UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

ATIVIDADES DE INSPEÇÃO NO CONTEXTO DE MÉTODOS ÁGEIS

KARINA MITIKO TOMA

São Carlos/SP

Agosto/2004

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

T655ai

Toma, Karina Mitiko.
Atividades de inspeção no contexto de métodos ágeis / Karina Mitiko Toma. -- São Carlos : UFSCar, 2007.
149 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2004.

1. Engenharia de software. 2. Métodos ágeis. 3. Inspeção de software. 4. Pacotes de laboratório. 5. Técnicas de leitura. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Atividades de Inspeção no Contexto de Métodos Ágeis"

KARINA MITIKO TOMA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Profa. Dra. Sandra Camargo Pinto Ferraz Fabbri
(Orientadora - DC/UFSCar)

Profa. Dra. Rosângela Ap. Dellosso Penteado
(DC/UFSCar)

Prof. Dr. Mauro de Mesquita Spinola
(POLI/USP)

São Carlos
Agosto/2004

Aos meus pais

Choiti e Keiko,

Às minhas irmãs

Denise e Regina e,

Ao meu amado

Claudio

pelo amor e compreensão,

dedico, com carinho, este trabalho.

AGRADECIMENTO ESPECIAL

---

*À Profa. Dra**. Sandra Camargo P. F. Fabbri,***

orientadora, "mestre", companheira e amiga, nos distintos momentos dessa caminhada. Aprendi a conhecê-la, respeitá-la e admirá-la nesses dois anos e meio de convívio. A certeza de que levo comigo seu entusiasmo pelo ensino, pautado nos princípios de sua vida de dedicação e amor à computação.

"Cada um que passa em nossa vida, passa sozinho, porque cada pessoa é única e para nós, nenhuma substitui a outra.
Cada um que passa em nossa vida, passa sozinho, mas não vai sozinho, nem nos deixa a sós...
Leva um pouco de nós mesmos e deixa um pouco de si mesmo.
Há os que levam muito, mas não há os que levam nada.
Há os que deixam muito, mas não há os que deixam nada.
Esta é a mais bela responsabilidade de nossa vida: a prova tremenda de que cada um é importante e de que ninguém se aproxima do outro por acaso."

(Anônimo)

AGRADECIMENTOS

À Deus, por fazer-me sentir, a todo o momento, amada e especial. Que eu possa sempre ser um pontinho da sua luz reconfortante àqueles que me cercam.

Ao ***Solo Fértil*** da Universidade Federal de São Carlos – UFSCar, pelos exemplos oferecidos através das inúmeras safras aqui colhidas anteriormente; pela oportunidade de contar com a precisa, amiga e experiente orientação da Professora Dra. Sandra C. P. F. Fabbri; e pela possibilidade de crescimento sempre sob a incansável regência dos Professores do Departamento de Computação.

Foi a fertilidade deste ***Solo*** que fez brotar a idéia de
comparar esta ***Dissertação*** a uma ***Colheita***,

colheita como sinônimo de felicidade, glória, conquista, progresso, missão cumprida, motivação para novas etapas...

este ar, inspirado neste instante, exala todos estes predicados... Quero dividir este momento com os ***amigos*** presentes durante a ***semeadura***:

Nori, Anderson, Ricardo, Frank, Karina, Pablo, Dona Eurides, Fernanda1, Luciana, Rafaela, Fernanda2, Margareth, Mariá, à todos os amigos conquistados durante esse período e aos funcionários do Departamento de Computação.

No entanto, graças à inquietação natural do homem, a que está por surgir será ainda mais farta, muito embora, neste momento estejamos oferecendo o nosso melhor.

O próximo passo, objetivando uma ***nova colheita***, será a ***seleção*** das melhores ***sementes***. Assim, em breve, estaremos ***plantando*** daquilo que já colhemos, ***sonhando*** com a possibilidade de garantir, sempre, uma ***boa safra***.

Hoje é o dia de uma colheita!!!
Obrigada a todos!!!

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Atividades de inspeção de software têm se tornado uma alternativa importante para avaliar artefatos de software a fim de alcançar uma maior qualidade no processo de desenvolvimento. Para apoiar essas atividades utilizam-se, em geral, técnicas de leitura como as OORTs/ProDeS que constituem um conjunto de técnicas para o paradigma Orientado a Objetos, baseando-se em um processo específico de desenvolvimento que utiliza a notação UML, denominado ProDeS/UML. Esse processo contempla um desenvolvimento gradativo, o que facilita a avaliação dos artefatos conforme eles são gerados e também a evolução dos mesmos. No entanto, esse processo é mais voltado para um desenvolvimento tradicional, no qual as informações devem estar bem documentadas, o que exige a elaboração de diversos tipos de artefatos. Considerando-se a abordagem mais atual dos Métodos Ágeis, o primeiro objetivo deste trabalho foi avaliar as características desses métodos e tornar o processo ProDeS/UML mais prático em termos de quantidade de documentação gerada, baseando-se nos artefatos utilizados por esses próprios métodos ou por outras iniciativas encontradas na literatura, de utilizá-los em conjunto com outras abordagens mais tradicionais. Estabelecido então o processo ProDeS/UML|pr, o segundo objetivo foi verificar quais as técnicas de leitura do conjunto OORTs/ProDeS permaneciam passíveis de aplicação nesse processo mais prático e elaborar para elas Pacotes de Laboratório que permitam com que essas técnicas possam ser validadas por meio de estudos experimentais. Finalmente, o terceiro objetivo, decorrente do estudo realizado até então, foi explorar atividades de inspeção para o método ágil mais utilizado na prática – o XP (*Extreme Programming*), estabelecendo uma estratégia de inspeção composta por um conjunto de técnicas de leitura que podem ser aplicadas no dia a dia, de forma ágil, condizente com o método, abordando principalmente as atividades de elicitação de requisitos. Como resultado do trabalho tem-se então o processo ProDeS/UML|pr, os Pacotes de Laboratório das técnicas de leitura que apóiam atividades de inspeção nesse processo e uma estratégia de inspeção para o XP. No contexto deste trabalho o Pacote de Laboratório de uma das técnicas de leitura para o processo ProDeS/UML|pr foi avaliado por um grupo de estudantes e mostrou-se apropriado para utilização em estudos experimentais controlados.

ABSTRACT

Software inspection activities have became an important alternative to evaluate software artifacts in order to reach more quality in the development process. To support such activities, generally, reading techniques as OORTs/ProDeS are used, which is a set of techniques for the Object Oriented paradigm. These techniques are applied in a specific development process that uses UML notation, named ProDeS/UML. This process contemplates a gradual development, which facilitates artifacts evaluation as they are generated as well as its evolution. However, this process is directed to a traditional development, where information must be well documented that demands elaboration of different types of artifacts. Taking into account the most recent approach of Agile Methods, the first objective of this work was to evaluate the characteristics of these methods and to make the ProDeS/UML process more practical, in relation to the quantity of generated documentation. This task was realized based on the artifacts used by the main agile methods or other initiatives found in literature that use them jointly with other more traditional approaches. Once established the ProDeS/UML|pr process, the second objective was to verify which reading techniques of the OORTs/ProDeS set remained feasible to apply in this practical process and to elaborate laboratory packages that allow validating this set through experimental studies. Finally, the third objective, derived from the study of these methods, was to explore inspection activities for XP (Extreme Programming) that is considered in practice the most used agile method. An inspection strategy composed of a reading technique set that considers the requirements elicitation activities of XP was established. This strategy can be applied daily, with agility, according to the objectives of this method. Thus, the results of this work are the ProDeS/UML|pr process, the laboratory packages of the reading techniques that support inspection activities for this process and an inspection strategy for XP. In the scope of this work the laboratory package of one of the reading techniques for the ProDeS/UML|pr was evaluated through a feasibility study that was assessed by a group of students. According to this study the lab package seemed appropriate for controlled empirical studies.

CAPÍTULO 1

INTRODUÇÃO

1.1 Contexto

Sistemas de software estão em todos os lugares: nos equipamentos eletrônicos, nas indústrias, nas escolas, no setor da saúde, etc. Em decorrência disso, a responsabilidade de se construir um software cada vez mais confiável e livre de defeitos tem se tornado essencial, principalmente em sistemas nos quais vidas humanas estão em risco.

Dessa forma, atividades de Garantia de Qualidade de Software, mais especificamente, atividades de VV&T (Validação, Verificação e Teste) tornam-se cada vez mais importantes para assegurar que ao longo do processo de desenvolvimento de software os artefatos gerados atendam os requisitos de qualidade necessários e estejam compatíveis com a especificação do usuário, de forma a gerar o produto esperado e com qualidade (SOMMERVILLE, 2003).

Dentre as atividades de VV&T, a Inspeção de Software é uma atividade estática para verificar se o software atende os seus requisitos (FAGAN, 1976, 1986). Ela pode ser aplicada em todos os estágios do processo de desenvolvimento, possibilitando que os defeitos sejam detectados, preferencialmente, nas próprias fases em que eles foram inseridos, evitando-se a procura de defeitos somente na fase de teste e manutenção, nas quais o custo de correção torna-se muito mais alto.

Uma das técnicas que se usa na atividade de inspeção para auxiliar o inspetor a detectar os defeitos com maior facilidade é a técnica de leitura. Uma técnica de leitura pode ser definida como uma série de passos ou procedimentos cujo propósito é orientar o inspetor a adquirir um entendimento profundo do software inspecionado (LAITENBERGER, 2001). A compreensão do software inspecionado é um pré-requisito para detecção de defeitos, sejam eles complexos ou não. Existem vários tipos de técnicas de leitura, usados em vários artefatos gerados por diversos processos de desenvolvimento de software.

Como o paradigma OO (Orientado a Objetos) tem sido muito utilizado no desenvolvimento de software, técnicas de leitura voltadas para este paradigma são muito importantes. Um conjunto de técnicas com esse propósito foi definido por Travassos et al. (2002b, 2002c), as OORTs (*Object Oriented Reading Techniques*), para serem usadas em alguns artefatos representados pela notação UML (*Unified Modeling Language*), que se aplicam à fase de projeto, considerando um processo de desenvolvimento de software OO genérico e simplificado.

Considerando a relevância de se abordar o paradigma OO e a linguagem UML, que tem sido amplamente utilizada nesse contexto e de se considerar também a importância de se adotar um processo de desenvolvimento gradativo, como menciona Andriole (1986), Marucci (2002) definiu um conjunto de técnicas de leitura denominado OORTs/ProDeS para apoiar atividades de inspeção em um processo específico, denominado ProDeS/UML (**Pro**cesso de **De**senvolvimento de **S**oftware para **UML**) (COLANZI, 1999). Esse processo considera um desenvolvimento de software gradativo, o que permite acompanhar constantemente a qualidade, corretitude dos artefatos gerados, bem como sua evolução. Além dessas características, esse processo possui uma estratégia de teste vinculada a ele. Isso faz do ProDeS/UML um processo mais do tipo tradicional e robusto, no qual dá-se grande ênfase aos aspectos de documentação. Assim, depois de realizado o trabalho de Marucci (2002) esse processo passou a contar também com uma estratégia de inspeção apoiada pelas técnicas de leitura OORTs/ProDeS, as quais foram propostas, mas ainda não foram validadas nesse contexto. Foi conduzido apenas um estudo de caso para avaliar a viabilidade de aplicação das mesmas, no qual duas técnicas foram exploradas.

No entanto, segundo Basili et al (1996b), um ponto importante na Engenharia de Software, devido à grande quantidade de técnicas, métodos, etc. propostos, é a caracterização dos mesmos para se estabelecer subsídios que apóiem os engenheiros de software na escolha desse ferramental. Não adianta simplesmente apresentar novas técnicas, pois isso não leva à solução dos problemas; é preciso utilizá-las e estabelecer suas vantagens e desvantagens. Uma forma de se fazer isso é através de estudos experimentais, com a replicação de experimentos, atividade esta que consiste em repetir o mesmo experimento, preferencialmente em diferentes contextos, pois quando pesquisadores diferentes chegam às mesmas conclusões, aumenta-se a aceitação das hipóteses do experimento (FUSARO et al., 1997). Os resultados de uma replicação são os que dão subsídios à caracterização do ferramental da Engenharia de Software que está sendo avaliado no referido experimento. Para a realização de um

experimento e também para suas replicações, é fundamental a elaboração de um Pacote de Laboratório, o qual deve conter todas as informações, técnicas, artefatos, etc. que permitem que o experimento seja conduzido, quer no contexto em que o experimento tenha sido definido ou em contextos diferentes, por outros pesquisadores.

Assim, dada a importância da caracterização do ferramental proposto na Engenharia de Software, atividades de experimentação têm se tornado cada vez mais relevantes e mais freqüentes. Um exemplo disso é o Projeto Readers II (MALDONADO et al., 2001), com o qual a proposta deste trabalho está relacionada. Esse projeto tem como objetivo principal explorar técnicas para analisar artefatos de software, tendo em vista a detecção de defeitos, tanto no nível de especificação como de código fonte. A iniciativa desse projeto é contribuir para a definição de uma família de tecnologia de análise de software, a qual deve ser validada empiricamente, por experimentos controlados e, posteriormente, empacotados em Pacotes de Laboratório de engenharia de software adaptáveis e reutilizáveis.

Dessa forma, um ponto importante de ser tratado que ficou em aberto no trabalho de Marucci (2002) foi a validação das técnicas de leitura OORTs/ProDeS, sendo que, para tanto, a elaboração dos Pacotes de Laboratório eram essenciais.

No entanto, um outro ponto relevante no contexto deste trabalho é a ênfase que tem sido dada aos métodos ágeis. As organizações modernas de software têm operado em um mercado altamente dinâmico, sob a pressão do custo e de prazos bastante restritos de desenvolvimento. O desenvolvimento do produto é altamente volátil devido às mudanças nos requisitos e nas necessidades do negócio e do mercado. Métodos de desenvolvimento de software ágeis têm sido propostos para tratar desses problemas, sendo que, dentre eles, desde a sua introdução, o XP (*Extreme Programming*) tornou-se o método de desenvolvimento de software ágil mais relatado na literatura, sendo alvo de um grande volume de novas pesquisas, artigos e relatos de experiências de seu uso (KÄHKÖNEN; ABRAHAMSSON, 2003), não obstante os vários problemas que são também apontados no que diz respeito a essa nova abordagem de desenvolvimento, principalmente no que se refere à elicitação dos requisitos (LEITE, 2001; NAWROCKI et al., 2002).

Posto isso, para avaliar as técnicas OORTs/ProDeS, considerando que o processo ProDeS/UML é um processo mais tradicional, viu-se a necessidade de explorar também esse "movimento ágil" que tanto tem sido abordado nos dias de hoje.

1.2 Motivação e Objetivos

Com base no contexto anterior, a motivação para a realização deste trabalho pode ser sintetizada pelos seguintes itens:

- Atualmente muitos sistemas são desenvolvidos utilizando o paradigma OO e a UML como linguagem de modelagem dos mesmos.
- Atividades de VV&T são essenciais para assegurar uma maior qualidade ao software que é desenvolvido.
- Técnicas de leitura de artefatos de software são essenciais para a realização de atividades de inspeção, que constituem um tipo de atividade de VV&T.
- As técnicas de leitura OORTs/ProDeS, para fazer inspeção de artefatos de software gerados nas etapas do processo específico que segue o paradigma OO – ProDeS/UML – não foram validadas.
- A avaliação de novas técnicas é muito importante para que elas possam ser caracterizadas e que suas vantagens e desvantagens possam ser estabelecidas.
- Uma maneira de caracterizar novas técnicas é por meio de replicação de experimentos, os quais, para serem realizados exigem o suporte de Pacotes de Laboratório.
- O movimento dos métodos ágeis tem ganhado bastante atenção na comunidade e várias iniciativas de uso dos mesmos tem sido relatado na literatura.
- Embora o XP tenha sido bastante explorado em ambientes industriais e acadêmicos, é fato que ele apresenta alguns pontos fracos que necessitam de alternativas para melhorá-los.

Assim, considerando-se essa motivação, os objetivos deste trabalho são:

i) tornar o processo ProDeS/UML mais prático, tendo em vista as características dos métodos ágeis e algumas propostas que os mesclam com outras abordagens mais tradicionais;

ii) verificar quais técnicas de leitura do conjunto OORTs/ProDeS permaneciam passíveis de aplicação nesse processo mais prático e elaborar para elas Pacotes de Laboratório para sua validação;

iii) explorar a atividade de inspeção no método XP, que é um método essencialmente ágil e, dentre os métodos propostos, parece ser o mais utilizado na prática.

1.3 Organização do Trabalho

O presente trabalho está organizado em sete capítulos, além da seção referente às referências bibliográficas e do Apêndice A.

Neste capítulo foram apresentados o contexto no qual o trabalho está inserido, os pontos motivadores para o seu desenvolvimento e também seus objetivos.

No Capítulo 2 apresenta-se uma breve revisão dos conceitos e dos processos de desenvolvimento de software OO pertinentes ao contexto no qual este trabalho se insere.

As técnicas de leitura, incluindo a OORTs/ProDeS e os processos de inspeção e experimentação utilizados para a realização deste trabalho são apresentados no Capítulo 3.

Os Métodos Ágeis são apresentados no Capítulo 4, destacando-se o XP. Também são comentados alguns trabalhos que estão relacionados ao contexto desta dissertação.

No Capítulo 5 apresenta-se a uma comparação realizada entre os artefatos do processo ProDeS/UML com os artefatos de alguns métodos ágeis, bem como os Pacotes de Laboratório preparados no contexto deste trabalho e o estudo de caso usado para avaliar a técnica de leitura que consta de um desses pacotes.

No Capítulo 6 descreve-se uma proposta de inspeção de software voltada para o contexto XP.

As conclusões e trabalhos futuros são apresentados no Capítulo 7.

Finalmente, no Apêndice A encontra-se um exemplo do Pacote de Laboratório, o qual contém a técnica de leitura ER1. No documento de trabalho (TOMA, 2004) encontram-se todos os Pacotes de Laboratório especificados neste trabalho.

CAPÍTULO 2

ORIENTAÇÃO A OBJETOS: CONCEITOS E PROCESSOS DE DESENVOLVIMENTO

2.1 Considerações Iniciais

Neste capítulo apresenta-se uma breve revisão bibliográfica sobre os conceitos OO uma vez que esse paradigma está diretamente relacionado com os objetivos deste trabalho, tanto no que diz respeito ao estudo e revisão do processo de desenvolvimento OO denominado ProDeS/UML, como também no que diz respeito ao desenvolvimento ágil, já que os conceitos desse paradigma estão presentes na maioria dos processos que apóiam essa abordagem.

Assim, são apresentados os principais conceitos do paradigma OO, os conceitos da linguagem UML, pois o processo ProDeS/UML utiliza essa notação, e também são comentados alguns processos que podem ser utilizados para se conduzir um desenvolvimento de software nesse contexto.

O capítulo está organizado da seguinte forma: na Seção 2.2 apresentam-se os conceitos de OO; na Seção 2.3 são abordados alguns processos de desenvolvimento OO, dando-se ênfase ao processo ProDeS/UML e na Seção 2.4 apresentam-se as considerações finais.

2.2 Paradigma Orientado a Objetos

Nas últimas décadas houve um aperfeiçoamento constante das técnicas e metodologias para a engenharia de software visando acompanhar a evolução natural do conhecimento. Com essas mudanças foram também surgindo novas técnicas e ferramentas para o desenvolvimento de software. O paradigma de desenvolvimento de software OO (Orientado a Objetos), que tem sido bastante utilizado mais recentemente, é um exemplo disso.

Rumbaugh et al. (1992) definem orientação a objetos como "uma nova maneira de pensar sobre os problemas utilizando modelos organizados a partir de conceitos do mundo real. O componente fundamental é o objeto que combina estrutura e comportamento em uma única entidade".

A visão tradicional no desenvolvimento de software adota a perspectiva de um algoritmo. Nessa visão, o principal bloco de construção do software é o procedimento ou a função. Já a visão contemporânea adota uma perspectiva orientada a objetos, na qual o principal bloco de construção de todos os sistemas de software é o objeto ou a classe (BOOCH et al., 2000). Os principais conceitos sobre Orientação a Objetos são apresentados na Tabela 2.1.

Considerando o paradigma OO com seus conceitos, muitas linguagens de modelagem OO foram surgindo entre a metade da década de 1970 e o final da década de 1980 à medida que os pesquisadores envolvidos com a metodologia, diante de um novo gênero de linguagens de programação orientadas a objetos e de aplicações cada vez mais complexas, começaram a experimentar métodos alternativos de análise e projeto (BOOCH et al., 2000). Dentre esses pesquisadores, Booch et al. (2000) definem uma linguagem de modelagem como uma "linguagem cujo vocabulário e regras têm o seu foco voltado para a representação conceitual e física de um sistema". Por volta dos anos 80, havia muitos métodos diferentes para orientação a objeto, como por exemplo o Booch, o OOSE (*Objetc-Oriented Software Engineering*) de Jacobson, e a OMT (*Object Modeling Technique*) de Rumbaugh. Cada método tinha a sua notação, ou seja, uma notação podia representar uma coisa em um método e algo totalmente diferente em outro método. O problema maior era que se diferentes pessoas usassem notações diferentes para o mesmo conceito, mais cedo ou mais tarde alguém tinha que fazer a conversão de uma notação para a outra (QUATRANI, 2001).

Tabela 2.1 - Principais conceitos da Orientação a Objeto (adaptado de FURLAN, 1998)

Palavra-chave	Breve definição
Atributo	Característica particular de uma ocorrência da classe
Classe	Agrupamento de objetos similares que apresentam os mesmos atributos e operações
Encapsulamento	Combinação de atributos e operações em uma classe
Especialização	Atributos e operações diferentes de uma subclasse, acrescentando ou substituindo características herdadas da classe pai
Estado	Situação de um objeto em um dado instante do tempo
Evento	Uma ocorrência significativa no mundo real que deve ser tratada
Generalização	Atributos e operações comuns compartilhados por classes
Herança	Compartilhamento, pela subclasse, dos atributos e operações da classe pai
Instância de classe	Uma ocorrência específica de uma classe. É o mesmo que objeto
Mensagem	Uma solicitação entre objetos para invocar certa operação
Objeto	Elemento do mundo real (natureza). Sinônimo de instância de classe
Operações	Lógica contida em uma classe para designar-lhe um comportamento
Polimorfismo	Habilidade para usar a mesma mensagem para invocar comportamentos diferentes do objeto
Subclasse	Elemento que recebe por herança a estrutura e o comportamento de uma superclasse
Superclasse	Elemento que contém a estrutura e o comportamento generalizado de outras classes (subclasses)

No ano de 1995, Booch, Rumbaugh e Jacobson se juntaram e criaram uma linguagem de modelagem unificada conhecida como UML (*Unified Modeling Language*) (OMG, 2003).

Ao iniciar a unificação, estabeleceram os seguintes objetivos (BOOCH et al., 2000):

a) Fazer a modelagem de sistemas, do conceito ao artefato executável, com a utilização de técnicas orientadas a objetos,
b) Tratar de assuntos de escala inerentes a sistemas complexos e tarefas críticas, e
c) Criar uma linguagem de modelagem a ser utilizada por seres humanos e por máquinas.

A UML é uma linguagem padrão para a modelagem de sistemas de software OO. Ela é utilizada para especificar, visualizar, documentar e construir artefatos de um sistema

e pode ser usada em diversos tipos de sistemas e fases de desenvolvimento (FURLAN, 1998). A UML é apenas uma linguagem de modelagem e, portanto, pode ser utilizada dentro de um método para desenvolvimento de software. A UML não está associada com um processo específico, significando que é possível utilizá-la com vários processos de engenharia de software (BOOCH et al., 2000).

Segundo Travassos et al. (2001), a UML não tem um processo de software associado a ela. A UML pode ser usada em vários ciclos de vida e modelos de processo de software que podem estar em diferentes contextos. Alguns modelos de processo de software podem ser usados como *frameworks* para organizar e configurar atividades de projeto. Alguns exemplos são *Catalysis* (D'SOUZA et al., 1998 apud TRAVASSOS et al., 2001), RUP (*Rational Unified Process*) (KRUTCHEN, 1999), *Unified Software Process* (JACOBSON et al., 1999 apud TRAVASSOS et al.,2001) e *Open Process* (GRAHAM et al., 1997 apud TRAVASSOS et al., 2001). Apesar desses modelos de processo serem considerados robustos, alguns deles possuem um custo alto, pois estão relacionados com ferramentas automatizadas e precisam de treinamento e planejamento detalhados para seu uso. Escolher qual deles deve ser usado, é uma questão complicada, pois eles não oferecem técnicas ou diretrizes para detecção de defeitos e nem facilidades de adaptação para classes específicas de problemas, como por exemplo, sistemas de tempo real e de *e-commerce*.

