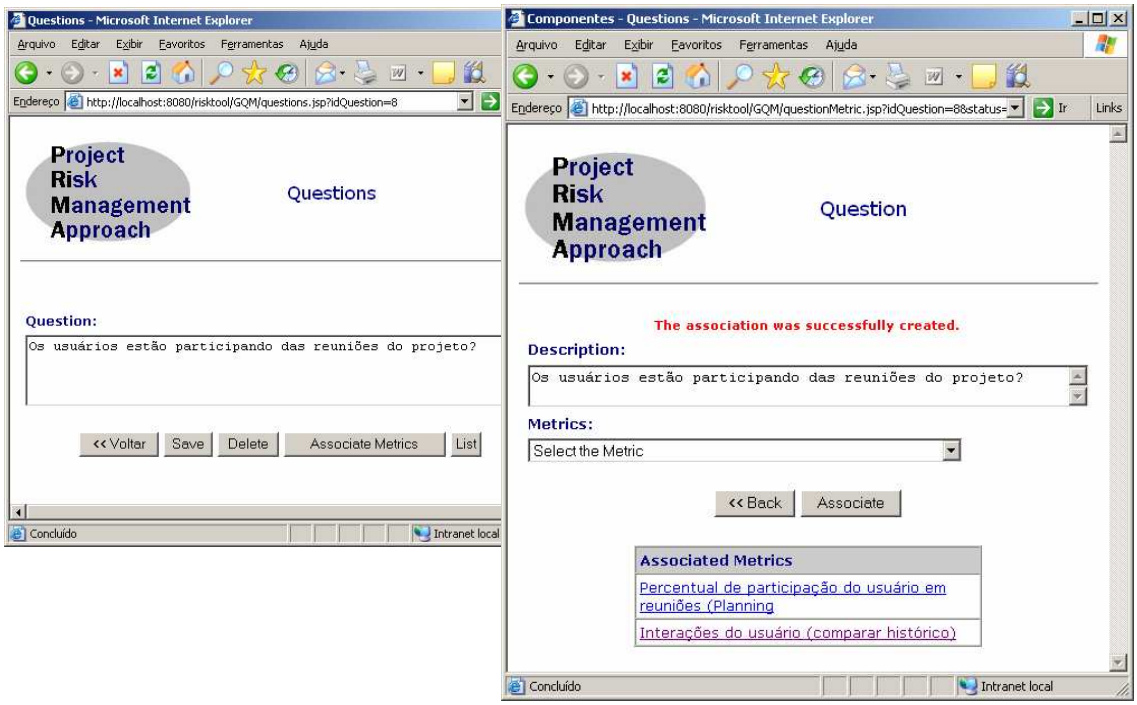


Figura 5.10: Formulários para definir questões (esquerda) e associar questões às métricas (direita)

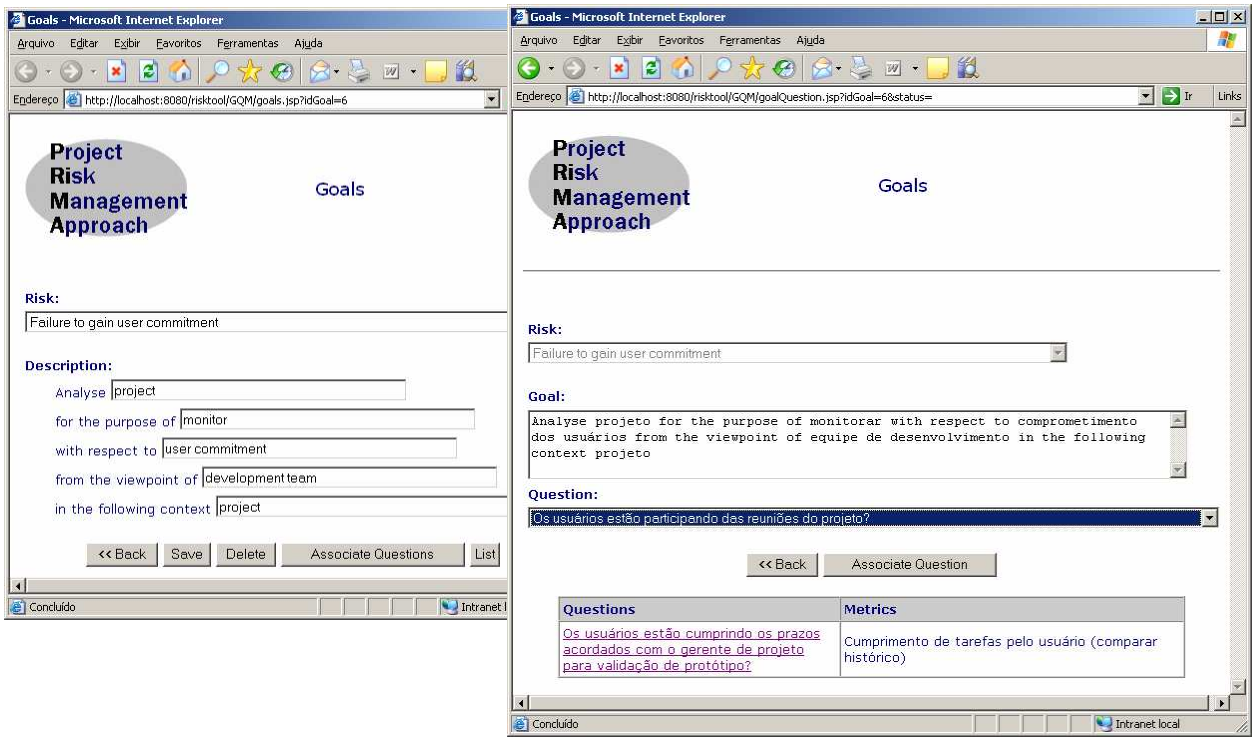


Figura 5.11: Formulário para definir metas (esquerda) e para associar metas a riscos e a questões (direita)

Os padrões são cadastrados por meio de PMT-Tool, a Figura 5.12 mostra o formulário usado para cadastrar padrões.

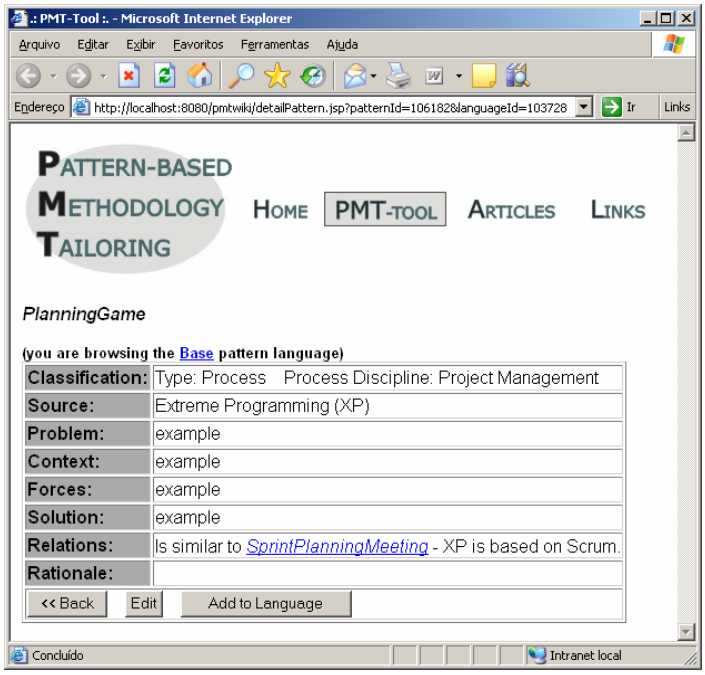


Figura 5.12: Formulário para cadastrar padrões

Os padrões são associados aos riscos por meio de regras. A Figura 5.13 mostra o formulário para definição de regras para associação de riscos a padrões (PMT).

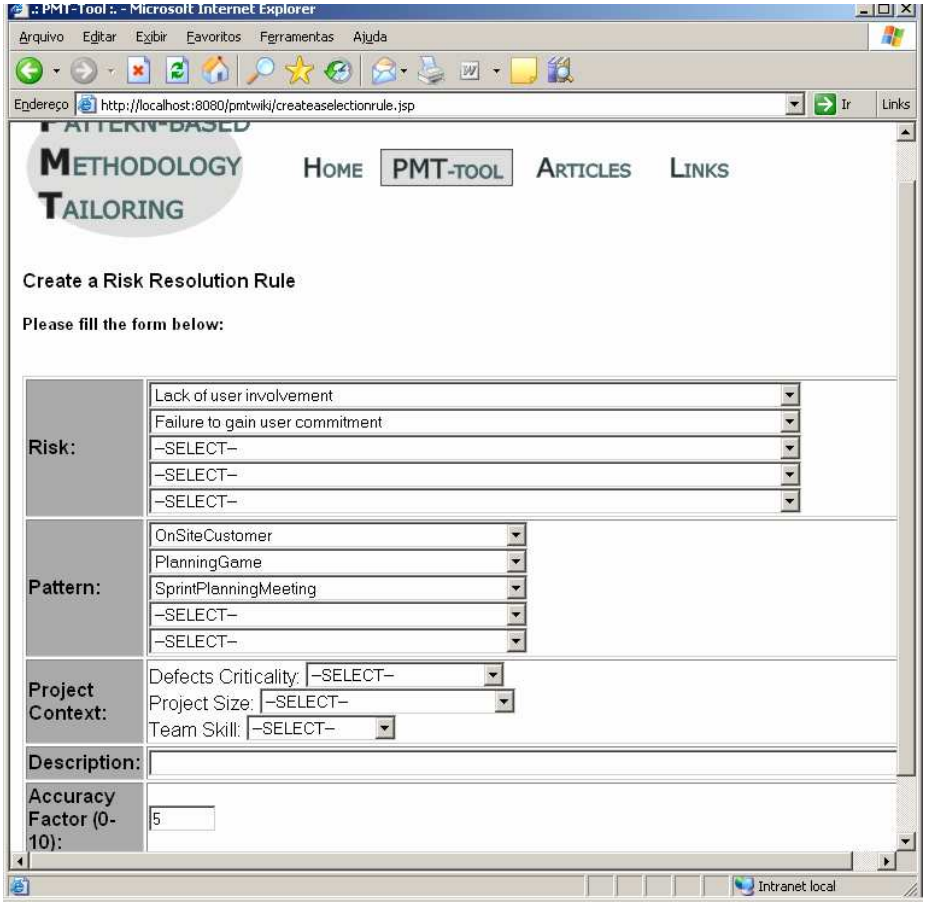


Figura 5.13: Formulário para definir regras para associação de riscos a padrões

É necessário descrever como os padrões de processo e organizacionais podem ser implantados em processos de software. No caso desta tese, os padrões são associados a instâncias de elementos de processo necessários à sua implantação. A Figura 5.14 mostra o formulário utilizado para associar os padrões às atividades necessárias à sua implantação. Ao selecionar as atividades, os demais elementos de processo (artefatos, papéis, disciplinas, modelos, guias, mentores e ferramentas) associados a essa atividade, no *framework* PRiMA-F, são automaticamente selecionados.

Após configurar o ambiente experimental, este pode ser utilizado para adaptar processos de software para projetos de acordo com as informações inseridas na base de conhecimento da organização.

A qualidade da adaptação dos processos quanto à prevenção de riscos e consistência dos modelos gerados vai depender da qualidade das informações e das regras de associação, definidas no *framework* de processo, e posteriormente armazenadas na base de conhecimento de PRiMA-Tool. Sugere-se que os processos utilizados em projetos de software sejam constantemente avaliados e que o resultado dessas avaliações reverta em melhoria no *framework* de processo, e conseqüentemente na base de conhecimento de PRiMA-Tool.

Figura 5.14: Formulário para associar padrões a elementos de processo

5.2.2 Adaptação de Processos

A Figura 5.15 mostra, usando um diagrama de atividades, as atividades necessárias para a adaptação de processos no ambiente experimental. As atividades de gerência de riscos são de responsabilidade do gerente de projetos e as de configuração do processo do SEPG, ou na falta deste do projetista ou engenheiro de processos.

A primeira atividade a ser realizada é a elaboração do processo de software da organização. O processo de software da organização é definido uma única vez, mas deve ser constantemente melhorado e otimizado. A existência de um processo definido para a organização é um pré-requisito para a adaptação do processo para um projeto, já que o processo para um projeto é obtido a partir do processo da organização.

O formulário exibido na Figura 5.16 (esquerda), possibilita incluir uma nova organização. O processo para a organização pode ser definido manualmente, isto é, as atividades são selecionadas uma a uma; ou a partir de configurações de processo. No caso de se utilizar configurações de processo pré-definidas, as atividades são selecionadas automaticamente de acordo com a configuração escolhida. A Figura 5.16 (direita) mostra a elaboração de um processo para uma organização a partir da configuração de processo *Essential RUP*.

O processo da organização pode ser modificado ou elaborado manualmente por meio do formulário exibido na Figura 5.17.

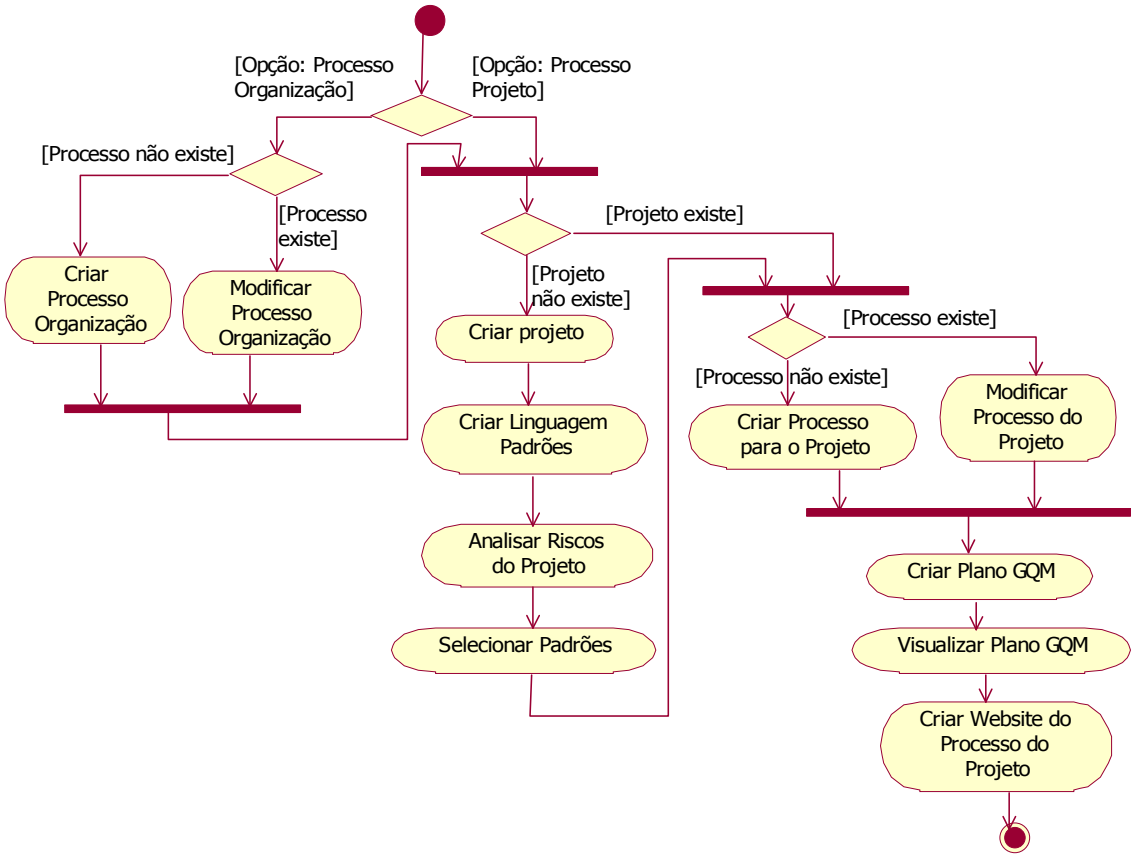


Figura 5.15: Atividades para adaptação de processos no ambiente experimental

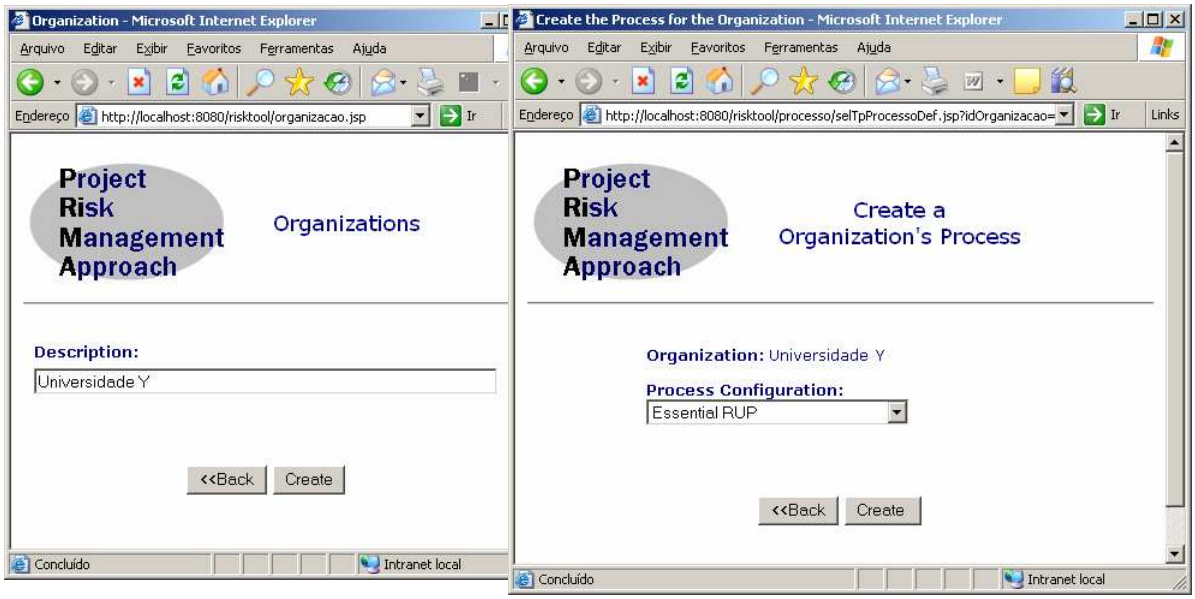


Figura 5.16: Formulário para inserir organização (esquerda) e para elaborar processo para a organização (direita)

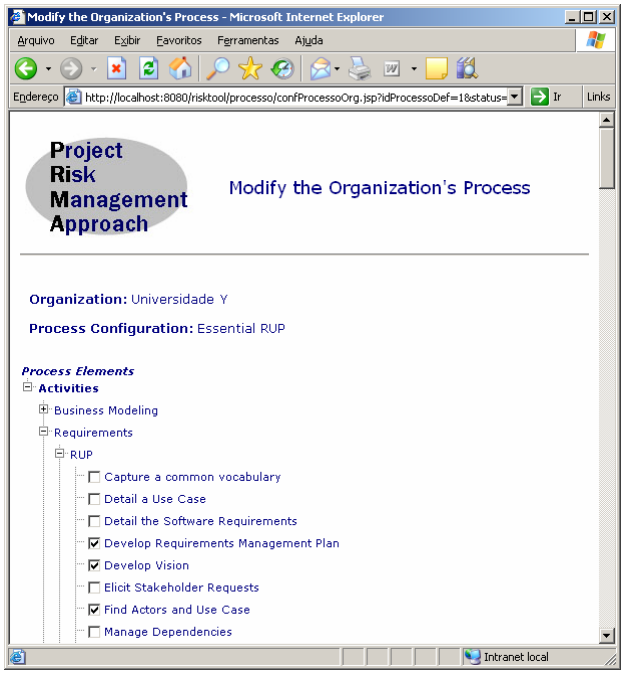


Figura 5.17: Configurar processo para a organização manualmente

Após a definição do processo da organização, os processos para projetos específicos dessa organização podem ser elaborados. O formulário, exibido na Figura 5.18 (esquerda), é utilizado para cadastrar o projeto de software. O próximo passo é criar uma linguagem de padrões, onde serão inseridos os padrões selecionados para prevenir os riscos. A Figura 5.18 (direita) mostra o formulário para criação da linguagem de padrões.

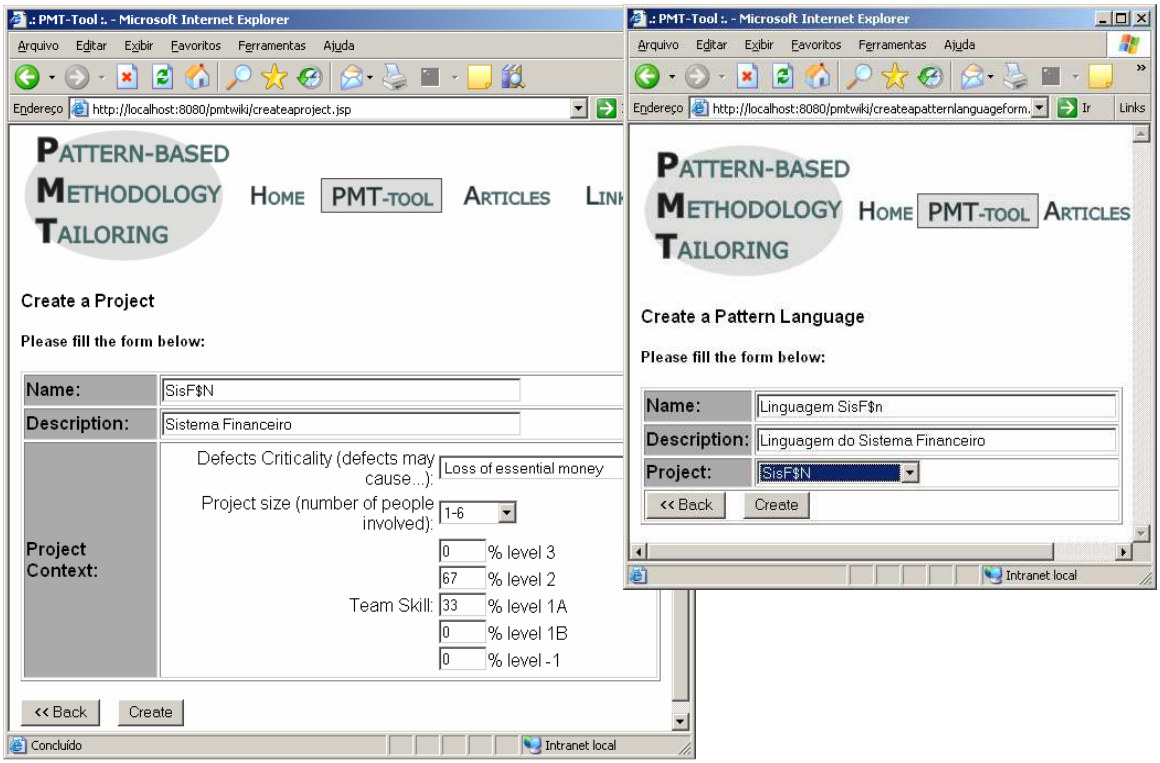


Figura 5.18: Formulário para cadastrar projetos (esquerda) e para criar linguagem de padrões para um projeto (direita)

A análise de exposição dos riscos é elaborada por meio do formulário de PMT-Tool, exibido na Figura 5.19. O gerente de projeto informa a probabilidade e a perda associada a cada risco e o fator de priorização de risco (RPF – *Risk Priorization Factor*). PMT-Tool calcula o percentual de exposição de riscos (ER) e prioriza os riscos com ER igual ao superior ao RPF informado pelo gerente.

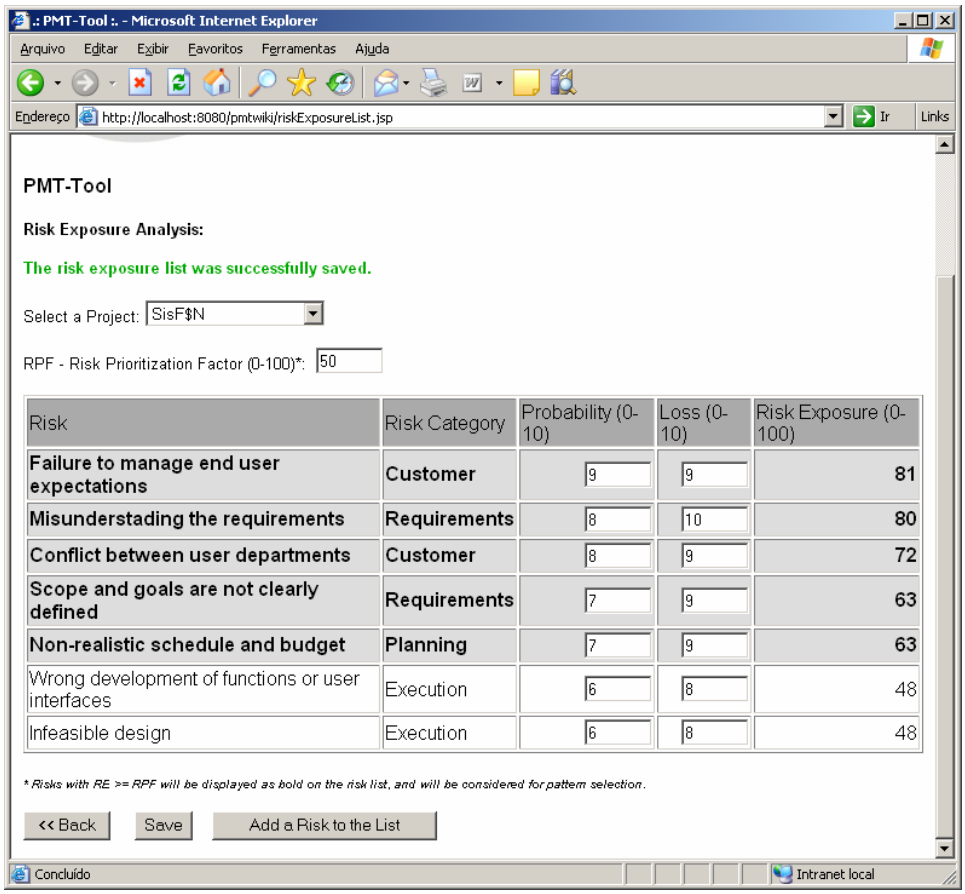


Figura 5.19: Análise dos riscos para o projeto

Após a priorização dos riscos do projeto, usando o formulário exibido na Figura 5.20, são selecionados da base de conhecimento, um conjunto de padrões, sugeridos por PMT-Tool, para prevenção dos riscos identificados para o projeto. O projetista seleciona os padrões que deseja inserir na linguagem de padrões do projeto.

O próximo passo é elaborar o processo para o projeto (Figura 5.21), informando a organização e o tipo do processo base. O processo para o projeto é elaborado automaticamente por PRiMA-Tool, inserindo no processo da organização as atividades necessárias para implantar os padrões selecionados para prevenção de riscos.

É possível modificar o processo definido para o projeto, inserindo neste outras atividades para atender necessidades específicas do projeto, tais como: tecnologias, clientes, etc. Essa modificação pode ser realizada por meio do formulário exibido na Figura 5.22.

A elaboração do Plano GQM, para gerenciamento dos riscos do projeto, é realizada por meio do formulário exibido na Figura 5.23. Nesse formulário são exibidas as metas, as questões e as métricas, previamente definidas no *framework,* para que o gerente de projeto selecione as métricas que deseja utilizar em determinado projeto. Ao selecionar a métrica são automaticamente selecionadas as questões e metas associada à métrica.

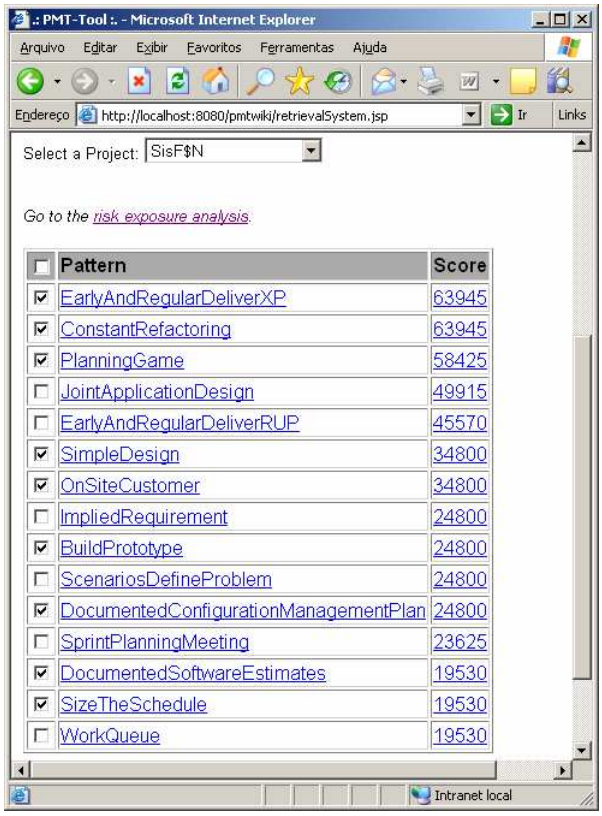


Figura 5.20: Formulário para seleção de padrões para prevenção de riscos do projeto

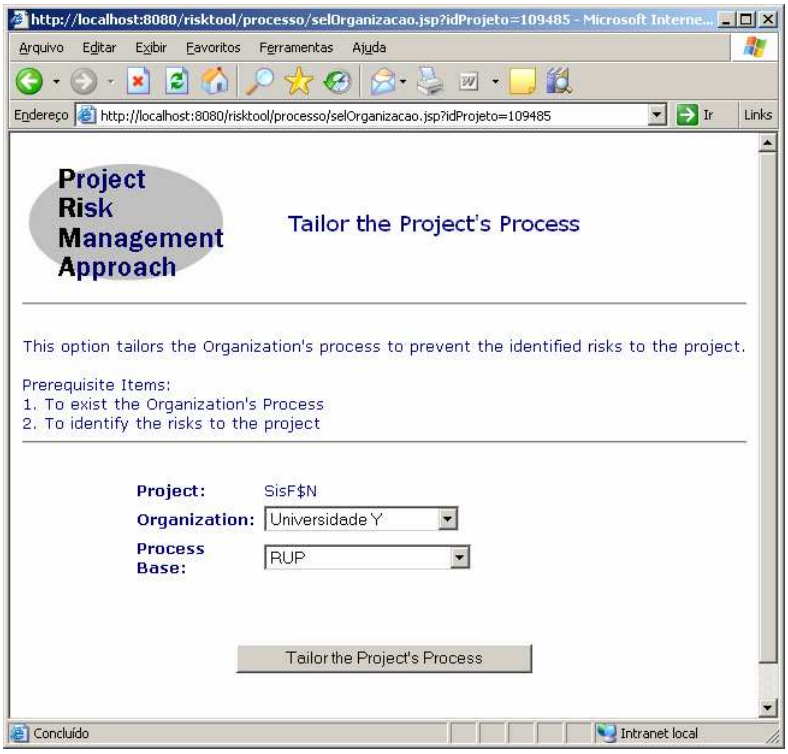


Figura 5.21: Formulário para elaborar o processo para o projeto

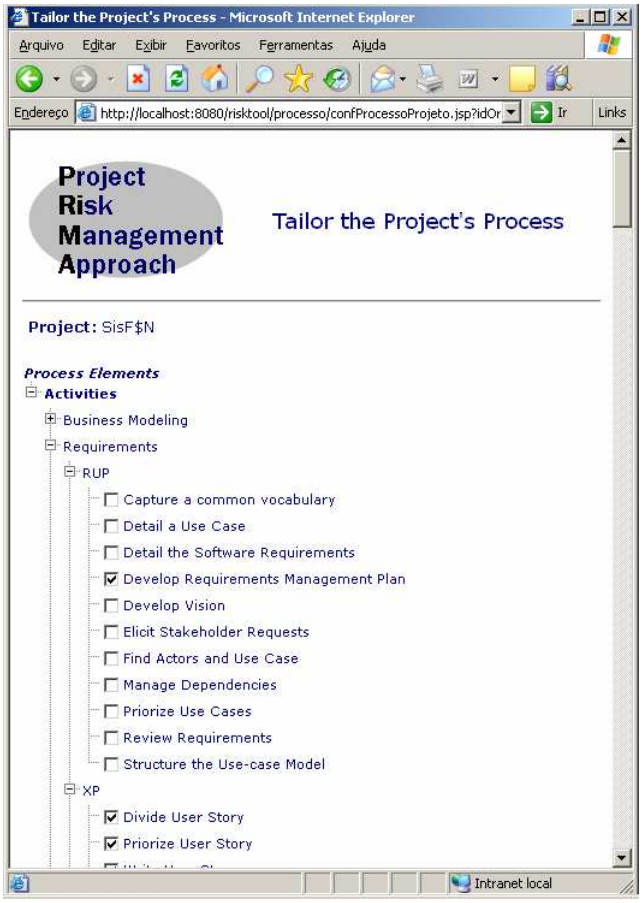


Figura 5.22: Formulário para modificação do processo do projeto

Após a elaboração do Plano GQM, é possível visualizar as métricas, metas e questões selecionadas por meio do formulário exibido na Figura 5.24. Neste formulário são também exibidos os padrões de processo e organizacionais selecionados para prevenir os riscos priorizados para o projeto.

O projetista pode gerar um *website* com a descrição do processo definido e do Plano GQM. Para isso, é necessário selecionar a opção *Create a Website*. O *site* descreve os elementos de processo instanciados para o projeto e as associações entre eles. A Figura 5.25 mostra, como um exemplo, um *website* gerado por PRiMA-Tool.

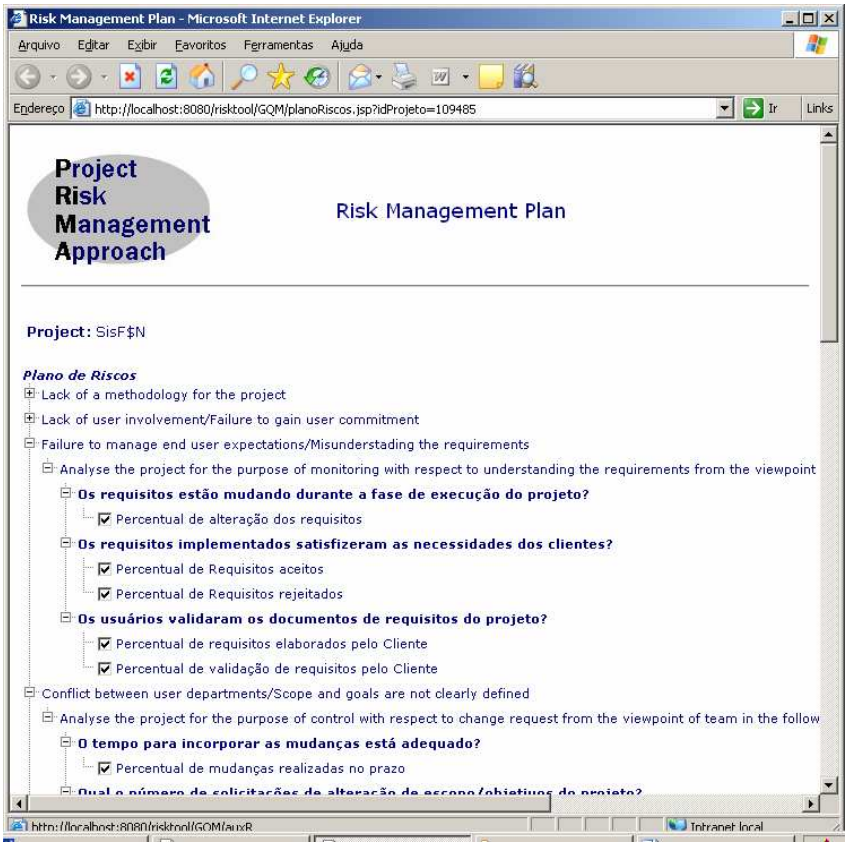


Figura 5.23: Formulário para elaborar Plano GQM

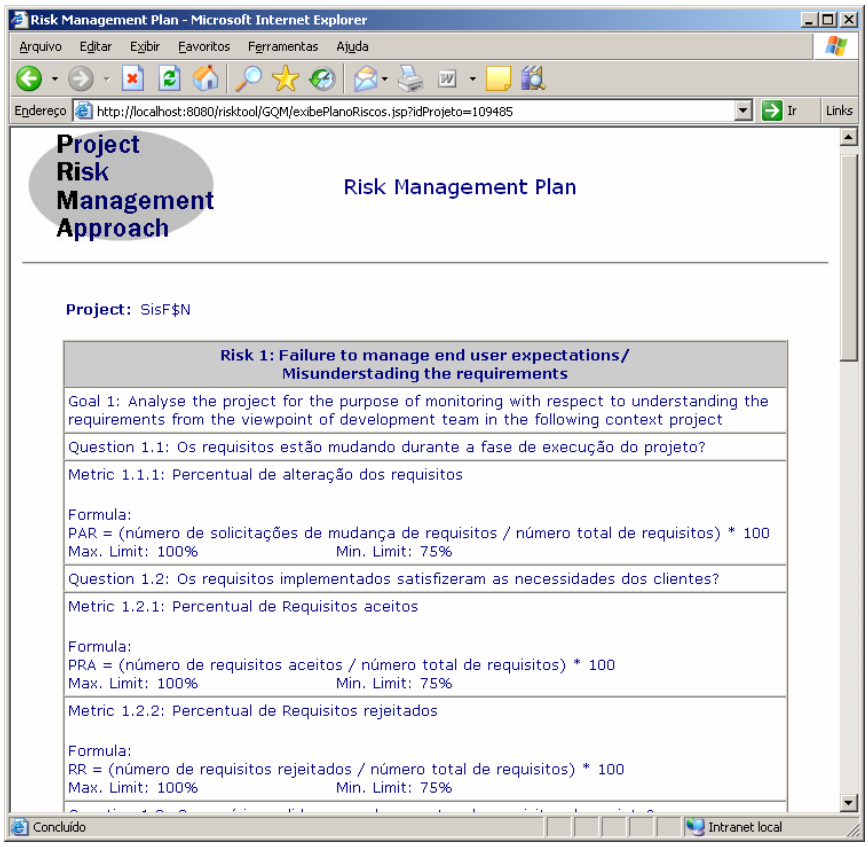


Figura 5.24: Formulário para exibir Plano GQM

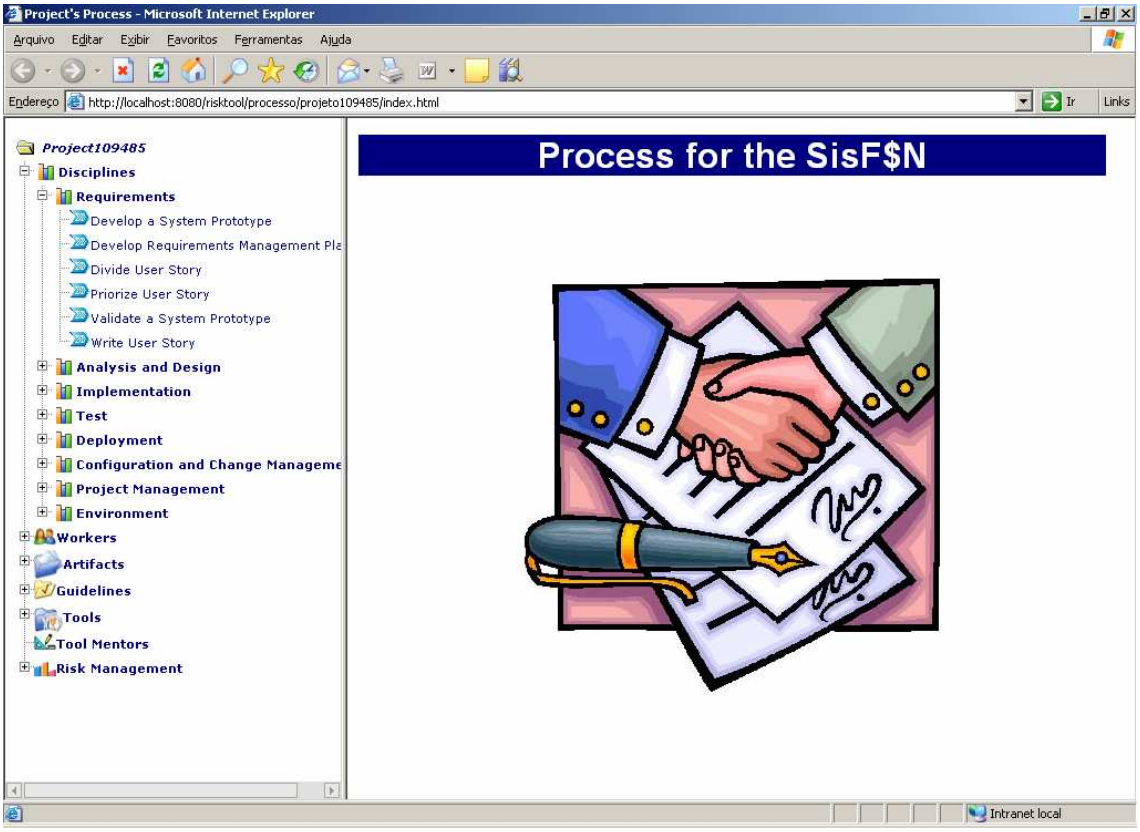


Figura 5.25: *Website* elaborado para projeto

6 ESTUDOS DE CASO

Este capítulo descreve dois estudos de caso realizados para validar a sistemática apresentada. Inicialmente será descrito o *framework* PRiMA-F, elaborado a partir dos processos RUP e XP e do conjunto de riscos descritos na seção 4.1 desta tese. Algumas configurações de processo são elaboradas para facilitar a tarefa de adaptação de processos. O estudo de caso foi realizado com o auxílio do ambiente experimental apresentado no capítulo anterior.

Os estudos de caso são elaborados considerando dois projetos de uma universidade, que será chamada de Universidade Y. Os projetos apresentam características bem específicas, sendo que, um dos projetos visa desenvolver um sistema financeiro em um dos câmpus da Universidade Y, e outro projeto visa desenvolver um sistema acadêmico de forma distribuída, por equipes localizadas em três diferentes câmpus da Universidade.

6.1 Elaboração do *Framework* PRiMA-F

PRiMA-F foi elaborado a partir dos processos *Rational Unified Process* (RATIONAL, 2003) e *Extreme Programming* e de atividades necessárias para implantar padrões associados a riscos em processos de software. O *framework* PRiMA-F foi elaborado a partir das seguintes etapas:

- Definição de elementos de processos: os elementos de processo descritos pelos processos RUP e XP foram definidos usando os conceitos descritos pelo metamodelo integrado (Anexo A);
- Elaboração de diagramas de atividades para cada disciplina de processo, descrevendo os aspectos dinâmicos do processo e a associação entre as atividades e os *hot spots* (Anexo B);
- Elaboração de exemplos de configurações de processo (Anexo C);
- Identificação de padrões para prevenção dos riscos de projeto citados na lista de riscos (Seção 4.3);
- Descrição dos padrões segundo conceitos propostos pelo metamodelo integrado (Anexo D);
- Definição do Plano GQM para os riscos descritos citados na lista de riscos (Seção 4.3).

6.2 Elaboração do Processo para a Organização

6.2.1 Descrição da Organização

O estudo de caso foi realizado em uma universidade que possui uma estrutura multicâmpus, com câmpus localizados em diferentes cidades. Alguns sistemas da Universidade são desenvolvidos localmente e outros são desenvolvidos de forma distribuída, por mais de uma equipe. As equipes desenvolvem os sistemas internos da Universidade, tais como: sistema acadêmico, sistema financeiro, sistema da biblioteca, sistema de almoxarifado, etc.

6.2.2 Processo de Software da Organização

O processo de software da Universidade Y é bastante simples, e se baseia em um pequeno conjunto de atividades do RUP. O processo da organização foi elaborado a partir da configuração de processo RUP Essencial (Seção 4.4.2.2), tendo sido adicionadas a essa configuração as atividades *Class Design e Database Design*, como mostra a Tabela 6.1.

Tabela 6.1: Processo de software padrão da Universidade Y

Discipline	***Activity***
Requirements	*Develop Requirements Management Plan*
Analysis and Design	*Architectural Analysis*
	Class Design
	Database Design
Implementation	*Execute Developer Tests*
	Implement Design Elements
Test	*Define Test Approach*
	Define Test Details
Deployment	*Create Deployment Unit*
	Develop Support Materials
Configuration and Change Management	*Confirm Duplicate or Rejected CR*
	Review Change Request
	Submit Change Request
	Update Change Request
Project Management	*Compile Software Development Plan*
	Develop Business Case
	Develop Iteration Plan
	Identify and Assess Risks
	Initiate Project
	Iteration Acceptance Review

	Lifecycle Milestone Review
	Prepare for Phase Close-out
	Prepare for Project Close-out
	Project Acceptance Review
	Report Status
Environment	*Select and Acquire Tools*
	Set Up Tools
	Tailor the Process for the Project

6.3 Sistema Financeiro

6.3.1 Descrição do Contexto

O sistema financeiro da Universidade Y constantemente apresentava problemas, tais como: erros de execução, resultados incorretos, necessidade de ajustes, novas definições, etc. Devido aos freqüentes erros no sistema, os funcionários da tesouraria utilizavam controles paralelos ao sistema, pois consideravam que este não era confiável. A equipe, diariamente, corrigia *bugs* no sistema existente. A maioria dos problemas relatados se enquadrava em: requisitos mal-definidos, erros de implementação, má qualidade do código existente, por isso as manutenções corretivas acabavam por gerar outros *bugs*.

O coordenador do Núcleo de Informática, apoiado pela direção, resolveu implementar um novo sistema financeiro. A equipe formada para desenvolver o sistema é composta por um gerente de projeto, dois desenvolvedores, dois estagiários e o coordenador do Núcleo de Informática, que atua como gerente sênior. A equipe é classificada como de alta habilidade, pois os desenvolvedores já são experientes na linguagem e já desenvolveram sistemas semelhantes.

6.3.2 Riscos Identificados

A Tabela 6.2 mostra os riscos selecionados para o projeto do sistema financeiro (SisF$n) e os percentuais de probabilidade e perda definidos para cada risco. O fator de priorização de risco informado foi de 50%, ocasionando a priorização dos riscos destacados na Tabela 6.2.

Tabela 6.2: Riscos identificados para o projeto SisF$n

Risco	Categoria	Probabilidade	Perda	ER
Falha na gerência de expectativas dos usuários	Cliente	9	9	81
Não-entendimento dos requisitos	Requisitos	8	10	80
Conflito entre os departamentos do usuário	Cliente	8	9	72
Escopo/objetivos não claros	Requisitos	7	9	63
Cronograma e orçamento não-realistas	Planejamento	7	9	63
Desenvolvimento errado das funções ou interface	Execução	6	8	48
Projeto (desenho) inviável	Execução	6	8	48

Os riscos selecionados estão relacionados a problemas de definição de requisitos de software, pois se identificou a existência de regras de negócio pouco claras e com muitos tratamentos de exceção. Os requisitos são complexos e os departamentos de contabilidade e tesouraria têm visões e expectativas diferentes quanto ao sistema a ser desenvolvido. Essa incerteza quanto aos requisitos, ocasiona dificuldades para estimar prazos e custos do projeto.

6.3.3 Padrões Selecionados

A Tabela 6.3 mostra a lista de padrões selecionados por PMT-Tool, ordenados por relevância de acordo com o contexto do projeto. Identifica-se, na lista de padrões sugeridos pelo mecanismo de seleção, que alguns padrões são provenientes de processos planejados (RUP), mas os primeiros padrões da lista, com relevância mais elevada, são oriundos de processos ágeis (XP, Scrum). Esse fato ocorre porque o contexto descrito para o projeto é de uma equipe pequena com alta habilidade e a criticidade do sistema é média (perda de dinheiro). Alguns padrões sugeridos são equivalentes, como o *EarlyAndRegularDeliverXP* e o *EarlyAndRegularDeliverRUP*. Neste caso, o projetista deve apenas selecionar um dos padrões. Os padrões selecionados, pelo projetista, para serem adicionados a linguagem de padrões do projeto estão marcados com ✓.

Tabela 6.3: Lista de padrões sugeridos por PMT-Tool para SisF$n

	Pattern
✓	*EarlyAndRegularDeliverXP*
✓	*ConstantRefactoring*
✓	*PlanningGame*
	JointApplicationDesign
	EarlyAndRegularDeliverRUP
✓	*SimpleDesign*
✓	*OnSiteCustomer*
	ImpliedRequirement
✓	*BuildPrototype*
	ScenariosDefineProblem
✓	*DocumentedConfigurationManagementPlan*
	SprintPlanningMeeting
✓	*DocumentedSoftwareEstimates*
✓	*SizeTheSchedule*
	WorkQueue

6.3.4 Processo Específico para o Projeto

O processo específico para o projeto SisF$n é gerado por adicionar as atividades usadas para implantar padrões no projeto padrão da organização. A Tabela 6.4 mostra as atividades que compõem o processo elaborado para o projeto SisF$n. As atividades destacadas são aquelas específicas do processo do projeto, e as demais são as atividades descritas pelo processo da organização. No Anexo D são descritos os padrões segundo os elementos necessários à sua implantação.

Tabela 6.4: Atividades do processo específico para SisF$n

Discipline	*Activity*
Requirements	*Develop Requirements Management Plan*
	Write User Story
	Divide User Story
	Priorize User Story
	Develop a System Prototype
	Validate a System Prototype
Analysis and Design	*Architectural Analysis*
	Class Design
	Database Design
	Write Tasks
Implementation	*Execute Developer Tests*
	Implement Design Elements
	Execute Unit Tests
	Implement Tasks
	Refactor Code
Test	*Define Test Approach*
	Define Test Details
	Execute Acceptance Tests
	Write Acceptance Tests
	Write Unit Tests
Deployment	*Create Deployment Unit*
	Develop Support Materials
Configuration and Change Management	*Confirm Duplicate or Rejected CR*
	Review Change Request
	Submit Change Request
	Update Change Request
	Establish Change Control Process
	Write CM Plan

Project Management	*Compile Software Development Plan*
	Develop Business Case
	Develop Iteration Plan
	Identify and Assess Risks
	Initiate Project
	Iteration Acceptance Review
	Lifecycle Milestone Review
	Prepare for Phase Close-out
	Prepare for Project Close-out
	Project Acceptance Review
	Report Status
	Accept Task
	Classify by Risk
	Collect Metrics
	Define Iteration Scope
	Define Monitoring and Control Processes
	Define the Scheduler
	Define Velocity
	Develop Measurement Plan
	Elaborate Release Plan
	Estimate Task
	Estimate User Story
	Monitor Status Project
	Negotiate the Scheduler with Team
	Negotiate with Customer
Environment	*Select and Acquire Tools*
	Set Up Tools
	Tailor the Process for the Project

O processo específico para o projeto apresenta características de métodos ágeis. A definição de requisitos é realizada por meio de *User Stories*, que são desdobradas em *Tasks* na disciplina de *Analysis and Design*. Devido a falta de clareza dos requisitos, foram incluídas atividades para desenvolvimento e validação de protótipo (*Develop a System Prototype* e *Validate a System Prototype*). Nas disciplinas *Implementation* e *Test* foram adicionadas atividades relacionadas a testes unitários e testes de aceitação. Considerando o risco instabilidade de requisitos, foram inseridas as atividades estabelecer processos de mudanças (*Establish Change Control Process)* e elaborar um plano de gerência de configuração (*Write a Configuration Management Plan*), na

disciplina *Configuration and Change Management*. Na disciplina *Project Management*, foram adicionadas atividades para implantar as práticas do *Planning Game*, que propõem a gerencia com base em *User Stories* e o desenvolvimento organizado em iterações.

Algumas atividades adicionadas ao processo específico para o projeto são equivalentes, isto é, tem objetivos comuns, não justificando serem ambas realizadas. Por exemplo, as atividades *Define the Scheduler, Elaborate Release Plan* e *Develop Iteration Plan* têm como objetivo elaborar o plano de projeto para a iteração. A tarefa de ajustar o processo definido, após a adaptação do processo, cabe ao projetista.

No processo definido para o SisF$n foram excluídas as atividades: *Define the Scheduler* e *Develop Iteration Plan* (equivalentes a *Elaborate Release Plan*); *Execute Developer Tests* (equivalente a *Execute Unit Tests*); *Monitorar Project Status* (equivalente a *Collect Metrics).* Essas equivalências podem ser descritas em guias de adaptação para auxiliar o projetista.

6.3.5 Plano de Gerência de Riscos

O Plano de Gerenciamento de Riscos (Plano GQM) foi elaborado por selecionar todas as métricas, sugeridas no *framework* PRiMA-F, associadas aos riscos priorizados para o projeto SisF$n, como mostra a Tabela 6.5.

Tabela 6.5: Plano GQM para SisF$n

Risco 1: Falha na gerência de expectativas dos usuários, Não-entendimento dos requisitos	
Meta 1: Analisar o projeto com a finalidade de monitorar com respeito à definição dos requisitos do ponto de vista da equipe de desenvolvimento.	
Questão 1.1: Os usuários validaram os documentos de requisitos do projeto?	*Métrica 1.1.1:* Percentual de validação de requisitos pelo Cliente PVRC = (número de documentos de requisitos validados/ número total de documentos de requisitos) * 100 *Métrica 1.1.2:* Percentual de requisitos elaborados pelo Cliente PRE = (número de requisitos elaborados pelos usuários/requisitos elaborados pelo pessoal de sistema) * 100
Questão 1.2: Os requisitos estão mudando durante a fase de execução do projeto?	*Métrica 1.2.1:* Percentual de alteração dos requisitos PAR = (número de solicitações de mudança de requisitos / número total de requisitos) * 100
Questão 1.3: Os requisitos implementados satisfizeram as necessidades dos clientes?	*Métrica 1.3.1:* Métrica 3.3a: Requisitos aceitos RA = (número de requisitos aceitos / número total de requisitos) * 100 *Métrica 1.3.2:* Requisitos rejeitados RR = (número de requisitos rejeitados / número total de requisitos) * 100

Risco 2: Conflito entre os departamentos do usuário, Escopo/objetivos não claros	
Meta 2: Analisar o projeto com a finalidade de controlar com respeito às solicitações de mudança do ponto de vista da equipe de desenvolvimento.	
Questão 2.1: Qual o número de solicitações de alteração de escopo/objetivos do projeto?	*Métrica 2.1:* Número de solicitações de mudança (comparar com dados históricos) NSM = Número de solicitações de mudança de escopo/objetivos
Questão 2.2: O tempo para incorporar as mudanças está adequado?	*Métrica 2.2:* 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
Risco 3: Cronograma e orçamento não-realistas	
Meta 3: Analisar o projeto com a finalidade de monitorar com respeito as estimativas do ponto de vista do gerente de projeto.	
Questão 3.1: Qual a precisão das estimativas?	*Métrica 3.1.1:* Percentual de casos de uso desenvolvidos iteração CUDI = (numero de casos de uso assinalados no inicio da iteração/casos de uso concluídos na interação) * 100 *Métrica 3.1.2:* Precisão das estimativas de esforço PEE = (esforço real da iteração/ esforço estimado da iteração) *100 *Métrica 3.1.3:* Precisão das estimativas de custo PEC = (custo real da iteração/ custo estimado da iteração)*100

6.4 Sistema Acadêmico

6.4.1 Descrição do Contexto

Cada câmpus da Universidade Y possuía um sistema acadêmico, o que tornava difícil a gerência dessas informações pela Reitoria. A reitoria da Universidade Y resolveu desenvolver um novo sistema acadêmico que será utilizado por todos os câmpus da Universidade. O sistema será desenvolvido para ambiente *Web*, instalado em um servidor na reitoria, e acessado pelos câmpus, por meio da Internet. A Reitoria tem como objetivo elaborar um sistema acadêmico que controle todos os procedimentos acadêmicos e auxilie na gerência de informações dos cursos de diferentes câmpus, estabelecendo comparativos de qualidade entre os cursos e um acompanhamento completo dos alunos, professores, disciplinas ofertadas, cursos, entre outros.

Como cada campus da Universidade possui uma equipe de desenvolvimento, a Reitoria decidiu desenvolver esse sistema de forma distribuída. Inicialmente foram definidos os requisitos do sistema, com analistas de requisitos de cada equipe e, posteriormente, os casos de uso elencados foram divididos por equipe. A gerência global do projeto cabe a Reitoria, e internamente cada equipe possui um gerente de projeto.

O sistema acadêmico (Sis@cad) foi desenvolvido por três equipes, totalizando 22 pessoas, assim distribuídas: 3 gerentes de projeto, 8 desenvolvedores, 10 estagiários e 1 gerente sênior, localizado na reitoria.

6.4.2 Riscos Identificados

A Tabela 6.6 mostra os riscos selecionados para o projeto Sis@cad e os percentuais de probabilidade e perda informados pelo gerente. O fator de priorização de risco informado foi de 50%, ocasionando a priorização dos riscos destacados da tabela.

Tabela 6.6: Riscos identificados para Sis@cad

Risco	Categoria	Probabilidade	Perda	ER
Falta de uma metodologia de Projeto	Planejamento	8	9	72
Falta de conhecimento/ habilidade da equipe de projeto	Planejamento	8	8	64
Não-entendimento dos requisitos	Requisitos	7	9	63
Introdução de novas tecnologias	Execução	7	8	56
Desenvolvimento errado das funções ou interface	Execução	7	8	56
Projeto (desenho) inviável	Execução	8	7	56
Falta de comprometimento da gerência sênior com o projeto	Planejamento	6	9	54
Acréscimos de funcionalidades idealizadas pela equipe sem o aval do cliente	Execução	6	8	48
Falta de envolvimento do usuário	Cliente	6	8	48
Conflito entre os departamentos do usuário	Cliente	6	8	48

Os riscos selecionados estão relacionados ao desenvolvimento distribuído e a utilização de novas tecnologias necessárias para desenvolvimento do sistema *web*, com uma boa performance. A equipe não tem conhecimento no desenvolvimento de sistemas *web* e nem no desenvolvimento distribuído. Cada equipe utiliza uma metodologia para desenvolvimento de software, mas nenhuma das equipes possui uma metodologia documentada. A gerência sênior desempenha um papel importante no controle global do desenvolvimento de Sis@cad, sendo um risco a falta de comprometimento da mesma.

6.4.3 Padrões Selecionados

A Tabela 6.7 mostra a lista de padrões selecionados por PMT-Tool, ordenados por relevância de acordo com o contexto do projeto. Identifica-se que os padrões sugeridos com maior relevância são propostos por métodos planejados, devido ao fato da equipe ser de mais de 20 pessoas e de habilidade baixa, devido ao grande número de estagiários e ao desconhecimento da tecnologia. A criticidade do projeto envolve perda de dinheiro, mas não caracteriza o projeto como de alto risco. Os oito últimos padrões são

provenientes de metodologias ágeis, o que indica que estes são pouco relevantes para o projeto em questão, não justificando sua seleção pelo projetista. Os padrões selecionados para serem adicionados a linguagem de padrões do projeto estão marcados com ✓.

Tabela 6.7: Lista de padrões sugeridos por PMT-Tool para Sis@cad

	Pattern
✓	*ConstantRefactoring*
	SimpleDesign
✓	*SoftwareLifeCycleIsDefined*
✓	*PeerReviews*
✓	*ArchitectureTeam*
	AcceptanceTests
	PairProgramming
✓	*ProjectProcessIsDefined*
✓	*ApprenticeShip*
	DayCare
	GenericsAndSpecifics
✓	*DocumentedConfigurationManagementPlan*
	ImpliedRequirement
	BuildPrototype
✓	*EarlyAndRegularDeliverRUP*
✓	*ScenariosDefineProblem*
✓	*ShunkWorks*
✓	*SeniorManagementReview*
	DefinedProcess
	PlanningGame
	OnSiteCustomer
	EarlyAndRegularDeliverXP
	SystemMetaphor
	SpikeSolutions
	IterationPlanning
	SprintPlanningMeeting

6.4.4 Processo Específico para o Projeto

O processo específico para o projeto Sis@cad é gerado por adicionar as atividades usadas para implantar padrões no projeto padrão da organização. A Tabela 6.8 mostra as atividades que compõem o processo específico para o Projeto Sis@cad.

Tabela 6.8: Atividades do processo específico para Sis@cad

Discipline	*Activity*
Requirements	*Develop Requirements Management Plan*
	Find Actors and Use Cases
	Structure the Use Case Model
	Detail a Use Case
	Review Requirements
Analysis and Design	*Architectural Analysis*
	Class Design
	Database Design
	Asses Viability of Architectural Proof-of-concept
	Construct Architectural Proof-of-concept
	Describe the Run-time Architecture
	Implement Innovative Idea
	Review the Architecture
	Review the Design
	Validate Innovative Idea
Implementation	*Execute Developer Tests*
	Implement Design Elements
	Execute Unit Tests
	Refactor Code
	Review Code
Test	*Define Test Approach*
	Define Test Details
	Execute Acceptance Tests
	Write Acceptance Tests
	Write Unit Tests
Deployment	*Create Deployment Unit*
	Develop Support Materials
Configuration and Change Management	*Confirm Duplicate or Rejected CR*
	Review Change Request
	Submit Change Request

	Update Change Request
	Establish Change Control Process
	Write CM Plan
Project Management	*Compile Software Development Plan*
	Develop Business Case
	Develop Iteration Plan
	Identify and Assess Risks
	Initiate Project
	Iteration Acceptance Review
	Lifecycle Milestone Review
	Prepare for Phase Close-out
	Prepare for Project Close-out
	Project Acceptance Review
	Report Status
	Assess Iteration
	Initiate Iteration
	Iteration Planning Review
	Plan Phases and Iterations
	Project Approval Review
	Project Planning Review
	Project Review Authority (PRA) Review
Environment	*Select and Acquire Tools*
	Set Up Tools
	Tailor the Process for the Project
	Develop Development Case
	Review the Software Process for the Project
Training	*Identify Training*
	Execute Training

O processo específico para o projeto apresenta características de métodos planejados, por ser uma equipe distribuída com muitos estagiários e sem experiência na tecnologia a ser utilizada no projeto.

A definição de requisitos é realizada por meio de *use cases*. Na disciplina *Analysis and Design*, são inseridas várias atividades relacionadas à elaboração e validação da arquitetura, pois a equipe não tem experiência no uso da tecnologia e existe o risco projeto inviável. Considerando o risco não-entendimento de requisitos, foram inseridas as atividades estabelecer processos de mudanças (*Establish Change Control Process)* e

elaborar um plano de gerência de configuração (*Write a Configuration Management Plan*), na disciplina *Configuration and Change Management*. Na disciplina *Project Management*, foram adicionadas atividades para implantar um processo iterativo, com revisões ao final das iterações, já que existem riscos relacionados à tecnologia, projeto e requisitos. Foram adicionadas atividades para definição do processo para uso no projeto, na disciplina *Environment*. Considerando que a equipe não em conhecimento da tecnologia a ser utilizada no projeto, foram adicionadas atividades para treinamento da equipe, na disciplina *Training*.

Não foram detectadas atividades equivalentes no processo definido para o projeto Sis@cad.

6.4.5 Plano de Gerência de Riscos

O Plano de Gerenciamento de Riscos, também chamado de Plano GQM, foi elaborado por selecionar todas as métricas, sugeridas no *framework* PRiMA-F, associadas aos riscos priorizados para o projeto Sis@cad, como mostra a Tabela 6.9.