Quanto aos diagramas, a UML define doze tipos de diagramas que são divididos em três categorias (OMG, 2003):

Diagramas estruturais: compreendem quatro tipos de diagrama que representam a estrutura estática da aplicação; são eles: Diagrama de classes, Diagrama de objetos, Diagrama de componentes e o Diagrama de distribuição.

Diagramas comportamentais: compreendem cinco diagramas que representam aspectos diferentes do comportamento dinâmico: Diagrama de casos de uso (usado por algumas metodologias durante a coleta de requisitos), Diagrama de seqüência, Diagrama de atividades, Diagrama de colaboração e Diagrama de estados.

Diagramas de gerenciamento: compreendem três diagramas que representam maneiras que se pode organizar e controlar os módulos de aplicação. Incluem os Pacotes, Subsistemas e Modelos.

A seguir são relacionados os diagramas da UML que foram conceituados pela OMG (2003):

- **Diagrama de classes**: apresenta a estrutura estática das classes de um sistema na qual estas representam as "coisas" que são gerenciadas pela aplicação modelada.
- **Diagrama de objetos**: é uma variação do diagrama de classes e utiliza a mesma notação. A diferença é que o diagrama de objetos mostra os objetos que foram instanciados das classes. O diagrama de objetos é como se fosse o perfil do sistema em um certo momento de sua execução.
- **Diagrama de casos de uso**: é usado para descrever e definir os requisitos funcionais de um sistema. Eles são escritos em termos de atores externos e de casos de uso do sistema modelado.
- **Diagrama de seqüência**: mostra a interação dinâmica entre os vários objetos de um sistema, alguma coisa que acontecerá em um ponto específico da execução do sistema. O mais importante aspecto desse diagrama é que a partir dele percebe-se a seqüência de mensagens enviadas entre os objetos. A interação de mensagens é mostrada com ênfase na decorrência do tempo.
- **Diagrama de colaboração**: mostra, de maneira semelhante ao Diagrama de seqüência, a colaboração dinâmica entre os objetos. No diagrama de colaboração, além de ser mostrada a troca de mensagens entre os objetos, percebe-se também os objetos com os seus relacionamentos. A interação de mensagens é mostrada com ênfase no contexto do sistema.
- **Diagrama de estados**: é tipicamente um complemento para a descrição das classes. Esse diagrama mostra todos os estados possíveis em que objetos de uma certa classe podem se encontrar e também apresenta quais são os eventos do sistema que provocam tais mudanças. Os diagramas de estado não são escritos para todas as classes de um sistema, mas apenas para aquelas que possuem um número definido de estados conhecidos e no qual o comportamento das classes é afetado e modificado pelos diferentes estados.
- **Diagrama de atividades**: capturam ações e seus resultados. Ele focaliza o trabalho executado na implementação de uma operação (método) e suas atividades numa instância de um objeto. O diagrama de atividades é uma variação do diagrama de estado e possui o propósito de capturar ações (trabalho e atividades que serão executados) e seus resultados em termos das mudanças de estados dos objetos.
- **Diagrama de componentes**: descreve os componentes de software e suas dependências entre si, representando a estrutura do código gerado. Os componentes são

a implementação na arquitetura física dos conceitos e da funcionalidade definidos na arquitetura lógica (classes, objetos e seus relacionamentos). Abrange a visão estática da implementação de um sistema.

- **Diagrama de distribuição**: apresenta a arquitetura *run-time* de processadores, componentes físicos (*devices*) e de software que rodam no ambiente em que o sistema desenvolvido será utilizado.
- **Pacote**: é um mecanismo de propósito geral para a organização de elementos em grupos. Um pacote é representado graficamente como uma pasta com uma guia.
- **Subsistema**: é um agrupamento de elementos, sendo que alguns constituem a especificação do comportamento oferecido por outros elementos nele contidos.
- **Modelo**: é uma simplificação da realidade, uma abstração de um sistema, criada com a finalidade de permitir uma melhor compreensão do sistema.

2.3 Abordagens usadas no Desenvolvimento OO

Um processo de desenvolvimento de software define a maneira como os conceitos do mundo real devem ser representados, interpretados e transformados em um sistema de software. Tipicamente, o processo de desenvolvimento de software é guiado por uma estratégia ou um paradigma. Existem alguns paradigmas como o estruturado (ou funcional) e o orientado a dados que são bem estabelecidos. Embora esses paradigmas possam ser usados para especificar e projetar sistemas de diferentes tipos de problemas, o uso deles produz um impacto na qualidade e produtividade do desenvolvimento de software porque nesses paradigmas, os desenvolvedores não usam adequadamente uma notação consistente durante todo o ciclo de vida do software. Conseqüentemente, é reduzida a liberdade de reorganização das atividades e sua adaptação nos modelos de ciclo de vida das organizações (TRAVASSOS et al., 2001). O paradigma OO surgiu para tratar desses assuntos e várias abordagens (métodos, processos, *frameworks*) foram propostas na literatura. A seguir, comentam-se algumas delas.

FUSION

Fusion é um método de desenvolvimento de software orientado a objetos elaborado por Coleman et al. (1994). Ele é baseado em um conjunto de notações com fácil entendimento e essas notações são bem definidas para a captura de decisões de análise e projeto.

O Fusion é dividido nas seguintes fases: Análise, Projeto e Implementação. Ele não possui a fase de Requisitos, pois essa fase é geralmente desempenhada por um cliente, que irá fornecer o documento de requisitos inicial. A Figura 2.1 apresenta o processo do método Fusion:

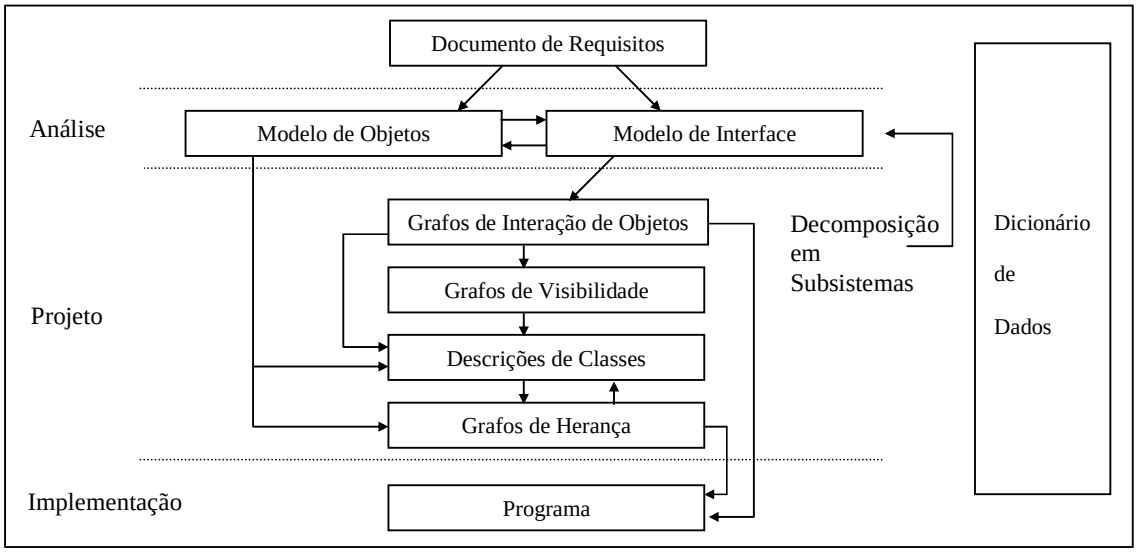


Figura 2.1 - Visão Geral do Método Fusion (COLEMAN et al., 1994)

Na fase de Análise define-se a finalidade comportamental do sistema. Os Modelos de Sistema que seriam o Modelo de Objetos e o Modelo de Interface são produzidos, os quais descrevem o seguinte:

1. classes e objetos que existem no sistema
2. relacionamentos entre essas classes
3. operações que podem ser desempenhadas no sistema
4. seqüências admissíveis dessas operações

O processo da Análise é definido pelas seguintes etapas:

1. Desenvolver um Modelo de Objetos para o domínio do problema.

2. Determinar a interface do sistema.
    - Identificar agentes, operações do sistema e eventos.
    - Produzir o Modelo de Objetos do sistema. O Modelo de Objetos do sistema é um refinamento do Modelo de Objeto desenvolvido na primeira etapa da Análise.
3. Desenvolver um Modelo de Interface.
    - Desenvolver um Modelo de Ciclo de Vida.
    - Desenvolver um Modelo de Operações.
4. Checar os modelos de análise.

Em todas as fases do Fusion é necessário construir e usar o dicionário de dados. Um dicionário de dados é um repositório central de definições de termos e conceitos. Sem ele, os modelos do Fusion têm pouco conteúdo semântico.

Na fase de Projeto decide-se como representar as operações através de interações de objetos relacionados e como estes objetos atingem o acesso uns aos outros. Existem quatro modelos de projeto que são desenvolvidos nessa fase:

- Grafo de Interação de Objetos: descreve como os objetos interagem em tempo de execução para dar suporte à funcionalidade especificada no modelo de operação.
- Grafo de Visibilidade: descreve o caminho de comunicação dos objetos.
- Descrição de Classes: fornece uma especificação da classe interface, atributos de dados, objeto referente aos atributos e assinaturas de métodos para todas as classes do sistema.
- Grafo de Herança: descreve as estruturas de herança da classe/subclasse.

Esses artefatos fornecem uma base para a implementação, teste e manutenção do software. Além deles, o dicionário de dados documenta os termos, conceitos e restrições construídas durante as fases de análise e projeto.

A fase de Implementação codifica o projeto em uma linguagem de programação. O processo de implementação é dividido em três partes, cada uma com sua própria subestrutura:

- Codificação: traduz os artefatos do projeto em código de linguagem de implementação.

- Desempenho: identifica e melhora as partes dos códigos que usam muito recurso e são pouco eficientes.
- Revisão: revisa o código que é produzido.

OMT

Um outro método de desenvolvimento Orientado a Objeto é o OMT (*Object Modeling Technique*) (RUMBAUGH et al., 1992) que engloba a Análise, Projeto e Implementação e foi desenvolvido para dar suporte à modelagem de sistemas de tempo real, por possuir o Modelo Dinâmico.

O método consiste de quatro fases: Análise, Projeto do Sistema, Projeto do Objeto e Implementação. A Figura 2.2 apresenta a visão geral do processo:

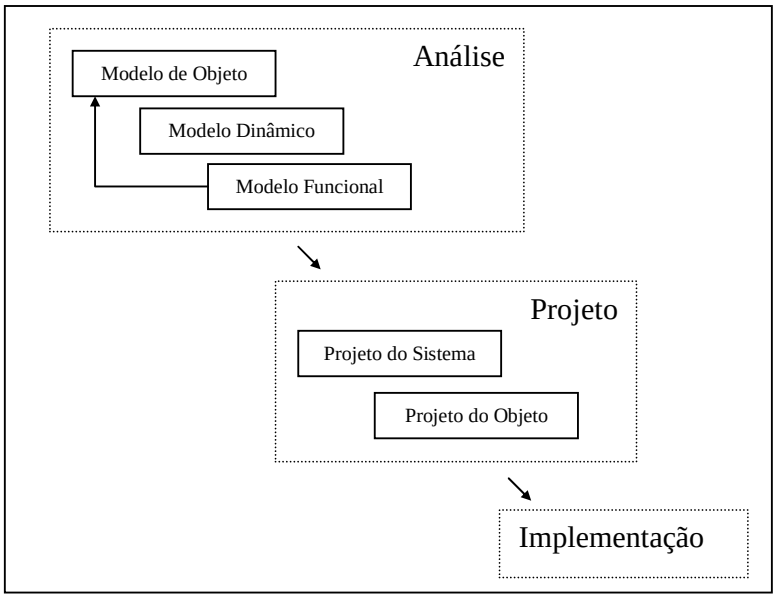


Figura 2.2 - Visão Geral do Método OMT (RUMBAUGH et al., 1992)

O propósito da fase da Análise é modelar o mundo real para que ele possa ser entendido.

Em relação à documentação, na fase da Análise, o OMT desenvolve três tipos de modelos que são refinados em todas as outras fases. São eles (JACOBSON et al., 1992):

- Modelo de Objeto: descreve a estrutura estática do sistema com classes e seus relacionamentos.

- Modelo Dinâmico: captura os aspectos temporais do Modelo de Objeto com eventos e estados dos objetos, permitindo então que sistemas de tempo real sejam modelados de forma mais apropriada.
- Modelo Funcional: descreve o processamento em termos de como valores de saída são derivados a partir dos valores de entrada, isto é, principalmente em termos das operações dos objetos.

A fase de Projeto não introduz nenhum tipo de modelo. Ao invés disso, ele é constituído de heurísticas ou diretrizes para se executar o projeto. Existem dois estágios:

- Projeto do Sistema: uma estratégia de alto nível é desenvolvida. O sistema é particionado em subsistemas e também são alocados os processadores e processos (tarefas).
- Projeto do Objeto: define os objetos em detalhes. Isto inclui a definição de suas interfaces, algoritmos e operações.

A última fase em OMT é a Implementação dos objetos. Isto é feito seguindo um número de diretrizes e regras de codificação para se adquirir um bom estilo de programação OO.

BOOCH

O método BOOCH (BOOCH, 1994) possui um processo descritivo e natural. Uma visão geral do processo é mostrada na Figura 2.3:

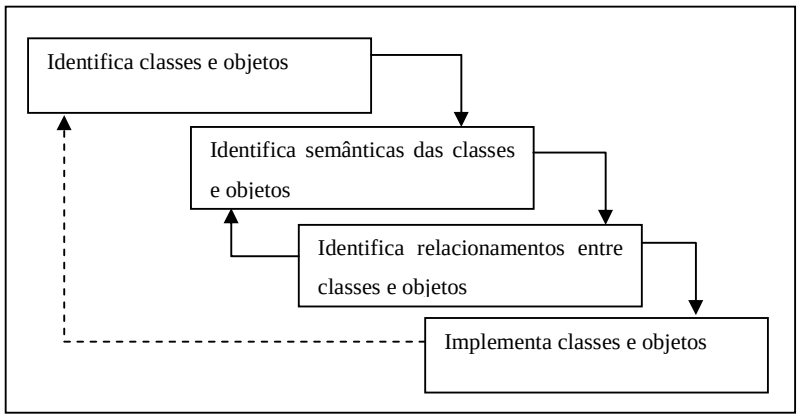


Figura 2.3 - Visão Geral do Método BOOCH (BOOCH, 1994)

A primeira fase corresponde à identificação das classes e objetos que dão forma ao sistema. Depois, a interface para as classes é construída, isto é, "identificação de semânticas" seguida pelos relacionamentos encontrados entre as classes. A descoberta de relacionamentos pode causar novas interfaces que devem ser adicionadas, sendo que essas duas etapas devem ser repetidas até que se obtenha um estado satisfatório. Em seguida, na fase da implementação decide-se a representação interna das classes (atributos e comportamento); isso pode resultar no retorno da aplicação do processo inteiro para o comportamento de uma única classe.

As notações usadas em BOOCH são seis, sendo que elas não possuem uma ordem de construção: Diagrama de Objetos, Diagrama de Classes, Diagramas de Tempo, Diagramas de Estado e Diagramas de Processo.

OBJECTORY

As fases do Objectory (JACOBSON et al., 1992) são: Requisitos, Análise e Construção. A seguir, a visão geral do processo é apresentada na Figura 2.4:

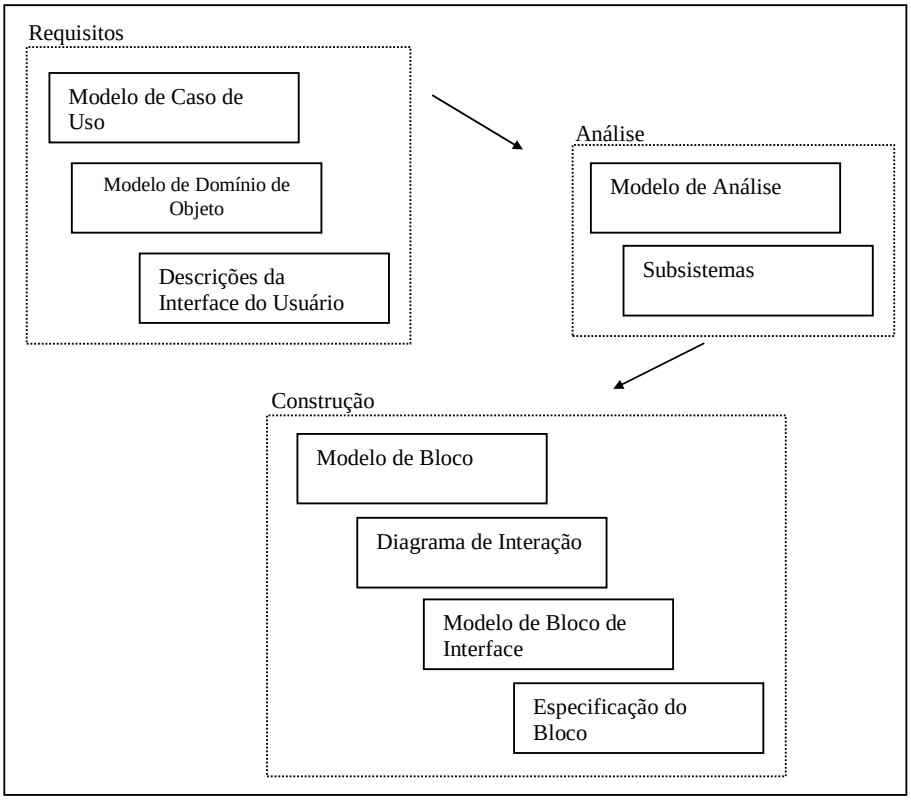


Figura 2.4 - Visão Geral do Método Objectory (JACOBSON et al., 1992)

O método Objectory é baseado na notação de Caso de Uso. Um caso de uso é um diálogo entre o sistema e um usuário, que é executado para alcançar alguns objetivos.

Na fase de Requisitos são construídos três modelos: Modelo de Caso de Uso, Modelo de Domínio de Objeto e Descrições da Interface do Usuário.

A fase de Análise refina os modelos de requisitos para produzir uma descrição ideal do sistema. O modelo de análise é uma forma de modelo entidade-relacionamento. Ele é construído incrementalmente considerando que objetos e classes são requeridos por cada caso de uso.

Na fase de Construção, os modelos de análise são refinados. A comunicação do objeto é mais precisamente definida e as características do ambiente de implementação são considerados. Quatro modelos são produzidos: Modelo de Bloco, Diagrama de Interação, Blocos de Interfaces e Blocos de Especificação.

RUP

O RUP (*Rational Unified Process*) (KRUTCHEN, 1999) é um modelo de processo de software que pode ser usado como um *framework* para organizar e configurar atividades de projeto (TRAVASSOS et al., 2001). Ele é genérico e completo o suficiente para ser usado por uma grande variedade de organizações de desenvolvimento de software. Em várias circunstâncias, esse processo de engenharia de software precisará ser modificado, ajustado, ampliado e adaptado para acomodar as características, as restrições e o histórico específicos da organização que o adota (RATIONAL, 2002).

A Figura 2.5 mostra a arquitetura geral do RUP:

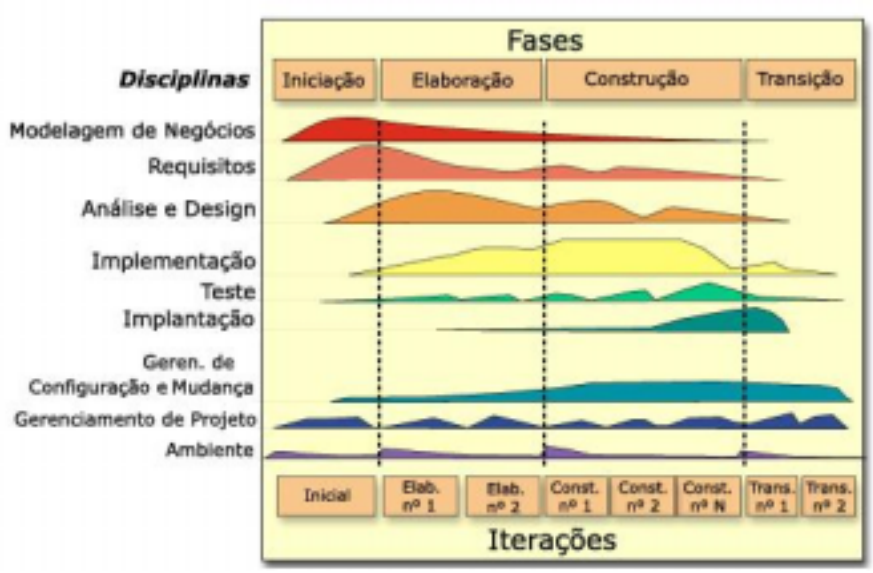


Figura 2.5- Visão Geral do RUP (RATIONAL, 2002).

O eixo horizontal representa o tempo e mostra os aspectos do ciclo de vida do processo à medida que se desenvolve e, o eixo vertical, representa o aspecto dinâmico do processo quando ele é aprovado e é expresso em termos de fases, iterações e marcos (RATIONAL, 2002).

Uma fase é o período de tempo entre dois importantes marcos de progresso do processo em que um conjunto bem-definido de objetivos é alcançado, artefatos são concluídos e decisões são tomadas em relação à passagem para a fase seguinte (BOOCH et al., 2000)

Em cada fase, ocorrem várias iterações. Uma iteração representa um ciclo completo de desenvolvimento, desde a captação de requisitos na análise até a implementação e a realização de testes, resultando na versão de um projeto executável (BOOCH et al., 2000).

Segundo Rational (2002), as fases do RUP possuem os seguintes objetivos:

- Fase de Iniciação: atingir o consenso entre todos os envolvidos sobre os objetivos do ciclo de vida do projeto.
- Fase de Elaboração: criar a *baseline* para a arquitetura do sistema a fim de fornecer uma base estável pra o esforço da fase de construção.
- Fase de Construção: esclarecer os requisitos restantes e concluir o desenvolvimento do sistema com base na arquitetura da *baseline*.
- Fase de Transição: assegurar que o software esteja disponível para seus usuários finais.

Todos os esforços, incluindo a modelagem, são organizados em *disciplinas* no RUP as quais são realizadas de uma maneira iterativa e incremental (AMBLER, 2002).

A Rational (2002) diz que uma disciplina mostra todas as atividades que se deve realizar para se produzir um determinado conjunto de artefatos. O RUP é composto por nove disciplinas que estão resumidas na Tabela 2.2:

Tabela 2.2 - Resumo das disciplinas do RUP (BOOCH, 2000; AMBLER, 2002)

Disciplina	Propósito
Modelagem de Negócios	Modelar o contexto do negócio, o escopo do sistema
Requisitos	Construir os requisitos para o projeto, incluindo a identificação, a modelagem e a documentação desses requisitos.
Análise e Projeto	Desenvolver uma arquitetura robusta para o sistema baseado nos requisitos, transformar os requisitos em projeto e assegurar que os assuntos do ambiente de implementação estão refletidos no projeto.
Implementação	Desenvolver o software, o teste de unidade e a integração.
Teste	Descrever os casos de teste, procedimentos e medidas para acompanhamento de erros.
Implantação	Descrever as atividades que garantem que o produto de software será disponibilizado a seus usuários finais.
Gerenciamento de Configuração e Mudança	Controlar as modificações e manter a integridade dos artefatos do projeto.
Gerenciamento de Projeto	Descrever as várias estratégias para o trabalho com um processo iterativo.
Ambiente	Tratar da infra-estrutura necessária para o desenvolvimento do sistema.

Cada atividade do RUP tem artefatos associados ou exigidos como uma entrada ou gerados como uma saída. Os modelos são o tipo mais importante de artefato do RUP. Esses modelos são baseados nos conceitos de objetos, classes e relacionamentos existentes entre eles e utilizam a UML como notação comum.

ProDeS/UML

O ProDeS/UML (**Pro**cesso de **De**senvolvimento de **S**oftware para **UML**) é um processo de desenvolvimento de software Orientado a Objeto, desenvolvido por Colanzi (1999), que se baseia no método Fusion e que adota a notação da UML nos seus artefatos. Engenharia de Requisitos, Análise, Projeto e Implementação são as quatro fases que compõem o ProDeS/UML. Durante todo o seu processo de desenvolvimento, modelos de teste são propostos visando Atividades de Garantia de Qualidade de Software.

Na Tabela 2.3 estão listados os artefatos que são gerados no ProDeS/UML, a notação usada em cada um deles e a procedência da notação, a qual pode ser a UML ou o

Fusion, uma vez que esse processo tem como base o Fusion, com a notação adequada para a UML.