Tabela 6.9: Plano GQM para Sis@cad

Risco 1: Não-entendimento dos requisitos	
Meta 1: Analisar o projeto com a finalidade de monitorar com respeito à definição dos requisitos do ponto de vista da equipe de desenvolvimento.	
Questão 1.1: Os usuários validaram os documentos de requisitos do projeto?	*Métrica 1.1.1:* Percentual de validação de requisitos pelo Cliente PVRC = (número de documentos de requisitos validados/ número total de documentos de requisitos) * 100 *Métrica 1.1.2:* Percentual de requisitos elaborados pelo Cliente PRE = (número de requisitos elaborados pelos usuários/requisitos elaborados pelo pessoal de sistema) * 100
Questão 1.2: Os requisitos estão mudando durante a fase de execução do projeto?	*Métrica 1.2.1:* Percentual de alteração dos requisitos PAR = (número de solicitações de mudança de requisitos / número total de requisitos) * 100
Risco 2: Falta de uma metodologia de projeto efetiva	
Meta 2: Analisar o projeto com a finalidade de monitorar com respeito ao uso da metodologia do ponto de vista da equipe de desenvolvimento.	

Questão 2.1: A equipe está utilizando a metodologia definida para o projeto?	*Métrica 2.1:* Número de ocorrência geradas em revisões (comparar dados históricos) NOR = Número de ocorrências geradas nas revisões de qualidade
Risco 3: Falta de compromisso da gerência sênior com o projeto	
Meta 3: Analisar o projeto com a finalidade de monitorar com respeito ao comprometimento da gerência do ponto de vista da equipe de desenvolvimento.	
Questão 3.1: A gerência sênior (Scrum Master, XP Manager) está participando das reuniões de acompanhamento do projeto?	*Métrica 3.1.1:* Percentual de participação do gerente sênior PPGS = (número de reuniões que o gerente sênior participou / número de reuniões gerenciais realizadas) * 100
Questão 3.2: A gerência sênior (Scrum Master, XP Manager) está solucionando os problemas da equipe a ela repassados?	*Métrica 3.2.1:* Percentual de solução de problemas PSP = (número de problemas solucionados / número de problemas em aberto) * 100 *Métrica 3.2.2:* Percentual de solução de problemas no tempo apropriado PSPT = (número de problemas solucionados no prazo/ número de problemas repassados a gerência sênior) * 100
Risco 4: Falta de conhecimento/ habilidade requerida do pessoal do projeto / Equipe insuficiente ou inadequada	
Meta 4: Analisar o projeto com a finalidade de avaliar com respeito à habilidade da equipe do ponto de vista do gerente de projeto.	
Questão 4.1: Qual o percentual de pessoas habilitadas para desempenhar seu papel na equipe?	*Métrica 4.1.1:* Percentual de pessoas habilitadas PPH = (número de pessoas habilitadas / número de pessoas da equipe) * 100
Questão 4.2: Existem recursos humanos suficientes para o projeto?	*Métrica 4.2.1:* Percentual de recursos humanos PRH = (número de RH / número de RH estimados pelo gerente de projeto)*100
Questão 4.3: Qual o percentual de treinamentos atendidos?	*Métrica 4.3.2:* Percentual de treinamentos atendidos PRH = (número de treinamentos atendidos / número de treinamentos solicitados)*100

Risco 5: Introdução de novas tecnologias	
Meta 5: Analisar o projeto com a finalidade de avaliar com respeito à nova tecnologia do ponto de vista da equipe de desenvolvimento.	
Questão 5.1: Qual o índice de retrabalho por problemas de tecnologia?	*Métrica 5.1.1:* Percentual de Retrabalho por tecnologia PRT = (quantidade de horas de retrabalho por defeitos ou falhas devido à tecnologia / quantidade total de horas de retrabalho) * 100
Questão 5.2: Qual o percentual de defeitos por tecnologia?	*Métrica 5.2.1:* Percentual de defeitos por tecnologia PDT = (quantidade de defeitos por tecnologia / quantidade total de defeitos)*100
Questão 5.3: Qual o percentual de pessoas na equipe que dominam a tecnologia?	*Métrica 5.3.2:* Percentual de pessoas que dominam a tecnologia PPDT = (número de pessoas que dominam a tecnologia / número total de pessoas)*100
Risco 6: Desenvolvimento errado das funções ou interface	
Meta 6: Analisar o projeto com a finalidade de avaliar com respeito à comunicação do ponto de vista da equipe de desenvolvimento.	
Questão 6.1: As solicitações de mudanças encaminhadas ao gerente de projeto são comunicadas a equipe?	*Métrica 6.1.1:* Percentual de solicitações de mudanças encaminhadas PSME = (Quantidade de solicitações de mudanças encaminhadas à equipe / Quantidade de solicitações de mudanças do cliente)*100
Risco 7: Projeto (desenho) inviável	
Meta 7: Analisar o projeto com a finalidade de avaliar com respeito ao projeto (desenho) do ponto de vista da equipe de desenvolvimento.	
Questão 7.1: A arquitetura definida é viável?	*Métrica 7.1.1:* Viabilidade da arquitetura EA = (número de atividades de desenho concluídas / número de atividades de desenho total)

6.5 Análise dos Estudos de Caso

Os estudos de caso foram realizados com projetos de características bastante distintas, gerando processos específicos também distintos.

No projeto para desenvolvimento do sistema financeiro (SisF$n), a equipe é pequena, com desenvolvedores que conhecem a tecnologia a ser utilizada no projeto, o cliente encontra-se no próprio campus onde o sistema será desenvolvimento, portanto as características do projeto propiciam o uso de metodologias ágeis. Os padrões de mais alta relevância, sugeridos pela sistemática PRiMA, são exatamente padrões que visam proporcionar maior agilidade ao processo padrão da organização, baseado no RUP. Algumas equivalências entre as atividades do processo específico para o projeto tiveram que ser eliminadas pelo fato que atividades selecionadas para implantar padrões para prevenção de riscos do projeto são redundantes a atividades propostas pelo processo padrão.

No processo para desenvolvimento do sistema acadêmico (Sis@cad), a situação apresentada é inversa ao do SisF$n, pois a equipe está distribuída em diferentes câmpus, e o cliente, no caso a Reitoria, está distante das equipes de desenvolvimento; a equipe é composta por vários estagiários, que podem permanecer na equipe apenas por períodos curtos. Os padrões de mais alta relevância, sugeridos pela sistemática PRiMA, são exatamente padrões que visam proporcionar mais planejamento e documentação ao processo de software. Considerando a dificuldade de comunicação face a face em equipes distribuídas, são gerados documentos para que as equipes se comuniquem e mantenham-se informadas.

Conclui-se que o mecanismo de seleção de padrões não elimina o papel do projetista, apenas facilita a sua atividade, identificando os padrões de mais alta relevância, de acordo com os riscos e contexto do projeto. A eficácia do mecanismo de seleção depende das associações entre riscos e padrões e entre padrões e atividades inseridas na base de conhecimento da organização.

Nesta tese são sugeridas regras de associação de riscos a padrões e de padrões a atividades como exemplo, a organização que deve definir suas próprias regras, de acordo com as suas necessidades. Somente o uso das regras definidas para adaptação de processos e avaliações nos resultados obtidos nos projetos permitirão a organização ajustar a base de conhecimento às suas necessidades.

7 CONSIDERAÇÕES FINAIS

Este trabalho propõe uma sistemática, chamada de *Project Risk Management Approach* – PRiMA, para gerenciar riscos em projetos de software, por meio da instanciação de processos de desenvolvimento adaptados com base no processo de software padrão da organização, de acordo com os riscos priorizados para projetos de software.

Essa sistemática provê instrumentos para facilitar a identificação dos riscos que podem afetar o projeto e a seleção de ações preventivas e corretivas aos riscos, descritas como padrões organizacionais e de processo. O monitoramento dos riscos de software é descrito por meio de Planos GQM.

A área de desenvolvimento de software é altamente dinâmica, os aspectos estruturais e organizacionais dos projetos e do cliente alteram-se rapidamente, requisitos do cliente tendem a mudar, e novas tecnologias precisam ser adotadas para que a organização desenvolva produtos de alta qualidade de forma competitiva. Isto implica que as organizações de desenvolvimento de software devem ser capazes de adaptar rapidamente seus processos, de acordo com as mudanças do ambiente (WEINBERG, 1997).

Outro aspecto importante é a necessidade de gerenciar os riscos dos projetos de software de forma sistemática. Projetos de software envolvem riscos, para que os riscos não impactem negativamente nos resultados dos projetos, esses precisam ser gerenciados.

Abordagens para gerência de riscos convencionais visam acompanhar os riscos e somente são tomadas ações quando o risco está próximo de se materializar (BOEHM 1991; HALL, 1998). Neste trabalho a prevenção aos riscos é o requisito principal usado na adaptação de processos.

7.1 Contribuições

7.1.1 Gerência de Riscos

PRiMA descreve um conjunto de atividades que visam elaborar um processo de software para uso em um projeto específico, visando prevenir os riscos priorizados para o projeto.

Essa abordagem facilita a identificação dos riscos, por parte do gerente de projeto, por prover uma lista com os riscos mais comum em projetos de software, elaborada por este trabalho a partir da literatura. Essa lista pode ser complementada de acordo com as necessidades da organização. Revisões *post mortem* podem ser uma fonte importante de

possíveis riscos que podem impactar os projetos da organização, já que as organizações tendem a repetir os mesmos erros de projetos anteriores (VERNER, 2005).

Ações preventivas e corretivas aos riscos são descritas por meio de padrões de processo e organizacionais. Padrões documentam práticas realizadas com sucesso em vários projetos, e devido ao grande número de padrões catalogados (CUNNINGHAM, 2004; OLDFIELD, 2002; COPLIEN, 2004; SCHWABER, 2001), seu uso é facilitado.

De acordo com as regras de associação de padrões a riscos, definidas pela organização, podem ser elaborados processos ágeis, planejados ou híbridos; de acordo com o contexto do projeto. Abordagens híbridas podem obter vantagens de ambos os métodos, enquanto mitigam riscos de projetos (BOEHM, 2003).

A abordagem *Goal/Question/Metric* é utilizada para monitoramento dos riscos. Metas, questões e métricas são associadas aos riscos facilitando a elaboração de planos para monitoramento de riscos. Segundo Fenton e Neil (2000), o futuro para métricas de software é prover informação para suportar a tomada de decisão gerencial durante o ciclo de vida, e que o bom suporte é aquele que avalia e reduz o risco. Enquanto Fenton e Neil propõem a modelagem causal como forma de avaliar e reduzir riscos, neste trabalho é utilizada a abordagem GQM.

7.1.2 Adaptação de Processos

A adaptação de processos proposta por este trabalho utiliza um *framework* de processo (PRiMA-F), definido a partir de um metamodelo (PRiMA-M). Enquanto o metamodelo oferece flexibilidade à elaboração dos processos, o *framework* oferece facilidade na instanciação de processos.

O metamodelo representa os conceitos usados na definição de processos de software, isto é, a partir dos conceitos descritos no metamodelo são instanciados elementos de processo (atividades, papéis, artefatos, ferramentas, etc.) que são usados na construção de modelos de processo. O metamodelo define a linguagem para expressar processos de software; descreve os conceitos e suas relações com a finalidade de construir e interpretar modelos de processo. O metamodelo de processo foi elaborado a partir dos metamodelos do método ágil *Extreme Programming* (XP) e do processo planejado *Rational Unified Process* (RUP). Os elementos de processo instanciados a partir do metamodelo podem ser utilizados para gerar novos processos, adaptações e alterações de processos existentes; indiferentemente se os modelos de processos são ágeis, planejados ou híbridos.

O metamodelo, por si só, não resolve conflitos entre os elementos de processo selecionados, não descreve e não garante a consistência entre esses componentes. Para solucionar esse problema, foi elaborado um *framework* de processo.

O *framework* PRiMA-F integra as atividades possíveis de serem executadas em processos de projetos de uma organização, incluindo atividades que visam implantar padrões de processo e organizacionais como ações preventivas ou corretivas aos riscos e planos GQM associados aos riscos. Os elementos descritos em PRiMA-F foram instanciados a partir do metamodelo proposto.

A partir do *framework* podem ser instanciados diferentes processos, tais como: o processo de software da organização e os processos específicos para os projetos; pela seleção de elementos de processo (previamente definidos no *framework*).

PRiMA prevê uma base de conhecimento da organização onde são organizados os bens da organização relacionados aos processos de software, tais como: elementos de processo possíveis de serem instanciados nos processos da organização, padrões de processo e organizacionais usados para prevenir riscos de software, metas, questões e métricas associadas aos riscos. Essa base deve evoluir com o tempo e com base na experiência dos desenvolvedores. A base de conhecimento possibilita que o conhecimento organizacional seja gerenciado de forma que possa ser reutilizado em outros projetos da organização e facilite a tarefa de adaptação de processos. O conhecimento, armazenado na base de conhecimento, é de responsabilidade da organização. Neste trabalho são sugeridos exemplos que devem ser adequados à realidade da organização.

A adaptação de processos pressupõe a existência de um processo padrão da organização. A adaptação com base em um processo de software padrão da organização garante que um conjunto de elementos de processo, considerados fundamentais, existiram em todos os processos da organização. Melhorias no processo padrão da organização serão refletidas em todos os processos da organização, já que os processos são elaborados a partir do processo padrão.

É consenso entre a comunidade de software que produtos de alta qualidade são gerados a partir de processos de alta qualidade. Várias normas e modelos são propostos na literatura para avaliar e melhorar a qualidade dos processos de software. Como contribuição deste trabalho cita-se a elaboração de configurações de processos típicas, definidas para atender o modelo CMM; e que podem ser usadas como base para a elaboração do processo padrão da organização. Outras configurações de processo podem ser definidas pela organização.

Alguns guias de adaptação são propostos visando facilitar a tarefa de adaptação de processos de software de acordo com os requisitos de projetos.

Durante o projeto, as necessidades e os riscos da organização podem mudar, sendo necessário adaptar o processo o andamento do projeto (*on the fly*). A sistemática PRiMA propõe que em intervalos de tempo, definidos pela organização, os riscos sejam reavaliados e o processo alterado caso haja uma modificação nos riscos priorizados para o projeto. Esse tipo de adaptação é chamado de adaptação dinâmica de processos.

Um conjunto de atividades para extensão do *framework* PRiMA-F é proposto.

A adaptação se baseia nas informações, regras e associações inseridas na base de conhecimento, a aplicabilidade e a eficácia do processo definido depende dessas informações, que devem ser adequadas à organização. Pequenos ajustes no processo de software definido para o projeto são necessários e cabem ao projetista.

7.1.3 Estudos de Caso

A partir dos estudos de caso realizados conclui-se que podem ser gerados diferentes processos a partir de um mesmo processo padrão, de acordo com as características do projeto.

No projeto para desenvolvimento do sistema financeiro (SisF$n), a equipe é pequena, com desenvolvedores que conhecem a tecnologia a ser utilizada no projeto, o cliente encontra-se no próprio campus onde o sistema será desenvolvimento, portanto as características do projeto propiciam o uso de metodologias ágeis. Os padrões de mais

alta relevância, sugeridos pela sistemática PRiMA, são exatamente padrões que visam proporcionar maior agilidade ao processo padrão da organização, baseado no RUP.

No processo para desenvolvimento do sistema acadêmico (Sis@cad), a situação apresentada é inversa ao do SisF$n, pois a equipe está distribuída em diferentes câmpus, e o cliente, no caso a Reitoria, está distante das equipes de desenvolvimento, a equipe é composta por vários estagiários, que podem permanecer na equipe apenas por períodos curtos. Os padrões de mais alta relevância, sugeridos pela sistemática PRiMA, são exatamente padrões que visam proporcionar mais planejamento e documentação ao processo de software. A dificuldade de comunicação face a face em equipes distribuídas é superada por meio da elaboração de documentos, que são trocados entre as equipes, viabilizando a comunicação e a troca de informações.

O objetivo da adaptação é elaborar um processo definido para um projeto adequado ao contexto do projeto, obtendo vantagens do uso de métodos ágeis, planejados ou híbridos.

7.1.4 Ferramenta PRiMA-Tool

PRiMA-Tool foi elaborada para apoiar o uso da sistemática proposta para gerenciar riscos em projetos de software. Conclui-se que uma vez definida a base de conhecimento da organização na ferramenta, a adaptação de processos de software torna-se uma tarefa fácil. PRiMA-Tool apóia as atividades para elaboração de processos, mas não suporta a execução dos processos gerados, por isso a atividade *Monitorar os Riscos do Projeto* não é apoiada por PRiMA-Tool.

7.2 Perspectivas Futuras

Como trabalhos futuros indicam-se, entre outros, o uso de um Sistema de *Workflow* para elaboração de ambientes de apoio a execução de processos, gerados a partir de PRiMA, mecanismos de monitoramento de riscos, estender a sistemática para prevenção de defeitos, uso de simulação para validar os processos gerados a partir de PRiMA-Tool, e melhoria nos guias de adaptação. A seguir, uma breve descrição dessas sugestões.

7.2.1 Elaboração de Ambientes para Execução de Processos usando SGW

PRiMA-Tool suporta a adaptação dos processos de software, restringindo-se a definição do processo a ser usado em determinado projeto. O *website* elaborado visa facilitar a comunicação entre as pessoas da equipe. É desejável a elaboração de um ambiente que suporte a execução do processo de software, apoiando as atividades do dia-a-dia da equipe.

Ambientes para execução de processos podem ser construídos por meio de Sistemas de Gerência de *Workflow*. Sistemas de Gerência de *Workflow* possibilitam a instanciação de processos descritos por meio de XPDL (XML *Process Definition Language*), uma linguagem elaborada pela *Workflow Management Coalition* (WfMC), a partir de XML (*eXtensible Markup Language*).

A *Workflow Management Coalition* (WfMC) é uma organização sem fins lucrativos que tem como objetivo estabelecer uma terminologia comum e padrões para a

tecnologia de *workflow*, permitindo a interoperabilidade entre os diversos sistemas de *workflow*.

Existem várias ferramentas livres compatíveis com o formato definido pelo WfMC, tais como: Enhydra Shark, JBpm, WfMOpen e OSWorkflow, etc.

PRiMA-Tool pode ser alterada para gerar uma definição de modelo de processo compatível com XPDL, para que o processo seja instanciado e executado em um Sistema de Gerência de *Workflow* livre.

7.2.2 Monitoramento de Riscos

Implementar mecanismos de coleta automática de métricas durante a execução dos projetos. A coleta automática de métricas é necessária para o monitoramento dos riscos. Essas métricas são definidas no Plano GQM.

A coleta de métricas pode ser realizada pela integração de PRiMA-Tool, com ferramentas de gerência de projetos, modelagem de software, gerência de configuração. Foram analisadas algumas ferramentas de gerência de projeto, tais como: MS-Project, Gantt Project, Xplanner, DotProject, Tutos, Workbench. Na análise considerou-se o fato da ferramenta ser livre e gerar um XML com os dados do planejamento e acompanhamento, pois a integração poderia ser realizada pelo XML.

Com base nas medidas coletadas, poderá ser desenvolvida uma espécie de módulo de painel de controle de projetos. O painel de controle permite uma visualização de indicadores de projeto e serve como o *display* de status do gerenciamento dos riscos, informando ao gerente de projeto situações de perigo ao projeto.

7.2.3 Prevenção de Defeitos

As áreas-chave Análise e Resolução Causal, proposta pelo CMMI (CMMI, 2002); e Prevenção de Defeitos, proposta pelo CMM (PAULK, 1993); têm como finalidade identificar a causa de defeitos e outros problemas e tomar ações para preveni-los de ocorrer no futuro.

Prevenção de defeitos envolve analisar os defeitos que são encontrados em projetos passados e tomar ações específicas para prevenir a ocorrência desses tipos de defeitos no futuro. Os defeitos podem ter sido identificados em outros projetos, como em estágios anteriores ou tarefas do próprio projeto. Prevenção de defeitos é também um mecanismo para comunicar lições aprendidas entre projetos (CMMI, 2002; PAULK, 1993).

A sistemática proposta por este trabalho pode ser estendida para gerenciar a causa dos defeitos e inserir ações preventivas no momento de adaptar o processo da organização para uso em projetos. A base de conhecimento pode armazenar lições aprendidas sobre defeitos e padrões podem ser selecionados para prevenir defeitos, de forma semelhante como hoje são prevenidos os riscos.

7.2.4 Simulação

Desenvolvimento de software é uma atividade complexa envolvendo um grande número de fatores na definição de sucesso. Experimentações reais são difíceis e

custosas. O objetivo da simulação de processos é compreender os processos de desenvolvimento de software, e com base nesse entendimento mitigar os problemas que continuam ocorrendo na indústria, por prover suporte para a tomada de decisão gerencial (KIRK, 2004). Simulação pode ser utilizada para comparar processos e explorar os efeitos das variações entre processos.

Segundo Raffo et al. (1999), o uso de modelos de simulação pode apoiar uma avaliação quantitativa dos riscos e a incerteza associada com alternativas de mudanças nos processos. Os autores propõem o uso da simulação de processos para alcançar níveis 4 e 5 do CMM.

Como um trabalho futuro sugere-se o uso de simulação para avaliar o processo elaborado como resultado da adaptação pela sistemática PRiMA, visando identificar a efetividade do processo na prevenção dos riscos identificados para o projeto. O projetista pode utilizar a simulação para auxiliá-lo a identificação a melhor alternativa de processo, já que diferentes processos podem ser instanciados a partir do *framework*, de acordo com os padrões selecionados.

Simulação pode também ser utilizada para avaliar extensões propostas ao *framework* PRiMA-F. Neste caso, o uso do *framework* em projetos pilotos, sugerido na abordagem de extensão, pode ser substituído pela simulação de processos.

7.2.5 Outras Sugestões

Outras sugestões futuras podem ser citadas tais como: melhoria dos guias de adaptação, o uso de sistemas de gerência de configuração para manter um controle das mudanças e alterações no *framework* e o uso da sistemática em projetos reais.

As ações preventivas e corretivas são tratadas pela sistemática da mesma forma, através do uso da sistemática em projetos reais pode-se identificar a necessidade de se tratar as ações corretivas de forma distinta das ações preventivas.

REFERÊNCIAS

ACUÑA, S. T. et al. A Process Model Applicable to Software Engineering and Knowledge Engineering. **International Journal of Software Engineering and Knowledge Engineering**, Singapore, v.9, n.5, p. 663-687, Oct. 1999.

ADDISON, T.; VALLABH, S. Controlling Software Project Risks – An Empirical Study of Methods used by Experienced Project Managers. In: SOUTH AFRICAN INSTITUTE OF COMPUTER SCIENTISTS & INFORMATION TECHNOLOGISTS, SAICSIT, 2002. **Proceedings...** Port Elizabeth, South Africa: South African Institute for Computer Scientists and Information Technologists, 2002.

AMBLER; S. W. **An Introduction to Process Pattern**. 1998. Disponível em: <http://www.ambysoft.com/processPatternsPaper.html>. Acesso em: set. 2005.

AMBRIOLA, V.; CONRADI, R.; FUGGETA A. Assessing Process-Centered Software Engineering Environments. **ACM Transactions on Software Engineering and Methodology**, New York, v. 6, n. 3, p. 283–328, July 1997.

APPLETON, B. **Patterns and Software:** Essential Concepts and Terminology. 2000. Disponível em: <http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>. Acesso em: mar. 2005.

APPULLUTTY, K.; AMMAR, H.; POPSTAJANOVA, K. G. Software Requirement Risk Assessment Using UML. In: ACS/IEEE INTERNATIONAL CONFERENCE ON COMPUTER SYSTEMS AND APPLICATIONS, AICCSA, 3., 2005. **Proceedings…** Cairo, Egypt: IEEE Computer Society, 2005.

BANDINELLI, S.; DI NITTO, E.; FUGGETA, A. Supporting Cooperation in the SPADE-1 Environment. **IEEE Transactions on Software Engineering**, New York, v. 22, n. 12, p. 841-865, Dec. 1996.

BARGHOUTI, N. S. et al. Information Management in Process-Centered Software Engineering Environments. In: FUGGETTA, A.; WOLF, A. (Ed.). **Software Process. Trends in Software.** New York: John Wiley & Sons, 1996. p. 53-87.

BASILI, V. R.; ROMBACH, D. The TAME Project: Towards Improvement-Oriented Software Environments. **IEEE Transactions on Software Engineering**, New York, v.14, n.6, p. 758-773, June 1988.

BASILI, V. R.; SEAMAN, C. The Experience Factory Organizational. **IEEE Software**, New York, v. 19, n. 3, p. 30-31, May 2002.

BATINI, C.; CERI, S.; NAVATHE, S. B. **Conceptual Database Design An Entity-Relationship Approach**. Redwood City: The Benjamin/Cummings Publications, 1992.

BECK, K. **Programação Extrema (XP) Explicada**: Acolha as Mudanças. Porto Alegre: Bookman, 2004.

BECK, K.; BOEHM, B. Agility through Discipline: a Debate. **Computer**, New York, v. 36, n. 6, p.39-43, June 2003.

BECK, K. Embracing Change with Extreme Programming. **IEEE Computer**, New York, v. 32, n. 10, p.70-77, Oct. 1999.

BELKHATIR, N.; ESTUBLIER, J.; MELO, W. L. Adele2: A Support to Large Software Development Process. In: INTERNATIONAL CONFERENCE ON THE SOFTWARE PROCESS, 1., 1991. **Proceedings…** Redondo Beach, CA: [s.n.], 1991.

BELL CANADÁ. **Trillium**: Model for Telecom Product Development and Support Process Capability, release 3.0. Montreal, 1994.

BENCOMO, A. **Extending the RUP, Part 1**: Process Modeling. [S.l.]: IBM, 2005. Disponível em: <http://www-128.ibm.com/developerworks/ rational/library/05/r-3320/>. Acesso em: mar. 2005.

BOEHM, B. Software Risk Management: Principles and Practices. **IEEE Software**, New York, v. 8, n.1, p. 32-41, Jan. 1991.

BOEHM, B. Get Ready for Agile Methods, with Care. **Computer**, New York, v.35, n.1, p. 64-69, Jan. 2002.

BOEHM, B.; TURNER, R. Using Risk to Balance Agile and Plan-Driven Methods. **Computer**, New York, v. 36, n. 6, p.39-43, June 2003.

BOEHM, B.; TURNER, R. **Balancing Agility and Discipline**. Massachusetts: Addison-Wesley, 2004.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML – Guia do Usuário**. Rio de Janeiro: Campus, 2002.

BRIAN, L. C.; MORASCA, S.; BASILI, V. R. An Operational Process for Goal-Driven Definition of Measures. **IEEE Transactions on Software Engineering**, New York, v. 28, n. 12, p.1106-1125, Dec. 2002.

BRIAN, L. C.; DIFFERDING, C. M.; ROMBACH, D. **Practical Guidelines for Measurement-Based Process Improvement**. Kaiserslautern: Department of Computer Science, University of Kaiserslautern, 1996. (ISERN-96-05).

BUSCHMANN, F. et al. **Pattern - Oriented Software Architecture:** a system of patterns. New York: John Wiley & Sons, 1996.

CARR, M. J. et al. **Taxonomy-Based Risk Identification**. Pittsburg: Software Engineering Institute, Carnegie Mellon University, 1993. (CMU/SEI-93-TR-6).

CHARETTE, R. **The decision is in:** Agile Versus Heavy Methodologies. 2002. Disponível em: <http://www.cutter.com/freestuff/ epmu0119.html>. Acesso em: mar. 2003.

CLARK, B. K. Quantifying the Effects of Process Improvement on Effort. **IEEE Software**, New York, v. 17. n. 6, p. 65-70, Nov. 2000.

COCKBURN, A. Selecting a Project's Methodology. **IEEE Software**, New York, v. 17. n. 4, p. 64-71, July 2000.

COCKBURN, A. **Agile Software Development**. Massachusetts: Addison-Wesley, 2002.

COPPENDALE, J. Managing Risk in Product and Process Development and Avoid Unpleasant Surprises. **Engineering Management Journal**, New York, v.5, n.1, p. 35-38, Feb. 1995.

COHN, M.; FORD, D. Introducing an Agile Process to an Organization. **Computer**, New York, v. 36, n. 6, p.39-43, June 2003.

CONRADI, R. et al. EPOS: Object-Oriented and Cooperative Process Modelling. In: FINKELSTEIN, A.; KRAMER, J.; NUSEIBEH, B. (Ed.). **Software Process Modelling and Technology**. New York: Research Studies Press Limited, John Wiley & Sons, 1994.

CONRADI, R.; FUGGETA, A. Improving Software Process Improvement. **IEEE Software**, New York, v. 19, n. 1, p. 92-99, July 2002.

COPLIEN, J. O.; HARRISON, N. B. **Organizational Patterns of Agile Software Development**. Upper Saddle River: Prentice Hall, 2004.

COPLIEN, J. O. **Sofware Patterns**. New York: SIGS Books and Multimedia, 1996. Disponível em: <http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/>. Acesso em: mar. 2005.

COPLIEN, J. O. **A Development Process Generative Pattern Language**. 1995. Disponível em: <http://www1.bell-labs.com/user/cope/Patterns/Process/index.html>. Acesso em: mar. 2005.

CORTELLESSA, V. et al. Model-Based Performance Risk Analysis. **IEEE Transactions on Software Engineering**, New York, v. 31, n.1, p. 3-20, Jan. 2005.

CUGOLA, G.; GHEZZI, C. Software process: a retrospective and path to the future. **Software Process – Improvement and Practice**, New York, v. 4, n.3, p. 101-123, Sept. 1998.

CUNNINGHAM, W. **Portland Pattern Repository**. 2004. Disponível em: <http://c2.com/ppr/index.html>. Acesso em: jun. 2005.

DASKALANTONAKIS, M. K. A Practical View of Software Measurement and Implementation Experiences Within Motorola. **IEEE Transactions on Software Engineering**, New York, v.18, n.11, p.998-1010, Nov. 1992.

DASKALANTONAKIS, M. K. Achieving Higher SEI Levels. **IEEE Software**, New York, v.11, n.4, p.17-24, July 1994.

DEMARCO, T.; LISTER, T. **Waltzing with Bears:** Managing Risk on Software Projects. New York: Dorset House Publishing Company, 2003.

DEVEDZIC, V. Software Patterns. In: CHANG, S.K. (Ed.). **Handbook of Software Engineering and Knowledge Engineering.** Singapore: World Scientific Publishing, 2002. v.2, p. 645-671.

DILLON, R. L.; PATÉ-CORNELL, E.; GUIKEMA, S. D. Optimal Use of Budget Reserves to Minimize Technical and Management Failure Risks During Complex Project Development. **IEEE Transactions on Software Engineering**, New York, v. 53, n.3, p.382-395, Oct. 2005.

FAIRLEY, R. Risk Management for Software Projects. **Computer**, New York, v.11, n.3, p. 57-67, May 1994.

FALBO, R. A.; BORGES, L. S. M.; VALENTE, F. F. R. Using Knowledge Management to Improve Software Process Performance in a CMM Level 3 Organization. In: INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE, QSIC, 4., 2004. **Proceedings...** Braunschweig, Germany: IEEE Computer Society, 2004.

FARIAS, L. et al. Planejamento de Riscos em Ambientes de Desenvolvimento de Software Orientados à Organização. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, SBQS, 2., 1993. Fortaleza. **Anais...** Fortaleza: SBC, 2003.

FEILER, P. H.; HUMPHREY, W. Software Process Development and Enactment: Concepts and Definition. In. INTERNATIONAL CONFERENCE ON SOFTWARE PROCESS, 2., 1993. **Proceedings...** [S.l.: s.n], 1993.

FENTON, N. E.; NEIL, M. Making Decisions: Using Bayesian Nets and MCDA. **Knowledge-Based Systems**, [S.l.], v. 14, n. 1, p. 307-325, Mar. 2001.

FENTON, N. E.; NEIL, M. Software Metrics: Roadmap. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 22, 2000. **Proceedings...** New York: ACM Press, 2000.

FENTON, N.; PLEEGER, S. L.; GLASS, R. L. Science and Substance: A Challenge to Software Engineering. **IEEE Software**, New York, v.11, n.4, p. 86-95, July 1994.

FIORINI, S. **Uma Arquitetura para Reutilização de Processos de Software**. 2001. 243 f. Tese (Doutorado em Informática), Departamento de Informática, PUC, Rio de Janeiro.

FITZGERALD, B.; O'KANE, T. A Longitudinal Study of Software Process Improvement. **IEEE Software**, New York, v.16, n.3, p. 37-45, May 1999.

FONTOURA, L. M.; PRICE, R. T. Usando GQM para Gerenciar Riscos em Projetos de Software. In: SIMPÓSIO BRASILEIRO EM ENGENHARIA DE SOFTWARE, SBES, 18., 2004, Brasília. **Anais...** Recife: UFPE: SBC, 2004. p. 39-54.

FONTOURA, L. M.; HARTMANN, J.; PRICE, R. T. Metamodelo para Adaptação de Processos de Software com Base nos Riscos. In: WORKSHOP IBEROAMERICANO DE INGENIERÍA DE REQUISITOS Y AMBIENTES DE SOFTWARE, IDEAS, 9., 2006, La Plata. **Anais...** La Plata: UNLP, 2006. p.67-80.

FOWLER, Martin. **Refactoring:** Improving the Design of Existing Code. Massachusetts: Addison Wesley, 2000.

FOWLER, M. **The New Methodology**. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em: fev. 2003.

FUGGETA, A. Functionality and Architecture of PSEEs. **Information and Software Technology**, Amsterdam, v. 38, n. 4, p. 289-293, Apr. 1996.

FUGGETA, A. Software Process: A Roadmap. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 22., 2000. **Proceedings...** New York: ACM Press, 2000.

GAMMA, E. et al. **Design patterns:** elements of reusable object-oriented software. Reading: Addison-Wesley, 1994.

GILB, T. **Software Metrics**. [S.l.]: Chartwell-Bratt, 1976.

GINSBERG, M. P.; QUINN, L. H. **Process Tailoring and the Software Capability Maturity Model**. [S.l.]: Software Engineering Institute, Carnegie Mellon University, 1995. (CMU/SEI-94-TR-024).

GLASS, R. Frequently Forgotten Fundamental Facts about Software Engineering. **IEEE Software**, New York, v.18, n.3, p. 110-112, May 2001.

GLEICK, James. A Bug e a Crash: Sometimes a Bug Is More Than a Nuisance. **New York Times Magazine**, New York, Dec. 1996. Disponível em: <http://www.around.com/ariane.html>. Acesso em: dez. 2005.

GNATZ, M. et al. The Living Software Development Process. **Software Quality Professional**, Milwaukee, v.5, n.3, p. 4-16, June 2003.

GRADY, R. B. Successfully Applying Software Metrics. **Computer**, New York, v. 27, n. 9, p. 18-25, Sept. 1994.

GRESSE, C.; ROMBACH, D.; RUHE, G. A Practical Approach For Building GQM-Based Measurement. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 10., 1996. São Carlos, 1996. **Proceedings...** São Carlos: UFSCAR, 1996.

HAASE, V. et al. Bootstrap: Fine-Tuning Process Assessment. **IEEE Software**, New York, v.11, n.4 p. 25-35, July 1994.

HAGEN, M.; GRUHN, V. Process Pattern – a Means to Describe Processes in a Flexible Way. In: INTERNATIONAL WORKSHOP ON SOFTWARE PROCESS SIMULATION AND MODELING, PROSIM, 5., 2004. **Proceedings...** Scotland, United Kingdom: IEE, 2004.

HALL, E. **Managing Risk:** Methods for Software Systems Development. New York: Addison-Wesley, 1998.

HARTMANN, J. **Utilizando Padrões Organizacionais e Avaliação de Risco para Adaptar a Metodologia de Desenvolvimento de Software**. 2005. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

HARTMANN, J.; FONTOURA, L. M.; PRICE, R. T. Tailoring Software Processes with Organizational Patterns Languages using Risk Analysis. In: SIMPÓSIO BRASILEIRO EM ENGENHARIA DE SOFTWARE, SBES, 19., 2005, Uberlândia. **Anais...** Rio de Janeiro: PUCRJ, 2005. p. 327-342.

HARJUMAA, L.; TERVONEN, I.; VUORIO, P. Improving Software Inspection Process with Patterns. In: INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE, 9., 2004. **Proceedings...** Braunschweig, Germany: IEEE Computer Society, 2004.

HENNINGER, S. Using Software Process to Support Learning Organizations. In: WORKSHOP ON LEARNING SOFTWARE ORGANIZATIONS, 1., 1999. **Proceedings…** Kaiserslaurten, Germany: [s.n.],1999.

HENDERSON-SELLERS, B. Object-Oriented Methods and Processes. In: SOFTWARE METHODS AND TOOLS CONFERENCE, SMT, 2000. **Proceedings…** Wollongong: University of Wollongong, 2000.

HENDERSON-SELLERS, B.; DUÉ, R.; GRAHAM, I. A Qualitative Comparison of Two Processes for Object-Oriented Software Development. **Information and Software Technology**, Amsterdam, v. 43, n. 12, p. 705-724, Nov. 2001.

HETZEL, W. The Measurement Process. In: OMAN, P.; PFLEEGER, S. L. (Ed.). **Applying Software Metrics**. California: IEEE Computer Society, 1997. p. 72-93.

HIGHTOWER, R.; LESIECKI, N. **Java Tools for Extreme Programming**. New York: John Wiley & Sons, 2002.

HIGUERA, R. P.; HAIMES, Y. Y. **Software Risk Management**, 1996. (Technical Report CMU/SEI-96-TR-012).

HILLSIDE. **Patterns Library**. 2003. Disponível em: <http://hillside.net/patterns/>. Acesso em: mar. 2005.

HIGHSMITH, J.; COCKBURN, A. Agile Software Development: The Business of Innovation. **Computer**, New York, v.34, n.9, p. 120-122, Sept. 2001.

HILGERS, M. G. Quasi-Monte Carlo Methods in Cash Flow Testing Simulations. In: WINTER SIMULATION CONFERENCE, 32., 2000. **Proceedings...** Orlando, Florida: Society for Computer Simulation International, 2000.

HIRSCH, M. Making RUP Agile. In: CONFERENCE ON OBJECT ORIENTED PROGRAMMING SYSTEMS LANGUAGES AND APPLICATIONS, OOPSLA, 2002. **Proceedings...** New York: ACM Press, 2002.

HUANG, H.; ZHANG, S. Hierarchical Process Pattern: Construct Software Processes in a Stepwise Way. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, SMC, 2003. **Proceedings...** Washington: IEEE Computer Society, 2003.

HULL, M.E.C et al. Software Development Processes - an Introduction. **Information and Software Technology**, Amsterdam, v. 44, n. 1, p. 1-12, Jan. 2002.

HUMPHREY, W. S. **Introduction on the Personal Software Process**. Reading, Massachusetts: Addison Wesley, 1997.

HUMPHREY, W. S. **Managing the Software Process**. Reading, Massachusetts: Addison Wesley, 1990.

INTERNATIONAL STANDARDS ORGANIZATION. **ISO 9000-3** Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software, ISO 9000-3. Geneva, 1991.

INTERNATIONAL STANDARDS ORGANIZATION. **ISO/IEC 12207:** Information Technology - Software Life-cycle Processes. Geneva, 1995.

INTERNATIONAL STANDARDS ORGANIZATION. **ISO 15504:** SPICE Software Process Assessment - Part 1: Concepts and Introductory Guide. Geneva, 1995.

JUNKERMANN G. et al. MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment. In: FINKELSTEIN, A.; KRAMER, J.; NUSEIBEH, B. (Ed.). **Software Process Modelling and Technology**. Taunton: John Wiley and Sons: Research Studies Press, 1994.

KEENAN, F. Agile Process Tailoring and Problem Analysis (APTLY). In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE, 26., 2004. **Proceedings…** Washington, DC: IEEE Computer Society, 2004.

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

KELTER, U. et al. Do We Need 'Agile' Software Development Tools? In: INTERNATIONAL CONFERENCE NETOBJECTDAYS, NODe, 2004. **Proceedings…** Erfurt, Germany: Spring Verlag, 2002.

KESHLAF, A. A. HASHIM, K. A Model and Prototype Tool to Manage Software Risks. In: ASIA-PACIFIC CONFERENCE ON QUALITY SOFTWARE, 1., 2000. **Proceedings...** Hong-Kong: IEEE Computer Society, 2000.

KIPER, J. D.; FEATHER, M. S. A Risk-based Approach to Strategic Decision-Making for Software Development. In: INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 38., 2005. **Proceedings...** Hawaii: IEEE Computer Society, 2005.

KIRK., D; TEMPERO, E. A Flexible Software Process Model. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE, 26., 2004. **Proceedings…** Washington, DC: IEEE Computer Society, 2004.

KLERK, A. M. The Value of Project Risk Management. In: PORTLAND INTERNATIONAL CONFERENCE ON MANAGEMENT OF ENGINEERING AND TECHNOLOGY, PICMET, 2001. **Proceedings...** Portland, Oregan: [s.n.], 2001.

KONTIO, J.; GETTO, G.; LANDES, D. Experiences in Improving Risk Mangement Processes Using the Concepts of the Riskit Method. In: INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, SIGSOFT, 6., 1998. **Proceedings...** Lake Buena Vista, Florida: ACM SIGSOFT, 1998.

KRUCHTEN, P. **The Rational Unified Process:** An Introduction. Boston: Addison Wesley, 2000.

LARMAN, C.; BASILI, V. R. Iterative and Incremental Development: A Brief History. **Computer**, New York, v. 36, n. 6, p.39-43, June 2003.

LATUM, F. V. et al. Adopting GQM-Based Measurement in an Industrial Environment. **IEEE Software**, New York, v.15, n. 1, p. 78-86, Jan. 1998.

LAVAZZA, L. Providing Automated Support for the GQM Measurement Process. **IEEE Software**, New York, v.17, n. 3, p. 56-62, May 2000.

LEPASAAR, M.; MÄKINEN, T. Integrating Software Process Assessment Models using a Process Meta Model. In: IEEE INTERNATIONAL ENGINEERING MANAGEMENT CONFERENCE, IEMC, 2002. **Proceedings...** Cambridge: IEEE Computer Society, 2002.

LIEBOWITZ, J. A Look at NASA Goddard Space Flight Center's Knowledge Management Initiatives. **IEEE Software**, New York, v. 19, n. 3, p.40-42, May 2002.

LIPPERT, M. Developing Complex Projects Using XP with Extensions. **Computer**, New York, v. 36, n. 6, p.67-73, June 2003.

LYCETT, M. et al. Migrating Agile Methods to Standardized Development Practice. **Computer**, New York, v. 36, n. 6, p.79-85, June 2003.

MACHADO, L. F.; OLIVEIRA, K. M.; ROCHA, A. R. Modelo para Definição de Processos de Software baseado na ISO/IEC12207, em Modelos de Maturidade e Características do Projeto. In: JORNADAS IBEROAMERICANAS DE INGENIERÍA DE REQUISITOS Y AMBIENTES DE SOFTWARE, IDEAS, 2000, Cancún. **Memorias** Cuernavaca: Centro Nacional de Investigación y Desarollo Tecnológico, 2000. p. 73-84.

MADACHY, R. J. System Dynamics Modeling of an Inspection-Based Process. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE, 18., 1996. **Proceedings...** Berlin, Germany: IEEE Computer Society, 1996.

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**, New York, v. 29, n. 2, p.181-192, Feb. 2003.

MARTIN, R. C. The Process. In: BOOCH, G. et al. (Ed.). **Object Oriented Analysis and Design with Applications**. Boston: Addison Wesley, 1998.

MAY, D.; TAYLOR, P. Knowledge Management With Patterns. **Communications of the ACM**, New York, v. 46, n. 7, p.94-99, July 2003.

MCCABE, B.; FORD, D. Using Belief Networks to Assess Risk. In: WINTER SIMULATION CONFERENCE, 2001. **Proceedings...** Arlington: ACM, 2001.

MIZUNO, O. et al. On Prediction of Cost and Duration for Risky Software Projects Based on Risk Questionnaire. In: ASIA-PACIFIC CONFERENCE ON QUALITY SOFTWARE, 2., 2001. **Proceedings...** Hong-Kong, China: IEEE Computer Society, 2001.

MURTHI, S. Preventive Risk Management for Software Projects. **IT Professional**, New York, v. 4, n. 5, p. 9-15, 2002.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. **Software Measurement Guidebook**. Maryland: NASA Goddard Space Flight Center, 1995. (SEL-94-102).

NICHOLS, R.; CONNAUGHTON, C. **Software Process Improvement Journey:** IBM Australia Application Management Services. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2005. (CMU/SEI-2005-TR-02).

NIX, L. Dragon Management 101. **Cutter Consortium**, [S.l.], v.3, n.1, Feb. 2002. Disponível em: <http://www.ksinc.com/articles/DragonMgmt101.pdf>. Acesso em: dez. 2005.

OLDFIELD, P.; ANGAY, H.; RAWSTHORNE, D. Appropriate Process Movement, 2002. Disponível em: <http://www.aptprocess.com/index.htm.>. Acesso em: nov. 2005.

OLSSON, T.; RUNESON, P. V-GQM: A Feed-Back Approach to Validation of a GQM Study. In: INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, Metrics, 7., 2001. **Proceedings...** Sydney, Australia: IEEE Computer Society, 2001.

OMAN, P.; PFLEEGER, S. L. **Applying Software Metrics**. California: IEEE Computer Society, 1997.

OBJECT MANAGEMENT GROUP (OMG). **Software Process Engineering Metamodel Specification**, Version 1.0. Massachusetts: Object Management Group, Inc. 2002. (Technical Report formal/02-11-14).

OIVO, M.; BASILI, V. R. Representing Software Engineering Models: The TAME Goal Oriented Approach. **IEEE Transactions on Software Engineering**, Los Alamitos, v. 18, n. 10, p. 886-898, Oct. 1992.

PALMER, Stephen R. FELSING, John M. **A Practical Guide to Feature-Driven Development**. Upper Saddle River: Prentice Hall, 2002.

PAULK, M. C. Using the Software CMM With Good Judgment. **ASQ Software Quality Professional**, [S.l.], v.1, n. 3, p. 19-29, June 1999.

PAULK, M. C. Effective CMM-Based Process Improvement. In: INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY, 16., 1996. **Proceedings...** New York: IEEE Computer Society, 1996.

PAULK, M. C. et al. **Key Practices of the Capability Maturity Model**, Version 1.1. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1993. (CMU/SEI-93-TR-025).

PAULK, M. C. **Capability Maturity Model for Software**, Version 1.1. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1993. (CMU/SEI-93-TR-024).

PFLEEGER, S. R.; MCGOWAN, C. Software Metrics in the Process Maturity Framework. **Journal of Systems and Software**, Amsterdam, v. 12, n.3, p.255-261, July 1990.

PFLEEGER, S. R.; FITZGERALD, J. C. Software Metrics Tool Kit: Support for Selection, Collection and Analysis. **Information and Software Technology**, Amsterdam, v. 33, n. 7, p. 477-482, Sept. 1991.

PFLEEGER, S.R. **Software Engineering:** Theory and Practice. Upper Saddle River: Prentice-Hall, 1998.

POLLICE, G. **Using the IBM Rational Unified Process for Small Projects**: Expanding Upon eXtreme Programming. New York: IBM, 2003.

PROJECT MANAGEMENT INSTITUTE (PMI). **A Guide to the Project Management Body of Knowledge**. Pennsylvania, 2000.

PRESSMAN, R. **Engenharia de Software**. 5. ed. São Paulo: McGraw Hill, 2002.

PRIETO-DÍAZ, R. Implementing Faceted Classification for Software Reuse. **Software Engineering Notes**, New York, v. 34, n. 5, p. 88-97, May 1991.

RAFFO, D. M.; VANDEVILLE, J. V.; MARTIN, R. H. Software Process Simulation to Achieve Higher CMM Levels. **Journal of Systems and Software**, Amsterdam, v.46, n.3, p. 163-172, Apr. 1999.

RATIONAL SOFTWARE CORPORATION. **Rational Unified Process:** Version 2001.03.00. Cupertino, 2001.

RATIONAL SOFTWARE CORPORATION. **Rational Process Workbench**: Process Developer's Guide. Cupertino, 2002.

RATIONAL SOFTWARE CORPORATION. **Rational Unified Process**: Version 2003.06.12. Cupertino, 2003.

RAUTIAINEN, K. et al. A Tentative Framework for Managing Software Product Development in Small Companies. In: INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 1., 2002. **Proceedings…** Hawaii: IEEE Computer Society, 2002.

ROY, G. C. A Risk Management Framework for Software Engineering Practice. In: AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE, ASWEC, 2004. **Proceedings...** Hawaii: IEEE Computer Society, 2004.

RUS, I.; LINDVALL, M. Knowledge Management in Software Engineering. **IEEE Software**, New York, v. 19, n. 3, p.26-38, May 2002.

SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. Upper Saddle River: Prentice Hall, 2001.

SCHNEIDER, K.; HUNNIUS, J. V.; BASILI, V. Experience in Implementing a Learning Software Organization. **IEEE Software**, New York, v. 19, n. 3, p.46-49, May 2002.

SCHUYLER, J. **Risk and Decision Analysis in Projects**. Pennsylvania: Project Management Institute, 2001.

SOFTWARE ENGINEERING INSTITUTE. **Capability Maturity Model Integration:** (CMMI/SM), Version 1.1, Continuous Representation. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2002. (CMU/SEI-2002-TR-011).

SILLITTI, A. et al. Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. In: EUROMICRO CONFERENCE, 29., 2003. **Proceedings...** Antalya, Turkey: IEEE Computer Society, 2003.

SOLINGEN, R. V.; BERGHOUT, E. Integrating Goal-Oriented Measurement in Industrial Software Engineering: Industrial Experiences with and Additions to the Goal/Question/Metric Method. In: INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, Metrics, 7., 2001. **Proceedings...** Sydney, Australia: IEEE Computer Society, 2001.

SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Addison Wesley, 2003.

STANDISH GROUP. Third Quarter Research Report. 2004. Disponível em: <http://www.standishgroup.com/sample_research/PDFpages/q3-spotlight.pdf>. Acesso em: dez. 2005.

STÖRRLE, H. Describing Process Pattern with UML. In: EUROPEAN WORKSHOP SOFTWARE PROCESS TECHNOLOGY, EWSPT, 8., 2001. **Proceedings...** Witten, Germany: Springer, 2001.

TAYLOR, R. N. et al. Foundations of the Arcadia Environment Architecture. **Software Engineering Notes**, New York, v. 13, n. 5, p. 1-13, May 1988.

VASCONCELOS, F.M de; WERNER, C.M.L. Organizing the Software Development Process Knowledge: An Approach Based on Patterns. **International Journal of Software Engineering and Knowledge Engineering**, Singapore, v. 8, n. 4, p. 461-482, Dec. 1998.

VERNER, J. M.; EVANCO, W. M. In-houce Software Development: What Project Management Practices Lead to Success. **IEEE Software**, New York, v. 22, n. 1, p.86-93, Jan. 2005.

XU, P.; RAMESH, B. A Tool for the Capture and Use of Process Knowledge in Process Tailoring. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, HICSS, 36., 2002. **Proceedings...** Hawaii: IEEE Computer Society, 2002.

ZHONG, X.; MADHAVJI, N.; EMAM, K. E. Critical Factors Affecting Personal Software Process. **IEEE Software**, New York, v. 17. n. 6, p. 76-83, Nov. 2000.

YACOUB, S. M.; AMMAR, H. H. A Methodology for Architecture-Level Reliability Risk Analysis. **IEEE Transactions on Software Engineering**, New York, v. 28, n. 6, p.529-547, June 2002.

YAU, C. A Quantitative Methodology for Software Risk Control. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, SMC, 1994. **Proceedings...** Washington: IEEE Computer Society, 1994.

YATCO, M. C. **Joint Application Design/Development**, University of Missouri, St-Louis. Disponível em: <http://www.umsl.edu/ ~sauter/analysis/JAD.html>. Acesso em: jul. 2004.

YOON, I.; MIN, S.; BAE, D. Tailoring and Verifying Software Process. In: ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, APSEC, 8., 2001. **Proceedings...** Macao, China: IEEE Computer Society, 2001.

WANG, Y.; HE, Q. A Pratical Methodology for Measurement Deployment in GQM. In: CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING, CCECE, 2003. **Proceedings...** Montreal, Canada: IEEE Computer Society, 2003.

WANGENHEIM, C. **Utilização do GQM no Desenvolvimento de Software**. São Leopoldo: Universidade do Vale do Rio dos Sinos, 2000.

WARBOYS, B. The IPSE 2.5 Project: Process Modeling as the Basis for a Support Environment. In: INTERNATIONAL CONFERENCE ON SYSTEM DEVELOPMENT ENVIRONMENTS AND FACTORIES, 1990. **Proceedings…** Berlin: [s.n.], 1990.

WEBSTER, K. P. B.; OLIVEIRA, K. M.; ANTIQUEL, N. A Risk Taxonomy for Software Maintenance. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2005. **Proceedings …** Budapest, Hungary: IEEE Computer Society, 2005.

WEI, C.; HU, P. J.; CHEN, H. Design and Evaluation of a Knowledge Management System. **IEEE Software**, New York, v. 19, n. 3, p.56-59, May 2002.

WEINBERG, G. M. **Quality Software Management:** Anticipating Change. New York: Dorset House Publishing Company, 1997.

WELLER, E. F. Using Metrics to Manage Software Projects. **Computer**, New York, v. 27, n. 9, p. 27-33, Sept. 1994.

WILLIAMS, L.; COCKBURN, A. Agile Software Development: It's about Feedback and Change. **Computer**, New York, v. 36, n. 6, p.39-43, June 2003.

ANEXO A ATIVIDADES PRIMA-F

Este anexo descreve as atividades descritas no *framework* de processo *Project Risk Management Approach – Framework* (PRiMA-F). Para cada atividade, são descritos os demais elementos de processo (papéis, artefatos de entrada e saída) associados à atividade. Para as atividades propostas pelo RUP são omitidas a descrição, por questões de espaço. Na base de conhecimento de PRiMA-Tool foram cadastrados todos os elementos de processo descritos pelo *framework*. A PRiMA-Tool pode ser acessada a partir do endereço: http://www.urisantiago.br/lisandra/prima/.

1. Rational Unified Process – Discipline: Business Modeling

Activities	Worker	Input Artifacts	Resulting Artifacts
Assess Target Organization	Business-Process Analyst		Target-Organization Assessment
Set and Adjust Objectives	Business-Process Analyst	Business Case Stakeholder Requests Target-Organization Assessment Vision	Business Vision
Maintain Business Rules	Business-Process Analyst	Business Analysis Model Business Architecture Document Business Glossary Business Use Case Model Business Vision Supplementary Business Specification	Business Rule
Define the Business Architecture	Business-Process Analyst	Business Analysis Model Business Glossary Business Use Case Model Business Vision Supplementary Business Specification	Business Architecture Document Business System
Capture a Common Business Vocabulary	Business-Process Analyst	Business Vision	Business Glossary
Find Business Actors and Use Cases	Business-Process Analyst	Business Glossary Business Vision	Business Actor Business Use Case Business Use Case Model Supplementary Business Specification
Structure the Business Use-Case Model	Business-Process Analyst	Business Actor Business Glossary Business Use Case Business Use Case Model Supplementary Business Specification	Business Actor Business Use Case Business Use Case Model
Identify Business Goals	Business-Process Analyst	Business Architecture Document Business Goal Business Use Case Model Business Vision Project Specific Guidelines	Business Goal

Detail a Business Use Case	Business Designer	Business Actor Business Glossary Business Goal Business Use Case Business Use Case Model Business Vision	
Find Business Workers and Entities	Business Designer	Business Architecture Document Business Glossary Business Use Case Model Supplementary Business Specification	Business Analysis Model Business Entity Business Event Business Use-Case Realization Business Worker
Detail a Business Worker	Business Designer	Business Analysis Model Business Rule Business System Business Use Case Business Use-Case Realization Business Worker Supplementary Business Specification	Business Worker
Detail a Business Entity	Business Designer	Business Analysis Model Business Entity Business Rule Business System Business Use Case Business Use-Case Realization Supplementary Business Specification	Business Entity Business Event
Define Automation Requirements	Business Designer	Business Analysis Model Business Architecture Document Business Glossary Business Use Case Model Supplementary Business Specification Target-Organization Assessment	Analysis Model Supplementary Specifications Use-Case Model
Review the Business Use-Case Model	Technical Reviewer	Business Glossary Business Use Case Business Use Case Model Supplementary Business Specification	Review Record

Review the Business Analysis Model	Technical Reviewer	Business Analysis Model Business Entity Business Event Business Glossary Business Rule Business System Business Use-Case Realization Business Worker	Review Record

2. Rational Unified Process – Discipline: Requirements

Activities	**Worker**	**Input Artifact**	**Resulting Artifact**
Capture a Common Vocabulary	System Analyst	Glossary Vision	Glossary
Find Actors and Use Case	System Analyst	Vision Use-case model	Use-case model Use-case Supplementary Specifications
Develop Requirements Management Plan	System Analyst	Software Development Plan	Requirements Management Plan
Manage Dependencies	System Analyst	Requirements Attributes Change Request Requirements Management Plan Use case model Supplementary Specifications Use case Vision	Requirements Attributes Vision (refined)
Develop Vision	System Analyst	Stakeholder Requests Requirements Management Plan	Vision
Elicit Stakeholder Requests	System Analyst	Change Request Vision Stakeholder Requests	Storyboard Stakeholder Requests
Structure the Use-case Model	System Analyst	Iteration Plan Use case model Use case package	Use case model Use case package
Priorize Use Cases	Software Architect	Vision Supplementary Specifications Use Case Model Use Case	Software Architecture Document Requirements Atributtes
Detail a Use Case	Requirements Specifier	Glossary Requirements Management Plan Stakeholders Requests Vision Supplementary Specifications Use Case	Use Case

Detail the Software Requirements	Requirements Specifier	Requirements Attributes Use case Supplementary Specifications	Requirements Attributes Use case Supplementary Specifications Software Requirements Specification
Review Requirements	Technical reviewer	All requirements artifacts	Review record

3. Rational Unified Process – Discipline: Analysis and Design

Activities	**Worker**	**Input Artifact**	**Resulting Artifact**
Architectural Analysis	Software Architect	Architectural Proof-of-Concept Design Model Glossary Software Architecture Document Supplementary Specifications Use-Case Model Vision	Analysis Class Deployment Model Design Model Software Architecture Document
Use-Case Analysis	Designer	Analysis Class Analysis Model Design Model Glossary Software Architecture Document Supplementary Specifications Use Case Use-Case Model Use-Case Realization	Analysis Class Analysis Model Use-Case Realization
Construct Architectural Proof-of-concept	Software Architect	Deployment Model Design Model Software Architecture Document	Architectural Proof-of-Concept
Assess Viability of Architectural Proof-of-concept	Software Architect	Architectural Proof-of-Concept Business Case Glossary Risk List Supplementary Specifications Use-Case Model Vision	Review Record
Identify Design Mechanisms	Software Architect	Analysis Class Design Model Software Architecture Document Supplementary Specifications	Design Class Design Model Design Package Design Subsystem Software Architecture Document

Identify Design Elements	Software Architect	Analysis Class Analysis Model Design Model Software Architecture Document Supplementary Specifications	Capsule Design Class Design Model Design Package Design Subsystem Event Interface Protocol Signal
Incorporate Existing Design Elements	Software Architect	Design Model Software Architecture Document	Design Class Design Model Design Package Design Subsystem Interface Software Architecture Document
Describe Run-Time Architecture	Software Architect	Capsule Design Model Software Architecture Document Supplementary Specifications	Design Model Software Architecture Document
Describe Distribution	Software Architect	Deployment Model Design Model Implementation Model Software Architecture Document Supplementary Specifications	Deployment Model Software Architecture Document
Review the Architecture	Technical Reviewer	Risk List Software Architecture Document Supplementary Specifications	Review Record
Review the Design	Technical Reviewer	Analysis Model Data Model Design Model Navigation Map Supplementary Specifications Use-Case Model User-Interface Prototype	Review Record
Design the User-Interface	User-Interface Designer	Stakeholder Requests Storyboard Supplementary Specifications Use Case Vision	Navigation Map
Prototype the User-Interface	User-Interface Designer	Navigation Map Storyboard Supplementary Specifications Use Case	User-Interface Prototype

Use-Case Design	Designer	Analysis Model Capsule Design Class Design Model Design Subsystem Interface Supplementary Specifications Use Case	Design Model Use-Case Realization
Class Design	Designer	Analysis Class Design Class Design Model Event Signal Storyboard Supplementary Specifications Use-Case Realization User-Interface Prototype	Design Class Design Model
Design Testability Elements	Designer	Design Class Design Package Testability Class Test Interface Specification	Design Package Testability Class
Define Testability Elements	Test Designer	Deployment Model Design Model Implementation Model Interface Test Automation Architecture Test Case Test Data Test Strategy Use-Case Realization Workload Analysis Model	Test Automation Architecture Test Design Test Interface Specification
Subsystem Design	Designer	Design Model Design Subsystem Interface	Capsule Design Class Design Model Design Subsystem Interface
Capsule Design	Capsule Design	Capsule Event Protocol Signal	Capsule Design Class Protocol
Database Design	Database Designer	Analysis Class Data Model Design Class Design Model Supplementary Specifications Use-Case Realization	Data Model

4. Rational Unified Process – Discipline: Implementation

Activities	Worker	Input Artifact	Resulting Artifact
Structure the Implementation Model	Software Architect	Design Model	Implementation Model Implementation Subsystem Software Architecture Document
Plan System Integration	Integrator	Implementation Model Integration Build Plan Iteration Plan Use-Case Realization	Integration Build Plan
Implement Design Elements	Implementer	Data Model Design Model Implementation Element Software Architecture Document Supplementary Specifications Testability Element	Implementation Element
Implement Testability Elements	Implementer	Implementation Element Implementation Subsystem Testability Class Testability Element	Testability Element Test Stub
Implement Developer Test	Implementer	Design Model Implementation Element Testability Element Test Design Test Stub	Developer Test
Execute Developer Test	Implementer	Developer Test Implementation Element	Test Log
Analyze Runtime Behavior	Implementer	Implementation Element Test Log	Test Results
Plan Subsystem Integration	Integrator	Implementation Element Implementation Model Implementation Subsystem Integration Build Plan Iteration Plan Use-Case Realization	Integration Build Plan
Review Code	Technical Reviewer	Implementation Element Project Specific Guidelines	Review Record
Integrate Subsystem	Integrator	Implementation Element Implementation Subsystem Integration Build Plan	Build Implementation Subsystem

Integrate System	Integrator	Implementation Subsystem Integration Build Plan	Build