Tabela 2.3 - Procedência e notação dos diagramas do ProDeS/UML (MARUCCI, 2002)

Fases do ProDeS/UML	Diagramas elaborados no ProDes/UML	Notação Correspondentes	Procedência	
			UML	Fusion
Engenharia de Requisitos	Diagrama de Casos de Uso	Diagrama de Casos de Uso	X	
	Especificação dos Casos de Uso	Diagrama de Casos de Uso	X	
	Diagrama de Classes do Domínio	Diagrama de Classes	X	
Análise	Cenários	Diagrama de Seqüência	X	
	Diagrama de Classes da Análise	Diagrama de Classes	X	
	Modelo de Operações	Modelo de Operação		X
	Modelo de Ciclo de Vida	Diagrama de Estados	X	
Projeto	Diagrama de Colaboração	Diagrama de Colaboração	X	
	Diagrama de Visibilidade	Diagrama de Classes	X	
	Diagrama de Classes da Análise Refinado	Diagrama de Classes	X	
	Descrição de Classes	Descrição de Classe		X
	Diagrama de Estados da Classe	Diagrama de Estados	X	
Implementação	Implementação das Classes	-		

A fase de Engenharia de Requisitos tem como objetivo descrever o que o sistema deve fazer e permitir que os desenvolvedores e clientes concordem com essa descrição. Nessa fase utiliza-se o documento de requisitos do sistema como entrada e, a partir dele, o Diagrama de Casos de Uso e as Especificações são gerados. O Diagrama de Classes do Domínio também é criado com o intuito de representar os conceitos existentes no domínio do problema e suas relações (COLANZI, 1999).

Na fase de Análise devem ser elaborados os seguintes documentos: Cenários, Diagrama de Classes da Análise, Modelo de Operações e Modelo de Ciclo de Vida. O Diagrama de Classes da Análise nada mais é do que o Diagrama de Classes do Domínio gerado na fase anterior, acrescido de classes necessárias para a implementação do software. Os Cenários do sistema são construídos para detectar os eventos e os dados de entrada e saída do sistema. O Modelo de Operações explica de forma declarativa, o comportamento de cada operação do sistema individualmente. O Modelo de Ciclo de Vida descreve o comportamento completo de como o sistema se comunica com o ambiente, desde sua criação até seu término.

A fase de Projeto tem como objetivo especificar como uma operação pode ser implementada por intermédio da interação de vários objetos que trocam mensagens entre si. Para saber como as classes interagem entre si para a execução de uma operação, é gerado o Diagrama de Colaboração. O Diagrama de Visibilidade que é produzido nessa

fase, corresponde ao Diagrama de Classes de Análise utilizando os relacionamentos de navegabilidade e dependência. Outro diagrama elaborado é o Diagrama de Classes de Análise Refinado, que corresponde ao Diagrama de Classes da fase anterior, acrescido dos objetos, das relações de herança (generalização/especialização) e das operações das classes. A Descrição das Classes descreve a estrutura interna de cada classe que é formada pelos atributos de dados e de objetos-valorados (é aquele que resulta de um relacionamento, como navegabilidade e dependência), métodos e posição hierárquica, caso exista uma hierarquia de herança. O Diagrama de Estados das Classes descreve o comportamento de cada uma das classes em particular, desde sua criação até o seu término.

O objetivo da fase de Implementação em relação aos artefatos OO criados na fase anterior é transformar esses artefatos em código. A Descrição das Classes produzidas na fase de Projeto, é usada para a descrição dos Métodos. O Diagrama de Colaboração, por apresentar as mensagens enviadas em resposta à chamada de um método, é utilizado para a implementação dos mesmos.

Na Figura 2.6 é mostrada a interação existente entre as fases do processo, os documentos (modelos) produzidos em cada uma delas e a origem das informações para a produção de cada modelo. Embora o Documento de Requisitos não esteja incluído na Figura 2.6, supõe-se que o usuário defina os requisitos do sistema ou que esse documento seja produzido em conjunto pelos usuários e profissionais envolvidos no projeto (COLANZI, 1999). Observe que na Figura 2.6 aparecem os modelos de teste que são gerados ao longo do processo, mas estes não foram comentados, pois não são de interesse para o trabalho.

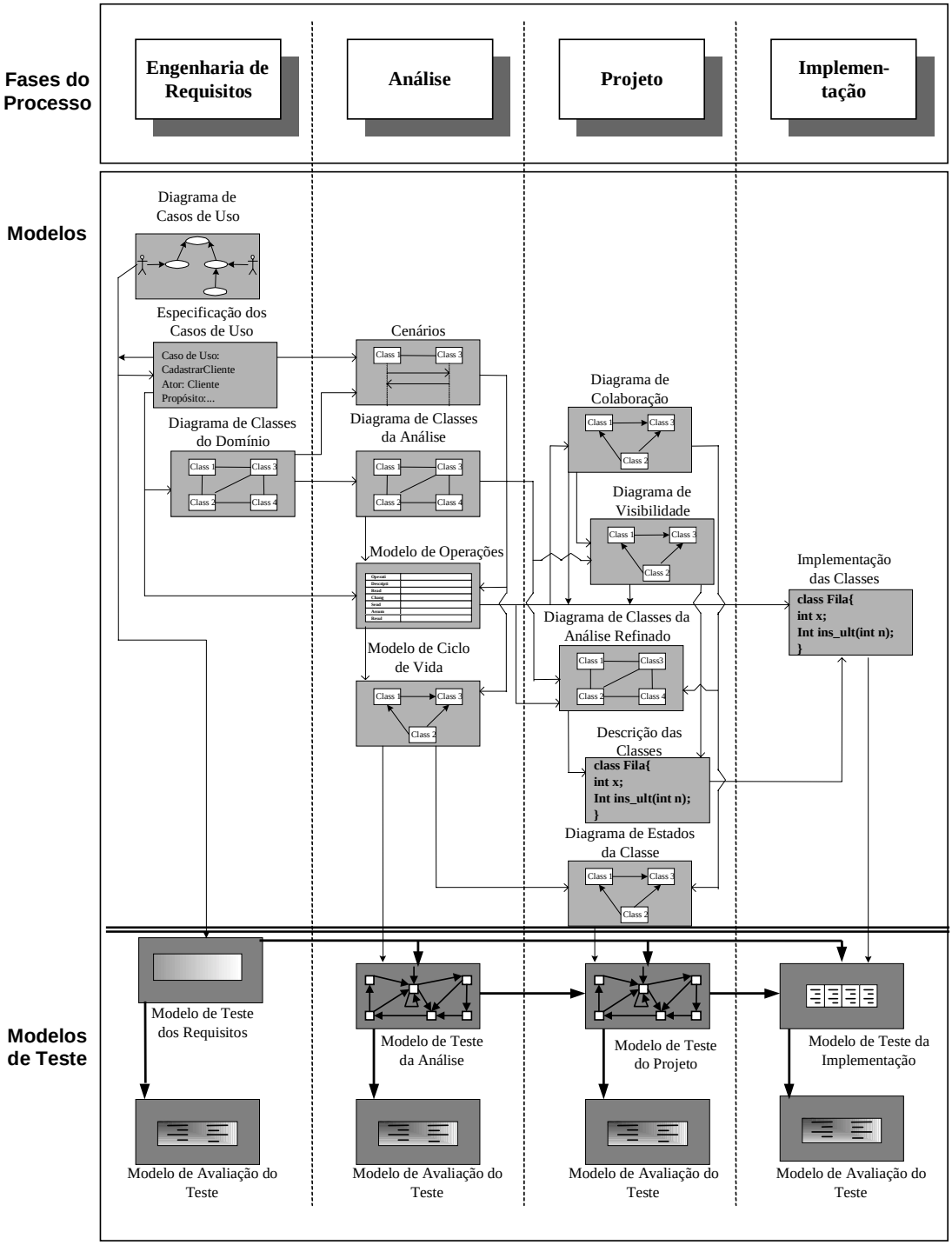


Figura 2.6 - Visão Geral do Processo ProDeS/UML (Adaptado MARUCCI, 2002)

2.4 Considerações Finais

Neste capítulo foram abordados, resumidamente, os conceitos de OO e da notação UML a qual é amplamente utilizada no contexto desse paradigma de desenvolvimento.

Também foram comentadas algumas abordagens de desenvolvimento de software OO como o Objectory, OMT, Fusion, RUP e o próprio ProDeS/UML, que é objeto de estudo neste trabalho. Observou-se que cada uma dessas abordagens possui as suas etapas e os seus artefatos de desenvolvimento, sendo que a maioria possui a sua própria notação. Este fato levou à definição da UML que é uma linguagem de modelagem que tem sido, atualmente, usada como um padrão de notação para esse paradigma.

Dentre as abordagens apresentadas, deu-se um destaque maior para o processo ProDeS/UML dada a sua relevância para o trabalho. Esse processo é baseado nas fases do método Fusion e na notação UML. As fases que compõem esse processo representam uma evolução gradativa e cada vez mais detalhada da solução do problema, o que facilita a introdução de Atividades de Garantia de Qualidade, como atividades de VV&T, ao longo do processo, permitindo pontos intermediários de avaliação do produto, possibilitando a detecção de defeitos à medida que eles vão surgindo. Esse processo possui agregado a ele uma estratégia de teste que propõe a elaboração de Modelos de Teste com base em alguns artefatos que são produzidos em cada fase e também uma estratégia de inspeção composta por um conjunto de técnicas de leitura denominado OORTs/ProDeS.

Dada a relevância dessas atividades de VV&T para melhorar a qualidade dos processos de desenvolvimento de software, particularmente para este trabalho, das atividades de inspeção, no próximo capítulo discutem-se as atividades de inspeção, incluindo as técnicas de leitura que podem apoiar essas atividades, em particular as OORTs/ProDeS.

CAPÍTULO 3

TÉCNICAS DE LEITURA E OS PROCESSOS DE INSPEÇÃO E DE EXPERIMENTAÇÃO

3.1 Considerações Iniciais

As técnicas de leitura dão suporte às atividades de inspeção, as quais são bastante efetivas na detecção de defeitos e podem ser aplicadas em vários tipos de artefatos de software. Atividades de inspeção podem ser aplicadas tão logo um artefato seja elaborado, permitindo que a detecção de defeitos seja feita na própria fase em que o artefato foi construído, minimizando os custos associados ao desenvolvimento.

As atividades de inspeção são realizadas com base em um Processo de Inspeção, no qual a inspeção do artefato propriamente dita é apenas uma das etapas, etapa esta na qual as técnicas de leitura são utilizadas para ajudar o inspetor fazer a leitura do documento que está sendo avaliado.

Várias técnicas de leitura são propostas na literatura, cada uma com propósitos específicos, como é o caso das OORTs/ProDeS, que têm o objetivo de apoiar a inspeção de artefatos de software, na sua maioria, baseados na notação UML e que foram definidas para apoiar atividades de inspeção, especificamente no processo ProDeS/UML apresentado no capítulo anterior.

Assim como as técnicas de leitura, várias outras técnicas, métodos e ferramentas são propostas na literatura e atuam tanto no âmbito de atividades de inspeção como de outras atividades. No entanto, independentemente da técnica e do contexto em que ela pode ser aplicada, cada nova proposta deve ser bem caracterizada, mostrando suas vantagens e desvantagens, para que os desenvolvedores tenham um suporte no momento em que tiverem que optar por alguma.

Essa caracterização de técnicas, métodos e ferramentas pode ser realizada por meio de estudos experimentais, com a aplicação de experimentos controlados que explorem a utilização do objeto de estudo em vários ambientes, cultura, etc., de modo a gerar uma boa caracterização do mesmo.

Da mesma forma que as atividades de inspeção, os estudos experimentais, para serem aplicados, seguem um Processo de Experimentação composto de algumas etapas. Por meio de resultados de experimentos é que o objeto de estudo pode ser bem caracterizado. Além disso, quanto mais resultados forem coletados sobre um objeto de estudo, mais significantes se tornam as conclusões que se chegam sobre ele. Na prática isso é feito pela replicação de experimentos que exploram uma determinada técnica que está sendo estudada. No entanto, para se replicar um experimento, é fundamental que se tenha o apoio de um Pacote de Laboratório, o qual reúne toda documentação necessária para que o experimento seja realizado da maneira mais fiel possível ao experimento original.

Assim, neste capítulo são explorados esses conceitos que darão subsídios às propostas realizadas neste trabalho, as quais estão diretamente relacionadas a técnicas de leitura no contexto de métodos ágeis e da elaboração de Pacotes de Laboratório que dêem apoio à replicação de experimentos para a caracterização de técnicas de leitura.

Dessa maneira, este capítulo está organizado da seguinte forma: na Seção 3.2 apresentam-se os conceitos sobre Inspeção de Software; na Seção 3.3 são abordadas algumas técnicas de leitura para o Processo de Inspeção, incluindo as OORTs/ProDeS que constituem um dos objetos de estudo deste trabalho; na Seção 3.4 é comentado o Processo de Experimentação juntamente com o empacotamento de experimento e na Seção 3.5 apresentam-se as considerações finais.

3.2 Inspeção de Software

Inspeção de software, segundo Laitenberger (2001), é um método que, comprovadamente, possibilita a detecção e remoção de defeitos em artefatos de software assim que estes são criados. A inspeção envolve atividades que ajudam um grupo de pessoas qualificadas determinar se o artefato criado possui as propriedades de qualidade desejada (LAITENBERGER, 2001). Com o uso da inspeção tem-se uma melhora

significante na qualidade do artefato, a qual vem acompanhada por baixos custos no desenvolvimento e uma grande redução de esforços na manutenção (FAGAN, 1986).

De acordo com Fagan (1986), o problema da falta de qualidade de um software é o resultado de defeitos no código e na documentação, provocando falhas, que não permitem que os requisitos do usuário sejam satisfeitos. Para eliminar os defeitos de um produto é necessária a prevenção ou a detecção. A correção desses defeitos deve ser feita perto do ponto de origem em que estes foram inseridos e não detectá-los somente nas fases de teste ou de manutenção. Com isso, há uma redução de custo de 10 a 100 vezes.

Nas inspeções, os indivíduos revisam um artefato individualmente e depois, eles se reúnem em grupos formados por quatro a cinco membros, com a finalidade de discutir e registrar os defeitos que, posteriormente, são enviados para o autor do documento com o objetivo de serem corrigidos (LAITENBERGER, 2001).

Nos estudos originais de Fagan (1976, 1986), os participantes da inspeção deviam ter habilidade e conhecimentos específicos e, além disso, eles exerciam uma determinada função com responsabilidades claras e específicas. As funções eram divididas em: Moderador, Autor, Leitor/Apresentador e o Testador.

Laitenberger (2001) em seu trabalho, afirma que atualmente existe um consenso entre vários trabalhos propostos sobre inspeção de software, em relação à definição das funções que podem ser atribuídas aos participantes de uma inspeção. Essas funções são apresentadas na Tabela 3.1.

Tabela 3.1 - Funções atribuídas aos participantes de uma inspeção segundo Laitenberger (2001).

Papel	Descrição
Autor	Desenvolvedor do produto a ser inspecionado e responsável pela correção dos defeitos.
Moderador	Membro da equipe que lidera e programa a inspeção, bem como controla as reuniões.
Inspetor	Responsável pela identificação de defeitos no produto, sendo que todos os participantes podem atuar como inspetores, além de executarem outras atribuições.
Organizador	Realiza o planejamento de todas as atividades relacionadas à atividade de inspeção.
Apresentador	Apresenta os artefatos a serem inspecionados de acordo com o ritmo dos participantes, caso uma reunião de inspeção seja realizada.
Redator	Classifica e registra os defeitos, bem como relata as questões surgidas durante a inspeção.
Coletor	Coleta os defeitos encontrados pelos inspetores, caso não seja realizada uma reunião de inspeção.

O processo de inspeção de Fagan (1976, 1986) é aplicável em qualquer fase de desenvolvimento do ciclo de vida do software incluindo análise de requisitos, documentos

de teste ou documentos de gerenciamento, bem como projeto e código. Ele consiste de seis etapas:

Planejamento: uma equipe de inspeção é formada e as funções são atribuídas aos membros da equipe.

Apresentação: uma etapa opcional na qual a equipe de inspeção é informada sobre o produto.

Preparação: inspetores inspecionam o material independentemente. O material de inspeção pode ser um produto da fase de análise, de projeto (tais como diagramas de entidade-relacionamento, fluxogramas, diagramas de transição de estado ou uma especificação em texto) ou código. O objetivo da preparação é fazer com que os inspetores aprendam sobre o material e cumpram as funções atribuídas a eles.

Inspeção: é também chamada de reunião de inspeção. O objetivo da reunião não é discutir ou avaliar a solução, mas procurar defeitos e colocá-los todos juntos. O moderador coordena a reunião e esta não deve durar mais que duas horas. Depois da reunião o moderador produz um relatório para assegurar que todos os defeitos identificados na reunião serão tratados nas fases de Retrabalho e Acompanhamento.

Retrabalho: Os defeitos são verificados pelo moderador e são corrigidas pelo Autor. Alguns produtos devem ser refeitos e reinspecionados várias vezes, se necessário.

Acompanhamento: O moderador checa e verifica cada correção.

Apesar das etapas principais do processo de inspeção de Fagan serem a Preparação, Inspeção e Retrabalho, Fagan (1986) diz que todas essas etapas são necessárias e que saltar ou combinar etapas não é recomendado.

Aurum et al.(2002) relatam que nos últimos 25 anos, várias modificações no método de inspeção de Fagan foram propostas para um melhor desempenho das inspeções de software. Essas modificações correspondem a reestruturações nos processos básicos da inspeção de Fagan, gerando modelos diferentes. Isso inclui mudanças nas atividades das etapas de preparação e de reunião de inspeção, mudança no número de participantes nas equipes, utilização de uma ou várias equipes, mudança na coordenação estratégica entre as equipes ou participantes de uma equipe.

Embora na inspeção original se enfatize muito o trabalho em grupo, alguns estudos empíricos recentes comprovaram que a detecção de defeitos é mais um trabalho individual do que em grupo. Ou seja, os resultados de uma inspeção dependem dos próprios

participantes da inspeção e de suas estratégias para o entendimento do artefato inspecionado. Portanto, dar suporte aos participantes da inspeção com técnicas específicas que os ajudem a detectar defeitos nos produtos de software, pode aumentar ainda mais a eficiência desses inspetores. Tais técnicas são conhecidas como Técnicas de Leitura e serão comentadas brevemente a seguir (LAITENBERGER, 2001):

3.3 Técnicas de Leitura para o processo de inspeção

Uma técnica de leitura pode ser definida como uma série de etapas ou procedimentos que têm por objetivo guiar um inspetor para que este possa ter uma melhor compreensão do artefato inspecionado (LAITENBERGER, 2001). Esse tipo de técnica é usado em análise individual de produto de software textual (requisitos, projeto, código e plano de teste) e tem como objetivo dar suporte ao inspetor para que este consiga o conhecimento necessário para uma tarefa específica, como por exemplo, detecção de defeitos, reuso e manutenção (BASILI et al., 1996a).

Basili et al.(1996a) desenvolveram uma família de técnicas de leitura que pode ser representada pela árvore da Figura 3.1. A parte superior da árvore (sobre a linha horizontal tracejada) modela os problemas que podem ser tratados pela leitura. Cada nível representa uma outra especialização do problema de acordo com a classificação de atributos, os quais são mostrados na coluna mais à direita da figura. A parte mais baixa da árvore (abaixo da linha tracejada) modela as soluções específicas de um problema em particular.

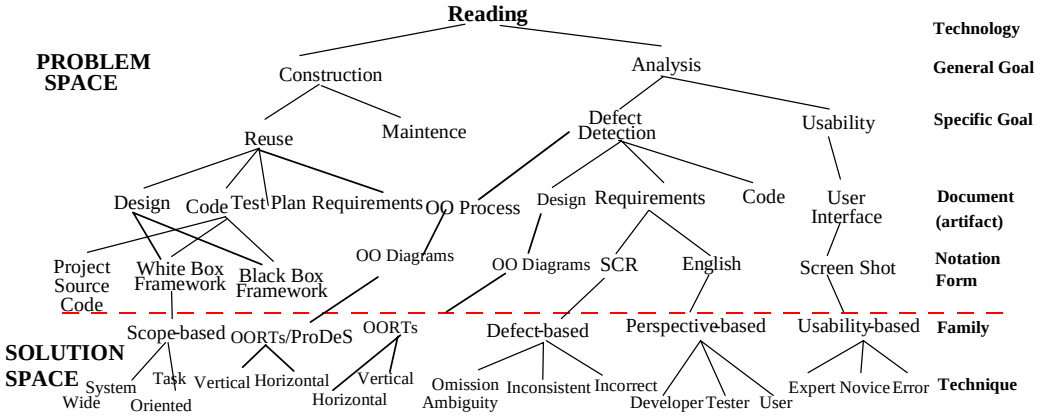


Figura 3.1 - Famílias de técnicas de leitura (adaptado BASILI et al., 1996a).

Segundo Basili et al.(1996a), o espaço solução da árvore consiste de famílias de Técnicas de Leitura. Cada família é associada com um objetivo particular, documento ou artefato de software, e a notação na qual o documento é escrito. As técnicas de leitura possuem as seguintes características:

- adaptável, baseada nas características do projeto e do ambiente;
- detalhada, pois fornece ao leitor um conjunto de passos bem definidos a serem seguidos;
- específica, pois o leitor tem um propósito particular ou específico para a leitura do documento e os procedimentos que apóiam esse objetivo;
- focalizada, pois fornece uma cobertura particular do documento e uma combinação de técnicas da mesma família fornece a cobertura do documento inteiro.
- estudada para determinar empiricamente se e quando é mais eficaz.

Em geral, as técnicas de leitura são baseadas em cenários. Um cenário pode ser um conjunto de perguntas ou uma descrição mais detalhada para um inspetor saber como realizar a inspeção no documento. Essencialmente, um cenário limita a atenção de um inspetor para a detecção de defeitos específicos (LAITENBERGER, 2001).

Alguns autores, segundo Aurum et al.(2002), classificam as Técnicas de Leitura em técnicas não-sistemáticas e técnicas sistemáticas. As técnicas não-sistemáticas são uma abordagem intuitiva que oferecem pouco ou nenhum suporte ao inspetor, como por exemplo, a Ad-hoc e o *Checklist*. As técnicas sistemáticas, como por exemplo, a Leitura Baseada em Perspectiva (PBR), fornecem um conjunto de instruções para os inspetores e explicam como ler o documento de software e o que eles devem procurar. Algumas técnicas serão comentadas a seguir:

- Ad-hoc

A técnica de leitura Ad-hoc não oferece nenhum auxílio ou procedimento de como conduzir a leitura e o que procurar. Nessa técnica, não há necessidade de treinamentos e a detecção de defeitos depende inteiramente da habilidade, do conhecimento e da experiência do inspetor (AURUM et al., 2002).

- *Checklist*

É uma técnica em que o inspetor responde a um questionário depois de ler o documento com o objetivo de ajudá-lo a identificar os principais defeitos ou a priorizar diferentes defeitos. Na inspeção proposta por Fagan (1986), o *Checklist* é a técnica utilizada (AURUM et al., 2002).

As questões de um *Checklist* são freqüentemente gerais e não são suficientemente adaptadas para um ambiente de desenvolvimento particular. Embora essas questões do *Checklist* forneçam um esquema geral sobre o que checar durante uma inspeção, ele não diz de maneira precisa como assegurar essa checagem. Desta maneira, o *Checklist* fornece pouco suporte para um inspetor entender o artefato inspecionado. Um *Checklist* não deve exceder uma página e as frases do questionário devem ser precisas (LAITENBERGER, 2001).

- *Stepwise Abstraction*

É uma técnica voltada para a leitura de código. Nessa técnica, cada revisor identifica subprogramas no software e determina suas funções. Depois, cada revisor determina a função completa do programa por combinação das sub-funções identificadas e redige uma especificação para o programa. Então, o revisor compara essa especificação derivada com a especificação oficial. Com base nessa comparação, são identificados as inconsistências e os defeitos (BASILI, 1997; AURUM et al., 2002).

- *Active Design Review* (ADR)

Essa técnica de leitura é usada para inspecionar documentos de projeto. O processo é dividido em três etapas e começa com uma visão geral, na qual o projetista apresenta um resumo do projeto e as reuniões são marcadas. A etapa seguinte é a de detecção de defeito na qual o autor fornece questionários para guiar os inspetores. As perguntas são projetadas de tal forma que podem somente ser respondidas através de um estudo cuidadoso do documento do projeto, isto é, os inspetores têm que elaborar a resposta em vez de indicar sim/não. A etapa final é a coleta de defeitos, a qual é realizada em reuniões de inspeção. Entretanto, cada reunião de inspeção é dividida em diversas partes menores, equipes especializadas, na qual cada equipe se preocupa em uma propriedade da qualidade do artefato (LAITENBERGER, 2001).

Active Design Review é uma variação importante da inspeção porque os inspetores de ADR são guiados por uma série de perguntas proposta pelo autor do projeto a fim incentivar uma etapa completa da detecção de defeito (LAITENBERGER, 2001).