5. Rational Unified Process – Discipline: Test

Activities	Worker	Input Artifact	Resulting Artifact
Identify Test Motivators	Test Manager	Change Request Iteration Plan Quality Assurance Plan Software Requirement Stakeholder Requests Test Plan Use-Case Model Vision Work Order	Test Plan
Agree on the Mission	Test Manager	Change Request Iteration Plan Quality Assurance Plan Test Automation Architecture Test Plan Vision Work Order	Test Plan
Define Test Approach	Test Designer	Deployment Model Iteration Plan Software Architecture Document Software Development Plan Software Requirements Specification Test Automation Architecture Test Environment Configuration Test Plan Use-Case Model Vision	Test Environment Configuration Test Plan Test Strategy
Identify Test Ideas	Test Analyst	Data Model Deployment Model Design Model Implementation Model Iteration Plan Software Architecture Document Software Requirements Specification Test-Ideas List Test Plan Test Strategy Use-Case Realization Vision	Test-Ideas List

Define Assessment and Traceability Needs	Test Analyst	Configuration Management Plan Iteration Plan Quality Assurance Plan Requirements Management Plan Software Development Plan Test Plan	Test Plan
Identify Targets of Test	Test Analyst	Data Model Deployment Model Implementation Model Integration Build Plan Iteration Plan Software Architecture Document Test Strategy Use-Case Realization	Test Strategy
Define Test Environment Configuration	Test Designer	Deployment Model Software Architecture Document Test Automation Architecture Test Data Test Environment Configuration Test Plan Test Strategy Workload Analysis Model	Test Environment Configuration
Identify Testability Mechanisms	Test Designer	Deployment Model Software Architecture Document Test Automation Architecture Test Interface Specification Test Strategy User-Interface Prototype	Test Automation Architecture Test Interface Specification
Define Testability Elements	Test Designer	Deployment Model Design Model Design Model Implementation Model Interface Test Automation Architecture Test Case Test Data Test Interface Specification Test Strategy Use-Case Realization Workload Analysis Model	Test Automation Architecture Test Design Test Interface Specification

Define Test Details	Test Analyst	Change Request Supplementary Specifications Test Case Test Data Test-Ideas List Test Interface Specification Test Strategy Use-Case Realization	Test Case Test Data Test Script Workload Analysis Model
Obtain Testability Commitment	Test Manager	Issues List Software Architecture Document Software Development Plan Test Automation Architecture Test Interface Specification Test Strategy	Test Plan
Implement Test	Tester	Build Development Infrastructure Testability Class Test Automation Architecture Test Case Test Data Test Environment Configuration Test-Ideas List Test Script Test Strategy Tools Workload Analysis Model	Test Script
Implement Test Suite	Tester	Build Development Infrastructure Implementation Model Test Automation Architecture Test Data Test Environment Configuration Test Script Test Strategy Test Suite Tools	Test Suite

Structure the Test Implementation	Test Designer	Development Infrastructure Project Specific Guidelines Test Automation Architecture Test Case Test Data Test Environment Configuration Test Interface Specification Test Plan Test Script Test Strategy Test Suite	Test Script Test Suite
Execute Test Suite	Tester	Build Development Infrastructure Iteration Plan Testability Element Test Data Test Plan Test Script Test Suite	Test Log
Determine Tests Results	Test Analyst	Test Case Test-Ideas List Test Log Test Strategy Workload Analysis Model	Test Evaluation Summary Test Results
Analyze Test Failure	Tester	Test Case Test Data Test Log Test Script Test Strategy Test Suite Tools Workload Analysis Model	Change Request
Assess and Advocate Quality	Test Manager	Iteration Plan Project Measurements Quality Assurance Plan Risk List Test Evaluation Summary Test Results	Test Evaluation Summary
Assess and Improve Test Effort	Test Manager	Change Request Iteration Assessment Iteration Plan Project Measurements Quality Assurance Plan Review Record Status Assessment Test Evaluation Summary Test Plan Test Results	Test Evaluation Summary Test Plan

6. Rational Unified Process – Discipline: Deployment

Activities	Worker	Input Artifact	Resulting Artifact
Develop Deployment Plan	Deployment Manager	Deployment Model Iteration Plan Product Acceptance Plan Software Development Plan	Deployment Plan
Define Bill of Materials	Deployment Manager	Iteration Plan Product Acceptance Plan Software Development Plan	Bill of Materials
Develop Support Materials	Technical Writer	Build Iteration Plan Navigation Map Software Requirements Specification	End-User Support Material
Develop Training Materials	Course Developer	Build Deployment Plan Navigation Map Software Requirements Specification	Training Materials
Support Development	System Administrator	Development Infrastructure Test Environment Configuration Tools	Development Infrastructure
Execute Test Suite (From Test)	Tester	Build Development Infrastructure Iteration Plan Testability Element Test Data Test Plan Test Script Test Suite	Test Log
Determine Test Results (From Test)	Test Analyst	Test Case Test-Ideas List Test Log Test Strategy Workload Analysis Model	Test Evaluation Summary Test Results
Manage Acceptance Tests	Deployment Manager	Configuration Audit Findings Deployment Plan Deployment Unit Product Acceptance Plan Test Evaluation Summary	Change Request Test Environment Configuration
Write Release Notes	Deployment Manager	Bill of Materials Deployment Plan Integration Build Plan Iteration Assessment	Release Notes
Develop Installation Artifacts	Implementer	Build End-User Support Material	Installation Artifacts

Create Deployment Unit	Configuration Manager	Bill of Materials Build Deployment Model Deployment Plan End-User Support Material Installation Artifacts Project Repository Release Notes Training Materials	Deployment Unit
Manage Beta Test	Deployment Manager	Deployment Plan Deployment Unit Stakeholder Requests	Change Request
Create Product Artwork	Graphic Artist	Bill of Materials Deployment Plan Iteration Plan	Product Artwork
Release to Manufacturing	Deployment Manager	Bill of Materials Deployment Unit Product Artwork	Product
Verify Manufactured Product	Deployment Manager	Bill of Materials Product	Product
Provide Access to Download Site	Deployment Manager	Deployment Plan Deployment Unit	Deployment Unit

7. Rational Unified Process – Discipline: Configuration and Change Management

Activities	**Worker**	**Input Artifact**	**Resulting Artifact**
Establish CM Policies	Configuration Manager	Configuration Management Plan Development Case Software Development Plan	Configuration Management Plan
Establish Change Control Process	Change Control Manager	Configuration Management Plan Development Case Development Infrastructure Software Development Plan	Configuration Management Plan
Write CM Plan	Configuration Manager	Development Case Software Development Plan	Configuration Management Plan
Setup CM Environment	Configuration Manager	Configuration Management Plan Development Case Development Infrastructure Implementation Model Iteration Plan Software Development Plan	Project Repository
Create Integration Workspace	Integrator	Development Process Project Repository Project Specific Guidelines	Workspace
Report on Configuration Status	Configuration Manager	Configuration Management Plan Project Repository	Project Measurements

Perform Configuration Audits	Configuration Manager	Configuration Management Plan Project Repository	Configuration Audits Findings
Create Development Workspace	Any Role	-	Workspace
Make Changes	Any Role	Work Order Workspace	Workspace
Deliver Changes	Any Role	Workspace	Project Repository Workspace
Update Workspace	Any Role	Project Repository Workspace	Workspace
Create Baselines	Integrator	Project Repository Software Development Plan Work Order Workspace	Workspace
Promote Baselines	Integrator	Project Repository Software Development Plan Workspace	Project Repository Workspace
Create Deployment Unit	Configuration Manager	Bill of Materials Build Deployment Model Deployment Plan End-User Support Material Installation Artifacts Project Repository Release Notes Training Materials	Deployment Unit
Submit Change Requests	Any Role	-	Change Request
Review Change Request	Change Control Manager	Change Request Iteration Plan Problem Resolution Plan Product Acceptance Plan Quality Assurance Plan	Change Request
Confirm Duplicate or Rejected CR	Change Control Manager	Change Request	Change Request
Update Change Request	Any Role	Change Request	Change Request
Verify Changes in Build	Test Analyst	Build Change Request Test Log	Change Request Test Results

8. Rational Unified Process – Discipline: Project Management

Activities	**Worker**	**Input Artifact**	**Resulting Artifact**
Identify and Assess Risk	Project Manager	Vision	Risk List
Develop Business Case	Project Manager	Vision	Business case
Initiate Project	Project Manager	Business case	Software Development Plan
Project Approval Review	Management Reviewer	Risk List Business Case Software Development Plan	Review record

Develop Measurement Plan	Project Manager	Risk List Business Case Software Development Plan	Measurement Plan
Develop Risk Management Plan	Project Manager	Risk List	Risk Management Plan
Develop Product Acceptance Plan	Project Manager	Vision Software Requirements Specification Business Case	Product Acceptance Plan
Develop Problem Resolution Plan	Project Manager	Software Development Plan	Problem Resolution Plan
Develop Quality Assurance Plan	Project Manager	Business Case Configuration Management Plan Measurement Plan Problem Resolution Plan Risk Management Plan Software Development Plan Test Plan Vision	Quality Assurance Plan
Define Project Organization and Staffing	Project Manager	Risk List Business Case Vision	Software Development Plan
Define Monitoring and Control Processes	Project Manager	Risk Management Plan Software Development Plan	Measurement Plan Software Development Plan
Plan Phases and Iterations	Project Manager	Business Case Development Case Risk List	Software Development Plan
Compile Software Development Plan	Project Manager	Business Case Configuration Management Plan Development Case Iteration Plan Measurement Plan Problem Resolution Plan Product Acceptance Plan Project Specific Guidelines Quality Assurance Plan Requirements Management Plan Risk Management Plan Vision	Software Development Plan
Project Planning Review	Management Reviewer	Business Case Risk List Software Development Plan Vision	Review Record

Develop Iteration Plan	Project Manager	Development Case Development Process Risk List Software Architecture Document Software Development Plan Vision	Iteration Plan
Iteration Plan Review	Management Reviewer	Iteration Plan Risk List Software Development Plan	Review record
Acquire Staff	Project Manager	Development Case Software Development Plan	Software Development Plan
Initiate Iteration	Project Manager	Iteration Plan Software Development Plan	Work Order
Assess Iteration	Project Manager	Business Case Development Case Development-Organization Assessment Issues List Iteration Plan Measurement Plan Software Development Plan Status Assessment Test Evaluation Summary Test Plan Vision	Iteration Assessment
Iteration Evaluation Criteria Review	Management Reviewer	Iteration Plan Software Development Plan Test Plan	Review Record
Iteration Acceptance Review	Management Reviewer	Iteration Assessment Iteration Plan	Review Record
Prepare for Project Close-Out	Project Manager	Issues List Iteration Assessment Software Development Plan	Issues List Software Development Plan Status Assessment
Project Acceptance Review	Management Reviewer	Iteration Assessment Product Acceptance Plan Software Development Plan	Review Record
Prepare for Phase Close-Out	Project Manager	Issues List Iteration Assessment Software Development Plan	Issues List Iteration Assessment Software Development Plan Status Assessment
Lifecycle Milestone Review	Management Reviewer	Business Case Iteration Assessment Software Development Plan Status Assessment	Review Record

Monitor Project Status	Project Manager	Issues List Iteration Plan Measurement Plan Project Measurements Review Record Risk List Risk Management Plan Software Development Plan	Issues List Project Measurements Risk List
Report Status	Project Manager	Issues List Project Measurements Risk List Status Assessment	Status Assessment
Schedule and Assign Work	Project Manager	Change Request Iteration Plan Software Development Plan	Iteration Plan Software Development Plan Work Order
Handle Exceptions and Problems	Project Manager	Configuration Management Plan Issues List Problem Resolution Plan Status Assessment	Issues List Work Order
Organize Review	Review Coordinator	Development Process Iteration Plan Project Specific Guidelines Review Record Software Development Plan	Review Record
PRA Project Review	Management Reviewer	Status Assessment	Review Record

9. Rational Unified Process – Discipline: Environment

Activities	**Worker**	**Input Artifact**	**Resulting Artifact**
Tailor the Process for the Project	Process Engineer	Development Process	Development Process
Develop Development Case	Process Engineer	Development Case Development Process Software Development Plan	Development Case
Select & Acquire Tools	Tool Specialist	Development Case Project Specific Guidelines Tools	Project Specific Guidelines
Prepare Guidelines for the Project	Process Engineer	Development Case Project-Specific Templates Tools	Project-Specific Templates
Prepare Templates for the Project	Process Engineer	Development Case Tools	Tools

Launch Development Process	Process Engineer	Development Case Development Process Project Specific Guidelines Project-Specific Templates Requirements Management Plan Tools	Change Request
Set Up Tools	Tool Specialist	Development Case Project Specific Guidelines Tools	Project Specific Guidelines Tools
Verify Tool Configuration and Installation	Tool Specialist	Development Case Development Infrastructure Project Specific Guidelines Tools	-
Develop Manual Style guide	Technical Writer	Development Case Manual Styleguide Software Development Plan Tools	Manual Styleguide
Support Development	System Administrator	Development Infrastructure Test Environment Configuration Tools	Development Infrastructure

10. Extreme Programming (XP) – Discipline: Requirements

Activities	Description	Worker	Input Artifact	Resulting Artifact
Divide User Story	If the User story is bigger then he may be split.	Customer	User Story	User Story
Priorize User Story	Business then decides which stories to implement in what order, as well as when and how often to produce a production release of the system.	Customer	User Story	User Story
Write User Story	Each feature is written out as a User Story, which gives the feature a name, and describes in broad strokes what is required. User stories are typically written on 4x6 cards.	Customer	-	User Story

11. Extreme Programming (XP) – Discipline: Analysis and Design

Activities	Description	Worker	Input Artifact	Resulting Artifact

Create Spike Solutions	Create spike solutions to figure out answers to tough technical or design problems. A spike solution is a very simple program to explore potential solutions. Build a system which only addresses the problem under examination and ignore all other concerns. Most spikes are not good enough to keep, so expect to throw it away. The goal is reducing the risk of a technical problem or increases the reliability of a user story's estimate. When a technical difficulty threatens to hold up the system's development put a pair of developers on the problem for a week or two and reduce the potential risk.	Developer	User Story	Code
Validate Spike Solutions	Validate Spike Solutions	Developer	Code	Test Results
Write System Metaphor	Choose a system metaphor to keep the team on the same page by naming classes and methods consistently. What you name your objects is very important for understanding the overall design of the system and code reuse as well. Being able to guess at what something might be named if it already existed and being right is a real time saver. Choose a system of names for your objects that everyone can relate to without specific, hard to earn knowledge about the system.	Developer	User Story	Metaphor Class, Responsibilities, and Collaboration (CRC) Cards
Write Tasks	For each of the user stories the team determines which tasks have to be completed in order to make the user story work. Each task typically takes between as few as 2 hours up to 2 days. A simple user story may consist of only one task. In general, each user story consists of several tasks.	Team	User Story	Task

12. Extreme Programming (XP) – Discipline: Implementation

Activities	**Description**	**Worker**	**Input Artifact**	**Resulting Artifact**
Execute Unit Tests	Execute unit tests	Developer	Code Unit tests	Test Results
Implement Tasks	All production code is written by two programmers sitting at one machine. Essentially, all code is reviewed as it is written.	Developer (pair)	Task	Code

Integrate code	All changes are integrated into the codebase at least daily. The tests have to run 100% both before and after integration.	Developer	Code	Buid
Refactor Code	Continual improvement of the structure of the code. Reestruturar o sistema para simplificar o código sempre que possível	Developer	Code	Code

13. Extreme Programming (XP) – Discipline: Deployment

Activities	Description	Worker	Input Artifact	Resulting Artifact
Create Release	The purpose of this activity is to create a deployment unit that is sufficiently complete to be used by the customer.	Developer	Code	Release

14. Extreme Programming (XP) – Discipline: Test

Activities	Description	Worker	Input Artifact	Resulting Artifact
Determine Tests Results	The tests score is published to the team. It is the team's responsibility to schedule time each iteration to fix any failed tests.	Developer	Test Results	Defect and associated data
Execute Acceptance Tests	In the next step the XP-team estimates the effort for every user story. Sometimes this will not be possible due to lack of semantic/business information. In this case the customer will be asked. Sometimes the estimation is not possible because solving a new software problem is required. In this case a prototype (or spike-solution) will be implemented. In some cases several variants of the prototype will be implemented in order to choose from.	Customer	Acceptance Tests Build	Test Results
Write Acceptance Tests	For every user story one or more acceptance tests will be created by the customer. Theses test server as a validation mean for both the XP-team and the customer.	Customer	User Story	Acceptance Tests
Write Unit Tests	Before programmers add a feature, they write a test for it. When the suite runs, the job is done.	Developer	Task	Unit Tests

15. Extreme Programming (XP) – Discipline: Project Management

Activities	Description	Worker	Input Artifact	Resulting Artifact
Accept Task	Developer accept the task.	Developer	Task	Task
Classify by Risk	The development classify the user story by risk.	Developer	User Story	Metrics
Collect Metrics	To collect metrics for measuring progress with numbers.	Tracker	Metrics	Metrics

Define Iteration Scope	After determining the required tasks, the XP-team estimates the required effort for each of the tasks. This may lead to the situation, which the sum of the efforts for all tasks is higher than the effort that can be handled in the iteration. In this case the task may be split, or the user story may be split, or the user story may deferred to the next iteration altogether. If the team detects that too many user stories have been assigned to the current release cycle, the customer will be informed. Then the customer will be able to decide which user story will be deferred into the next release cycle.	Customer	User Story	User Story Release Plan
Define Velocity	Development determines at what speed they can perform the project.	Developer	Metrics User Story	Velocity
Elaborate Release Plan	After user stories have been written you can use a release planning meeting to create a release plan. The release plan specifies exactly which user stories are going to be implemented for each system release and dates for those releases. This gives a set of user stories for customers to choose from during the iteration planning meeting to be implemented during the next iteration. These selected stories are then translated into individual programming tasks to be implemented during the iteration to complete the stories.	Coach	Task User Story	Release Plan
Estimate Task	Development estimates how much effort each task will take.	Developer	Task	Metrics: development time
Estimate User Story	Development estimates how much effort each story will take, and how much effort the team can produce in a given time interval (the iteration).	Developer	User Story	Metrics: development time
Guide Team	Coach guides the team.	Coach	Release Plan	Release Plan
Negotiate with customer	Negotiate with customer.	Team	Metrics Release Plan	Release Plan
Report Progress	The developers report progress to the tracker.	Tracker	Metrics	Report Progress

16. Others Processes – Discipline: Requirements

Atividades	Descrição	Discipline	Papel	Artefato Entrada	Artefato saída	Fonte
Develop a system prototype	Build an isolated prototype solution whose purposes are to understand requirements, to validate requirements with customers, to explore the cost and benefits of design decisions and, where necessary, to explore new technology.	Requirements	System Analyst	Software Requirements	Prototype	Generic
Name chunks of functionality	Early in the requirements gathering process, the requirements are needed for delimiting scope and for estimation purposes. Select and name chunks of required functionality that will act as placeholders for implied detailed requirements.	Requirements	System Analyst	User Requirements	Software Requirements	Generic
Priorize Backlog Product	The team meets with Product Owner, management, and users to figure out what functionality to build during the next Sprint.	Requirements	Scrum Master	Product Backup	Product Backlog	SCRUM
Priorize Implied Requirements	Produce a prioritized list of ImpliedRequirements. Sequencing by priority allows us to estimate end-dates for given scopes, or estimate scope for given end-dates, where available development effort per time period is assumed to be constant.	Requirements	Client	User Requirements	User Requirements	Generic

Review Requirements with Customer	To formally verify that the results of Requirements conform to the customer's view of the system with the customer	Requirements	Technical Reviewer	Glossary Software Requirements Specification Supplementary Specifications Use-case Use-case model Use-case package Vision	Review Record	Generic
Validate a system prototype	To Validate a System prototype with customer to define the requirements.	Requirements	System Analyst	System Prototype	Review Record	Generic

17. Others Processes – Discipline: Analysis and Design

Atividades	Descrição	Discipline	Papel	Artefato Entrada	Artefato saída	Fonte
Create a Model Class	Create a class model.	Analysis and Design	Developer	Software Requirement	Analysis Model	Generic
Implement Innovative Idea	Have a small specialist team develop an innovative idea outside the constraints of project development, to build confidence in the idea.	Analysis and Design	System Analyst	Software Requirements	Code	Generic
Validate Innovative Idea	Have a small specialist team develop an innovative idea outside the constraints of project development, to build confidence in the idea	Analysis and Design	System Analyst	Code Test Case	Test Results	Generic

18. Others Processes – Discipline: Project Management

Atividades	Descrição	Discipline	Papel	Artefato Entrada	Artefato saída	Fonte
Conduct the Daily Meetings	Each Scrum Team meets daily 15-minute status meeting called the Daily Scrum. During the meeting, the team explains what it has accomplished since the last meeting, what it is going to do before the next meeting, and what obstacles are in its ways.	Project Management	Scrum Master	Sprint Backlog		SCRUM
Define a project Plan	Detail the requirements, analyze, design and implement in (approximate) order of sequence on the prioritized list.	Project Management	Project Manager	Requirements Specification	Project Plan	Generic
Define a Scheduler	Define a sheduler	Project Management	Project Manager	Software Requirements	Scheduler	
Negotiate the Scheduler with Team	Negotiate the Scheduler with Team.	Project Management	Project Manager	Scheduler	Scheduler	Generic

Plan the Sprint	The Team works by itself to figure out how it is going to build this functionality into a Product increment during the Sprint. Input to this meeting is Product Backlog, the latest increment of product, and the capabilities and past performance of the team.	Project Management	Scrum Master	Product Backlog	Sprint Backlog	SCRUM

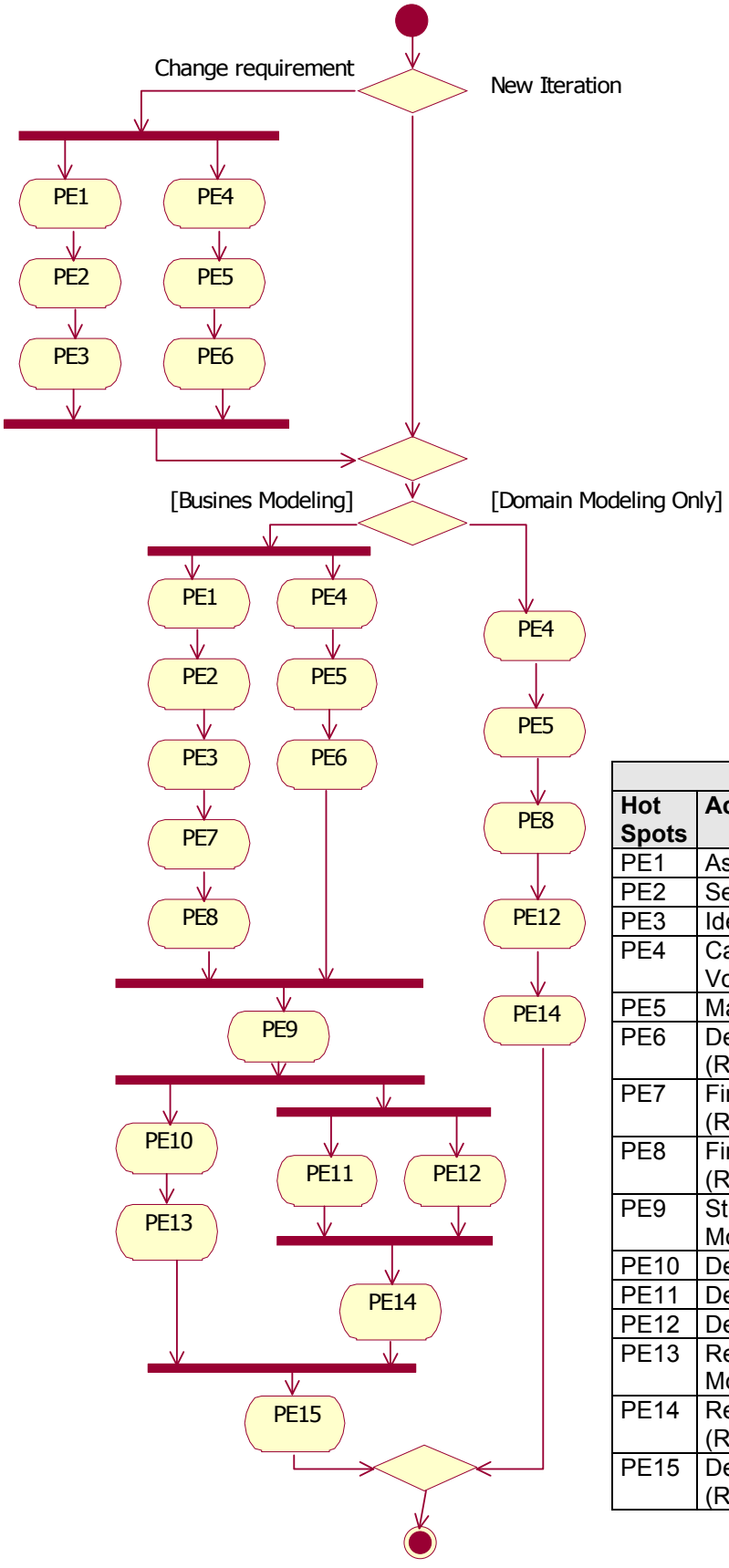
19. Others Processes – Discipline: Environment

Atividades	**Descrição**	**Discipline**	**Papel**	**Artefato Entrada**	**Artefato saída**	**Fonte**
Elaborate the Software Process for the Organization	Define a software process for an organization.	Environment	Process Engineer	Organization Software Process	Organization Software Process	Generic
Review the Software Process for the Organization	Review the software process for the organization periodically.	Environment	Process Engineer	Software Process for the Organizacion	Review Record	CMM
Review the Software Process for the Project	Review the software process for the project periodically.	Environment	Process Engineer	Software Process for the Project	Review Record	CMM

ANEXO B DIAGRAMAS DE ATIVIDADES PRIMA-F

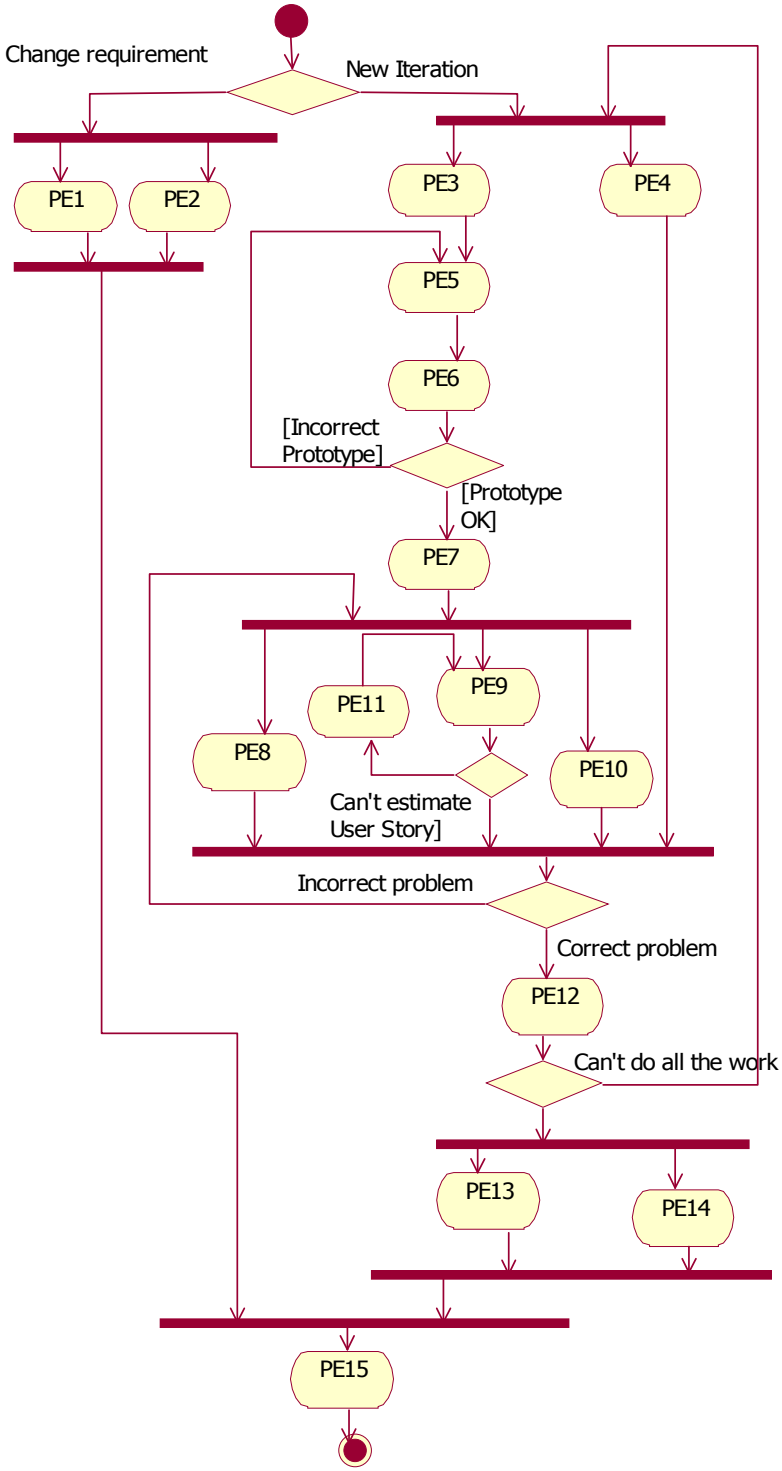
Diagramas de Atividades, elaborados para PRiMA-F, descrevendo as atividades possíveis de serem instanciadas em cada *hot spot*.