- Leitura Baseada em Defeito

Essa técnica foi desenvolvida para identificar defeitos em um modelo de requisitos usando uma notação de máquina de estado. Nela os defeitos são classificados nas seguintes classes: inconsistência de tipo de dado, funções incorretas e ambigüidade ou falta de informação. As questões de análise são geradas por combinação e abstração de um conjunto de questões que são usadas em *Checklists* para a avaliação da corretitude e confiabilidade dos documentos de requisitos (BASILI et al., 1996a).

- Leitura Baseada em Perspectiva (PBR)

Essa técnica é utilizada para inspeção de documento de requisitos escrito em linguagem natural. Ela ajuda o revisor a inspecionar o produto sob diferentes perspectivas, como por exemplo, a perspectiva de um projetista de software, de um testador e de um usuário final (BASILI et al., 1996a; BASILI, 1997). As questões usadas para apoiar a leitura do documento são geradas enfocando predominantemente vários tipos de erros de requisitos, como por exemplo, fato incorreto, omissão, ambigüidade e inconsistência, de acordo com a perspectiva assumida pelo leitor (por exemplo, as perguntas para a perspectiva de testador conduzem o leitor a verificar se os requisitos estão livres desses tipos de erros, de forma a permitir a elaboração de casos de teste que poderiam ser utilizados para testar o software depois de implementado) (BASILI et al., 1996a; BASILI, 1997).

Aurum et al. (2002) dizem que, de acordo com alguns autores, os revisores que usam PBR para inspecionar documentos de requisitos tendem a detectar mais defeitos do que aqueles que usam uma técnica menos estruturada. Eles também enfatizam que PBR tem qualidades benéficas porque ela é sistemática, bem definida, adaptável e transferível via treinamento.

- Leitura Baseada em Pontos de Função

Essa abordagem é baseada em Análise de Pontos de Função (FPA). FPA define um sistema de software em termos de suas entradas, arquivos, consultas e saídas. Os cenários

associados com a técnica de pontos de função são desenvolvidos com base nesses itens. Eles consistem de questões que direcionam o foco de um inspetor para um item do ponto de função específico dentro do documento de requisitos inspecionado. (LAITENBERGER et al., 1999; LAITENBERGER, 2001).

- OORTs

OORTs (*Object Oriented Reading Techniques*) é uma técnica de leitura voltada para dar orientação específica e prática para identificação de defeitos em documentos de projeto de alto nível, representados por diagramas UML, considerando um processo de software genérico e simplificado (TRAVASSOS et al., 2002b, 2002c).

Essa família OORTs consiste de 7 diferentes técnicas que dão suporte à leitura de diferentes diagramas, como por exemplo: diagrama de classe, de interação (seqüência e colaboração) e de estado. Geralmente, esses diagramas são os principais diagramas da UML que os desenvolvedores constroem para o projeto de alto nível OO e que capturam as visões estática e dinâmica do problema. Um projeto de alto nível pode ser entendido como um conjunto de artefatos preocupados com a representação dos conceitos do mundo real (TRAVASSOS et al., 2000, 2002b, 2002c).

As OORTs, conforme a Figura 3.2, podem ser divididas em dois grandes grupos: Horizontal e Vertical. As técnicas Horizontais fazem comparações dentro da mesma fase e tendem a procurar defeitos de ambigüidades e inconsistências, enquanto que as Verticais fazem comparações de documentos entre fases e tendem a encontrar mais defeitos de omissão e funcionalidade incorreta (TRAVASSOS et al.,2000, 2002b, 2002c).

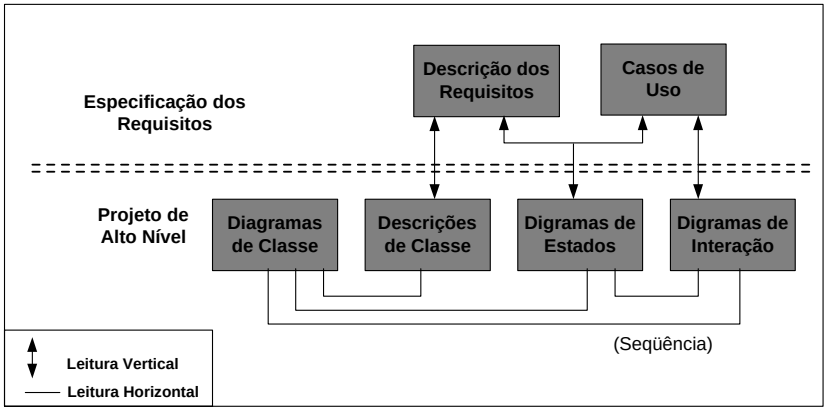


Figura 3.2 - Conjunto de Técnicas de Leitura OORTs (adaptado TRAVASSOS et al., 2002c).

Na Tabela 3.2 são apresentados as técnicas de leitura pertencentes às OORTs, sendo que para cada leitura são listados os documentos a serem inspecionados, o seu tipo e o objetivo.

Tabela 3.2 - Técnicas de Leitura OORTs (MARUCCI, 2002).

Técnica	Documentos inspecionados	Tipo	Objetivo
P1	Diagramas Seqüência x Classes	Horizontal	Verificar se um diagrama de classes para um sistema descreve as classes e seus relacionamentos de forma que os comportamentos especificados nos diagramas de seqüência estão capturados corretamente. Para fazer isso, primeiramente deve-se verificar se as classes e objetos especificados no diagrama de seqüência aparecem no diagrama de classes. Assim, será verificado se o diagrama de classes descreve os relacionamentos, comportamentos e condições que capturam a dinâmica dos serviços como estão descritos no diagrama de seqüência.
P2	Diagramas de Estado x Descrição de Classes	Horizontal	Verificar se as classes estão descritas de forma a capturar a funcionalidade especificada pelo diagrama de estados.
P3	Diagramas de Seqüência x Estados	Horizontal	Verificar se toda transição de estado para um objeto pode ser realizada pelas mensagens enviadas e recebidas pelo objeto.
P4	Diagrama de Classes x Descrições de Classe	Horizontal	Verificar se as descrições detalhadas das classes contêm toda a informação necessária de acordo com o diagrama de classes e se a descrição da classe possui um sentido semântico.
P5	Descrições de Classes x Descrição de Requisitos	Vertical	Verificar se os conceitos e serviços descritos pelos requisitos funcionais estão capturados apropriadamente pela descrições das classes.
P6	Diagramas de Seqüência x Casos de Uso	Vertical	Verificar se os diagramas de seqüência descrevem uma combinação apropriada de objetos e mensagens que trabalham em conjunto para capturar a funcionalidade descrita pelo caso de uso.
P7	Diagrama de Estado x Descrição de Requisitos e Casos de Uso	Vertical	Verificar se os diagramas de estado descrevem apropriadamente os estados dos objetos e eventos que disparam as trocas de estado conforme descritos nos requisitos e casos de uso.

Na subseção a seguir, comentam-se, mais detalhadamente, as técnicas de leitura OORTs/ProDeS que são fundamentais no contexto deste trabalho.

3.3.1 OORTs/ProDeS

As OORTs/ProDeS são um conjunto de técnicas de leitura definido por Marucci (2002), que é utilizado em uma estratégia de inspeção para o processo de desenvolvimento de software OO denominado ProDeS/UML (Processo de Desenvolvimento de Software para UML) apresentado no Capítulo 2. Essas técnicas de leitura foram baseadas nas técnicas de leitura OORTs (TRAVASSOS et al., 2000, 2002b, 2002c) que já estavam definidas para um processo de software genérico e simplificado.

Como foi mencionado no Capítulo 2, o ProDeS/UML é composto por quatro fases: Engenharia de Requisitos, Análise, Projeto e Implementação e possui agregado a ele, atividades de teste que se estendem por todo o processo (COLANZI, 1999).

A cada fase do processo, ocorre uma evolução gradativa e cada vez mais detalhada da solução do problema. Com isso, é facilitada a introdução de Atividades de Garantia de Qualidade ao longo do processo, possibilitando a detecção de defeitos à medida que eles vão surgindo (MARUCCI, 2002). Assim, por todas essas características apresentadas anteriormente em relação ao processo ProDeS/UML e pela necessidade de um processo padrão que a UML não possui, este processo foi o escolhido para o trabalho de Marucci (2002).

Marucci (2002) estabeleceu uma estratégia de inspeção baseada em técnicas de leitura para documentos UML para permitir uma avaliação gradativa e continuada dos artefatos construídos no ProDeS/UML. Essa estratégia seguiu as recomendações propostas por Andriole (1986), o qual considera fundamental em um processo de desenvolvimento de software os seguintes tipos de atividades: i) verificação tem como objetivo contrapor os diversos artefatos de software entre eles mesmos, para assegurar que eles estão sendo gerados de forma correta e consistente; ii) validação tem como objetivo contrapor artefatos de software com o documento de requisitos, para assegurar que o software continue representando-os apropriadamente e também para assegurar que esses requisitos permaneçam atualizados durante todo o processo; iii) checagem de adequação e suficiência, cujo objetivo é comparar a solução com a compreensão que se tem do problema no momento corrente, para assegurar que a solução seja suficiente e a desejada e iv) checagem de evolução, cujo objetivo é assegurar a completitude e consistência entre níveis de especificação, sendo que o segundo nível é um refinamento do primeiro.

Essas técnicas foram elaboradas numa seqüência de passos, análoga às seqüência das técnicas OORTs e são formadas por duas fases principais: uma fase verifica a sintaxe do documento, enquanto a outra fase avalia a semântica do documento (MARUCCI, 2002).

Inicialmente, para definir o conjunto de técnicas de leitura ao longo das fases que compõem o ProDeS/UML, Marucci (2002) realizou uma análise de adequação das técnicas de leitura da OORTs (TRAVASSOS et al., 2000, 2002b, 2002c), avaliando-se a aplicação das mesmas nos diversos artefatos construídos ao longo desse processo. Foi feita uma comparação entre os artefatos utilizados pelas técnicas de leitura OORTs e os artefatos produzidos no processo ProDeS/UML, constatando-se que as OORTs poderiam ser utilizadas somente na Fase de Projeto do ProDeS/UML com algumas adaptações (MARUCCI et al., 2002). Na Figura 3.3 apresentam-se as OORTs no contexto do processo ProDeS/UML. Como pode ser observado, somente quando se chega na fase de Projeto desse processo é que as OORTs são passíveis de aplicação. Ressalta-se que o Diagrama de Colaboração que era produzido na fase de Projeto foi substituído pelo Diagrama de Seqüência já que era este o utilizado pelas OORTs e, segundo Silva (2001), o Diagrama de Seqüência é o dual do Diagrama de Colaboração e possui o mesmo tipo de informação desse.

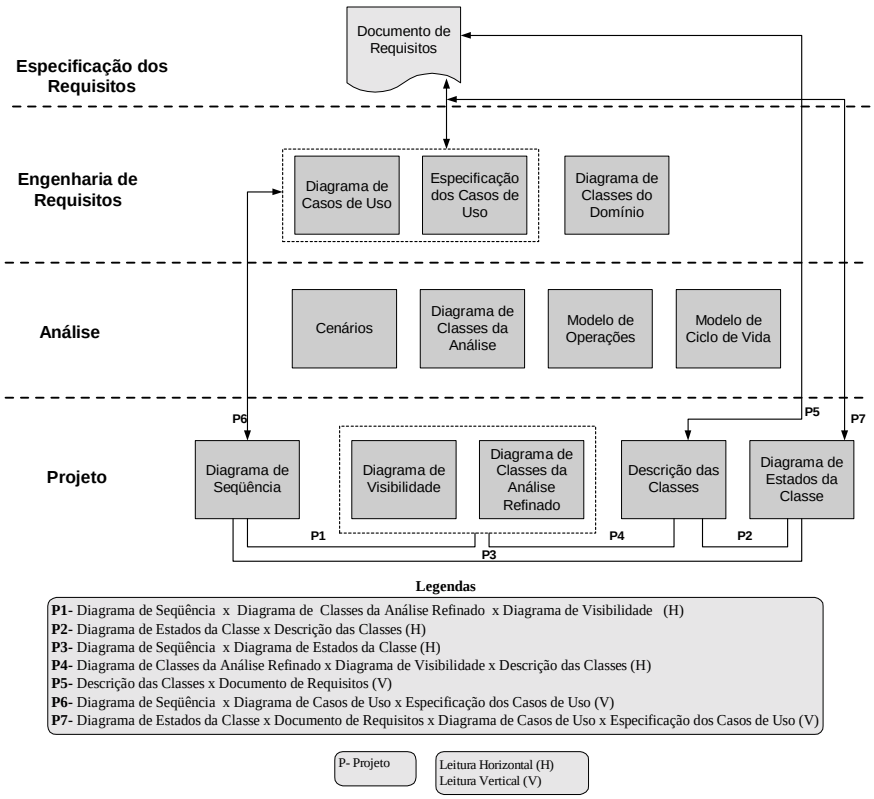


Figura 3.3 - Como as OORTs se aplicam nos Artefatos do ProDeS/UML (MARUCCI, 2002).

Depois dessa análise de adequação das OORTs nos artefatos gerados no ProDeS/UML, Marucci (2002) observou que novas técnicas deveriam ser elaboradas para os artefatos que são gerados na Fase de Engenharia de Requisitos, na Fase de Análise e mesmo na fase de Projeto, para avaliar alguns pontos que não eram tratados pelas OORTs, pelo fato desta não considerar um processo evolutivo de desenvolvimento. Essas novas técnicas são apresentadas na Figura 3.4 e são referenciadas como ER1, ER2, A1, A2, A3, A4, P8 e P9. Elas possuem como objetivo manter a rastreabilidade das informações entre os artefatos e a consistência das informações entre eles e entre os requisitos.

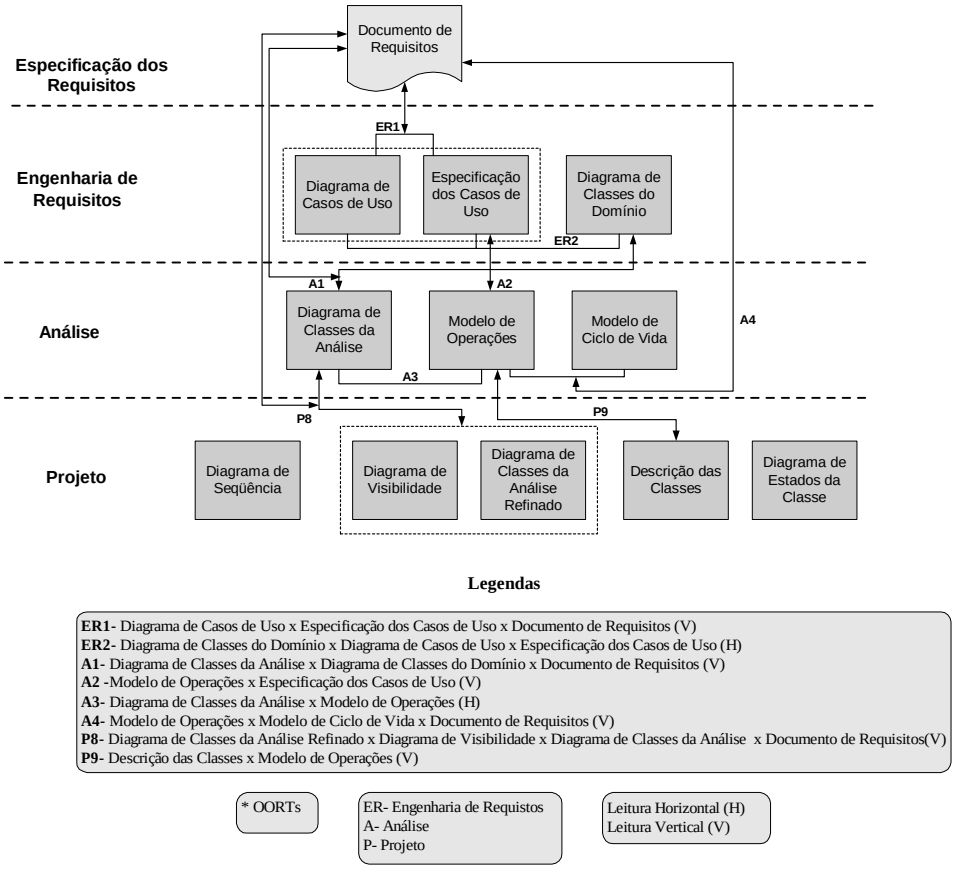


Figura 3.4 - Novas Técnicas de Leitura para o ProDeS/UML (MARUCCI, 2002).

A seguir é descrito como Marucci (2002) distribuiu essas novas Técnicas de Leitura no processo ProDeS/UML e quais são as finalidades de cada uma delas.

Na Fase de Engenharia de Requisitos foram identificadas duas Técnicas de Leitura: ER1 e a ER2 que são apresentadas na Figura 3.5. A ER1 é classificada como uma técnica de validação e é do tipo vertical. Já a ER2 é uma técnica de verificação do tipo horizontal.

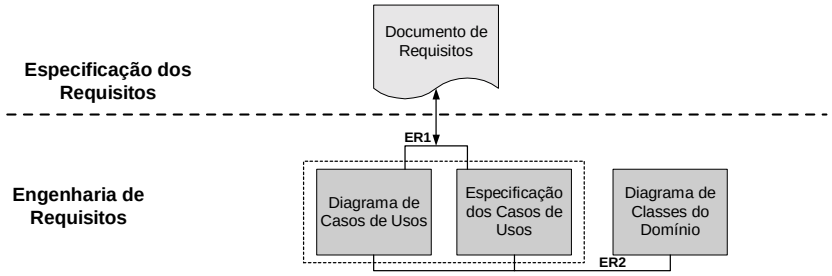


Figura 3.5 - Técnicas de Leitura para a Fase de Engenharia de Requisitos (MARUCCI, 2002).

A ER1 analisa os seguintes documentos: Diagrama de Casos de Uso, Especificação dos Casos de Uso e o Documento de Requisitos. Essa técnica tem como objetivo verificar se os conceitos e as funcionalidades que foram especificadas no Documento de Requisitos, o qual foi elaborado na Fase de Especificação de Requisitos, estão respectivamente contidos e descritos corretamente no Diagrama de Casos de Uso e na Especificação dos Casos de Uso.

A técnica de leitura ER2 que se caracteriza como uma checagem de adequação e suficiência, avalia se o Diagrama de Classes do Domínio possui os conceitos (atores, classes candidatas e atributos) e os relacionamentos exatos aos que estão descritos no Diagrama e Especificação dos Casos de Uso.

Na Fase de Análise foram identificadas as técnicas de leitura A1, A2, A3 e A4. Essas técnicas são mostradas na Figura 3.6 e englobam todos os artefatos produzidos nas fases anteriores que são as Fases de Especificação e Engenharia de Requisitos e também da Fase de Análise, com exceção dos Cenários. Como a elaboração deste artefato é opcional para ajudar a construir o Modelo de Operações e o Modelo de Ciclo de Vida, Marucci (2002) não definiu nenhuma técnica envolvendo esse documento.

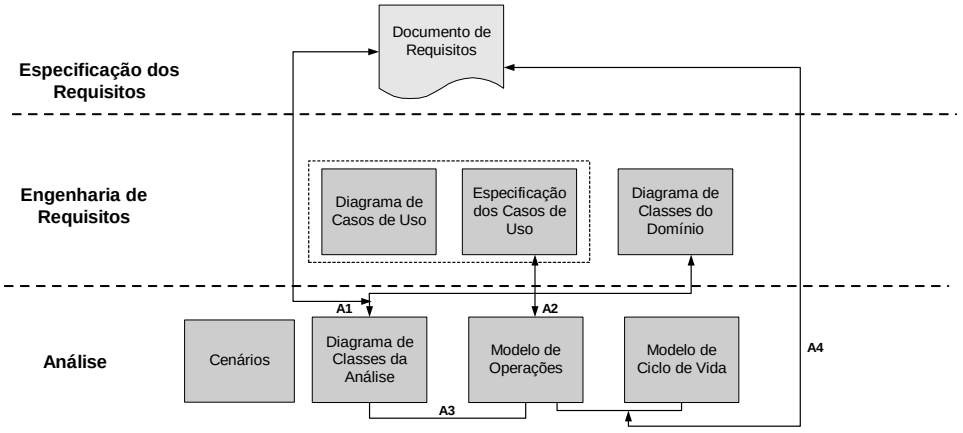


Figura 3.6 - Técnicas de Leitura para a Fase de Análise (MARUCCI, 2002).

Enquanto a Técnica de Leitura A3 é do tipo horizontal, as outras são do tipo vertical. As Técnicas de Leitura A3 e A2 são classificadas como técnicas de verificação e a A1 e A4 são de validação.

A Técnica de Leitura A1 é caracterizada como uma checagem de evolução, de adequação e suficiência. Ela tem como objetivo verificar se os conceitos (atores, classes ou atributos) e os relacionamentos presentes no Diagrama de Classes da Análise estão de acordo com o que está contido no Diagrama de Classes do Domínio. Também em A1 é checado se as novas informações existentes no Diagrama de Classes da Análise estão compatíveis com o Documento de Requisitos.

A Técnica de Leitura A2 é caracterizada como uma checagem de evolução e realiza uma comparação entre as informações contidas no Modelo de Operações e a Especificação dos Casos de Uso que foi produzido na fase anterior.

A Técnica de Leitura A3 é caracterizada como uma checagem de integridade, adequação e suficiência. Para se construir o Modelo de Operações utilizam-se informações contidas no Diagrama de Classes da Análise, por isso essa técnica verifica se as etapas para a realização de uma operação no Modelo de Operações, estão de acordo com o que está representado no Diagrama de Classes de Análise.

A Técnica de Leitura A4 é caracterizada por ser uma checagem de adequação e suficiência. A4 verifica se os eventos que disparam a troca de estados do sistema apresentados no Modelo de Ciclo de Vida estão compatíveis com as operações descritas no Modelo de Operações. O Modelo de Ciclo de Vida descreve o comportamento completo de

como o sistema se comunica com o ambiente, desde sua criação até seu término, por esta razão este artefato é analisado junto as informações contidas no Documento de Requisitos.

Na Fase de Projeto, mesmo com o uso das Técnicas de Leitura OORTs após algumas alterações realizadas em alguns artefatos do ProDeS/UML, houve a necessidade da criação de mais duas Técnicas de Leitura: P8 e P9, pois alguns artefatos não eram analisados pelas OORTs. As duas técnicas que estão apresentadas na Figura 3.7, são do tipo vertical, sendo que a P8 é classificada como técnica de validação e a P9 como técnica de verificação.

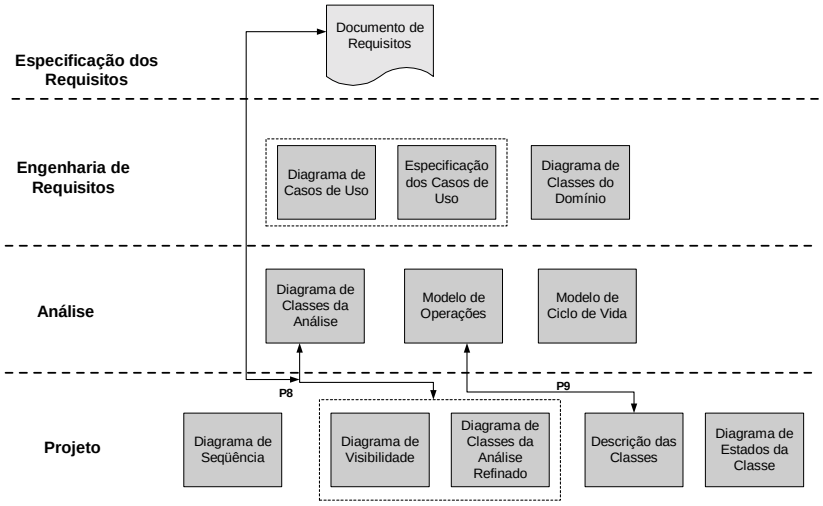


Figura 3.7 - Técnicas de Leitura para a Fase de Projeto (MARUCCI, 2002).

A Técnica de Leitura P8 caracteriza-se como uma checagem de evolução, de adequação e suficiência. Ela tem como objetivo avaliar se o Diagrama de Classes da Análise Refinado é realmente uma evolução do Diagrama de Classes da Análise e se as informações do Diagrama de Visibilidade foram baseadas no Documento de Requisitos.

A Técnica de Leitura P9 é considerada uma checagem de adequação e suficiência. Essa técnica verifica se as Descrições das Classes descrevem todas as informações para a execução das operações descritas no Modelo de Operações.

Na Tabela 3.3 é apresentado um resumo das Técnicas de Leitura definidas no trabalho de Marucci (2002).

Tabela 3.3 - Técnicas de Leitura desenvolvidas para apoiar o ProDeS/UML (MARUCCI, 2002).

Técnica	Documentos inspecionados	Tipo	Objetivo
ER1	Diagrama de Casos de Uso x Especificação dos Casos de Uso x Documento de Requisitos	Validação	Verificar se os conceitos e funcionalidades que estão descritos no Documento de Requisitos foram capturados apropriadamente pelo Diagrama de Casos de Uso e se as Especificações dos Casos de Uso estão descritas de forma precisa.
ER2	Diagrama de Classes do Domínio x Diagrama de Casos de Uso x Especificação dos Casos de Uso	Verificação	Verificar se os conceitos (atores, classes candidatas ou atributos) e os relacionamentos representados no Diagrama de Classes do Domínio capturam apropriadamente os conceitos sobre os atores, entidades a serem mantidas no sistema e atributos relacionados a elas, os quais estão descritos no Diagrama e Especificações dos Casos de Uso.
A1	Diagrama de Classes da Análise x Diagrama de Classes do Domínio x Documento de Requisitos	Validação	Verificar se os conceitos (atores, classes ou atributos) e os relacionamentos representados no Diagrama de Classes da Análise estão compatíveis com as informações contidas no Diagrama de Classes do Domínio e se as novas informações presentes nesse diagrama estão retratando corretamente as informações contidas no Documento de Requisitos.
A2	Modelo de Operações x Especificação dos Casos de Uso	Verificação	Verificar se a descrição de cada operação representada pelo Modelo de Operações está condizente com a Especificação dos Casos de Uso.
A3	Diagrama de Classes da Análise x Modelo de Operações	Verificação	Verificar se os passos para realizar a operação são consistentes e se os conceitos (atores, classes ou atributos) e os relacionamentos que estão sendo usados para que a operação seja realizada são consistentes e compatíveis com a forma como estão representados no Diagrama de Classes da Análise.
A4	Modelo de Operações x Modelo de Ciclo de Vida x Documento de Requisitos	Validação	Verificar se o Modelo de Ciclo de Vida descreve apropriadamente os eventos que disparam a troca de estados do sistema, se esses eventos correspondem a cada operação descrita no Modelo de Operações e se o estado alcançado pelo sistema após a execução da operação é apropriado. Além disso, verificar também se as representações em ambos os artefatos estão precisas conforme a descrição contida no Documento de Requisitos.
P8	Diagramas de Classes da Análise Refinado x Diagrama de Visibilidade x Diagrama de Classes da Análise x Documento de Requisitos	Validação	Verificar se os conceitos (atores, classes ou atributos) e os relacionamentos representados no Diagrama de Classes da Análise Refinado estão compatíveis com as informações contidas no Diagrama de Classes da Análise e se as novas informações presentes nesse diagrama, bem como se as relações de dependência e navegabilidade do Diagrama de Visibilidade estão condizentes com o Documento de Requisitos.
P9	Descrição de Classes x Modelo de Operações	Verificação	Verificar se as Descrições das Classes contêm toda a informação necessária para realização das operações descritas pelo Modelo de Operações.

Na Figura 3.8 apresentam-se todas as técnicas de leitura que podem ser aplicadas no processo ProDeS/UML, sendo que as técnicas ER1, ER2, A1, A2, A3, A4, P8 e P9 foram definidas no trabalho de Marucci (2002) e as técnicas de P1 a P7 foram definidas por Travassos et al. (2000, 2002b, 2002c). Todas elas compõem o conjunto de técnicas de leitura denominado OORTs/ProDeS.

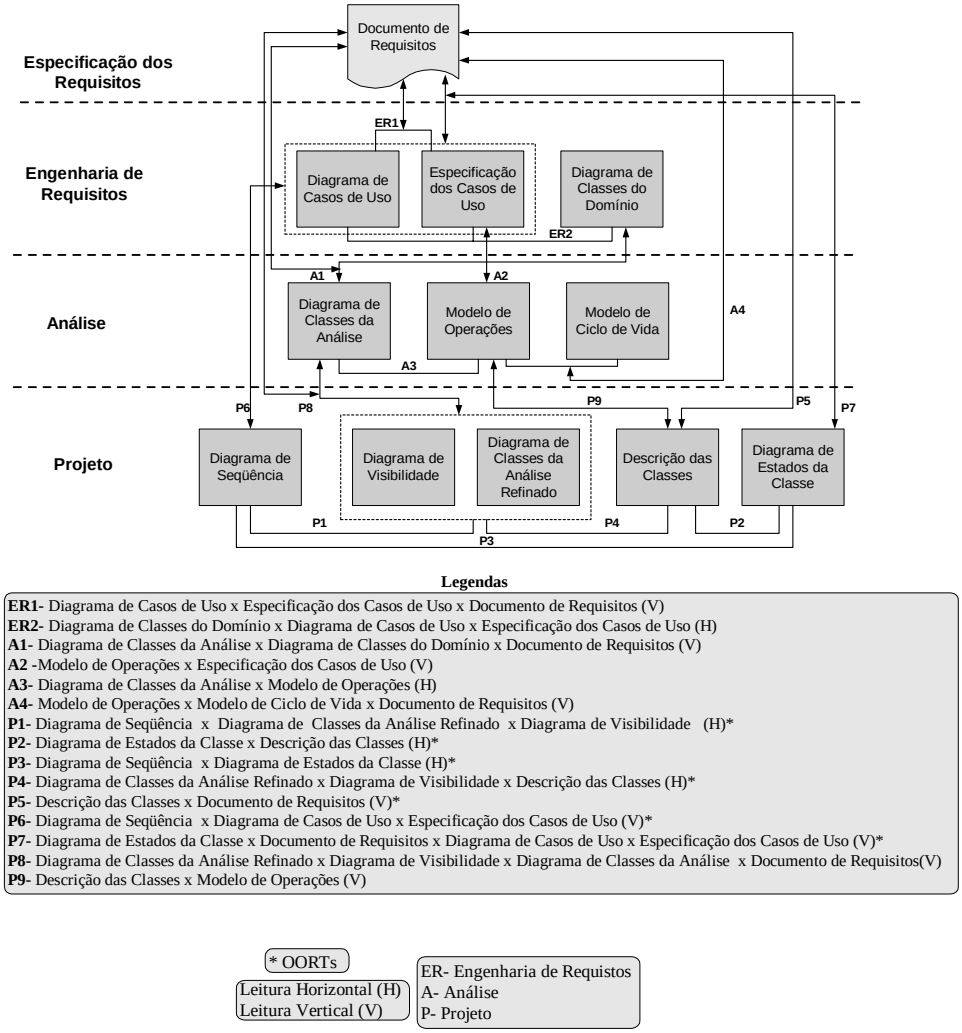


Figura 3.8 - Técnicas de Leitura OORTs/ProDeS (MARUCCI, 2002).

Para utilização dessas técnicas, Marucci (2002) definiu também uma estratégia de aplicação. Nessa estratégia, a ordem de aplicação dessas Técnicas de Leitura é um fator primordial para a obtenção de uma maior eficiência na detecção de defeitos no processo ProDeS/UML. A aplicação de todas essas técnicas, de acordo com a estratégia proposta

por Marucci, deve seguir a seguinte ordem: ER1, ER2, A1, A2, A3, A4, P6, P8, P1, P9, P5, P4, P7, P3 e P2 e o processo de inspeção aplicado no ProDeS/UML obedece as seguintes etapas (MARUCCI, 2002):

1. Aplicar uma técnica de leitura desde que se tenha um conjunto mínimo necessário de artefatos para sua aplicação;
2. Coletar as discrepâncias identificadas;
3. Analisar as discrepâncias coletadas, gerando uma lista com aquelas consideradas defeitos a serem corrigidos;
4. Corrigir os defeitos e retornar ao passo 1 até que se obtenha um artefato mais livre de defeitos possível.

3.4 O Processo de Experimentação

O propósito de abordar o processo de experimentação no contexto deste trabalho é que:

i) para se avaliar novas técnicas, métodos, ferramentas, etc., tem-se dado, atualmente, grande relevância aos estudos empíricos. Esses constituirão uma atividade que deverá ser conduzida em trabalhos futuros para que sejam avaliados o material de suporte produzido neste trabalho para a condução de experimentos e algumas técnicas de leitura propostas no contexto de XP (*Extreme Programming*);

ii) uma das tarefas do processo de experimentação é justamente elaborar os Pacotes de Laboratório para que os experimentos possam ser replicados, isso constitui uma das tarefas realizadas neste trabalho.

Wohlin et al.(2000) explicam que os experimentos são apropriados para investigar diferentes aspectos, incluindo:

- Confirmar teorias, isto é, testar teorias.
- Confirmar sabedoria convencional, isto é, testar as concepções das pessoas.
- Explorar relacionamentos, isto é, testar que um certo relacionamento se conserve.
- Avaliar a precisão dos modelos

- Validar medidas

O processo de experimentação definido por Wohlin et al. (2000) envolve diferentes etapas, conforme pode ser observado na Figura 3.9:

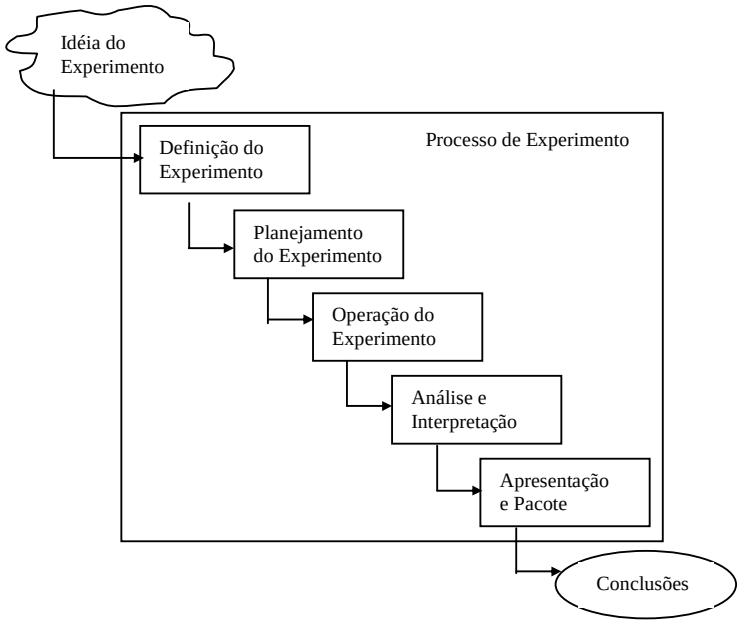


Figura 3.9 - Visão Geral do Processo de Experimentação (adaptado WOHLIN et al., 2000).

- **Definição**

É a primeira etapa, na qual o experimento é definido em termos de problema, objetivo e idéias. Em outras palavras, pode-se dizer que é determinada a base de fundação do experimento. Se a fundação não é propriamente apresentada, pode ocorrer um re-trabalho ou até mesmo o pior, o experimento pode não ser usado para o estudo que era intencionado.

Nessa fase é elaborado um esqueleto de definição de idéias com o propósito de assegurar que todos os aspectos importantes do experimento sejam definidos antes que o planejamento e a execução aconteçam. A seguir, são apresentadas a estrutura do esqueleto de idéias e a definição dos termos usados:

Análise <objeto de estudo>

com o propósito de <propósito>

com respeito a <foco de qualidade>

do ponto de vista da <perspectiva>

no contexto de <contexto>

Sendo que:

<objeto de estudo> é a entidade que é estudada no experimento;

<propósito> qual é a intenção do experimento;

<foco de qualidade> indica o principal aspecto da qualidade que está sendo estudado, por exemplo, eficiência, confiabilidade, custo;

<perspectiva> exibe o ponto de vista no qual os resultados do experimento são interpretados, por exemplo, perspectiva do desenvolvedor, cliente, gerente de projeto e pesquisador;

<contexto> é o ambiente no qual o experimento será realizado.

- **Planejamento**

A fase anterior determinava a base fundamental para o experimento, ou seja, porque o experimento será executado. A Fase de Planejamento prepara o "como" o experimento vai ser conduzido.

Aqui é selecionado o ambiente no qual o experimento será executado; a formulação de hipóteses, a escolha das variáveis, o projeto experimental, implementação do experimento e a avaliação da validade do experimento.

- **Operação**

Quando um experimento é projetado e esboçado ele deve ser executado com o objetivo de coletar os dados que deverão ser analisados. É exatamente isso que a Fase de Operação faz; executa o experimento.

Esta fase é dividida em três etapas:

1. Preparação: os participantes são escolhidos, organizados e preparados; o material utilizado no experimento também é preparado.
2. Execução: os participantes desempenham suas tarefas de acordo com o que foi planejado no projeto experimental e os dados são coletados.
3. Validação de Dados: os dados que foram coletados na etapa anterior são validados, no sentido de conferir se tudo foi anotado apropriadamente.

- **Análise e Interpretação**

Nessa etapa, os dados devem ser analisados e interpretados.

A primeira etapa a ser realizada é a estatística descritiva, a qual explora a tendência central, dispersão, etc. Depois disso, os dados com pontos anormais e falsos são excluídos, ocorrendo uma redução do conjunto de dados para um conjunto de dados válidos, sobre o qual serão aplicados outros métodos estatísticos para que as hipóteses possam ser avaliadas.

- **Apresentação e Pacote**

Depois que um experimento foi realizado, o objetivo, geralmente, é apresentar os resultados obtidos e documentá-lo de tal forma que possa ser repetido. Isto pode ser feito em um relatório para conferência, um pacote para replicação de experimento ou material educacional. O empacotamento tem como finalidade permitir que outros pesquisadores reutilizem o estudo e repitam o experimento. Ele pode ser também realizado dentro de empresas com a finalidade de melhorar e entender diferentes processos.

Considerando-se que apenas com grande volume de dados e com várias replicações de um experimento é que as hipóteses e as técnicas avaliadas podem ser caracterizadas, o empacotamento é uma atividade fundamental. A seguir, essa atividade é comentada com mais detalhe, pois um dos produtos deste trabalho foi material para compor Pacotes de Laboratório.

Empacotamento de Experimento:

Conforme o trabalho de Amaral e Travassos (2002), grande parte das publicações da área de Ciência da Computação não apresenta um experimento para comprovar o que foi escrito. Foi realizada uma comparação com as outras ciências e foi constatado que o percentual de pesquisadores que validam o seu trabalho é bem inferior em relação às outras áreas científicas.

Um fator que torna ainda mais grave essa situação é a falta de visibilidade dos experimentos já realizados. Ou seja, somente o cientista que elaborou o experimento conhece detalhadamente o processo aplicado no experimento, as técnicas e os resultados. Quando um novo cientista for estudar o mesmo assunto, praticamente ele vai começar o estudo do zero (AMARAL; TRAVASSOS, 2002).

Para que isto não ocorra, o empacotamento de experimento é necessário. Um empacotamento de experimento possui como objetivo armazenar todos os artefatos que foram usados durante o processo de experimentação, impedindo assim, a perda de informações importantes durante a execução do mesmo (AMARAL; TRAVASSOS, 2002). Deste modo, o conhecimento de um cientista torna-se público aos demais que estão pesquisando o mesmo assunto.

Por se tratar de uma área nova na engenharia de software, o empacotamento de experimentos não possui normas internacionais aprovadas (TRAVASSOS et al., 2002a).

Segundo Visaggio (2002), a utilização de um pacote de experimento permite que a aquisição de tecnologia seja mais rápida e também possa ser adotado por diferentes especialistas para um determinado projeto.

Amaral e Travassos (2002) propuseram uma alteração no processo de experimentação de Wohlin et al. (2000), no qual a etapa de Apresentação e Empacotamento é executada paralelamente com as outras fases, com a intenção de evitar a perda de informações e conhecimento ao longo do processo. Também são definidos momentos para tomada de decisão, com o objetivo de um maior controle sobre a execução do processo. Na Figura 3.10 apresenta-se o processo de experimentação, de acordo com a proposta de Amaral e Travassos (2002).

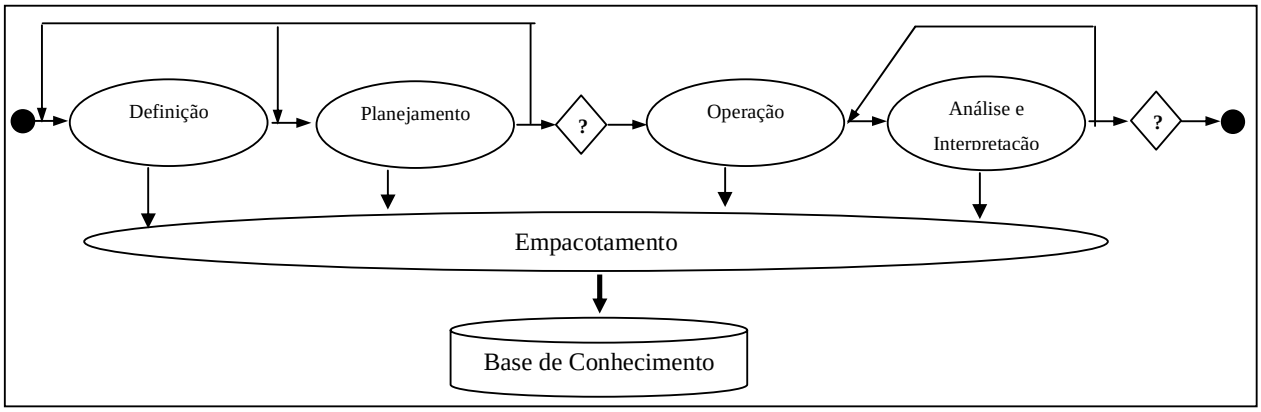


Figura 3.10 - Processo para Empacotamento de Experimento (AMARAL; TRAVASSOS, 2002).

Em cada fase são produzidos vários tipos de documentos para a armazenagem de informações importantes e necessárias para um empacotamento, que podem ser gerados parcialmente ou integralmente (AMARAL; TRAVASSOS, 2002).

No empacotamento de experimento é elaborado como produto o Pacote de Laboratório. Este descreve o experimento em termos específicos, fornece material para replicação, destaca oportunidades de variação e constrói um contexto para combinação de resultados em diferentes tipos de tratamentos experimentais. Ele também constrói uma infra-estrutura experimental para dar suporte a futuras replicações (SHULL et al., 2002).

Dessa maneira, o Pacote de Laboratório estabelece a base para confirmação/negação dos resultados originais, complementa o experimento original e adapta o objeto de estudo para um contexto experimental específico (SHULL et al., 2002).

A replicação é o atributo mais importante associado ao processo de experimentação e permite que outros pesquisadores reproduzam o experimento, possibilitando com isso, que as hipóteses sejam validadas. Se existem replicações suficientes de um determinado experimento, o risco dos resultados desse experimento estarem incorretos fica reduzido (ZELKOWITZ; WALLACE, 1997).

3.5 Considerações Finais

Neste capítulo foi apresentado o Processo de Inspeção o qual é composto por várias fases, sendo que uma delas, a de revisão propriamente dita, é apoiada por técnicas de leitura que servem para direcionar a leitura do inspetor, com o intuito de ajudá-lo a encontrar o maior número de defeitos possível. A atividade de inspeção é uma atividade de VV&T e, por conseguinte, uma atividade de Garantia de Qualidade de Software que tem se mostrado bastante efetiva, na prática, para a detecção de defeitos.

Foram comentadas também algumas técnicas de leitura, dando-se uma maior ênfase às técnicas OORTs/ProDeS, as quais são usadas em uma estratégia de inspeção para o processo de desenvolvimento OO ProDeS/UML. Salienta-se que a estrutura gradativa de desenvolvimento desse processo, como foi apresentado no Capítulo 2, é que permitiu com que essa estratégia de inspeção apresentada neste capítulo pudesse contemplar as características de um bom processo de desenvolvimento, segundo Andriole (1986). Esse processo e essas técnicas constituem um dos alvos de estudo deste trabalho, como será visto nos Capítulos 5 e 6.

Além disso, foi também apresentado neste capítulo o Processo de Experimentação, pois é através de estudos experimentais que novo ferramental proposto na literatura, como

por exemplo, novas técnicas de leitura para apoiar atividades de inspeção, pode ser avaliado e caracterizado, de forma a subsidiar a escolha dos projetistas, quando isso se fizer necessário. A avaliação e caracterização decorrentes dos estudos experimentais terão maior validade quanto mais dados derem suporte às conclusões a que se pode chegar, da condução desses estudos. Assim, quanto mais estudos puderem ser conduzidos para avaliar o mesmo ferramental em ambientes ou culturas diferentes, mais significativa será a caracterização. Em outras palavras, isso implica que o mesmo experimento seja replicado diversas vezes, da forma mais fiel possível, o que pode ser conseguido com o apoio de Pacotes de Laboratórios que contenham todo material necessário que permita com que pesquisadores diferentes possam replicar o experimento, mesmo que distantes geograficamente dos pesquisadores que definiram e aplicaram o experimento inicialmente.

Assim, os conceitos apresentados neste capítulo dão suporte ao entendimento das propostas realizadas neste trabalho, as quais serão apresentadas nos Capítulos 5 e 6, em que as técnicas de leitura OORTs/ProDeS serão discutidas no contexto de um processo mais prático, o ProDeS/UML|pr e as atividades de inspeção e técnicas de leitura serão exploradas e definidas no contexto de métodos ágeis, em particular o XP, respectivamente.

CAPÍTULO 4

DESENVOLVIMENTO ÁGIL

4.1 Considerações Iniciais

Dados os objetivos deste trabalho de explorar as características dos métodos ágeis no contexto do ProDeS/UML de forma a torná-lo um processo mais prático e também de definir uma estratégia de inspeção no contexto dos métodos ágeis propriamente ditos, foram necessários vários estudos relacionados com diferentes abordagens para o desenvolvimento ágil para se conhecer as suas características e as suas práticas.

Agilidade, para uma organização de desenvolvimento de software, é a habilidade de se adaptar e de reagir prontamente e apropriadamente às mudanças em seu ambiente e às demandas impostas por esse ambiente. Um método ágil é aquele que facilmente permite e dá suporte a esse tipo de adaptabilidade (KRUCHTEN, 2001).

Por esse motivo, os métodos ágeis de desenvolvimento de software têm chamado a atenção de engenheiros de software e pesquisadores do mundo todo. Mesmo assim, pesquisas acadêmicas sobre o assunto ainda são escassas, pois a maioria das publicações existente é escrita por praticantes ou consultores (ABRAHAMSSON et al., 2003; GRÜENBACHER, 2003). No entanto, dada a grande popularidade que esse "movimento ágil" vem ganhando, considerou-se relevante abordar esse paradigma no contexto do trabalho aqui proposto.

Dessa forma, este capítulo está elaborado da seguinte maneira: na Seção 4.2 são apresentados alguns dos principais métodos ágeis, sendo que o XP é mais detalhado por ser alvo de uma das propostas feitas no contexto deste trabalho; na Seção 4.3 são apresentados dois trabalhos que discutem alguns pontos fracos do XP, principalmente no que se refere às atividades de levantamento de requisitos e na Seção 4.4 apresentam-se as considerações finais.

4.2 Abordagens Ágeis

Em muitos métodos de desenvolvimento, o trabalho começa com a solicitação e a documentação de um conjunto completo de requisitos. Em meados de 1990, alguns projetistas que praticavam esse procedimento, passaram a considerar essas etapas iniciais de desenvolvimento frustrantes e talvez impossíveis. Isso seria decorrente da indústria e da tecnologia estarem em constante transformação, dos requisitos também se alterarem, sem contar que, muitas vezes, os clientes são incapazes de relatar suas necessidades. Como conseqüência, vários consultores desenvolveram, independentemente, algumas abordagens (métodos e metodologias) para responder às mudanças inevitáveis que eles vivenciaram (COHEN et al., 2003).

Tais abordagens são conhecidas como “Métodos Ágeis” e correspondem a uma coleção de técnicas (ou práticas) diferentes que compartilham os mesmos valores e princípios básicos, ou seja, possuem características comuns (COHEN et al., 2003; CHAU et al., 2003). Como exemplo dessas características, pode-se citar o desenvolvimento iterativo, o foco na interação, a comunicação e a redução de artefatos intermediários. O desenvolvimento em iterações permite que a equipe de desenvolvimento se adapte rapidamente às mudanças de requisitos. O trabalho em um ambiente em que exista proximidade, concentrado na comunicação e que promova o aspecto interativo permite que as próprias equipes possam tomar decisões e ações imediatamente, ao invés de ficar esperando troca de correspondências. A redução de artefatos intermediários, que não adicionam nenhum valor à entrega final, pode significar mais recursos a serem dedicados ao desenvolvimento do produto, fazendo com que este possa ser completado mais rapidamente (COHEN et al., 2003).

Essas abordagens foram desenvolvidas em três diferentes continentes: na Europa o *Dynamic Systems Development Method* (STAPLETON, 1997); na Austrália o *Feature-Driven Development* (COAD et al., 1999) e nos EUA o *Extreme Programming* (BECK, 2000), o *Crystal* (COCKBURN, 2002), o *Adaptive Software Development* (HIGHSMITH, 2000) e o *Scrum* (SCHWABER; BEEDLE, 2002). Embora suas práticas e filosofias compartilhem similaridades fundamentais, esses praticantes foram autores de suas abordagens independentemente (WILLIAMS; COCKBURN, 2003).