Business Modeling



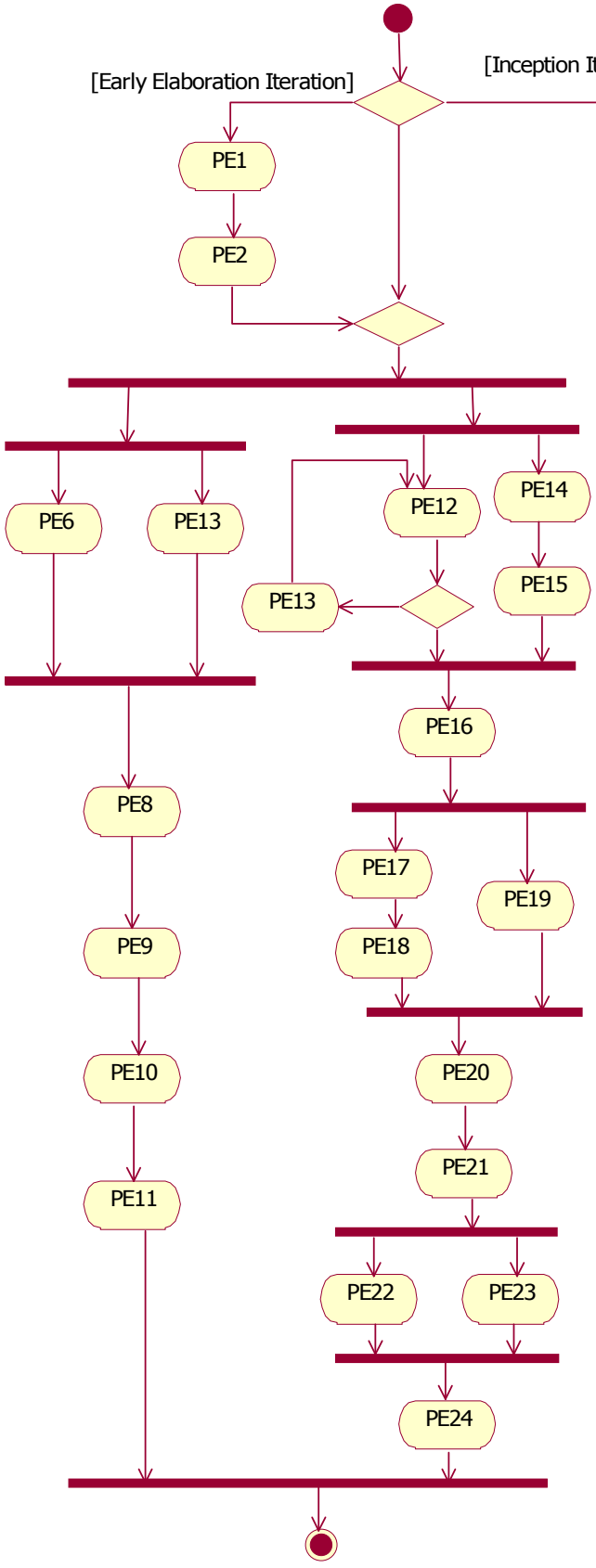
Legend	
Hot Spots	**Activities**
PE1	Assess Target Organization (RUP)
PE2	Set and Adjust Objectives (RUP)
PE3	Identify Business Goals (RUP)
PE4	Capture a Common Business Vocabulary (RUP)
PE5	Maintain Business Rules (RUP)
PE6	Define the Business Architecture (RUP)
PE7	Find Business Actors and Use Cases (RUP)
PE8	Find Business Workers and Entities (RUP)
PE9	Structure the Business Use-Case Model (RUP)
PE10	Detail a Business Use Case (RUP)
PE11	Detail a Business Worker (RUP)
PE12	Detail a Business Entity (RUP)
PE13	Review the Business Use-Case Model (RUP)
PE14	Review the Business Analysis Model (RUP)
PE15	Define Automation Requirements (RUP)

Requirements



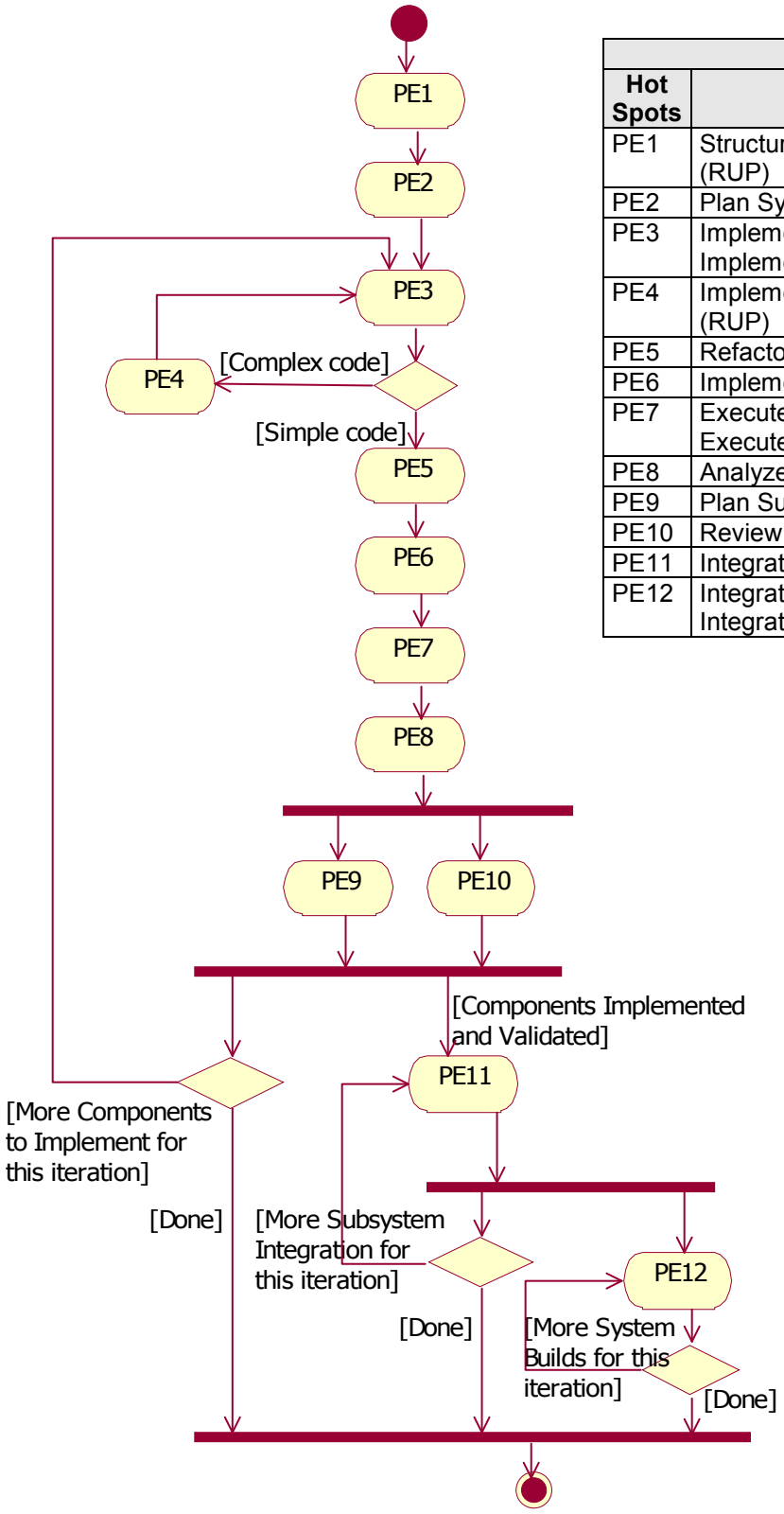
Legend	
Hot Spots	**Activities**
PE1	Structure the Use-case Model (RUP)
PE2	Manage Dependencies (RUP)
PE3	Elicit Stakeholder Requests (RUP) Write User Story (XP)
PE4	Develop Requirements Management Plan (RUP)
PE5	Develop a System prototype
PE6	Validate a system prototype
PE7	Develop Vision (RUP)
PE8	Capture a Common Vocabulary (RUP)
PE9	Find Actors and Use Case (RUP) Write User Story (XP) Name chunks of functionality
PE10	Manage Dependencies (RUP)
PE11	Divide User Story (XP)
PE12	Priorize Use Cases (RUP) Priorize User Story (XP) Priorize Product Backlog Priorize the Software Requirements
PE13	Detail a Use Case (RUP)
PE14	Detail the Software Requirements (RUP)
PE15	Review Requirements (RUP) Review the Requirements with Customer

Analysis and Design



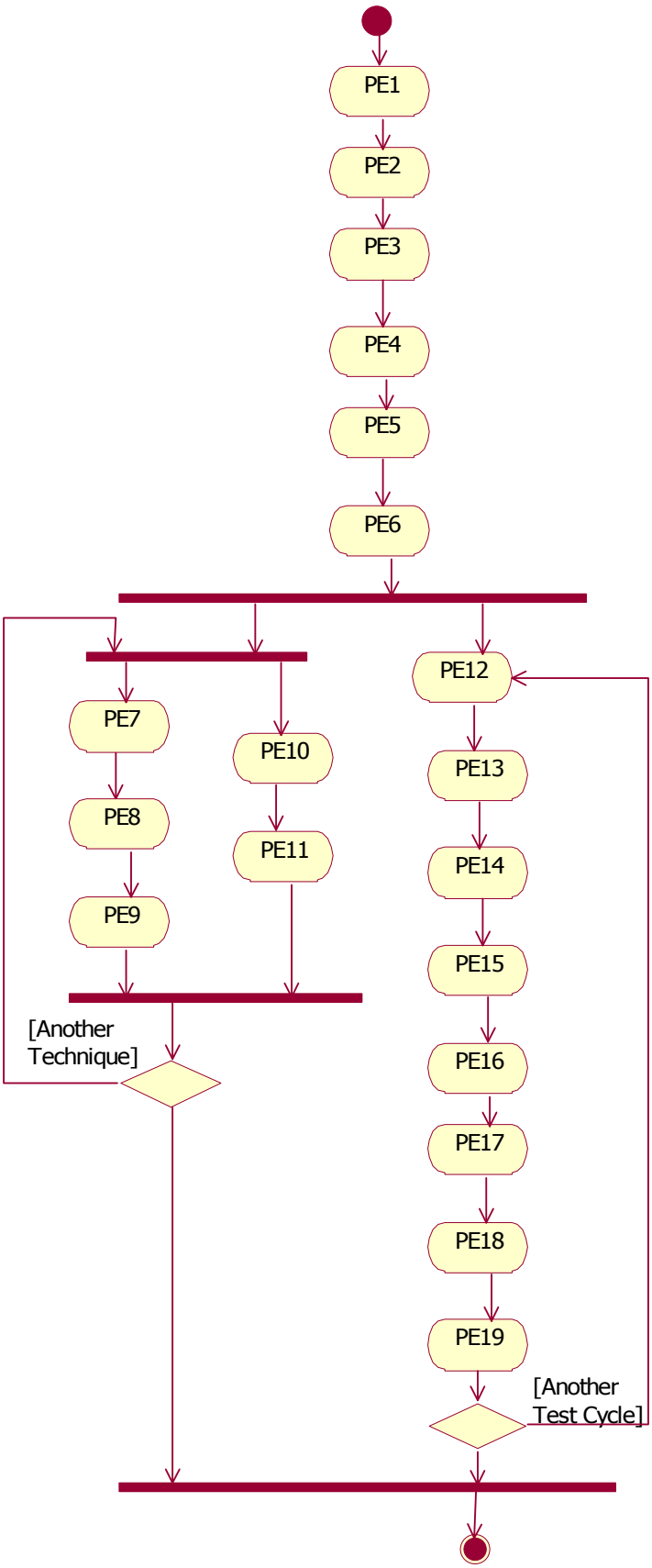
Legend	
Hot Spots	**Activities**
PE1	Architectural Analysis (RUP) Write a System Metaphor (XP)
PE2	Use-Case Analysis (RUP) Create a Model Class
PE3	Architectural Analysis (RUP)
PE4	Construct Architectural Proof-of-concept (RUP) Create Spike Solutions (XP) Implement Innovative Idea
PE5	Assess Viability of Architectural Proof-of-concept (RUP) Validate Spike Solutions (XP) Validate Innovative Idea
PE6	Identify Design Mechanisms (RUP)
PE7	Identify Design Elements (RUP) Write Tasks (XP)
PE8	Incorporate Existing Design Elements (RUP)
PE9	Describe Run-Time Architecture (RUP)
PE10	Describe Distribution (RUP)
PE11	Review the Architecture (RUP)
PE12	Identify Design Elements (RUP) Write Tasks (XP)
PE13	Divide/Combine Task (XP)
PE14	Design the User-Interface (RUP)
PE15	Prototype the User-Interface (RUP)
PE16	Review the Design (RUP)
PE17	Use-Case Design (RUP)
PE18	Class Design (RUP)
PE19	Design Testability Elements (RUP)
PE20	Define Testability Elements (RUP)
PE21	Subsystem Design (RUP)
PE22	Capsule Design (RUP)
PE23	Database Design (RUP)
PE24	Review the Design (RUP)

Implementation



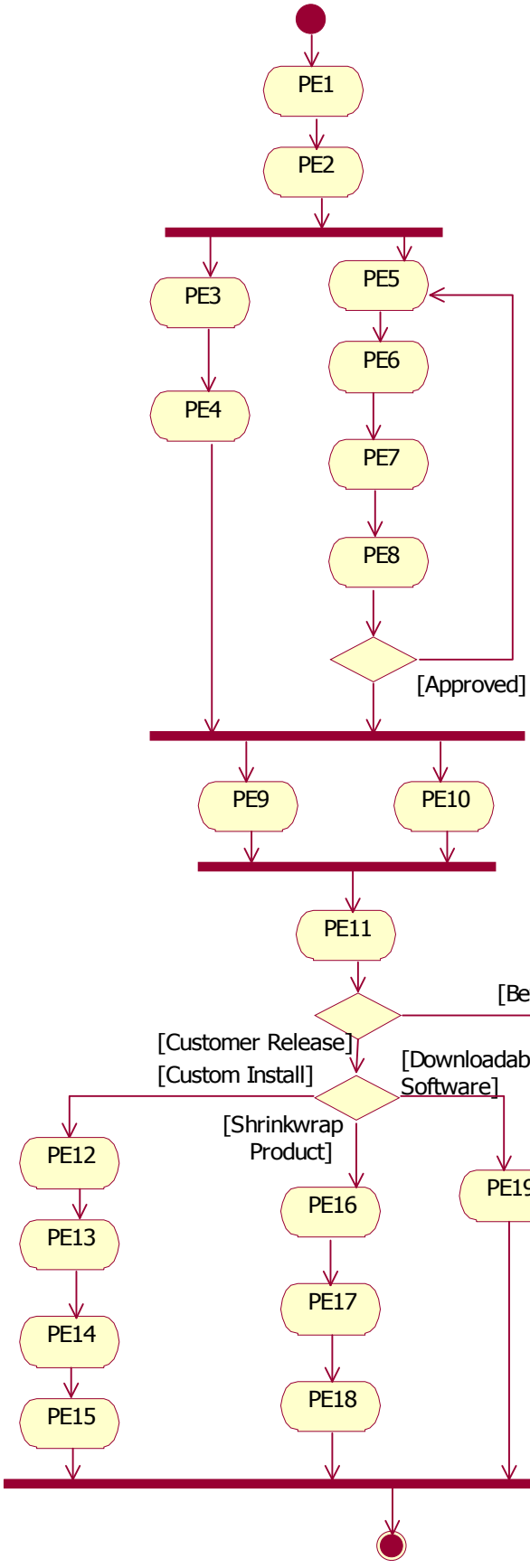
Legend	
Hot Spots	**Activities**
PE1	Structure the Implementation Model (RUP)
PE2	Plan System Integration (RUP)
PE3	Implement Design Elements (RUP) Implement Tasks (XP)
PE4	Implement Testability Elements (RUP)
PE5	Refactor Code (XP)
PE6	Implement Developer Test (RUP)
PE7	Execute Developer Test (RUP) Execute Unit Tests (XP)
PE8	Analyze Runtime Behavior (RUP)
PE9	Plan Subsystem Integration (RUP)
PE10	Review Code (RUP)
PE11	Integrate Subsystem (RUP)
PE12	Integrate System (RUP) Integrate Code (XP)

Test



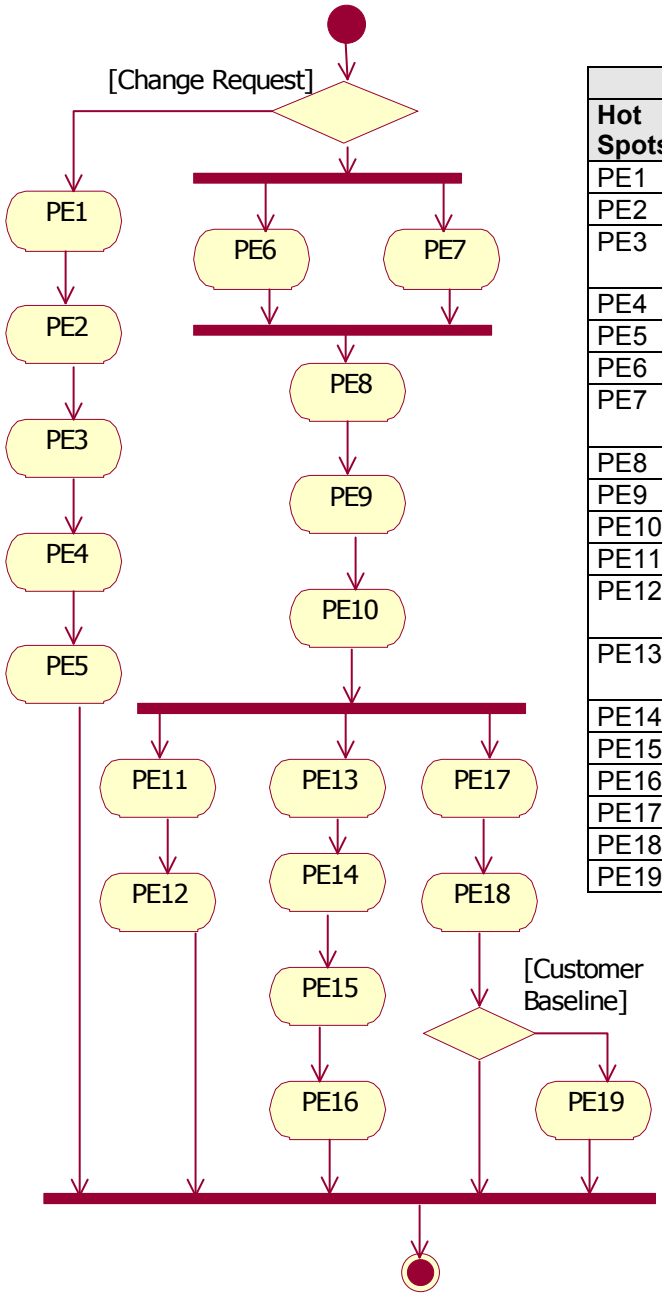
Legend	
Hot Spots	**Activities**
PE1	Identify Test Motivators (RUP)
PE2	Agree on the Mission (RUP)
PE3	Define Test Approach (RUP)
PE4	Identify Test Ideas (RUP)
PE5	Define Assessment and Traceability Needs (RUP)
PE6	Identify Targets of Test (RUP)
PE7	Define Testability Elements (RUP)
PE8	Identify Testability Mechanisms (RUP)
PE9	Define Test Environment Configuration (RUP)
PE10	Define Test Details (RUP)
PE11	Obtain Testability Commitment (RUP)
PE12	Structure the Test Implementation (RUP) Write Acceptance Tests (XP)
PE13	Implement Test (RUP) Write Unit Tests (XP)
PE14	Implement Test Suite (RUP)
PE15	Execute Test Suite (RUP) Execute Unit Tests (XP) Execute Acceptance Tests (XP)
PE16	Determine Tests Results (RUP) Determine Test Results (XP)
PE17	Analyze Test Failure (RUP)
PE18	Assess and Advocate Quality (RUP)
PE19	Assess and Improve Test Effort (RUP)

Implantação

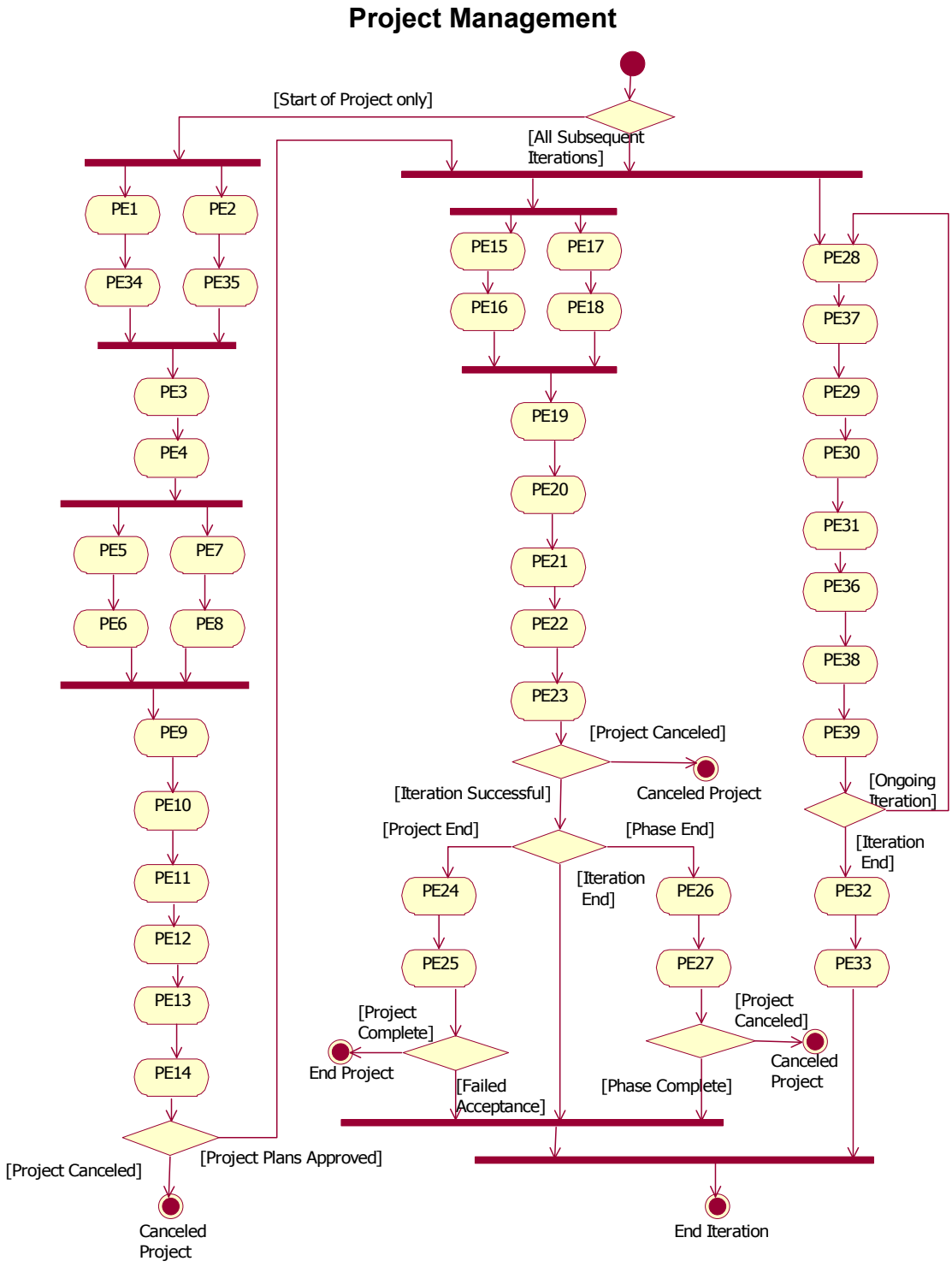


Legend	
Hot Spots	**Activities**
PE1	Develop Deployment Plan (RUP)
PE2	Define Bill of Materials (RUP)
PE3	Develop Support Materials (RUP)
PE4	Develop Training Materials (RUP)
PE5	Support Development (RUP)
PE6	Execute Test Suite (RUP)
PE7	Determine Test Results (RUP)
PE8	Manage Acceptance Tests (RUP)
PE9	Write Release Notes (RUP)
PE10	Develop Installation Artifacts (RUP)
PE11	Create Deployment Unit (RUP) Create Release (XP)
PE12	Support Development (RUP)
PE13	Execute Test Suite (RUP) Execute Acceptance Tests (xp)
PE14	Determine Test Results (RUP)
PE15	Manage Acceptance Tests (RUP)
PE16	Create Product Artwork (RUP)
PE17	Release to Manufacturing (RUP)
PE18	Verify Manufactured Product (RUP)
PE19	Provide Access to Download Site (RUP)
PE20	Manage Beta Test (RUP)

Configuration and Change Management



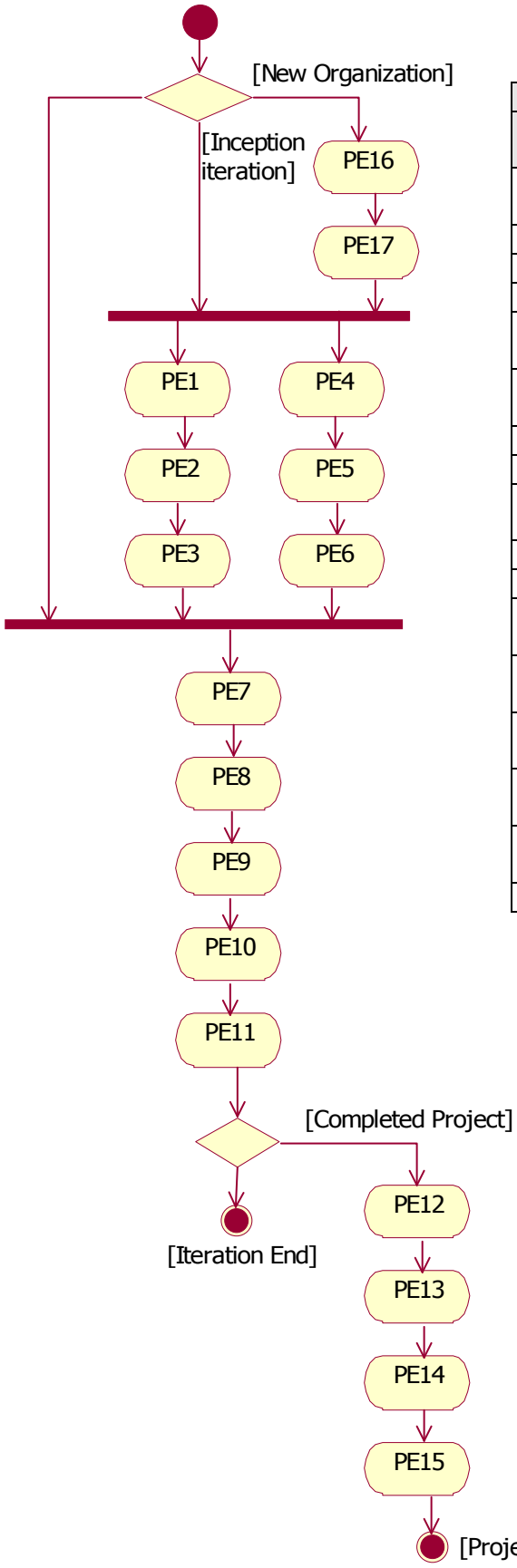
Legend	
Hot Spots	**Activities**
PE1	Submit Change Requests (RUP)
PE2	Review Change Request (RUP)
PE3	Confirm Duplicate or Rejected CR (RUP)
PE4	Update Change Request (RUP)
PE5	Verify Changes in Build (RUP)
PE6	Establish CM Policies (RUP)
PE7	Establish Change Control Process (RUP)
PE8	Write CM Plan (RUP)
PE9	Setup CM Environment (RUP)
PE10	Create Integration Workspace (RUP)
PE11	Perform Configuration Audits (RUP)
PE12	Report on Configuration Status (RUP)
PE13	Create Development Workspace (RUP)
PE14	Update Workspace (RUP)
PE15	Make Changes (RUP)
PE16	Deliver Changes (RUP)
PE17	Create Baselines (RUP)
PE18	Promote Baselines (RUP)
PE19	Create Deployment Unit (RUP)



Legend	
Hot Spots	**Activities**
PE1	Identify and Assess Risk (RUP)
PE2	Develop Business Case (RUP)
PE3	Initiate Project (RUP) Define Velocity (XP) Define the Project Plan
PE4	Project Approval Review (RUP)

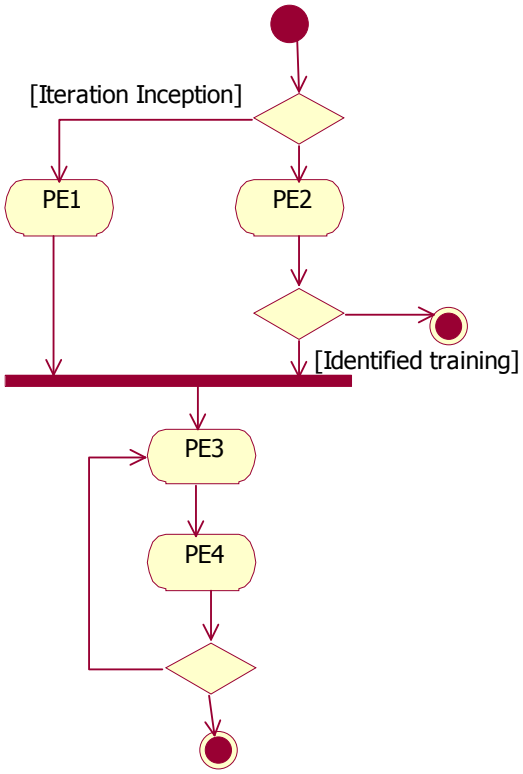
PE5	Develop Measurement Plan (RUP) Estimate User Story (XP)
PE6	Develop Risk Measurement Plan (RUP) Classify by Risk (XP)
PE7	Develop Problem Resolution Plan (RUP)
PE8	Develop Product Acceptance Plan (RUP)
PE9	Develop Quality Assurance Plan (RUP/Extended RUP)
PE10	Define Project Organization and Staffing (RUP)
PE11	Define Monitoring and Control Processes (RUP)
PE12	Plan Phases and Iterations (RUP)
PE13	Compile Software Development Plan (RUP)
PE14	Project Planning Review (RUP/Extended RUP)
PE15	Develop Iteration Plan (RUP) Elaborate Release Plan (XP) Plan the Sprint Define the Scheduler
PE16	Iteration Planning Review (RUP) Negotiate the Scheduler with Team
PE17	Identify and Assess Risk (RUP) Define Iteration Scope (XP)
PE18	Develop Business Case (RUP)
PE19	Acquire Staff (RUP)
PE20	Initiate Iteration (RUP) Accept Task(XP)
PE21	Assess Iteration (RUP) Estimate Task(XP)
PE22	Iteration Evaluation Criteria Review (RUP)
PE23	Iteration Acceptance Review (RUP)
PE24	Prepare for Project Close-Out (RUP)
PE25	Project Acceptance Review (RUP)
PE26	Prepare for Phase Close-Out (RUP)
PE27	Lifecycle Milestone Review (RUP)
PE28	Monitor Project Status (RUP) Collect Metrics (XP) Conduct the Daily Scrum
PE29	Report Status (RUP) Report Progress (XP)
PE30	Handle Exceptions and Problems (RUP) Negotiate with Client (XP) Guide Team (XP)
PE31	Schedule and Assign Work (RUP) Accept Task (XP)
PE32	Organize Review (RUP)
PE33	PRA Project Review (RUP)
PE34	Estimate resources and funds (Extended RUP)
PE35	Estimate critical computational resources (Extended RUP)
PE36	Track critical computational resources (Extended RUP)
PE37	Track Software Quality Assurance Activities (Extended RUP)
PE38	Software Quality Assurance Activities Review (Extended RUP)
PE39	Report Audits Results (Extended RUP)

Environment



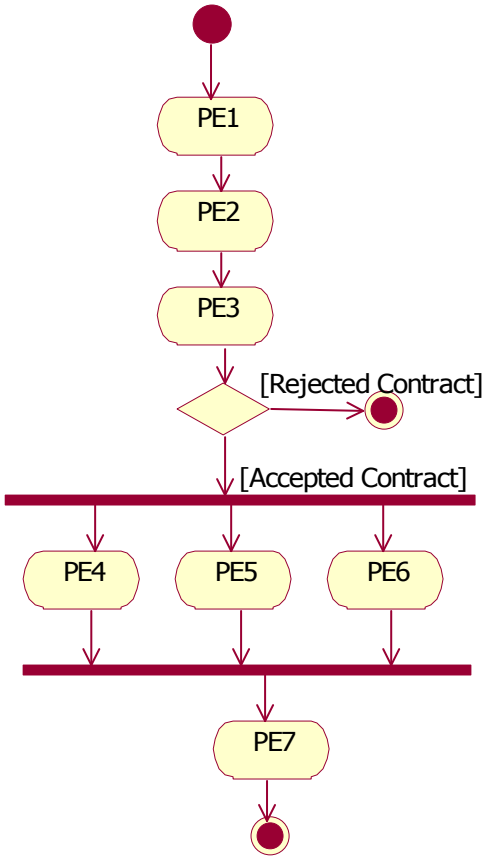
Legend	
Hot Spots	**Activities**
PE1	Tailor the Process for the Project (RUP)
PE2	Develop Development Case (RUP)
PE3	Review Process for the Project
PE4	Select & Acquire Tools (RUP)
PE5	Prepare Guidelines for the Project (RUP)
PE6	Prepare Templates for the Project (RUP)
PE7	Launch Development Process (RUP)
PE8	Set Up Tools(RUP)
PE9	Verify Tool Configuration and Installation (RUP)
PE10	Develop Manual Style guide (RUP)
PE11	Support Development (RUP)
PE12	Assess Current Organizational (Extended RUP)
PE13	Organizational Assessment Review (Extended RUP)
PE14	Develop Process Improvement Plan (Extended RUP)
PE15	Process Improvement Plan Review (Extended RUP)
PE16	Elaborate Process for the Organization
PE17	Review Process for the Organization