Em Fevereiro de 2001, dezessete desses praticantes e autores se reuniram para discutir as similaridades fundamentais e suas experiências com os seus respectivos

Métodos Ágeis. Como resultado dessa reunião, eles formaram uma "Aliança Ágil" e também escreveram um "Manifesto para o Desenvolvimento de Software Ágil" (BECK et al., 2001), que descreve os quatro valores essenciais:

- Indivíduos e interações mais que processos e ferramentas,
- Software operante mais que documentações completas,
- Colaboração do cliente mais que negociações contratuais,
- Responder às mudanças mais que seguir um planejamento.

A seguir serão comentados, brevemente, alguns Métodos Ágeis mais conhecidos, sendo que o XP será apresentado com mais detalhes, pois está mais relacionado com o trabalho aqui proposto.

- *Adaptive Software Development* (ASD): propõe uma nova maneira de ver o desenvolvimento de software em uma organização, promovendo um paradigma adaptativo. Ele oferece soluções para o desenvolvimento de sistemas grandes e complexos. O método encoraja o desenvolvimento incremental e iterativo com constante prototipação (ABRAHAMSSON et al., 2002, 2003).
- *Agile Modeling* (AM): é uma metodologia baseada na prática que visa a uma modelagem e a uma documentação eficazes no desenvolvimento de sistemas de software. A metodologia AM é uma coleção de práticas, guiadas por princípios e valores e não possui procedimentos detalhados de como criar um certo tipo de modelo. Ao invés disso, ela fornece conselhos de como ser um modelador eficaz. Por este motivo, AM não é um método de desenvolvimento de software completo e deve ser usado juntamente com um outro processo de desenvolvimento de software, como por exemplo o XP (*Extreme Programming*) e o UP (*Unified Process*) (AMBLER, 2002).
- *Crystal Family*: A família *Crystal* inclui vários métodos diferentes sendo que dentre eles deve ser selecionado o mais apropriado para cada um dos projetos individualmente. Além dos métodos, a abordagem *Crystal* também inclui princípios para a adaptação desses métodos em diversas circunstâncias de projetos diferentes. Cada método da família *Crystal* é marcado com uma cor indicando o seu "peso". *Crystal* propõe a escolha de um método de cor apropriada, baseando-se no tamanho e no aspecto crítico do projeto. Métodos *Crystal* estão abertos para qualquer prática

de desenvolvimento, ferramenta ou produtos de trabalho (COCKBURN, 2002; ABRAHAMSSON et al., 2003).

- *Dynamic Systems Development Method* (DSDM): A primeira versão do método foi proposta em 1994 e este pode ser visto como o primeiro método verdadeiro de desenvolvimento de software ágil (ABRAHAMSSON et al., 2003). Segundo Tuffs et. al (1999), DSDM é mais um framework do que um método, pois ele não diz como as coisas devem ser feitas com detalhes, mas fornece um esqueleto de processo e descrições de produto que devem ser adaptadas para um projeto particular ou uma organização particular. A idéia fundamental por trás do DSDM é que ao invés de ajustar a quantidade de funcionalidade em um produto e então depois ajustar o tempo e os recursos para alcançá-la, ele prefere ajustar primeiro o tempo e os recursos, para depois ajustar a quantidade de funcionalidade (ABRAHAMSSON et al., 2002).
- *Feature-Driven Development (FDD)*: é uma abordagem ágil e adaptativa para o desenvolvimento de sistemas. Seu foco está nas fases de projeto e construção. A abordagem FDD agrega desenvolvimento iterativo, enfatiza aspectos de qualidade durante todo o processo e inclui entregas freqüentes e tangíveis, com monitoramento cuidadoso ao longo do progresso do projeto. (ABRAHAMSSON et al., 2003).
- *Scrum*: foi desenvolvido para gerenciar o processo de desenvolvimento de software em um ambiente no qual os requisitos estão em constante mudança (ABRAHAMSSON et al., 2003). Nesse método os projetos progridem em uma série de iterações mensais chamados de *sprints* (período curto de atividade intensa). As equipes de desenvolvimento se encontram com o cliente antes de cada *sprint* para priorizar o trabalho a ser feito e selecionar as tarefas que a equipe pode completar no *sprint* que está por ser iniciado. Durante o *sprint*, a equipe é conduzida por breves reuniões diárias. No final de cada sprint, a equipe entrega um produto potencialmente incrementado (COHN; FORD, 2003).

Além desses, um outro método ágil bastante conhecido e talvez o mais utilizado na prática é o XP. Como, no contexto deste trabalho, tem-se um maior interesse por esse método, ele será descrito com maior detalhe a seguir.

4.2.1 Extreme Programming (XP)

XP é uma nova maneira de melhorar a agilidade de projetos de software. XP confia em vários princípios (teste automatizado, programação em pares, etc) que encurtam o ciclo de vida do projeto entre as versões. Mas estes princípios são também planejados para a obtenção de uma melhoria geral da qualidade de software e da satisfação do usuário, evitando problemas devido à demora e orçamentos excessivos (SANZ, 2002).

XP (*Extreme Programming*) foi criada por Kent Beck (2000) e é definido como um método leve voltado para equipes pequenas e médias que desenvolvem software com mudanças de requisitos rápidas ou vagas.

O XP tem como fundamento quatro valores que são Comunicação, Simplicidade, *Feedback* e Coragem. A partir desses valores foram definidas as doze práticas usadas em XP (BECK, 2000):

- Planejamento do Jogo (*Planning Game*): Determina rapidamente o escopo das próximas versões, combinando as prioridades de negócio e as estimativas técnicas.
- Pequenas Versões (*Small releases*): Coloca rapidamente um sistema simples em produção e o atualiza freqüentemente em ciclos bastante curtos.
- Metáfora (*Metaphor*): Ajuda as pessoas envolvidas no projeto a compreender o sistema sem a necessidade de um conhecimento específico, as vezes difícil de adquirir. O sistema é descrito em poucos parágrafos, deixando de lado o uso de termos técnicos, facilitando a compreensão não só dos desenvolvedores, mas também dos clientes.
- Projeto simples (*Simple design*): O sistema deve ser projetado o mais simples possível. Qualquer complexidade extra é removida assim que ela é descoberta.
- Testes (*Testing*): Os programadores escrevem testes de unidade continuamente. Esses testes devem ser executados e o seu resultado deve ser aprovado para que o desenvolvimento continue. Por outro lado, os clientes também escrevem testes para validar se os requisitos estão sendo atendidos.
- Refatoramento (*Refactoring*): Os programadores procuram aperfeiçoar o código do sistema durante todo o desenvolvimento, mantendo a clareza do software: sem ambigüidade, com alta comunicação, simples, porém completo.

- Programação em pares (*Pair programming*): Todo código produzido é feito em duplas, ou seja, dois programadores trabalhando juntos na mesma máquina.
- Propriedade coletiva (*Collective ownership*): Todo o código pertence a toda equipe. Então, qualquer um pode alterar qualquer código em qualquer momento.
- Integração contínua (*Continuous integration*): Integra e constrói o sistema de software várias vezes por dia. A todo o momento, uma tarefa é completada.
- Semana de 40 horas (*40-hour week*): Como regra, não se deve trabalhar mais que 40 (quarenta) horas na semana. Programadores exaustos cometem mais erros.
- Cliente junto aos desenvolvedores (*On-site customer*): A equipe de XP tem o "cliente real", disponível todo o tempo, que responde às dúvidas que surgem.
- Padronização do Código (*Coding standards*): Os programadores escrevem o código da mesma forma, de acordo com regras que enfatizam a comunicação através do código.

O XP trata de assuntos de mudanças de requisito e seus custos simplificando tarefas de gerenciamento e documentação.

Resumidamente, o processo de desenvolvimento de software no XP pode ser explicado da seguinte forma: tudo começa com o Planejamento do Jogo, o qual pode ser dividido em Planejamento de Versões e Planejamento de Iterações. Durante o Planejamento do Jogo, o cliente escreve as Estórias do Usuário, as quais são estimadas pelos desenvolvedores e, posteriormente, priorizadas pelos clientes. Planejamento do Jogo é a etapa de desenvolvimento do XP quando os requisitos, que são as Estórias, são elicitados, estimados e selecionados para a versão. Ele é executado para cada versão. Uma versão é dividida em iterações. Para cada uma dessas é feito o Planejamento de Iteração que consiste em selecionar um subconjunto de Estórias baseado na prioridade e no tamanho. A partir disso, os desenvolvedores dividem as Estórias em Tarefas e dão uma estimativa para cada Tarefa. A próxima etapa do processo XP é o Desenvolvimento, quando as iterações são produzidas e lançadas. Então os testes de aceitação são usados para validar a conclusão das Estórias (KÄÄRIÄINEN et al, 2003).

De acordo com Beck (2000), o ciclo de vida de um projeto ideal em XP consiste de seis fases: Exploração, Planejamento, Iterações até a Primeira Versão, Produção

(*Productionizing*), Manutenção e Morte. Abrahamsson et al. (2002) resumem as fases definidas por Beck (2000) da seguinte maneira:

Na fase de Exploração, o cliente escreve as Estórias do Usuário que eles desejam que sejam incluídas na primeira versão. Cada Estória é escrita no Cartão de Estória e Tarefa do Usuário que descreve uma funcionalidade que será adicionada ao sistema. Ao mesmo tempo a equipe de projeto familiariza-se com as ferramentas, tecnologias e práticas que eles irão usar no projeto. A tecnologia usada será testada e as possibilidades de arquitetura para o sistema são exploradas construindo-se um protótipo do mesmo. A fase de Exploração dura de poucas semanas a poucos meses, dependendo basicamente de como os programadores estão familiarizados com a tecnologia.

Na fase de Planejamento, o cliente coloca as Estórias em ordem prioritária e faz um acordo de conteúdo da primeira versão. Primeiro, os programadores estimam quanto de esforço cada Estória necessitará e combinam um cronograma. O cronograma da primeira versão normalmente não excede dois meses. A fase de Planejamento tem a duração de alguns dias.

A fase de Iterações até a Primeira Versão inclui várias iterações do sistema antes da primeira versão. O cronograma elaborado na fase de Planejamento é quebrado em algumas iterações sendo que cada uma terá a duração de uma a quatro semanas para ser implementada. O cliente decide as Estórias que serão selecionadas em cada iteração. Essas Estórias selecionadas são divididas em tarefas e escritas nos Cartões de Tarefa do Programador, pelos programadores. Os testes funcionais criados pelo cliente são executados no final de cada iteração. No final da última iteração o sistema estará pronto para a produção.

A fase de Produção (*Productionizing*) requer testes extras e checagem de desempenho do sistema antes dele ser entregue para o cliente. Nessa fase, novas mudanças podem ainda ocorrer e é tomada a decisão se elas serão incluídas na versão corrente. Durante essa fase, as iterações devem ser mais rápidas durando de uma a três semanas. As idéias e sugestões adiadas são documentadas para que sejam implementadas mais tarde, isto é, na fase de manutenção.

Depois que a primeira versão é produzida para uso do cliente, o XP deve tanto manter o sistema em processo de produção como também deve produzir novas iterações. Para tanto, na fase de Manutenção também é requerido um esforço do cliente na ajuda das tarefas. Desse modo, a velocidade do desenvolvimento pode ser desacelerada depois que o

sistema está em produção. A fase de Manutenção pode precisar incorporar novas pessoas na equipe gerando, com isso, uma mudança na sua estrutura.

A fase de Morte está próxima quando o cliente não tem mais nenhuma Estória para ser implementada. Para isso, é preciso que o sistema satisfaça as necessidades do cliente também em outros aspectos como o desempenho e a confiabilidade. Essa é a hora, no XP, de se elaborar a documentação necessária para o sistema, a qual é escrita finalmente sem nenhuma mudança a mais na arquitetura, projeto ou código. Nessa fase também pode acontecer do sistema não estar entregando os resultados desejados para o cliente ou então, do desenvolvimento adicional ficar muito caro.

Em XP, existem alguns papéis para a execução de tarefas e propósitos diferentes durante o processo e suas práticas. A seguir, esses papéis são apresentados de acordo com Beck (2000):

- Programador: É o coração do XP, responsável por estimar prazos das Estórias do Usuário; definir os Cartões de Tarefa a partir dos Cartões de Estórias; fazer estimativas do sistema com base nos Cartões de Tarefa; definir e executar os testes de unidade que devem ser feitos antes do código; trabalhar em par e implementar o código.
- Cliente: Escreve as Estórias do Usuário e os Testes de Aceitação. Ele também decide quando cada requisito é satisfeito. O cliente determina a prioridade da implementação dos requisitos.
- Testador: Ajuda o cliente a escrever os Testes de Aceitação. Eles executam os Testes de Aceitação regularmente e publicam os resultados para a equipe.
- Acompanhador (*Tracker*): Coleta os sinais vitais do projeto (métricas) uma ou duas vezes por semana e garante que todos estejam bem cientes dos mesmos.
- Treinador ou Técnico (*Coach*): Garante que o projeto permaneça extremo e ajuda no que for necessário.
- Consultor (*Consultant*): É um membro externo que possui conhecimento técnico específico necessário. O consultor guia a equipe para resolver problemas específicos.
- Gerente (*Big Boss*): Toma as decisões. Além disso, ele se comunica com a equipe do projeto para determinar a situação corrente e distingue qualquer dificuldade ou deficiência no processo.

Salienta-se que de acordo com a literatura estudada, torna-se evidente que o XP assume o seu desenvolvimento baseado no paradigma OO.

Além desses métodos, que são considerados os principais no contexto de métodos ágeis, no Capítulo 5 serão abordados, bem resumidamente, alguns trabalhos da literatura que se propuseram a combinar esses métodos com outros métodos tradicionais. Esse levantamento foi realizado e é comentado apenas naquele capítulo, para efeito de uma comparação lá apresentada, a qual subsidiou uma parte dos resultados apresentados neste trabalho.

4.3 Trabalhos Relacionados

Nesta seção apresentam-se dois trabalhos: Leite (2001) e Nawrocki et al. (2002). Eles apontam alguns problemas e algumas soluções para o XP relacionados ao levantamento de requisitos que serão usados na proposta deste trabalho.

Apesar do XP ser muito popular e propor algumas práticas muito interessantes (forte orientação do cliente, programação em pares, etc.), ele também possui algumas fraquezas (NAWROCKI et al., 2002). Uma delas, citada na literatura, é que XP não expõe uma estratégia explícita para gerenciamento de requisitos e apresenta vários problemas na sua maneira informal de lidar com eles (LEITE, 2001; LEONARDI; LEITE, 2002):

1. A suposição de que, no planejamento do jogo, o negócio pode ser representado por apenas uma pessoa;
2. A falta de consideração dos requisitos não funcionais do ponto de vista do negócio;
3. A falta de ligação explícita entre Cartões de Estórias e de Tarefas para o código;
4. A falta de um processo para a produção de testes de aceitação;
5. A falta de um processo para a produção de Estórias e Tarefas.

Tentando aliviar os problemas citados anteriormente, Leite (2001) propôs o XR (*Extreme Requirements*) com o objetivo de melhorar a qualidade do XP, definindo uma estratégia de requisitos que pode ser acoplada ao XP, já que este não define uma etapa de requisitos (LEITE, 2001; LEONARDI; LEITE, 2002). XR introduz uma maior sistemática ao levantamento de requisitos em XP utilizando a LEL (*Language Extended Lexicon*), os Modelos de Cenários e as Regras de Negócio.

A LEL é uma representação de símbolos que possui o propósito de capturar o vocabulário do Universo do Discurso (U*of*D). Seu principal objetivo é que os engenheiros de software compreendam a linguagem utilizada pelos clientes, entendendo seus termos sem se preocupar com o problema. Os símbolos da LEL definem objetos (entidades passivas), sujeitos (entidades ativas), frases verbais e estados (LEONARDI; LEITE, 2001).

Um Cenário é uma descrição técnica que otimiza a compreensão das descrições das tarefas relacionadas e a comunicação entre os *stakeholders*, que são as pessoas que terão alguma influência direta ou indireta sobre os requisitos do sistema (SOMMERVILLE, 2003). Seu principal objetivo é compreender o comportamento do problema. Eles são expressos em linguagem natural e estão conectados com a LEL.

As Regras de Negócio são definidas como sentenças que retratam a forma com que a empresa realiza os negócios. Refletem as políticas do negócio, e têm por finalidade satisfazer os objetivos do negócio, satisfazer os clientes, fazer um bom uso dos recursos e respeitar as normas convencionais da empresa (LEONARDI; LEITE, 2001).

As Regras podem ser obtidas a partir de entrevistas estruturadas com os clientes. Basicamente o cliente pensa em todos os aspectos de sua organização e denota limites, direitos e responsabilidades das partes envolvidas no sistema de software que se deseja construir. Nesta etapa aparecem os termos do vocabulário do problema que se está analisando. Identificam-se os termos apropriados da aplicação (um recurso, um processo, uma pessoa de um setor). Dessa maneira se cria a LEL para definir o vocabulário, eliminar qualquer tipo de ambigüidade e o uso incorreto do mesmo. Com ela obtém-se um mecanismo de rastreabilidade, pois permite que todos os modelos obtidos nesta etapa e nas etapas posteriores estejam relacionados pelo uso dessa linguagem restringida, resolvendo o problema três apresentado no início da seção.

Com os Cenários, ao invés de Estórias do Usuário, os aspectos não funcionais são devidamente representados solucionando assim o problema dois.

Com relação ao problema um, Leite (2001) relata que raramente encontra-se uma pessoa que possa ser um representante real de vários interesses diferentes ou perspectivas de negócio. Por essa razão, uma vez que o Cenário ou um conjunto de Cenários é escrito pelo cliente, ele é inspecionado por outros representantes do negócio, não envolvidos diretamente com a equipe de XP, a fim de assegurar que o Cenário não seja uma visão particular do negócio. O processo de inspeção é, na maioria das vezes, conduzido em uma simples reunião ou um simples ciclo de leitura com os outros representantes do negócio.

Para amenizar o problema quatro, propõe-se construir testes de aceitação a partir do Cenário. Assim um cliente facilmente constrói os testes, colocando variantes em cada parte dos Cenários para se criar falhas nos objetivos estabelecidos pelos mesmos.

Relacionado ao problema cinco, é oferecida uma estratégia para a produção de Cenários na qual atividades de Validação e Verificação são levadas em consideração. O processo de construção de Cenários começa a partir da aplicação léxica, produzindo uma primeira versão de cenários derivados exclusivamente da LEL. Esses Cenários são melhorados usando-se outros recursos de informação e organizados a fim de se obter um conjunto consistente de Cenários que represente a aplicação. Durante ou depois dessas atividades, os Cenários são verificados e validados com os clientes e usuários para a detecção de Discrepâncias, Erros e Omissões (DEO). O processo é composto de cinco atividades e é apresentado na Figura 4.1:

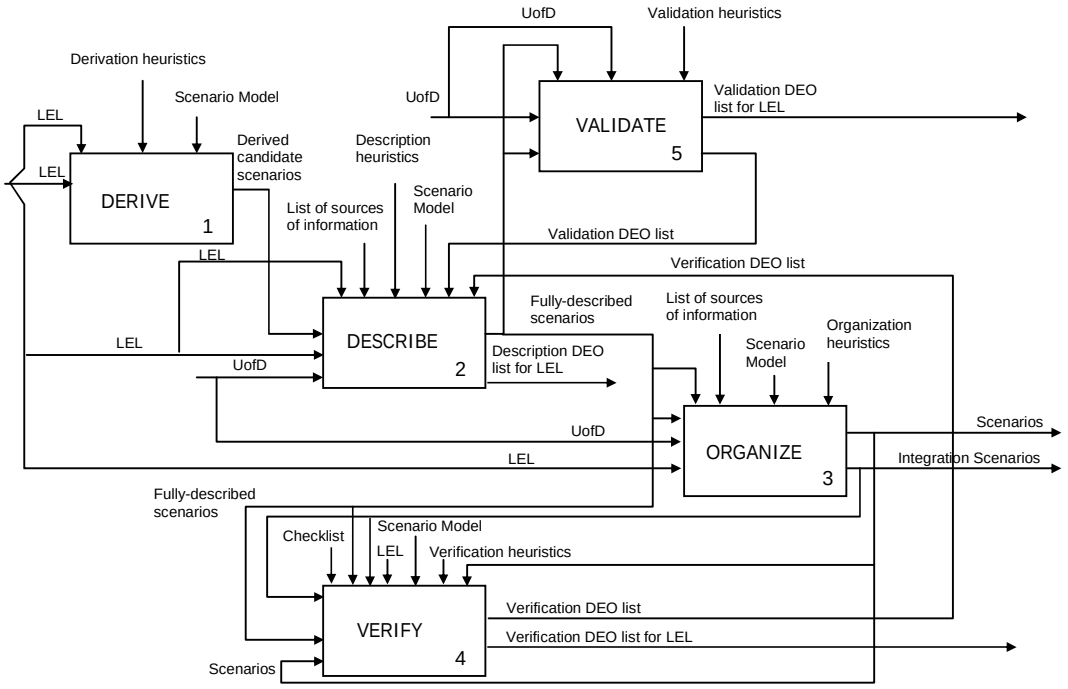


Figura 4.1 - Processo de Construção de Cenários (LEONARDI; LEITE, 2002).

A atividade *Derive* está preocupada com a elicitação dos requisitos e é desempenhada pelo cliente.

A atividade *Describe* detalha os Cenários produzidos na etapa anterior.

A atividade Organize é usada para manter um relacionamento entre os diferentes Cenários que estão sendo produzidos.

A atividade *Verify* é efetuada pelo menos duas vezes durante o processo de construção de Cenários: a primeira vez sobre o conjunto completo dos Cenários descritos e a segunda depois da atividade *Organize*. Como conseqüência dessa atividade, duas listas DEO são produzidas e usando essas listas, os Cenários e o LEL são modificados. Se maiores correções forem necessárias, uma nova verificação deve ser requerida. Esse processo de verificação é conduzido por checagem de sintaxe, por cruzamento de referências entre o LEL e os Cenários e por um *Checklist.*

Finalmente, os Cenários são validados com os *stakeholders* utilizando-se normalmente entrevistas estruturadas e reuniões.

Embora XR acrescente mais documentos e um dos objetivos do XP é tentar reduzir a documentação, ele é útil pois resolve muitos problemas em XP, principalmente a questão da Engenharia de Requisitos.

Um outro trabalho que propõe algumas alterações no XP é o de Nawrocki et al. (2002). Eles sugerem as seguintes alterações:

- A documentação de requisitos é gerenciada por um analista/testador.

De acordo com Beck (2000), um testador em XP é responsável por ajudar o cliente a escolher e escrever testes de aceitação e então executá-los. Assim, Nawrocki et al. (2002) considera que os requisitos e os testes de aceitação estão no mesmo nível de abstração; conseqüentemente, o testador parece ser a melhor pessoa para fazer o trabalho de analista responsável pela documentação e gerenciamento de requisitos. Uma boa prática é a ligação dos casos de teste com os requisitos. Se um requisito muda, um caso de teste deverá ser revisado. Deste modo, um testador bem organizado irá gerenciar os requisitos de qualquer maneira. Se os requisitos não forem documentados, o trabalho do testador fica difícil.

- O Planejamento do Jogo é modificado e permite que se tenha múltiplos clientes representativos.

XP assume que há somente um cliente representativo para o projeto. No caso de muitos clientes representativos é assumido que eles falam unanimemente a mesma coisa. Infelizmente, em alguns casos existem muitos clientes representativos e eles não falam a mesma coisa.

Para se solucionar essa questão, Nawrocki et al. (2002) propõem o seguinte: no lado do cliente existirão os clientes representativos e um cliente sênior. Cada cliente

representativo possui o conhecimento de algum domínio e ele será um usuário final do sistema. É possível que seus requisitos sejam contraditórios com os requisitos de outros clientes representativos.

O cliente sênior representa os interesses da organização inteira. É possível que ele não tenha o conhecimento do domínio dos clientes representativos na hora de discutir os detalhes do sistema a ser construído. Por esse motivo, tanto o cliente sênior como os clientes representativos são necessários. A principal função do cliente sênior é resolver os conflitos entre os clientes representativos sempre que necessário.

Os clientes representativos participam do Planejamento do Jogo modificado, elaborando os Cartões de Estória. Depois disso, os desenvolvedores estimam o esforço em horas e avaliam os riscos associados com cada Estória. Cada cliente representativo recebe uma certa quantia de orçamento representando o tempo que a equipe XP pode gastar trabalhando com ele. O orçamento é expresso em horas e o cliente sênior atribui esse orçamento para cada cliente representativo. Cada cliente paga as Estórias mais valiosas para os desenvolvedores. Se *n* clientes representativos compram uma estória, cada um paga 1/*n* pelo preço.

Se um cliente representativo não for capaz de comprar uma Estória valiosa (todas as suas Estórias são caras), ele pode retornar seu orçamento para o cliente sênior. Então o cliente sênior oferece o "dinheiro" para o restante dos clientes representativos. É uma espécie de leilão. A pessoa que oferecer um preço maior leva o "dinheiro" e será capaz de comprar uma outra Estória. O preço será pago no próximo Planejamento do Jogo para a pessoa que renunciou o "dinheiro".

- A fase de Engenharia de Requisitos no começo do projeto fornecendo, deste modo, uma ampla visão do sistema que está sendo desenvolvido.

Essa fase não precisa ser muito longa; algumas semanas já são suficientes. Durante essa fase a equipe do XP pode fazer as seguintes tarefas:

  - Juntar os Cenários de Uso. Cenários relativos ao sistema corrente são coletados com os clientes representativos e eles descrevem os problemas e as motivações para a construção de um novo sistema. Isto é uma forma simples de estudo de viabilidade. Isto também é um bom início para a coleta do conhecimento de domínio e o entendimento dos problemas. Cenários relacionados ao novo sistema podem ser escritos pelos clientes

representativos e/ou pela equipe do XP. Eles mostram a visão de como o sistema deve trabalhar.

- Avaliar a viabilidade do sistema: Uma das perguntas que deveria ser respondida é se o desenvolvimento do sistema de acordo com a método de XP seria a decisão certa.
- Identificar os *stakeholders* do sitema
- Descrever de modo simples o ambiente operacional do sistema
- Procurar as principais restrições do domínio
- Elaborar protótipo simples dos requisitos entendidos: Nesse estágio, o protótipo pode ser simples e deve, principalmente, focar a comunicação homem-máquina.

Nawrocki et al. (2002) dizem que em suas experiências essas modificações sugeridas são simples o suficiente e o método resultante é quase "leve" como o XP propriamente dito.

4.4 Considerações Finais

Neste capítulo foram abordados os conceitos relacionados com os "Métodos Ágeis" os quais são uma reação às maneiras tradicionais de desenvolvimento de software. Foram apresentados alguns métodos mais conhecidos e as suas principais características, pois isso dará suporte à revisão do processo ProDeS/UML na tarefa de torná-lo mais prático, em termos de documentação gerada, e mais factível de utilização na prática.

O método XP foi abordado de uma forma mais específica pois, dado o seu crescente uso na prática, este trabalho propõe uma estratégia de inspeção ágil que pode melhorar a qualidade do desenvolvimento de software baseado no mesmo.

Assim, embora esse "Movimento Ágil" esteja ganhando uma grande popularidade entre os desenvolvedores de software e seus seguidores apontem uma série vantagens associadas a esse novo paradigma de desenvolvimento, como pôde ser observado, algumas pesquisas também apontam alguns pontos fracos associados com ele. Esse fato não deixa de ser uma motivação para a definição da estratégia de inspeção ágil proposta neste trabalho para o XP, a qual é apresentada com maiores detalhes no Capítulo 6.

CAPÍTULO 5

INSPEÇÃO DE SOFTWARE NO CONTEXTO DE ProDeS/UML|pr

5.1 Considerações Iniciais

Como foi visto no Capítulo 2 o ProDeS/UML (COLANZI, 1999) é um processo que apóia o paradigma OO, usa a notação UML e foi criado com base no método Fusion (COLEMAN et al., 1994), com etapas de desenvolvimento gradativo. De acordo com Andriole (1986) o desenvolvimento gradativo é um ponto muito importante, pois isso permite que os artefatos de software intermediários sejam avaliados a fim de verificar sua qualidade e corretitude, bem como verificar sua evolução. Com isso, consegue-se estabelecer marcos durante o processo para que essas checagens sejam realizadas. Além disso, o ProDeS/UML possui uma etapa de teste associada a ele, na qual atividades de teste são realizadas com base nos artefatos gerados ao longo do processo. Assim, no trabalho de Marucci (2002) foi agregado a tudo isso uma estratégia de inspeção, com a definição de técnicas de leitura que avaliam os diversos artefatos criados ao longo do processo. Esse processo, com todos esses recursos, tornou-se um processo com características de um processo mais tradicional, mais "pesado" e com ênfase na documentação, o que faz dele um processo relativamente robusto, bem definido e voltado para atividades de Garantia de Qualidade de Software.

No entanto, por outro lado, observa-se, atualmente, um "Movimento Ágil" que tem por objetivo impor uma maior agilidade ao processo de desenvolvimento de software, dando uma maior ênfase ao aspecto de implementação do que de documentação.

Assim, considerando-se a importância das atividades de inspeção durante o processo de desenvolvimento de software e também se considerando a ênfase que tem sido dada ao movimento ágil, o objetivo deste capítulo é apresentar dois dos resultados a que este trabalho se propôs: i) o processo ProDeS/UML|pr, como uma alternativa mais prática

do processo ProDeS/UML, discutindo as técnicas de leitura propostas no trabalho de Marucci (2002), que permanecem aplicáveis nesse processo e ii) os documentos que compõem os Pacotes de Laboratório para as técnicas de leitura que constituem a estratégia de inspeção do processo ProDeS/UML|pr.

Como foi dito anteriormente, todos os métodos, técnicas, etc. propostos na literatura, para serem realmente úteis para a comunidade que os utiliza, deveriam ser avaliados formalmente de maneira que suas vantagens e desvantagens pudessem estar bem caracterizadas, de forma a apoiar a sua escolha.

Isso pode ser feito por meio de estudos experimentais que tenham por objetivo explorar tais métodos e técnicas. Em geral, o experimento é idealizado por algum pesquisador – normalmente quem propõe o objeto de estudo – e seus resultados terão maior significância quanto maior for a quantidade de dados analisados e a diversidade do ambiente em que o experimento foi realizado. No entanto, para que isso possa ser realizado, faz-se necessário um material de apoio, os Pacotes de Laboratório, que permita com que o experimento possa ser replicado por outra pessoa, em outro ambiente ou cultura diferente, inclusive distante geograficamente do pesquisador original e, muitas vezes, com certa dificuldade de comunicação. Assim, os Pacotes de Laboratório precisam conter uma série de artefatos que descrevam, pormenorizadamente, todo esse contexto do experimento.

O capítulo está organizado da seguinte maneira: na Seção 5.2 é apresentado um levantamento de artefatos dos métodos ágeis propostos na literatura bem como de trabalhos que propõem uma combinação desses métodos com métodos mais tradicionais, na tentativa de torná-los mais ágeis; na Seção 5.3 é explicado o que foi alterado no novo processo ProDeS/UML|pr e quais são as técnicas de leitura OORTs/ProDeS que serão usadas para esse processo; a Seção 5.4 apresenta cada um dos elementos do Pacote de Laboratório, descrevendo-os de forma resumida, fornecendo-se um pequeno exemplo de cada elemento. Também é comentada uma breve avaliação que foi feita de um dos Pacotes de Laboratório, a qual foi realizada por um grupo de estudantes. A Seção 5.5 apresenta as considerações finais.

5.2 Métodos Ágeis e os Artefatos de Software

Comum a todos os processos de desenvolvimento de software, em qualquer projeto tem-se a necessidade de capturar e compartilhar conhecimento sobre os requisitos e os documentos gerados, sobre o próprio processo de desenvolvimento, o domínio de negócio do cliente e a situação do projeto. Na maioria das abordagens de desenvolvimento tradicionais esses conhecimentos são armazenados em uma grande quantidade de documentos para assegurar que todas as informações sejam tratadas e capturadas. Apesar da maior parte do conhecimento ser documentado, existe o problema de garantir que essa documentação esteja atualizada. Por esta razão, os Métodos Ágeis defendem que a documentação deve ser pouca e significativa o suficiente (CHAU et al., 2003).

No XP, por exemplo, os requisitos são guardados em cartões indexados. O AM sugere que o domínio de conhecimento e o projeto do sistema devam ser armazenados em forma de modelos somente se os mesmos facilitarem uma melhor comunicação ou o entendimento do sistema, do contrário, esses modelos não precisam ser detalhados. Já no FDD é proposto o uso de modelos detalhados, mas é recomendado o uso de ferramentas CASE sofisticadas para se reduzir a quantidade de esforço manual requisitada para a geração e alteração desses modelos (CHAU et al., 2003).

Assim, com o objetivo de tornar o processo de desenvolvimento de software OO ProDeS/UML mais prático, em termos de tipos de documentos elaborados, fez-se um levantamento sobre os artefatos de software mais utilizados pelos principais métodos ágeis, apresentados no Capítulo 4, bem como por algumas iniciativas existentes na literatura que utilizam os métodos ágeis ou suas principais práticas em outras abordagens mais tradicionais de desenvolvimento.

No levantamento para identificação dos artefatos produzidos no contexto de Métodos Ágeis consideram-se alguns dos Métodos Ágeis mais conhecidos, apresentados no Capítulo 4, e outros processos que se caracterizavam como sendo ágeis, mas que eram abordagens mais tradicionais usadas com princípios dos Métodos Ágeis.

Dos Métodos Ágeis mais conhecidos, apresentados no Capítulo 4, foram considerados nesse levantamento os métodos FDD, AM, *Crystal* e XP, pois para esses conseguiu-se identificar quais eram os documentos produzidos. Já os métodos ASD, DSDM e Scrum não foram considerados, pois não se conseguiu identificar quais documentos eles produzem.

Os artefatos utilizados pelos métodos FDD, AM e *Crystal* apresentados na Tabela 5.1, foram identificados em trabalhos que combinaram o uso destes com outra abordagem. No caso do FDD, apresentado no item 1 da Tabela 5.1, obteve-se a informação no livro *Java modeling color with UML* (COAD et al., 1999), no qual o método é usado com Java e seus cinco processos, juntamente com os artefatos que são produzidos, são apresentados detalhadamente.

Os artefatos do AM foram identificados no livro *Agile Modeling* (AMBLER, 2002), que apresenta AM com XP e AM com UP apresentados na Tabela 5.1, nos itens 2 e 3, respectivamente. Na utilização do AM com o XP, segundo o autor, embora XP inclua claramente modelagem como parte de seu processo, não é explicado como fazer essa modelagem. Com as práticas de AM pode-se ter um melhor entendimento. Além disso, os princípios de AM estão intimamente ligados com as práticas de XP.

Já na utilização do AM com o UP (*Unified Process*), as técnicas de AM descrevem como modelar e documentar de modo eficaz e ágil. Assim, o processo de modelagem do UP juntamente com AM pode ser feito de uma maneira bem mais ágil. A modelagem no UP parece possuir vários problemas. Um deles é que ela é muito complexa e alguns dos artefatos utilizados precisam ser repensados. Com o AM existe uma grande oportunidade de simplificar a maioria dos artefatos do UP.

Os artefatos do *Crystal*, apresentados no item 4 da Tabela 5.1, foram identificados no livro *Agile Software Development* (COCKBURN, 2002). O livro aborda a família *Crystal*, principalmente o *Crystal Clear* e o *Crystal Orange*. Todos os métodos *Crystal* começam com um conjunto de funções, produtos de trabalho, técnicas e notações, sendo que esse conjunto vai expandindo à medida que a equipe aumenta ou o método torna-se muito compacto.

O método XP, apresentado no item 5 da Tabela 5.1, teve seus artefatos identificados no livro *Extreme Programming explained* (BECK, 2000). O XP foi detalhado no Capítulo 4 e o *layout* de seus artefatos podem ser encontrados no Capítulo 6.

A seguir, apresentam-se resumos de outros trabalhos identificados na literatura que correspondem, principalmente, da utilização de um método ágil usado em conjunto com uma abordagem mais tradicional. Essa apresentação está estruturada da seguinte forma: o título do artigo ou livro, o nome do autor e uma pequena descrição. Salienta-se que os documentos gerados por cada um deles também estão relacionados na Tabela 5.1 a partir do item 6, apresentada posteriormente a esses resumos.

- *A Rapid Development Process with UML* (ARMANO; MARCHESI, 2000)

Descreve uma aplicação de software que usa um ciclo de vida que segue basicamente as diretrizes sugeridas por XP (*Extreme Programming*). Foi usada como linguagem de programação o Smalltalk e ferramentas CASE poderosas como a UMLTALK. A UML foi selecionada para modelar todo o processo, pois ela incorpora diagramas bem conhecidos para a descrição de uma aplicação de software em diferentes perspectivas. Essa abordagem mostrou ser muito eficaz em projetos de tamanho pequeno. Os autores dizem que os diagramas de colaboração, seqüência e atividades da UML são usados muito raramente. A única motivação para usar esses diagramas é documentar um cenário complexo de interação entre os objetos, a fim de fazer um modelo mais compreensível.

- *Making RUP Agile* (HIRSCH, 2002)

Apresenta detalhes de como fazer o RUP ágil, como ele foi aplicado em dois projetos pequenos com equipes de 3 a 5 desenvolvedores, e como foi configurado. Nesse trabalho, HIRSCH (2002) diz que um dos pontos chave para o sucesso da aplicação do RUP em pequenos projetos é a seleção cuidadosa de um subconjunto de artefatos, mantendo esses artefatos concisos e livres de formalismo desnecessário. Os principais critérios para inclusão e exclusão de um artefato eram as questões: “Qual o valor deste artefato para o cliente?” e “Quais são, provavelmente, as conseqüências se nós não tivermos este artefato?”. Se não fosse identificado um valor convincente do artefato, esse não era utilizado.

- *Applying Agile Methods in Rapidly Changing Environments* (KUTSCHERA; SCHÄFER, 2002)

Descreve um desenvolvimento de software ágil para projetos de desenvolvimento de software comercial baseado no Método *IBM Global Services*. Esse método é tradicional, mas tem como base a engenharia de software incremental e adapta isto para especificar as necessidades em ambientes de mudanças rápidas.

Com o intuito de tornar o método ágil, uma das decisões foi reduzir o número de artefatos produzidos em todo o projeto com o objetivo de se focalizar mais no trabalho de software que na documentação.

- *Applying Use Case Driven Object Modeling with UML* (ROSENBERG; SCOTT, 2001)

Descreve o processo ICONIX que é considerado um processo intermediário entre os Métodos Ágeis e os Tradicionais. O ICONIX está entre a complexidade e abrangência do RUP *(Rational Unified Processe*s) e a simplicidade e o pragmatismo do XP, mas sem eliminar as tarefas de análise e de projeto que o XP não contempla muito bem. Por este motivo, para muitos autores, o ICONIX é considerado um método "prático". O ICONIX faz uso de um subconjunto mínimo e eficiente de quatro diagramas da UML, que os autores acham necessários para se conduzir um bom desenvolvimento de software. Nas etapas básicas que estão incluídas em todo o processo ICONIX, estão associados alguns pontos ou marcos que ajudam na produção de um software de melhor qualidade.

- *A Proposal for a Lightweight Rigorous UML-Based Development Method for Reliable Systems* (PAIGE; OSTROFF, 2001)

Propõe um método de desenvolvimento de software para construção de sistemas de software confiável. Um sistema confiável dá ênfase nos requisitos de correção, exigindo que ele satisfaça rigorosamente suas especificações e nos requisitos de robustez, funcionando muito bem mesmo quando ele recebe entradas inesperadas ou quando há uma falha em um componente com o qual o sistema interage. Esse método tenta combinar a ênfase da codificação do XP com a prática da modelagem. Ele é construído com base em um subconjunto de diagramas UML.

- *The Process* (BOOCH et al., 1998)

Apresenta o dX como um processo mínimo do RUP. Esse processo foi usado em vários projetos com sucesso significativo. O dX não faz uso explícito da UML ou qualquer outra notação. O único artefato intermediário que o dX usa são os cartões de caso de uso. Isto não quer dizer que os engenheiros que estão usando o dX não possam usar a UML quando eles forem criar um modelo. dX simplesmente não diz quando tais modelos devem ser criados.

- *A simplified approach to RUP* (EVANS, 2003)

Apresenta uma breve descrição de um processo mínimo de software iterativo, ou seja, mostra as etapas mínimas para se usar o RUP com sucesso, descrevendo esse "mini" RUP juntamente com os artefatos UML produzidos.

A primeira iteração fornece uma oportunidade para se explorar os requisitos e os objetivos do sistema. Depois disso, as próximas iterações são usadas para verificar o que o sistema deverá fazer e como o sistema deverá alcançar os objetivos.

- *Using the RUP for small projects: Expanding upon Extreme Programming* (POLLICE, 2003)

Apresenta como aplicar o RUP de uma maneira "leve" em projetos pequenos. O autor enfatiza em como tornar um processo mais leve usando dois métodos populares: o RUP (*Rational Unified Process*) e o XP. O RUP é adaptável a projetos pequenos e trata de muitas áreas que não são consideradas pelo XP. Essa combinação dá à equipe do projeto as orientações necessárias para aliviar os riscos e alcançar os objetivos da entrega do software. Combinando a abrangência do RUP com algumas técnicas do XP, é alcançada a medida certa do processo que agrada todos os membros de um projeto.

Ressalta-se que alguns desses trabalhos mencionados anteriormente, não declaravam explicitamente os documentos elaborados, mas sim as informações que eram registradas e documentadas. Dessa forma, considerando que o intuito era justamente identificar os artefatos para que, com base nisso, se pudesse estudar uma redução do processo ProDeS/UML para torná-lo mais prático ("leve", "enxuto") e mais viável de ser aplicado na prática, o que se fez foi o seguinte: de acordo com o tipo de informação que os trabalhos diziam elaborar ou registrar, analisaram-se e identificaram-se em quais artefatos do ProDeS/UML aquelas informações seriam registradas.

Assim, a Tabela 5.1 apresenta um resumo dos artefatos utilizados nos trabalhos comentados anteriormente. Na coluna "ID" está o número da comparação, na coluna "Artigo/Livro" está o título do artigo ou do livro; na coluna "Artefatos do Artigo/Livro" estão os artefatos mencionados no trabalho corrente; na coluna "Artefatos ProDeS/UML" estão os artefatos do ProDeS/UML correspondentes aos artefatos do trabalho corrente; a coluna XP diz se o trabalho corrente tem alguma ligação com o XP e a última coluna mostra as técnicas de leitura do conjunto de técnicas OORTs/ProDeS que podem ser usadas com base nos artefatos do ProDeS/UML relacionados na outra coluna. Observa-se que a coluna XP foi criada com o intuito de verificar quantas propostas, dentre as consultadas, utilizavam o método XP ou algumas de suas práticas, combinando com outras abordagens. Isso subsidiou a proposta apresentada no Capítulo 6.

Tabela 5.1 – Relação dos artefatos gerados nos Métodos Ágeis, seus equivalentes no ProDeS/UML e as técnicas de leitura OORTs/ProDeS aplicáveis.

ID	Artigo/Livro	Artefatos do Artigo/Livro	Artefatos ProDeS/UML	XP	Leituras
1	Java modeling color with UML (FDD)	Casos de Uso Diagrama de Classes Diagrama de Seqüência	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classe do Domínio Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado Diagrama de Seqüência Diagrama de Visibilidade		ER1 ER2 A1 P1 P6 P8
2	Agile Modeling (XP e AM)	Estórias do Usuário Cartões de Tarefa Cartões CRC (Classe - Responsabilidade – Colaborador) Esqueletos de Diagrama de classes feitos a mão Diagrama de Seqüência	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classe do Domínio Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado Diagrama de Visibilidade Diagrama de Seqüência	X	ER1 ER2 A1 P1 P6 P8
3	Agile Modeling (UP-Unified Process e AM)	Diagrama de Casos de Uso (ou DFD), Especificação dos Casos de Uso, Diagrama de Robustez, Diagrama de Seqüência, Diagrama de Classe (ou CRC cards), Diagrama de Componentes, Diagrama de Distribuição	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classe do Domínio Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado Diagrama de Visibilidade Diagrama de Seqüência		ER1 ER2 A1 P1 P6 P8
4	Agile Software Development (Crystal Clear e Crystal Orange)	Documento de Requisitos; Seqüência de Versões; Cronograma; Casos de Uso Anotado ou Descrições de Características; Esquemas de Projeto e Anotações Necessárias; Modelo de Objeto Simples; Manual do Usuário; Código Fonte; Casos de Teste; Código de Migração.	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classe do Domínio Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado Diagrama de Visibilidade		ER1 ER2 A1 P8
5	Extreme Programming	Estórias do Usuário Cartões de Tarefa Cartões CRC	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classe do Domínio Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado Diagrama de Visibilidade	X	ER1 ER2 A1 P8
6	A Rapid Development Process with UML	Casos de Uso Cartões CRC Diagrama de Classe	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classes do Domínio Diagrama de Classe da Análise Diagrama de Classe da Análise Refinado Diagrama de Visibilidade	X	ER1 ER2 A1 P8
7	Making RUP Agile	Visão do Documento, Modelo de Caso de Uso, Documento da Arquitetura de Software, Modelo de Projeto, Modelo de Implementação, Lista de Defeitos, Anotações de Versões, Artefatos de Instalação, Plano de Gerenciamento de Configuração, Lista de Mudanças Requisitada, Plano de Desenvolvimento de Software, Plano de Iteração, Avaliação da Iteração, Desenvolvimento de Caso, Diretrizes da Programação	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classes do Domínio Diagrama de Classe da Análise Refinado Diagrama de Visibilidade		ER1 ER2
8	Applying Agile Methods in Rapidly Changing Environments (Método IBM Global Service)	Requisitos Não Funcionais, Modelo de Casos de Uso, Diagrama de Contexto do Sistema, Diagrama da Arquitetura, Modelo de Componente, Modelo Operacional, Padões, Plano de Versão, Diretrizes de Codificação, Idéias Incrementais; Plano de Orgamização, Estratégia de Teste.	Diagrama de Casos de Uso Especificação dos Casos de Uso Cenário Diagrama de Classes da Análise Refinado Diagrama de Seqüência Diagrama de Visibilidade Descrição de Classes Diagrama de Estados da Classe		ER1 P1 P2 P3 P4 P5 P6 P7

Tabela 5.1 – Relação dos artefatos gerados nos Métodos Ágeis, seus equivalentes no ProDeS/UML e as técnicas de leitura OORTs/ProDeS aplicáveis. (Continuação)

ID	Artigo/Livro	Artefatos do Artigo/Livro	Artefatos ProDeS/UML	XP	Leituras
9	Applying Use Case Driven Object Modeling with UML (ICONIX)	Modelo de Casos de Uso Diagrama de Robustez Diagrama de Seqüência Modelo de Domínio Diagrama de Classe	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classes do Domínio Diagrama de Seqüência Diagrama de Visibilidade Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado		ER1 ER2 A1 P1 P6 P8
10	A Proposal for a Lightweight Rigorous UML – Based Development Method for Reliable Systems	Diagramas de Classe Diagramas de Colaboração Diagramas de Casos de Uso	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classes da Análise Refinado Diagrama de Seqüência Diagrama de Visibilidade	X	ER1 P1 P6
11	The process (dX)	Principais Casos de Uso; Cronograma do Projeto; Arquitetura inicial do sistema; Análise e projeto são executados pelos desenvolvedores em sessões de projeto. Os desenvolvedores podem usar diagramas UML, Cartões CRC, ou qualquer outro que signifique a construção de modelos de análise e projeto; Os modelos de projeto são completamente leais ao código.	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classe do Domínio Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado Diagrama de Seqüência Diagrama de Visibilidade Modelo de Operações Descrição de Classes	X	ER1 ER2 A1 A2 A3 P1 P4 P5 P6, P8, P9
12	A simplified approach to RUP (“mini-RUP”)	Casos de Uso Cartões CRC Diagrama de Classe do Domínio Diagrama de Classe Diagrama de Seqüência/ Diagrama de Colaboração Diagrama de Estados	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classe do Domínio Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado Diagrama de Seqüência Diagrama de Visibilidade Diagrama de Estados da Classe		ER1 ER2 A1 P1 P3 P6 P7 P8
13	Using to RUP for small projects: Expanding upon Extreme Programming	Um Caso de Negócio aprovado, Lista de Riscos, Plano de Projeto Preliminar, Plano de Aceitação do Projeto; Um plano para a Elaboração da Iteração Inicial; Modelo de Caso de Uso Inicial; Documento de Arquitetura do Software; Planos de Iteração para Construção; Componentes; Materiais de Treinamento; Plano de Organização; Plano de Iteração da Fase de Transição; Anotações de Versão; Material e Documentação de treinamento.	Diagrama de Casos de Uso Especificação dos Casos de Uso Diagrama de Classe do Domínio Diagrama de Classes da Análise Diagrama de Classes da Análise Refinado Diagrama de Seqüência Diagrama de Visibilidade	X	ER1 ER2 A1 P1 P6 P8

Fazendo uma síntese dos artefatos utilizados em cada proposta encontrada na literatura, na Tabela 5.2 apresenta-se o relacionamento inverso, ou seja, para cada artefato do ProDeS/UML, associam-se os trabalhos que o utilizam.

Tabela 5.2 – Artefatos do ProDeS/UML relacionados com as informações armazenadas ou documentos gerados nas abordagens ágeis.