Training



Legend	
Hot Spots	**Activities**
PE1	Identify Training (Extended RUP)
PE2	Assess Training Plan (Extended RUP)
PE3	Execute Training (Extended RUP)
PE4	Assess Concluded Training (Extended RUP)

Subcontract Manager



Legend	
Hot Spots	**Activities**
PE1	Develop a Subcontract Management Plan (Extended RUP)
PE2	Select Subcontractor (Extended RUP)
PE3	Elaborate Contract and Define Commitments (Extended RUP)
PE4	Monitor Project Status (RUP)
PE5	PRA Project Review (RUP)
PE6	Perform Configuration Audits (RUP)
PE7	Assess Artifacts Developed (Extended RUP)

ANEXO C CONFIGURAÇÕES DE PROCESSO

Business Modeling							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Assess Target Organization	RUP		X				
Set and Adjust Objectives	RUP		X				
Identify Business Goals	RUP		X				
Capture a Common Business Vocabulary	RUP		X				
Maintain Business Rules	RUP		X				
Define the Business Architecture	RUP		X				
Find Business Actors and Use Cases	RUP		X				
Find Business Workers and Entities	RUP		X				
Structure the Business Use-Case Model	RUP		X				
Detail a Business Use Case	RUP		X				
Detail a Business Worker	RUP		X				
Detail a Business Entity	RUP		X				
Review the Business Use-Case Model	RUP		X				
Review the Business Analysis Model	RUP		X				
Define Automation Requirements	RUP		X				

Requirements							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Develop a system prototype	Outros						
Priorize the Software Requirements	Outros						
Review the Requirements with Customer	Outros						
Name chunks of functionality	Outros						
Validate a system prototype	Outros						
Capture a Common Vocabulary	RUP						
Detail a Use Case	RUP		X	X	X		
Detail the Software Requirements	RUP		X	X	X		
Develop Requirements Management Plan	RUP	X	X	X	X		
Develop Vision	RUP		X	X	X		
Elicit Stakeholder Requests	RUP		X	X	X		
Find Actors and Use Case	RUP		X	X	X		
Manage Dependencies	RUP		X	X	X		

Priorize Use Cases	RUP		X	X	X		
Review Requirements	RUP		X	X	X		
Structure the Use-case Model	RUP		X	X	X		
Priorize Backlog Product	SCRUM						
Divide User Story	XP						X
Priorize User Story	XP						X
Write user story	XP						X

Analysis and Design							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Create a Model Class	Outros						
Implement Innovative Idea	Outros						
Validate Innovative Idea	Outros						
Architectural Analysis	RUP	X	X	X	X		
Assess Viability of Architectural Proof-of-concept	RUP		X		X		
Capsule Design	RUP		X				
Class Design	RUP		X				
Construct Architectural Proof-of-concept	RUP		X		X		
Database Design	RUP		X				
Define Testability Elements	RUP		X				
Describe Distribution	RUP		X				
Describe Run-Time Architecture	RUP		X				
Design Testability Elements	RUP		X				
Design the User-Interface	RUP		X				
Identify Design Elements	RUP		X		X		
Identify Design Mechanisms	RUP		X				
Incorporate Existing Design Elements	RUP		X				
Prototype the User-Interface	RUP		X				
Review the Architecture	RUP		X		X		
Review the Design	RUP		X		X		
Subsystem Design	RUP		X				
Use-Case Analysis	RUP		X		X		
Use-Case Design	RUP		X				
Create Spike Solutions	XP						X
Divide/Combine Task	XP						X
Validate Spike Solutions	XP						X
Write System Metaphor	XP						X
Write Tasks	XP						X

Implementation							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Analyze Runtime Behavior	RUP		X				
Execute Developer Test	RUP	X	X	X	X		
Implement Design Elements	RUP	X	X	X	X		
Implement Developer Test	RUP		X				
Implement Testability Elements	RUP		X				
Integrate Subsystem	RUP		X				
Integrate System	RUP		X		X		
Plan Subsystem Integration	RUP		X				
Plan System Integration	RUP		X				

Review Code	RUP		X		X		
Structure the Implementation Model	RUP		X		X		
Execute Unit Test	XP						X
Implement Tasks	XP						X
Integrate Code	XP						X
Refactor Code	XP						X

Test							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Agree on the Mission	RUP		X				
Analyze Test Failure	RUP		X				
Assess and Advocate Quality	RUP		X				
Assess and Improve Test Effort	RUP		X				
Define Assessment and Traceability Needs	RUP		X				
Define Test Approach	RUP	X	X	X	X		
Define Test Details	RUP	X	X	X	X		
Define Test Environment Configuration	RUP		X				
Define Testability Elements	RUP		X				
Determine Tests Results	RUP		X				
Execute Test Suite	RUP		X				
Identify Targets of Test	RUP		X				
Identify Test Ideas	RUP		X				
Identify Test Motivators	RUP		X				
Identify Testability Mechanisms	RUP		X				
Implement Test	RUP		X				
Implement Test Suite	RUP		X				
Obtain Testability Commitment	RUP		X				
Structure the Test Implementation	RUP		X		X		
Determine Tests Results	XP						X
Execute Acceptance Tests	XP						X
Write Acceptance Tests	XP						X
Write Unit Tests	XP						X

Deployment							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Create Deployment Unit	RUP	X	X	X	X		
Create Product Artwork	RUP		X				
Define Bill of Materials	RUP		X				
Determine Test Results	RUP		X				
Develop Deployment Plan	RUP		X				
Develop Installation Artifacts	RUP		X				
Develop Support Materials	RUP	X	X	X	X		
Develop Training Materials	RUP		X				
Execute Test Suite	RUP		X				
Manage Acceptance Tests	RUP		X		X		
Manage Beta Test	RUP		X				

Provide Access to Download Site	RUP		X				
Release to Manufacturing	RUP		X				
Support Development	RUP		X				
Verify Manufactured Product	RUP		X				
Write Release Notes	RUP		X		X		
Create Release	XP						X

Configuration and Change Management							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Confirm Duplicate or Rejected CR	RUP	X	X	X	X		
Create Baselines	RUP		X	X	X		
Create Deployment Unit	RUP		X	X	X		
Create Development Workspace	RUP		X	X	X		
Create Integration Workspace	RUP		X	X	X		
Deliver Changes	RUP		X	X	X		
Establish Change Control Process	RUP		X	X	X		
Establish CM Policies	RUP		X	X	X		
Make Changes	RUP		X	X	X		
Perform Configuration Audits	RUP		X	X	X		
Promote Baselines	RUP		X	X	X		
Report on Configuration Status	RUP		X	X	X		
Review Change Request	RUP	X	X	X	X		
Setup CM Environment	RUP		X	X	X		
Submit Change Requests	RUP	X	X	X	X		
Update Change Request	RUP	X	X	X	X		
Update Workspace	RUP		X	X	X		
Verify Changes in Build	RUP		X	X	X		
Write CM Plan	RUP		X	X	X		

Project Management							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Develop Project Plan	Outros						
Develop the Schedule	Outros						
Negotiate the Schedule with Team	Outros						
Acquire Staff	RUP		X				
Assess Iteration	RUP		X				
Compile Software Development Plan	RUP	X	X	X	X		
Define Monitoring and Control Processes	RUP		X	X	X		
Define Project Organization and Staffing	RUP		X	X	X		
Develop Business Case	RUP	X	X	X	X		
Develop Iteration Plan	RUP	X	X	X	X		
Develop Measurement Plan	RUP		X	X	X		

Develop Problem Resolution Plan	RUP		X				
Develop Product Acceptance Plan	RUP		X	X	X		
Develop Quality Assurance Plan	RUP		X				
Develop Risk Measurement Plan	RUP		X	X	X		
Handle Exceptions and Problems	RUP		X				
Identify and Assess Risk	RUP	X	X	X	X		
Initiate Iteration	RUP		X				
Initiate Project	RUP	X	X	X	X		
Iteration Acceptance Review	RUP	X	X				
Iteration Evaluation Criteria Review	RUP		X				
Iteration Planning Review	RUP		X	X	X		
Lifecycle Milestone Review	RUP	X	X	X	X		
Monitor Project Status	RUP		X	X	X		
Organize Review	RUP		X	X	X		
Plan Phases and Iterations	RUP		X	X	X		
PRA Project Review	RUP		X	X	X		
Prepare for Phase Close-Out	RUP	X	X	X	X		
Prepare for Project Close-Out	RUP	X	X	X	X		
Project Acceptance Review	RUP	X	X	X	X		
Project Approval Review	RUP		X	X	X		
Project Planning Review	RUP		X				
Report Status	RUP	X	X	X	X		
Schedule and Assign Work	RUP		X	X	X		
Develop Quality Assurance Plan	RUPx			X	X		
Estimate critical computational resources	RUPx			X	X		
Estimate resources and funds	RUPx			X	X		
Project Planning Review	RUPx			X	X		
Report Audits Results	RUPx			X	X		
Software Quality Assurance Activities Review	RUPx			X	X		
Track critical computational resources	RUPx			X	X		
Track Software Quality Assurance Activities	RUPx			X	X		
Conduct the Daily Scrum	Scrum						
Plan the Sprint	Scrum						
Accept Task	XP						X
Collect Project Metrics	XP						X
Define Iteration Scope	XP						X
Define Velocity	XP						X
Estimate User Story	XP						X
Guide the Team	XP						X
Negotiate with Client	XP						X
Report Progress	XP						X

Environment							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Elaborate Process for the Organization	Outros		X				

Review Process for the Organization	Outros						
Review Process for the Project	Outros						
Develop Development Case	RUP		X	X	X		
Develop Manual Style guide	RUP		X		X		
Launch Development Process	RUP		X		X		
Prepare Guidelines for the Project	RUP		X		X		
Prepare Templates for the Project	RUP		X				
Select & Acquire Tools	RUP	X	X	X	X		
Set Up Tools	RUP	X	X	X	X		
Support Development	RUP		X				
Tailor the Process for the Project	RUP	X	X	X	X		
Verify Tool Configuration and Installation	RUP		X		X		
Assess Current Organizational	RUPx				X		
Develop Process Improvement Plan	RUPx				X		
Organizational Assessment Review	RUPx				X		
Process Improvement Plan Review	RUPx				X		

Subcontract Management							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Monitor Project Status (Ger. Projetos)	RUP			X	X		
Perform Configuration Audits (GCS)	RUP			X	X		
PRA Project Review	RUP			X	X		
Assess Artifacts Developed	RUPx			X	X		
Assess Artifacts Developed	RUPx			X	X		
Develop a Subcontract Management Plan	RUPx			X	X		
Elaborate Contract and Define Commitments	RUPx			X	X		
Select Subcontractor	RUPx			X	X		

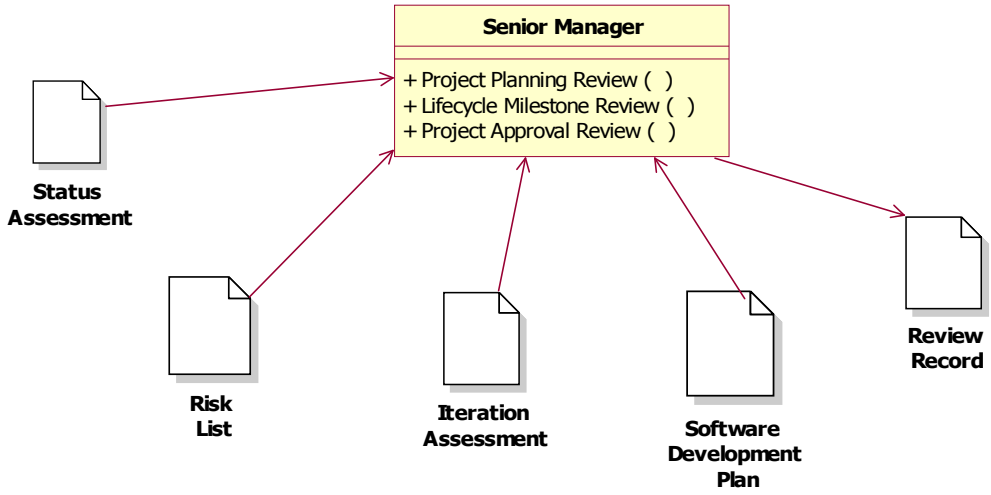
Training							
Activities	**Process**	**Essencial RUP**	**RUP**	**RUP/ CMM 2**	**RUP/ CMM 3**	**XP/ CMM**	**XP**
Identify Training	RUPx				X		
Assess Training Plan	RUPx				X		
Execute Training	RUPx				X		
Assess Concluded Training	RUPx				X		

ANEXO D PADRÕES ORGANIZACIONAIS E DE PROCESSO

Essa seção descreve os padrões de processo descritos segundo o metamodelo integrado proposto por esta tese.

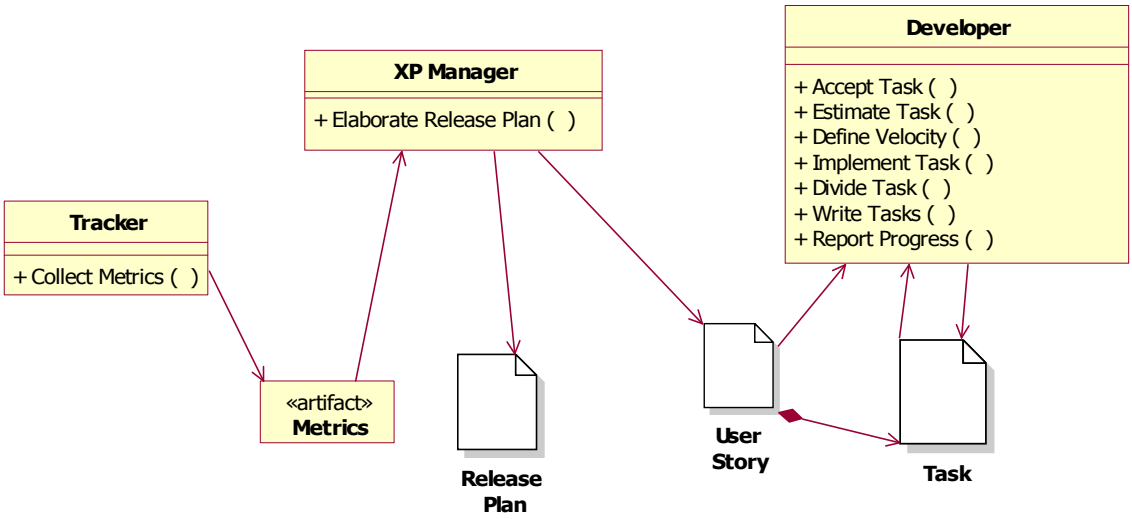
1. SeniorManagementReview

- **Descrição:**
  - O gerente sênior deve participar das reuniões de acompanhamento nos marcos dos projetos e ser informado a execução das atividades dos projetos da organização periodicamente.
- **Fonte:** Capability Maturity Model (PAULK, 1993-a)



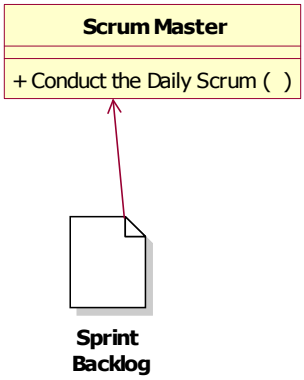
2. IterationPlanning

- **Descrição:**
  - Após o cliente ter priorizado as *User Stories* (*Planning Game*), o XP Manager (*coach*) agenda uma reunião para planejamento da iteração (*Iteration Planning*), onde será elaborado o plano do release (*Release Plan*). Essa reunião ocorre no início de cada iteração. O plano do release especifica exatamente quais tarefas serão implementadas em cada iteração. Uma iteração dura de uma a quatro semanas.
- **Fonte:** Extreme Programming (BECK, 99), Portland Pattern Repository (CUNNINGHAM, 2004)



3. DailyScrumMeeting

- **Descrição:**
  - Cada equipe Scrum encontra-se diariamente para uma reunião de 15 minutos para discutir o status do projeto. Essa reunião é chamada Daily Scrum Meeting.
  - No Daily Scrum, o progresso é revisado e são identificados impedimentos para serem removidos pelo gerenciamento. O Daily Scrum é um excelente lugar para observar quanto de progresso a equipe está fazendo.
- **Fonte:** SCRUM (SCHWABER, 2001)



4. Planning Game

- **Descrição:**

O objetivo dessa prática é determinar o escopo da iteração seguinte. Os usuários decidem escopo e prioridades do próximo release. A equipe técnica, por outro lado, decide estimativas, conseqüências técnicas, processo de desenvolvimento e detalha cronograma.

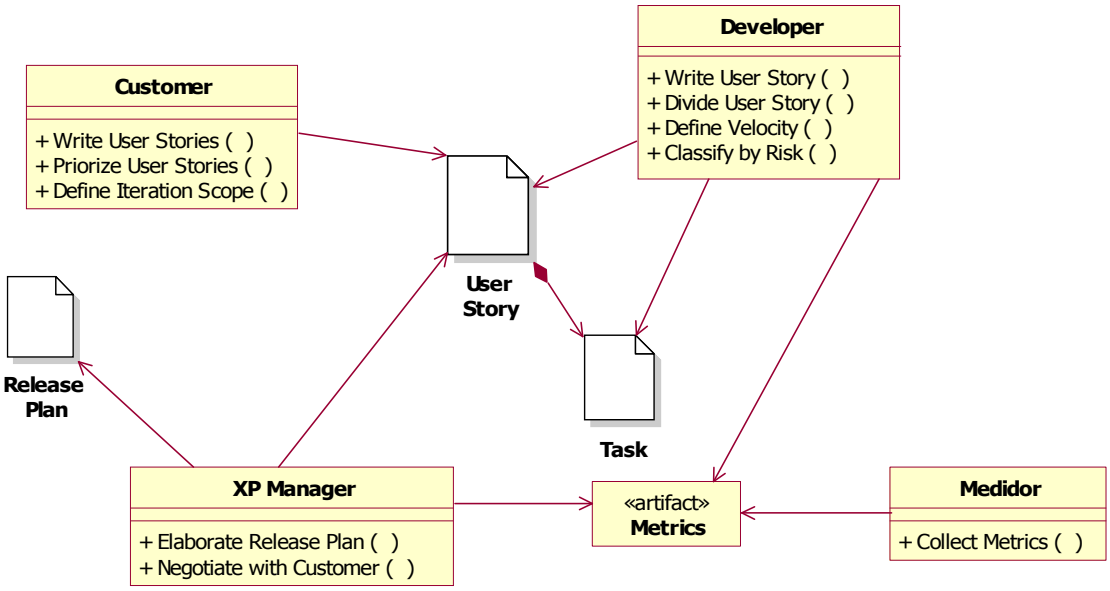
Regras básicas:

1. O cliente apresenta uma lista de funcionalidades desejadas para o sistema. Cada funcionalidade é escrita como um ***User Story*** **(US)**. Dá-se um nome para o *User Story* e descreve-se rapidamente a funcionalidade requerida, normalmente em cartões indexados (*index cards*) 4X6.

2. O desenvolvimento estima quanto de esforço cada US exige, e quanto de esforço a equipe pode produzir em um determinado intervalo de tempo (iteração).

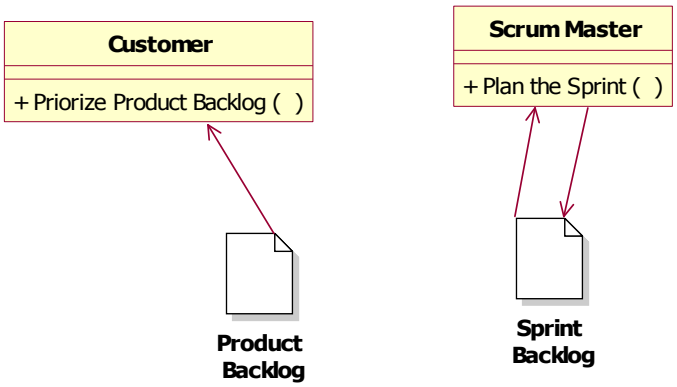
3. O cliente decide quais US devem ser implementados e as prioridades. O cliente também deve definir a freqüência dos *releases* de produção para o sistema.

- **Fonte:** Extreme Programming (BECK, 99), Portland Pattern Repository (CUNNINGHAM, 2004)



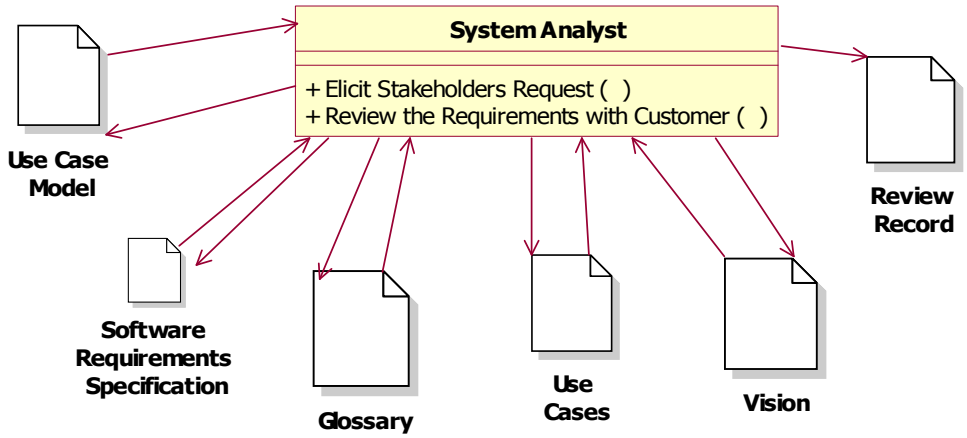
5. SprintPlanningMeeting

- **Descrição:**
  - Em processos Scrum ocorre uma reunião de planejamento a cada 30 dias, quando se inicia um novo *Sprint*. Nessa reunião se reúnem os clientes, os usuários, o gerenciamento, o dono do produto e a equipe Scrum para determinar os objetivos e as funcionalidades do próximo *Sprint*.
  - A reunião de planejamento *Sprint* consiste de duas reuniões consecutivas. Na primeira reunião, a equipe encontra-se com o dono do produto, gerenciamento, e usuários para definir quais funcionalidades serão construídas no próximo *Sprint*. Nessa reunião a equipe e o dono do produto concordam com as funcionalidades do Backlog do produto que serão implementadas no próximo *Sprint*. Na segunda reunião, a equipe se reúne para definir como construirá as funcionalidades requeridas no próximo incremento do produto.
- **Fonte:** SCRUM (SCHWABER, 2001)



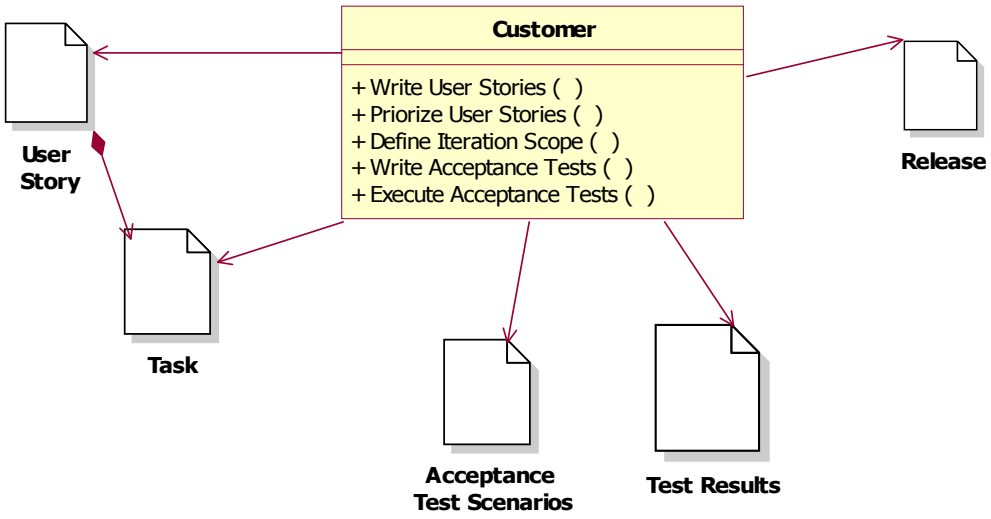
6. EngageCustomer

- **Descrição:**
  - Assegura e mantém a satisfação dos clientes por encorajar a participação do cliente no projeto.
- **Fonte:** James Coplien (COPLIEN, 1996)



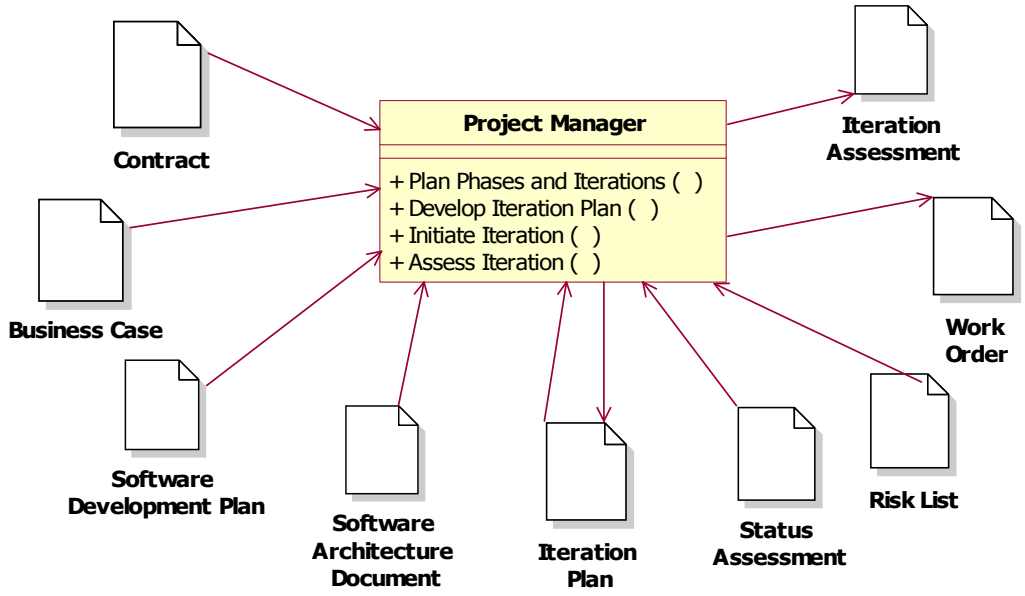
7. OnSiteCustomer

- **Descrição:**
  - O cliente deve estar disponível quando os desenvolvedores precisam solucionar dúvidas, definir prioridades, e resolver pendências. Se possível, o cliente deve trabalhar na mesma sala dos desenvolvedores. O cliente deve estar disponível para esclarecer *user stories*. O cliente deve participar do *Planning Game*. Somente o cliente pode criar ou priorizar *user stories*.
- **Fonte:** Extreme Programming (BECK, 99), Portland Pattern Repository (CUNNINGHAM, 2004)



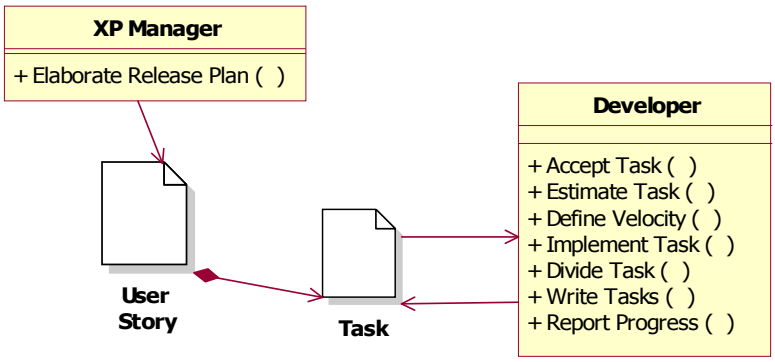
8. EarlyAndRegularDeliveryRUP

- **Descrição:**
  - Desenvolvimento incremental, onde versões do software são entregues freqüentemente ao cliente. Adaptação do padrão EarlyAndRegularDelivery, proposto por James Coplien, para processos RUP.
- **Fonte:** James Coplien (COPLIEN, 1996)



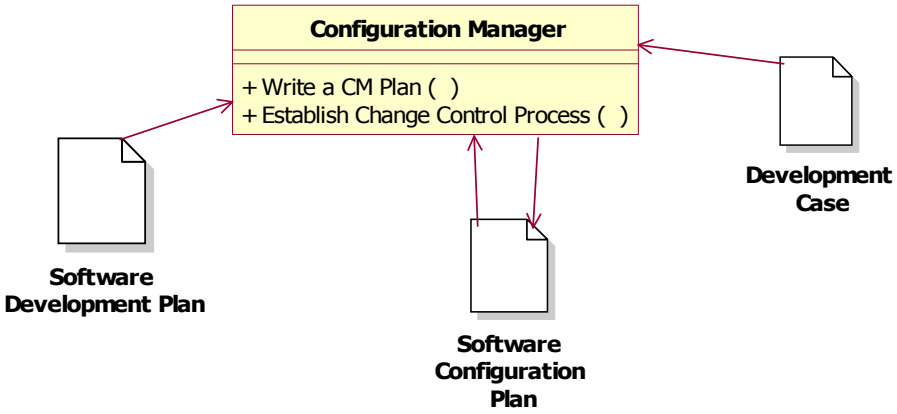
9. EarlyAndRegularDeliveryXP

- **Descrição:**
  - Desenvolvimento incremental, onde versões do software são entregues freqüentemente ao cliente. Adaptação do padrão EarlyAndRegularDelivery, proposto por James Coplien, para processos XP.
- **Fonte:** James Coplien (COPLIEN, 1996)



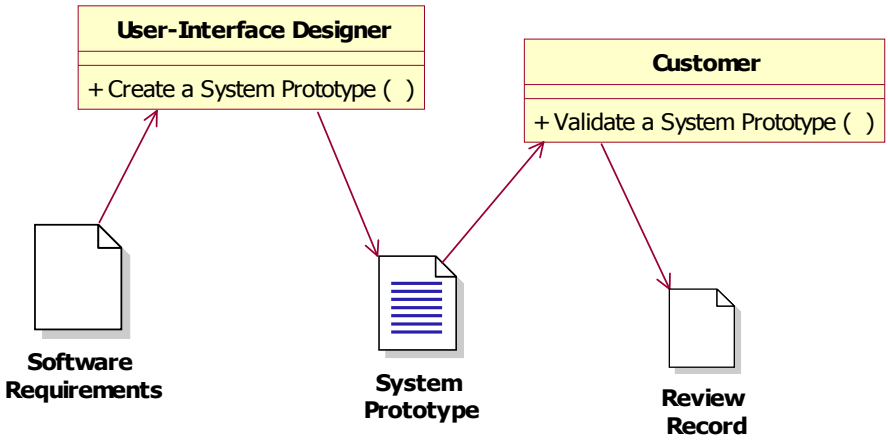
10. DocumentedConfigurationManagementPlan

- **Descrição:**
  - Um plano de gerência de configuração de software (GCS) é elaborado para cada projeto de acordo com um procedimento documentado. Este procedimento especifica que o plano deve ser desenvolvido nos estágios iniciais do projeto, e em paralelo, com o planejamento do projeto; o plano é revisado por todos os grupos afetados; o plano é gerenciado e controlado.
  - O plano de GCS documentado e aprovado é usado como base para executar as atividades de gerência de configuração de software. O plano cobre as atividades que serão executadas, o cronograma das atividades, responsáveis e recursos necessários.
- **Fonte:** Capability Maturity Model (PAULK, 1993-a)



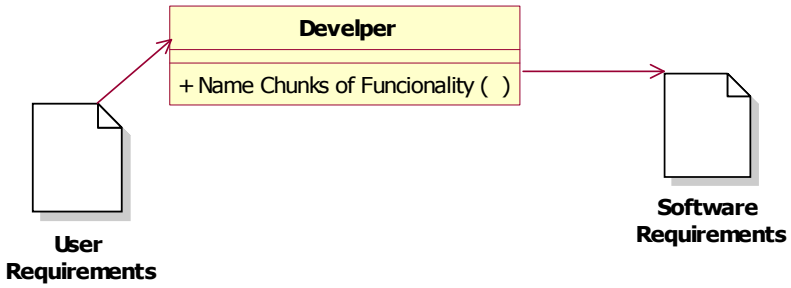
11. BuildPrototype

- **Descrição:**
  - Elaborar um protótipo do sistema para entender, validar e explorar o custo e benefícios de decisões de projeto. Validar o protótipo, junto ao cliente, visando definir os requisitos do sistema.
- **Fonte:** James Coplien (COPLIEN, 1996)



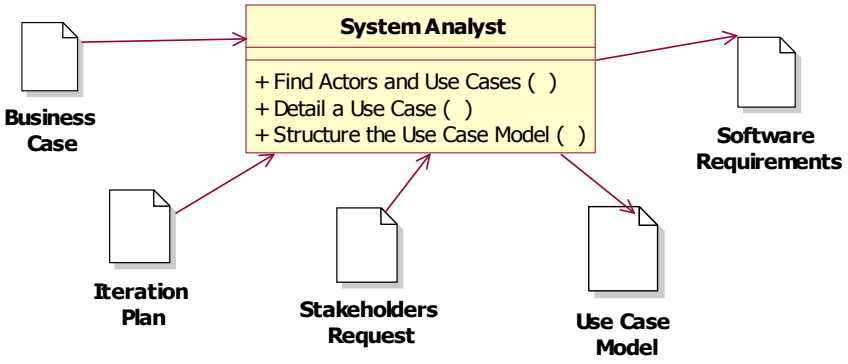
12. ImpliedRequirement

- **Descrição:**
  - Selecione e atribua um nome para um conjunto de funcionalidades relacionadas.
  - No início do processo de definição de requisitos, os requisitos são necessários para delimitar o escopo e estimar o software. Nomeie conjuntos de funcionalidades que representam os requisitos detalhados implícitos.
- **Fonte:** James Coplien (COPLIEN, 1996)



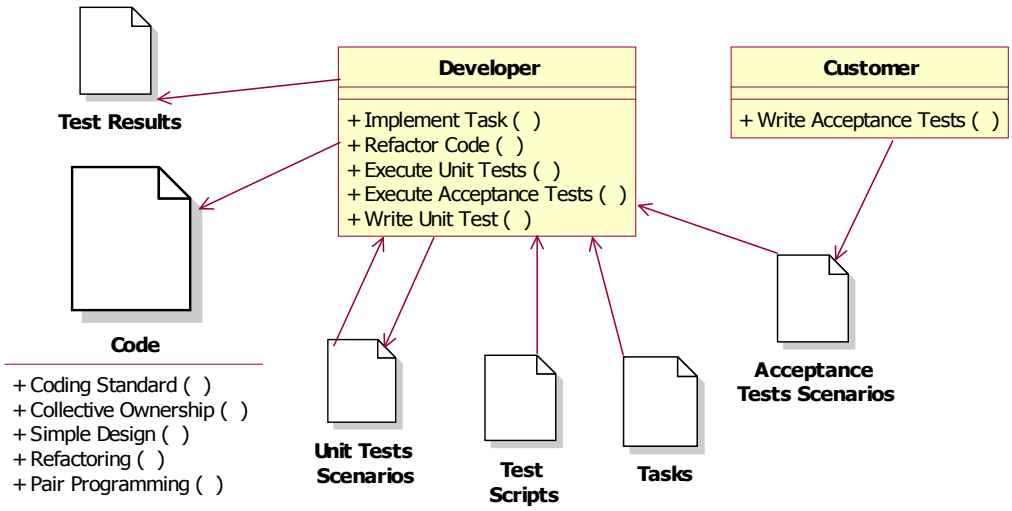
13. ScenariosDefineProblem

- **Descrição:**
  - Documentos de projeto são inefetivos para comunicar a visão do cliente de como o sistema deve funcionar. Para engajar o cliente no desenvolvimento do software, capture os requisitos funcionais como casos de uso, conforme proposto por Jacobson.
- **Fonte:** James Coplien (COPLIEN, 1996)



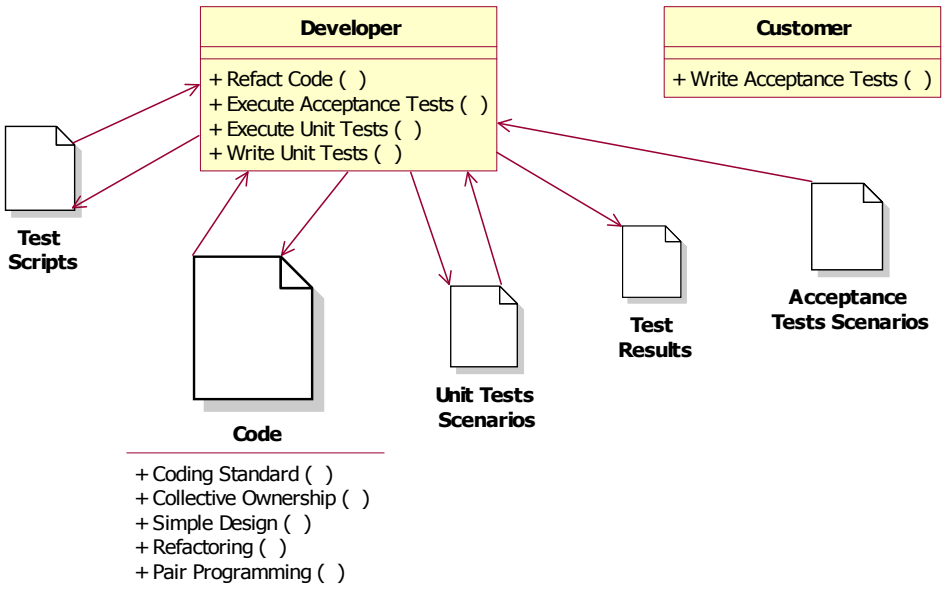
14. SimpleDesign

- **Descrição:**
  - Implemente os requisitos atuais da forma mais simples possível. XP apresenta três diretrizes ou guias com essa finalidade: considere a implementação mais simples que pode funcionar, implemente somente o código necessário para suportar os requisitos, não duplique código.
- **Fonte:** Extreme Programming (BECK, 99), Portland Pattern Repository (CUNNINGHAM, 2004)



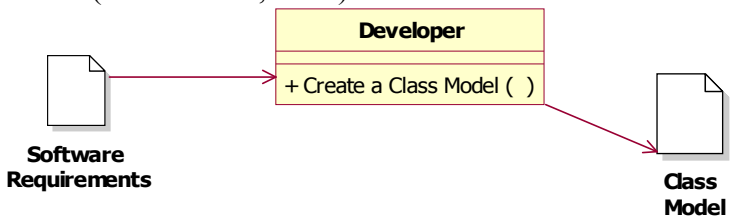
15. ConstantRefactoring

- **Descrição:**
  - Melhorar continuamente a estrutura do código. *Refactoring* é o processo de mudar a estrutura do código sem mudar a sua semântica, isto é, sem mudar sua funcionalidade.
- Fonte: Fowler (FOWLER, 2000)



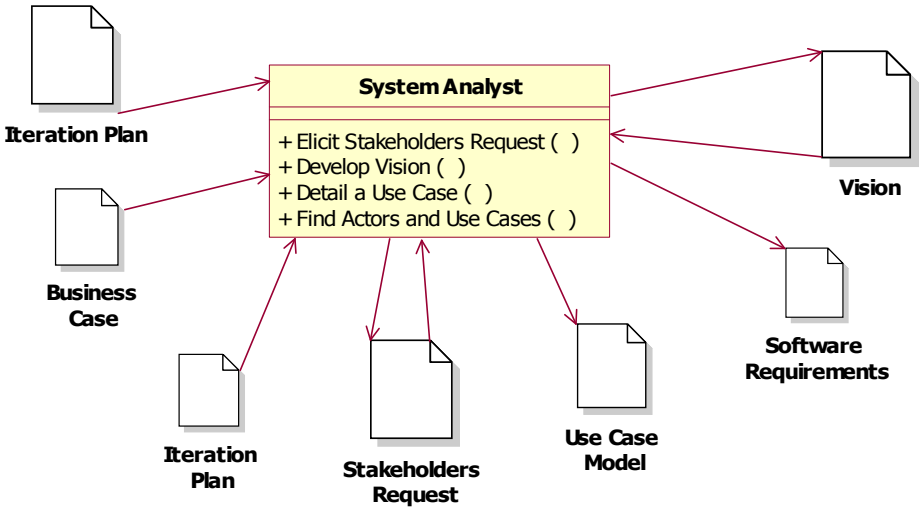
16. ModelTheDomain

- **Descrição:**
  - Crie modelos de classe que descrevem conceitos do domínio, seus inter-relacionamentos, colaborações e restrições.
- **Fonte:** Paul Oldfield (OLDFIELD, 2002)



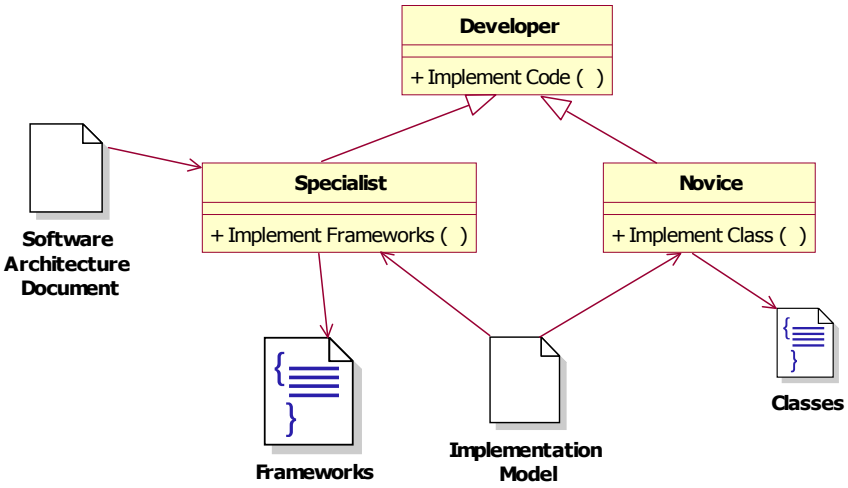
17. JointApplicationDesign

- **Descrição:**
  - Assegure que todos os envolvidos interessados estão representados quando os requisitos são elicitados. Conflitos nos requisitos podem ser detectados e resolvidos nos estágios iniciais do projeto, se cada stakeholder ouvir o que outros stakeholders estão solicitando, podendo intervier para esclarecer os requisitos.
- **Fonte:** Mei C. Yatco (YATCO, 1999)



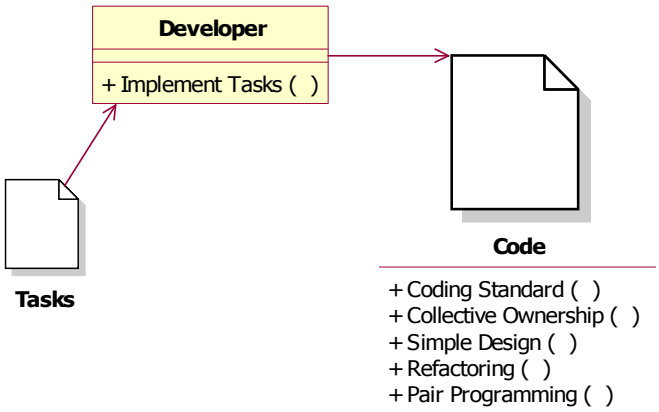
18. GenericsAndSpecifics

- **Descrição:**
  - Reunir pessoas com diferentes níveis de habilidade em projetos OO
  - Separe partes genéricas de partes específicas de um problema. Use um especialista, projetista de *framework* para projetar partes projeto genéricas. Deixe os programadores novatos projetar as partes específicas.
- **Fonte:** James Coplien (COPLIEN, 1996)



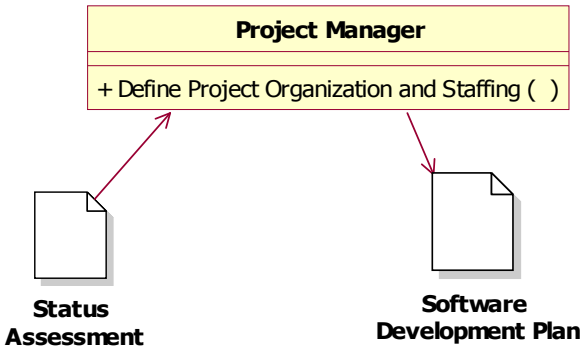
19. DevelopingInPairs

- **Descrição:**
  - Existem muitos benefícios no desenvolvimento em pares, tais como: constante comunicação, revisão em pares, flexibilidade, transferência de habilidades, disseminação de conhecimento, entre outros.
  - Todo o código de produção deve ser desenvolvido por duas pessoas trabalhando juntas no mesmo computador. Enquanto um dos programadores digita o código o outro pensa nas funcionalidades, fica atento a *bugs* e cuida para que a codificação esteja no rumo certo.
- **Fonte:** Extreme Programming (BECK, 99), Portland Pattern Repository (CUNNINGHAM, 2004)



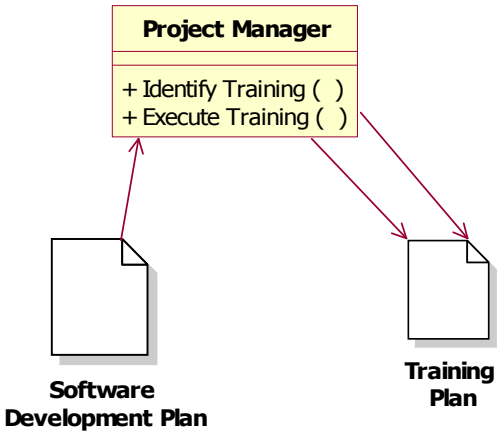
20. DayCare

- **Descrição:**
  - Quando os especialistas da equipe estão gastando muito tempo auxiliando os novatos; dedique um dos especialistas para auxiliar os novatos, e deixe os outros desenvolverem o sistema.
- **Fonte:** James Coplien (COPLIEN, 1996)



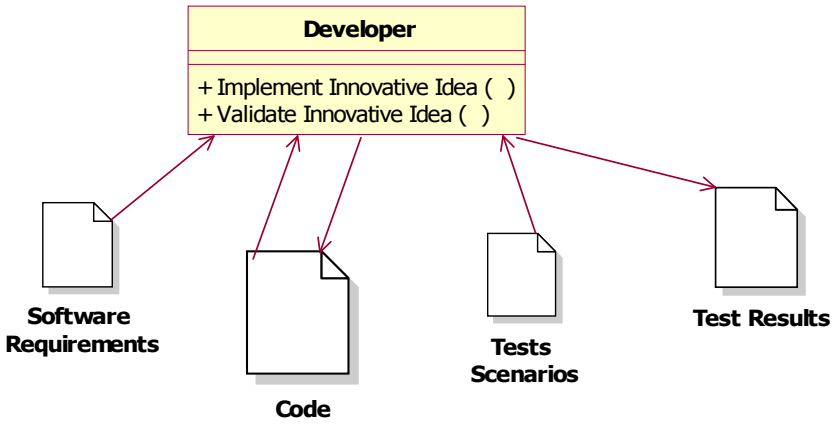
21. ApprenticeShip

- **Descrição:**
  - Treine os novos contratados, por meio de um programa de treinamento, para estes se tornarem especialistas.
- **Fonte:** James Coplien (COPLIEN, 1996)



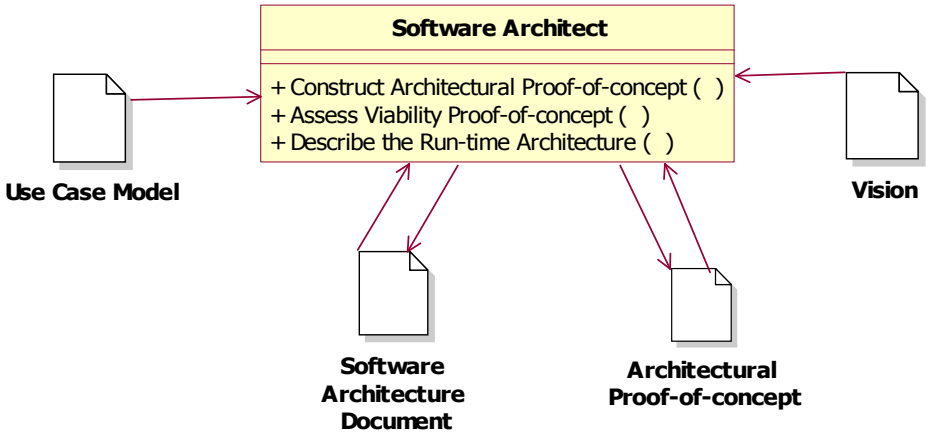
22. SkunkWorks

- **Descrição:**
  - Tenha uma equipe de especialistas desenvolvendo uma idéia inovadora em um contexto externo ao projeto de software e as restrições de desenvolvimento de projeto, para obter segurança na implementação da idéia.
  - Isole o objetivo principal do projeto das idéias inovadoras até que elas tenham sido avaliadas e os riscos minimizados. Um produto precisa de inovação para permanecer competitivo, mas não deve arriscar o projeto inteiro, introduzindo inovação diretamente e sem testar e avaliar previamente essas novas soluções.
- **Fonte:** James Coplien (COPLIEN, 1996)



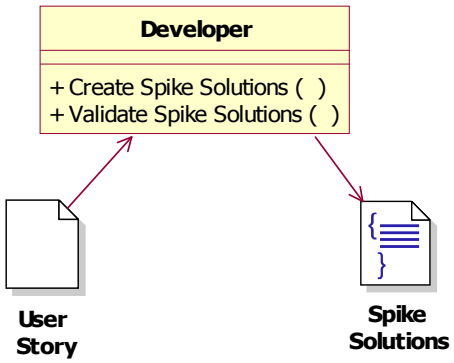
23. ArchitectureTeam

- **Descrição:**
  - Contexto: é necessário construir uma arquitetura simples e coesa, para acomodar os componentes constituintes do sistema.
  - Solução: crie uma pequena equipe para definir a arquitetura inicial, de maneira que a equipe tenha conhecimento de todas as partes do sistema. Membros da equipe devem ser responsáveis por suas próprias partes do sistema e devem trabalhar com outros membros para definir a arquitetura completa.
- **Fonte:** James Coplien (COPLIEN, 1996)



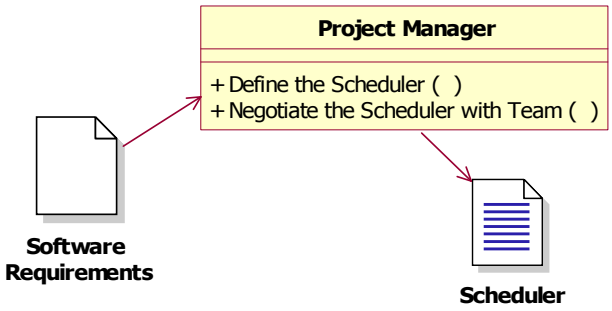
24. SpikeSolutions

- **Descrição:**
  - SpikeSolutions são soluções rápidas. Servem para reduzir riscos e testar a viabilidade de uma solução. Um *spike* é um código em que se testa apenas uma solução, ignorando todo o resto do programa. Ou seja, isolando o problema.
  - Na prática, Spike Solutions são usadas para atacar problemas tecnológicos.
- **Fonte:** Extreme Programming (BECK, 99), Portland Pattern Repository (CUNNINGHAM, 2004)



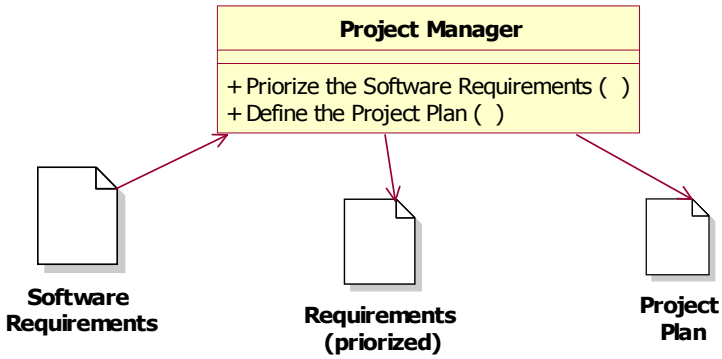
25. SizeTheSchedule

- **Descrição:**
  - O trabalho a ser realizado é definido e o tamanho do projeto é estimado. Negocie o cronograma do projeto com a equipe de desenvolvimento.
- **Fonte:** James Coplien (COPLIEN, 1996)



26. WorkQueue

- **Descrição:**
  - Produza uma lista priorizada dos requisitos implícitos. Detalhe os requisitos, analise, projete e implemente na ordem da lista priorizada.
- **Fonte:** James Coplien (COPLIEN, 1996)

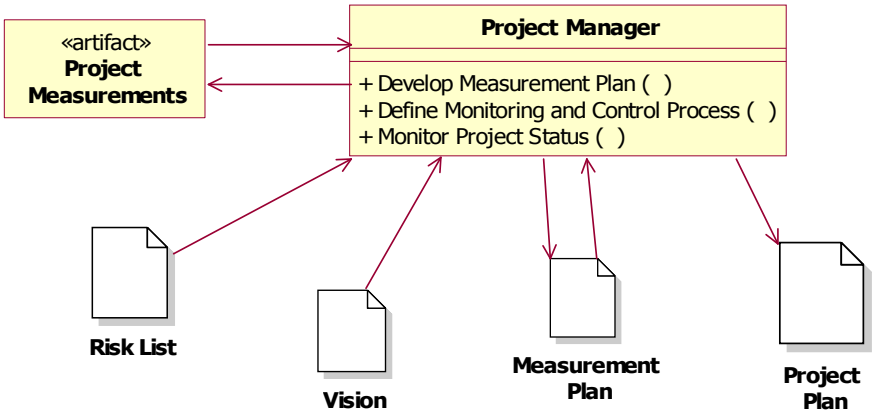


27. DocumentedSoftwareEstimates

- **Descrição:**
  - As estimativas de software são usadas no planejamento e acompanhamento de projetos de software. Estimativas de tamanho, custo, tempo, recursos críticos e fundos são derivadas de acordo com procedimentos documentados.

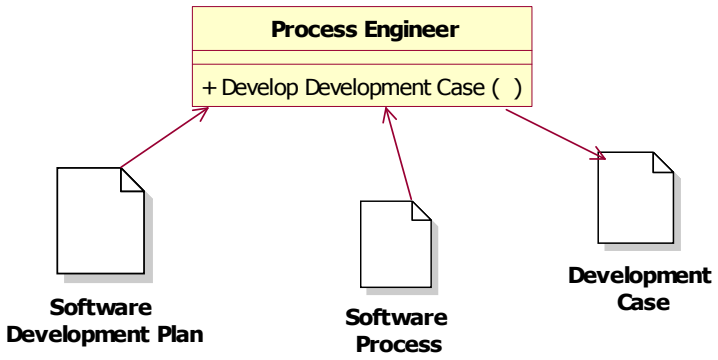
Estimativas devem ser documentadas em uma base histórica e armanzenadas em um banco de dados.

- **Fonte:** Capability Maturity Model (PAULK, 1993-a)



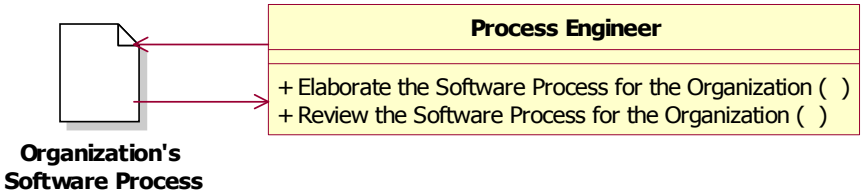
28. SoftwareLifecycleIsDefined

- **Descrição:**
  - Definir um ciclo de vida com estágios pré-definidos e de tamanho gerenciável para o projeto.
- **Fonte:** Capability Maturity Model (PAULK, 1993-a)



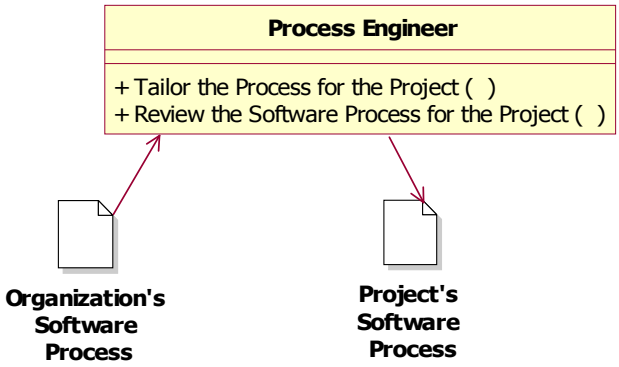
29. DefinedProcess

- **Descrição:**
  - Elabore um processo de software para descrever as tarefas que precisam ser executadas para desenvolver o software. Um processo definido possibilita a melhoria do processo. Trate as tarefas de rotina e as tarefas criativas de forma diferente.
- **Fonte:** Portland Pattern Repository (CUNNINGHAM, 2004)



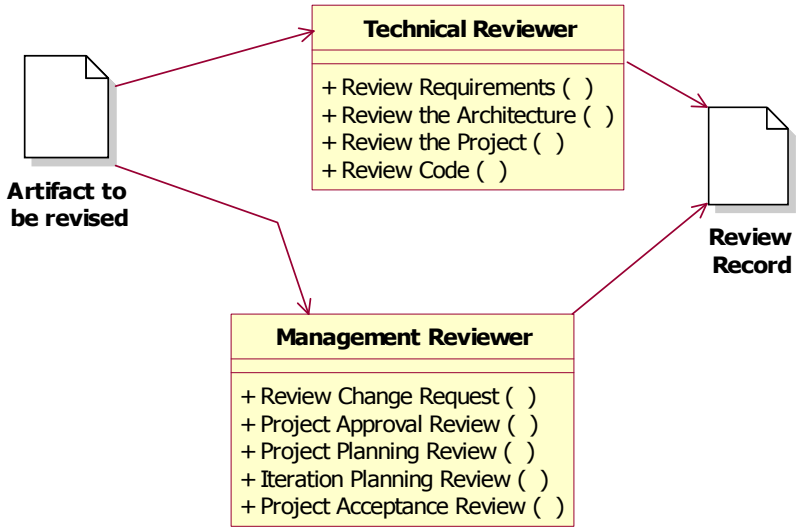
30. ProjectProcessIsDefined

- **Descrição:**
  - Para cada projeto deve existir uma definição operacional do processo que será usado no projeto. O processo de software definido do projeto é descrito em termos de padrões, procedimentos, procedimentos, ferramentas e métodos.
- **Fonte:** Capability Maturity Model (PAULK, 1993-a)



31. PeerReviews

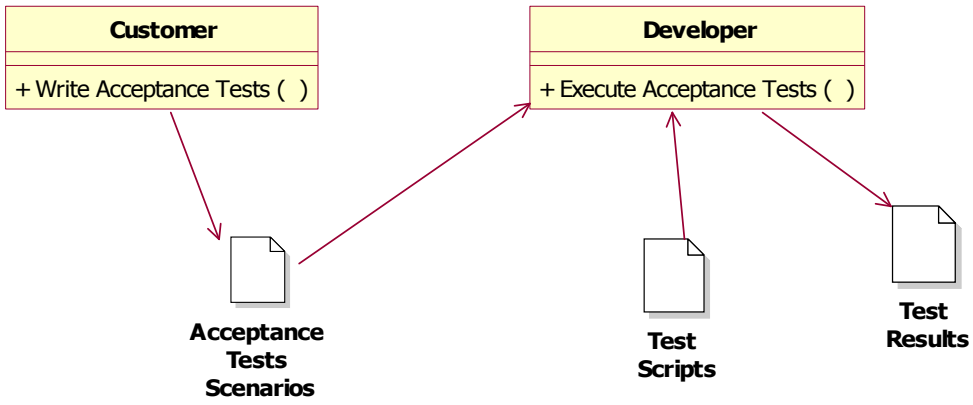
- **Descrição:**
  - Executar revisões por pares. As revisões por pares envolvem um exame metodológico de produtos de trabalho de software para identificar defeitos e áreas onde mudanças são necessárias.
- **Fonte:** Capability Maturity Model (PAULK, 1993-a)



32. AcceptanceTests

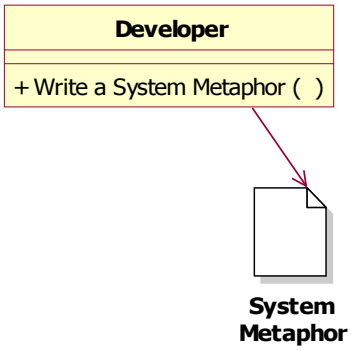
- **Descrição:**
  - Para cada *User Story* deve existir pelo menos um teste de aceitação. Os testes de aceitação são especificados pelo cliente e implementados, com execução automática, pelos desenvolvedores. Uma *User Story* é considerada terminada somente quando todos os seus testes de aceitação executarem completamente.

- **Fonte:** Extreme Programming (BECK, 99), Portland Pattern Repository (CUNNINGHAM, 2004)



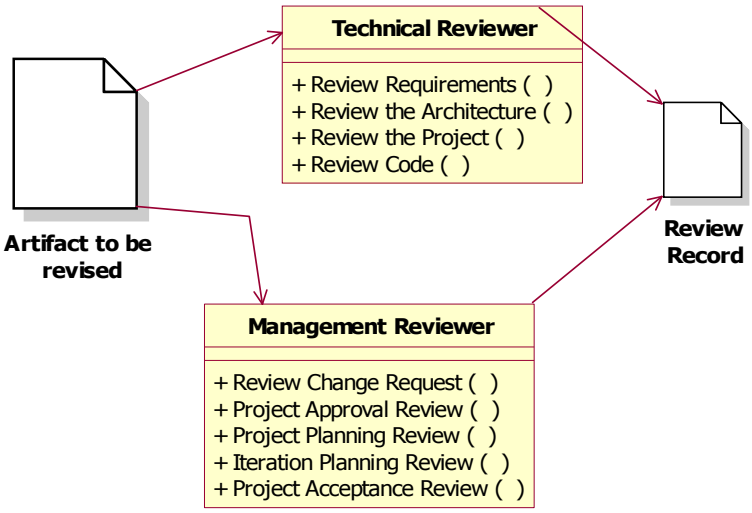
33. SystemMetaphor

- **Descrição:**
  - Em XP, a arquitetura do sistema é descrita como uma metáfora. Uma metáfora é uma história compartilhada entre a equipe de como o sistema funciona. Essa história tipicamente envolve várias classes e padrões que formam o fluxo principal do sistema sendo construído.
- **Fonte:** Extreme Programming (BECK, 99), Portland Pattern Repository (CUNNINGHAM, 2004)



34. GroupValidation

- **Descrição:**
  - A equipe de desenvolvimento valida o projeto
- **Fonte:** James Coplien (COPLIEN, 1996)



35. DocumentedSubcontractManagementPlan e SubcontractManager

- **Descrição:**
  - Definir um processo de gerência de subcontratação – definir marcos de acompanhamento das atividades ao longo do projeto para garantir que os prazos e os custos estejam sendo cumpridos, e verificar a qualidade dos produtos desenvolvidos por terceiros. Designar um gerente de subcontratação para gerenciar as atividades da subcontratada.
- **Fonte:** Capability Maturity Model (PAULK, 1993-a)