Artefato do ProDeS/UML	Identificador dos trabalhos da Tabela 5.1												
	1	2	3	4	5	6	7	8	9	10	11	12	13
Diagrama de Casos de Uso	X	X	X	X	X	X	X	X	X	X	X	X	X
Especificação dos Casos de Uso	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagrama de Classe do Domínio	X	X	X	X	X	X	X		X		X	X	X
Diagrama de Classes da Análise	X	X	X	X	X	X			X		X	X	X
Diagrama de Classes da Análise Refinado	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagrama de Visibilidade	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagrama de Seqüência	X	X	X					X	X	X	X	X	X
Cenário								X					
Modelo de Operações											X		
Descrição de Classes								X			X		
Diagrama de Estados da Classe								X				X	

Com base na Tabela 5.2 pode-se observar que os artefatos do ProDeS/UML comuns à maioria dos trabalhos são: o Diagrama de Casos de Uso, as Especificações dos Casos de Uso, o Diagrama de Classe do Domínio, o Diagrama de Classe da Análise, o Diagrama de Classe da Análise Refinado, o Diagrama de Visibilidade e o Diagrama de Seqüência.

Destaca-se que durante as comparações, o Diagrama de Visibilidade e o Diagrama de Classe da Análise Refinado foram utilizados em conjunto, uma vez que Marucci et al. (2002) dizem que eles representam todas as características de um Diagrama de Classes da UML. Essas comparações darão subsídios à definição do processo ProDeS/UML|pr que será explicado na seção seguinte.

5.3 ProDeS/UML|pr

Na tentativa de tornar o processo ProDeS/UML mais prático, em termos dos documentos gerados durante o desenvolvimento do software, na seção anterior foi realizada a comparação dos artefatos utilizados pelos principais métodos ágeis ou por outras propostas da literatura que procuram combiná-los com outros métodos. O que pode ser observado é que alguns artefatos são bastante usados nas várias propostas e também que, em algumas delas, havia a intenção de tornar um método com características mais tradicionais (“pesado”) apenas em um método um pouco mais ágil (“leve”). Outro fato

importante observado é que mesmo nos métodos ágeis, o desenvolvimento é gradativo e quase todos eles possuem como característica o desenvolvimento iterativo.

Levando em consideração esses pontos observados e tendo como objetivo de deixar o ProDeS/UML mais ágil, isto é, um processo "prático", mantendo suas características de tal forma que a estratégia de inspeção proposta por Marucci (2002) pudesse permanecer aplicável, parcialmente, na nova versão do processo, a abordagem adotada para reduzir o ProDeS/UML foi a seguinte: estabeleceu-se como base de comparação o processo ICONIX, uma vez que este é considerado um processo prático, isto é, não tão ágil como o XP, mas também não tão "pesado" como os processos mais tradicionais e, a partir dos artefatos utilizados por ele, identificaram-se os outros trabalhos cujo conjunto de artefatos produzidos era parecido com o do ICONIX. A partir daí, analisaram-se as intersecções existentes e chegou-se à conclusão de que o conjunto formado pelos seguintes artefatos contemplava a proposta de um método mais prático que um tradicional e menos ágil que o XP: Diagrama de Casos de Uso, Especificação dos Casos de Uso, Diagrama de Classes do Domínio, Diagrama de Classes da Análise, Diagrama de Seqüência, Diagrama de Visibilidade e o Diagrama de Classes da Análise Refinado.

A Figura 5.1 apresenta os artefatos do ProDeS/UML que devem ser desconsiderados (aqueles que estão marcados com um "x"), de acordo com o estudo realizado.

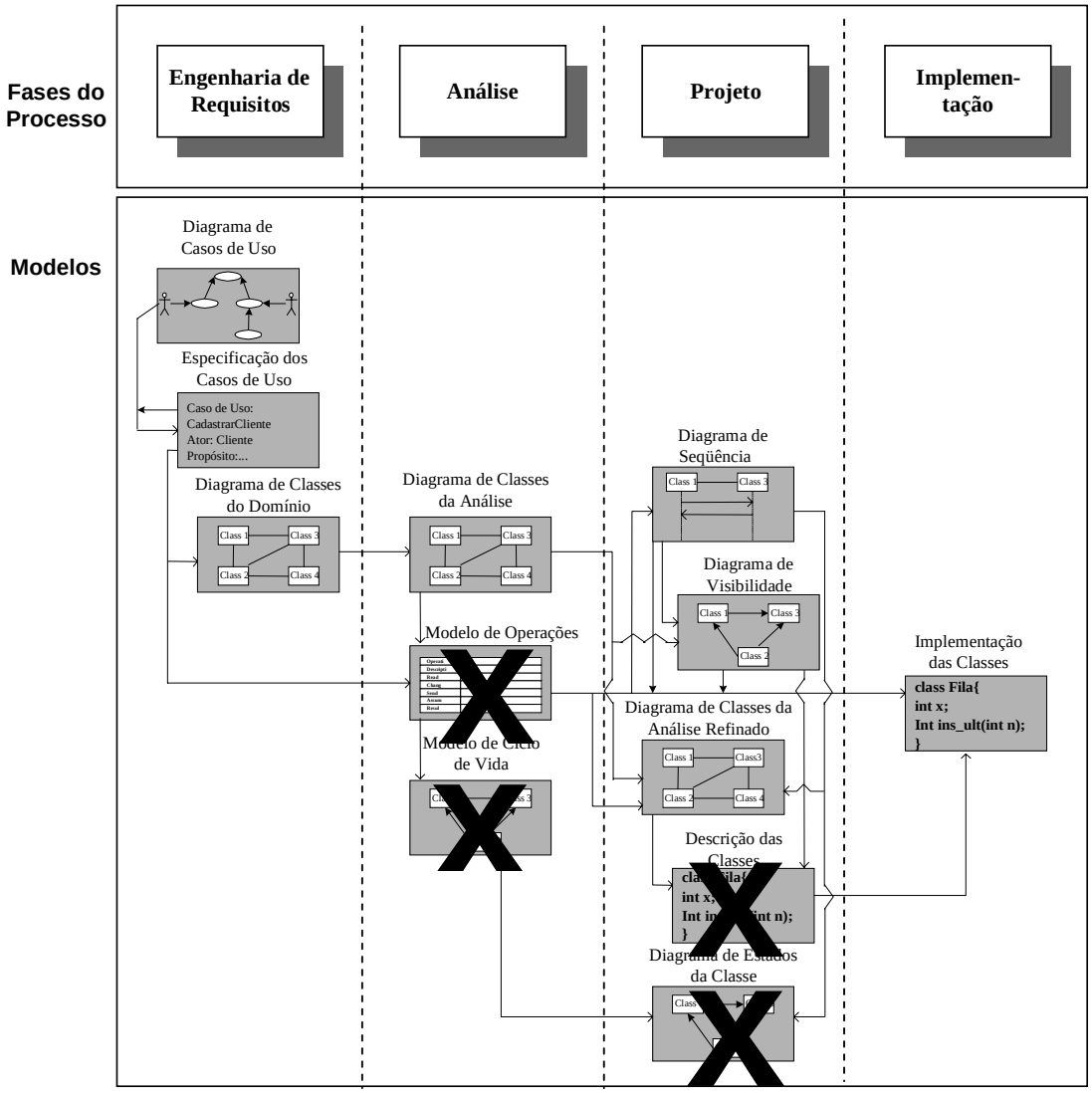


Figura 5.1 - Artefatos excluídos do ProDeS/UML (Adaptado MARUCCI, 2002)

Como pode ser observado, os artefatos que devem ser desconsiderados são: o Modelo de Operações e Modelo de Ciclo de Vida da fase de Análise; o Diagrama de Estados da Classe e a Descrição de Classes da fase de Projeto.

Excluindo-se efetivamente esses documentos do processo ProDeS/UML tem-se então o processo ProDeS/UML|pr,o qual está apresentado na Figura 5.2.

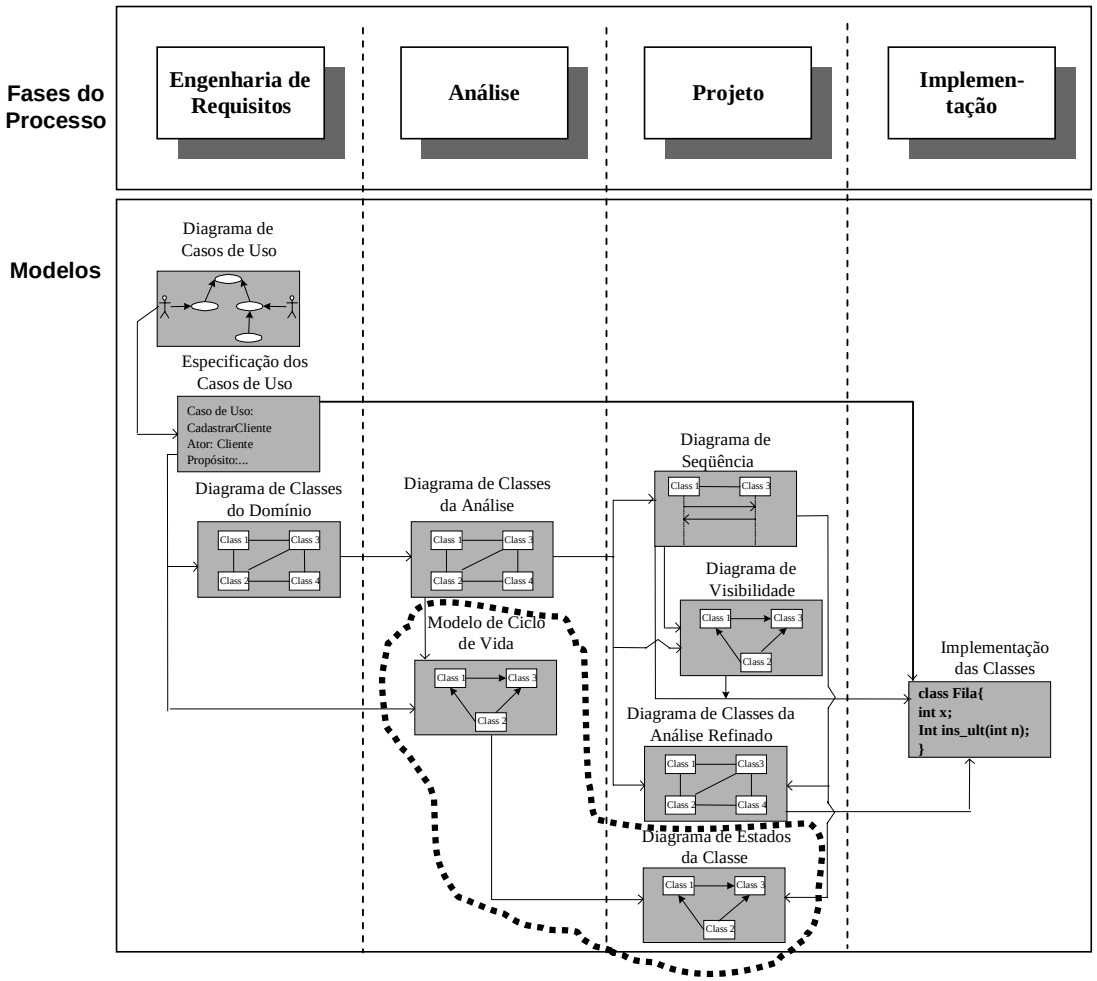


Figura 5.2 - Visão Geral do ProDeS/UML|pr

Observe que como a Descrição de Classes e o Modelo de Operações foram eliminados, as informações do Diagrama de Classes da Análise Refinado e da Especificação dos Casos de Uso foram atribuídas diretamente a implementação.

A exemplo do ICONIX, esse novo processo não elimina as fases de Engenharia de Requisitos, Análise, Projeto e Implementação. Ele possui as mesmas fases de desenvolvimento do ProDeS/UML seguindo os mesmos objetivos, só que com alguns artefatos a menos.

A fase de Engenharia de Requisitos não tem alteração, ou seja, tanto os artefatos gerados nessa fase bem como as técnicas de leitura do OORTs/ProDeS permanecem os mesmos. Já na fase de Análise só permanece a técnica de leitura A1. Na fase de Projeto três técnicas de leitura permanecem aplicáveis: P1, P6 e P8.

Desta forma, as técnicas de leitura do OORTs/ProDeS que podem ser utilizadas no ProDeS/UML|pr são: ER1, ER2, A1, P1, P6 e P8.

Segundo Rosenberg e Scott (2001), no ICONIX pode-se criar como documentos opcionais o Diagrama de Colaboração e o Diagrama de Estados. Por esse motivo, o ProDeS/UML|pr, tendo como base o ICONIX, também têm como documentos opcionais o Modelo de Ciclo de Vida e o Diagrama de Estados da Classe, destacados com uma linha tracejada na Figura 5.2. O Diagrama de Colaboração não foi incluído pelo fato de já existir o Diagrama de Seqüência que também é um Diagrama de Interação e possui as mesmas informações desse diagrama. Se esses documentos opcionais forem criados, mais duas técnicas de leitura serão incluídas no ProDeS/UML|pr: P3 e P7.

A Figura 5.3 mostra as técnicas de leitura em conjunto com os artefatos do ProDeS/UML|pr:

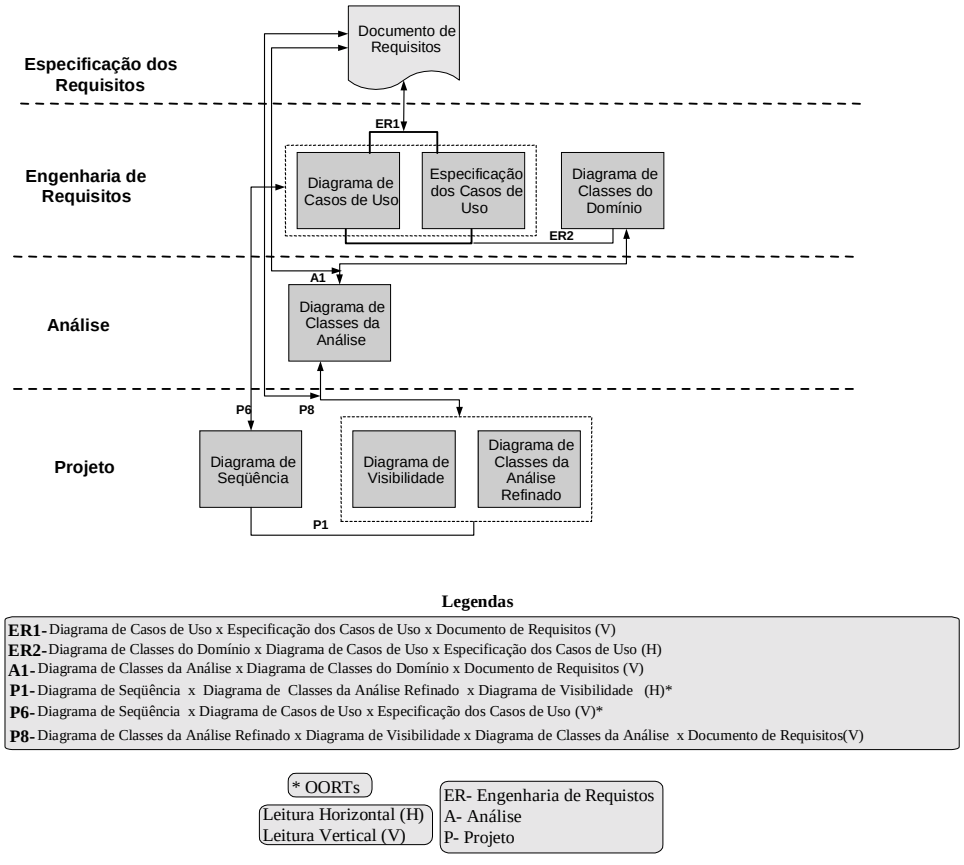


Figura 5.3 - Técnicas de Leitura do ProDeS/UML|pr

Com relação às técnicas de leitura que são utilizadas no ProDeS/UML|pr, a ER1, ER2, A1 e P8 foram elaboradas por Marucci (2002) e somente a A1 não sofreu nenhuma alteração. A P1 e a P6 são técnicas definidas por Travassos et al.(2000, 2002b, 2002c).

As técnicas ER1 e ER2, depois de revisadas, sofreram algumas alterações e a técnica P8 foi especificada no contexto deste trabalho.

A estratégia de aplicação das técnicas de leitura no ProDeS/UML|pr permanece a mesma já que isso é dependente da ordem de elaboração dos artefatos, o que não foi alterado. Assim, a seqüência de aplicação é a seguinte: ER1, ER2, A1, P6, P8 e P1.

Para as técnicas de leitura ER1, ER2, A1 e P8, que permaneceram no ProDeS/UML|pr e que foram propostas no trabalho de Marucci (2002), desenvolveram-se Pacotes de Laboratórios para que elas possam ser aplicadas e validadas. A elaboração desses pacotes será abordada com mais detalhes na próxima seção e os pacotes dessas técnicas, propriamente ditos, constam no Apêndice A.

5.4 Pacotes de Laboratório para as técnicas de leitura aplicáveis no ProDeS/UML|pr

Como foi dito anteriormente, todos os métodos, técnicas, etc. propostos na literatura, para serem realmente úteis para a comunidade que os utiliza, deveriam ser avaliados formalmente de maneira que suas vantagens e desvantagens pudessem estar bem caracterizadas, de forma a apoiar a sua escolha.

Isso pode ser feito por meio de estudos experimentais que tenham por objetivo explorar tais métodos e técnicas. Em geral, o experimento é idealizado por algum pesquisador – normalmente quem propõe o objeto de estudo – e seus resultados terão maior significância quanto maior for a quantidade de dados analisados e a diversidade do ambiente em que o experimento foi realizado. No entanto, para que isso possa ser realizado, faz-se necessário um material de apoio, os Pacotes de Laboratório, que permita com que o experimento possa ser replicado por outra pessoa, em outro ambiente ou cultura diferente, inclusive distante geograficamente do pesquisador original e, muitas vezes, com certa dificuldade de comunicação. Assim, os Pacotes de Laboratório precisam conter uma série de artefatos que descrevam, pormenorizadamente, todo esse contexto do experimento.

Como um dos objetivos deste trabalho foi justamente compor os Pacotes de Laboratório para as técnicas de leitura que constituíram a estratégia de inspeção do

processo ProDeS/UML|pr, ou seja, as técnicas ER1, ER2, A1 e P8, esta seção apresenta os elementos que compõem esses pacotes.

5.4.1 Material Elaborado

Para cada uma das técnicas de leitura que compõem a estratégia de inspeção do ProDeS/UML|pr foi elaborado um Pacote de Laboratório, ou seja, as técnicas ER1, ER2, A1 e P8[1]. Cada Pacote de Laboratório é composto dos seguintes artefatos, que serão comentados resumidamente:

- Questionário de Caracterização do Participante
- Técnica de Leitura
- Treinamento
- Artefatos para a inspeção
- Relatório de Discrepâncias

- Questionário de Caracterização do Participante

Esse documento tem como objetivo avaliar o conhecimento e a experiência do participante que irá fazer parte da inspeção. Sua importância está nos relacionamentos que podem ser feitos posteriormente, na análise dos resultados, associando, por exemplo, a experiência do inspetor com uma determinada categoria de discrepância. A Figura 5.4 apresenta uma parte desse questionário.

[1] Para a técnica P8 faltam alguns dos diagramas UML.

ID do Revisor: _____ Data: ___/___/___

Este formulário possui algumas questões sobre seu conhecimento e experiência

Conhecimento Geral

1. Qual é sua experiência anterior com desenvolvimento de software na prática? (Marque o item mais apropriado)

Eu nunca desenvolvi software.	Eu desenvolvi software por minha própria conta.	Eu desenvolvi software como parte de uma equipe, como parte de um curso.	Eu desenvolvi software como parte de uma equipe, na indústria.

Quantos anos / meses de experiência você tem nesta prática? _____

Experiência em Desenvolvimento de Software

Experiência com Requisitos

2. Qual é a sua experiência em escrever requisitos?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

3. Qual é a sua experiência na elaboração Casos de Uso?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

4. Qual é a sua experiência em revisão de requisitos?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

5. Qual é a sua experiência em revisão de Casos de Uso?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

6. Qual é a sua experiência em alterar requisitos como decorrência de manutenção?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

Figura 5.4 - Questionário de Caracterização do Participante

- Técnica de Leitura

As técnicas de leitura abordadas neste trabalho são aquelas que permaneceram no processo ProDeS/UML|pr, ou seja, ER1, ER2, A1 e P8. As três primeiras foram revisadas, aplicando-as em um exemplo, sendo que a ER1 e a ER2 sofreram algumas alterações.

Embora a P8 tenha sido proposta no trabalho de Marucci (2002), ela não estava finalizada. Terminou-se então a redação dessa técnica neste trabalho e o Pacote de

Laboratório dela é o único que não está totalmente completo, faltando alguns artefatos UML que devem ser analisados durante sua aplicação.

Como pode ser observado, essas técnicas possuem uma parte de avaliação sintática, em que se procuram conceitos básicos presentes nos artefatos inspecionados e outra parte de avaliação semântica, em que se conduz o inspetor a procurar discrepâncias, levando-se em consideração os conceitos marcados anteriormente.

Na Figura 5.5 apresenta-se uma parte da técnica P8.

Observe que a Etapa I corresponde à avaliação sintática e a partir da Etapa II tem-se a avaliação semântica.

Leitura P8 - Diagrama de Classes da Análise Refinado x Diagrama de Visibilidade x Diagrama de Classes da Análise x Documento de Requisitos (V)

Objetivo: Verificar se os conceitos (atores, classes ou atributos) e os relacionamentos representados no Diagrama de Classes da Análise Refinado estão compatíveis com as informações contidas no Diagrama de Classes da Análise e se as novas informações presentes nesse diagrama, bem como se as relações de dependência e navegabilidade do Diagrama de Visibilidade estão condizentes com o Documento de Requisitos.

Entradas para o processo:
1. Um Diagrama de Classes da Análise Refinado e um Diagrama de Visibilidade.
2. Um Diagrama de Classes da Análise.
3. Documento de Requisitos.

I. Leia o Documento de Requisitos para identificar os conceitos e funcionalidades do sistema.

ENTRADAS: Um conjunto de Requisitos Funcionais (RFs).
SAÍDAS: Substantivos candidatos a atores ou entidades que serão mantidas (ou geradas) no sistema ou atributos das classes (marcados em azul nos RFs);
Verbos ou descrições de ações candidatos a funcionalidades do sistema (marcados em verde nos RFs).

Para cada Requisito Funcional do Documento de Requisitos faça:
A. Leia o Requisito para entender a funcionalidade que ele descreve.
B. Encontre os substantivos que constam desse Requisito e que sejam candidatos a tornarem-se atores ou entidades que serão mantidas (ou geradas) no sistema, ou atributos dessas entidades. Sublinhe esses substantivos com uma caneta azul.
C. Encontre os verbos ou descrições de ações, os quais sejam candidatos a funcionalidades realizadas pelo sistema. Sublinhe esses verbos ou descrições de ações com uma caneta verde.

II. Inspecione o Diagrama de Classes da Análise Refinado para verificar se os conceitos e os relacionamentos entre eles são compatíveis com os conceitos e relacionamentos representados no Diagrama de Classes da Análise e se as novas informações presentes no mesmo, bem como se as relações de dependência e navegabilidade do Diagrama de Visibilidade estão condizentes com os Requisitos.

ENTRADAS: Diagrama de Classes da Análise Refinado (DCAR);
Diagrama de Visibilidade (DV);
Diagrama de Classes da Análise (DCA);
Substantivos candidatos a atores, entidades que serão mantidas (ou geradas) no sistema ou atributos das entidades (marcados em azul nos RFs);
Verbos ou descrições de ações candidatos a funcionalidades do sistema (marcados em verde nos RFs).
SAÍDAS: Conceitos e Relacionamentos (marcados com "x" em azul no DCAR e DCA);
Conceitos (marcados com "x" em azul no DV);
Métodos (marcados com "x" em verde no DCAR e RFs);
Relacionamentos de Dependência e Navegabilidade (marcados com "x" em verde no DV);
Relatório de Discrepâncias.

Figura 5.5 - Técnica de Leitura P8

- Treinamento

Os treinamentos são um ponto chave, pois eles têm que ser bastante explicativos para que se possa ter confiança de que os participantes do experimento (os inspetores) tenham realmente entendido a técnica que eles irão aplicar. Observa-se que o ideal é que houvesse uma avaliação sobre esse aprendizado antes que os participantes realizassem o experimento propriamente dito, uma vez que o objetivo do experimento é avaliar a técnica ou método que está sendo o objeto de estudo.

Todos os treinamentos das técnicas de leitura foram elaborados em Powerpoint e mostram a aplicação da técnica, passo a passo, em um sistema exemplo. Esse sistema é o Sistema de Apoio à Escrita (SAPES) que tem como objetivo principal auxiliar a pesquisa bibliográfica (MARUCCI, 2002). Alguns *slides* do treinamento da técnica A1 são mostrados na Figura 5.6.

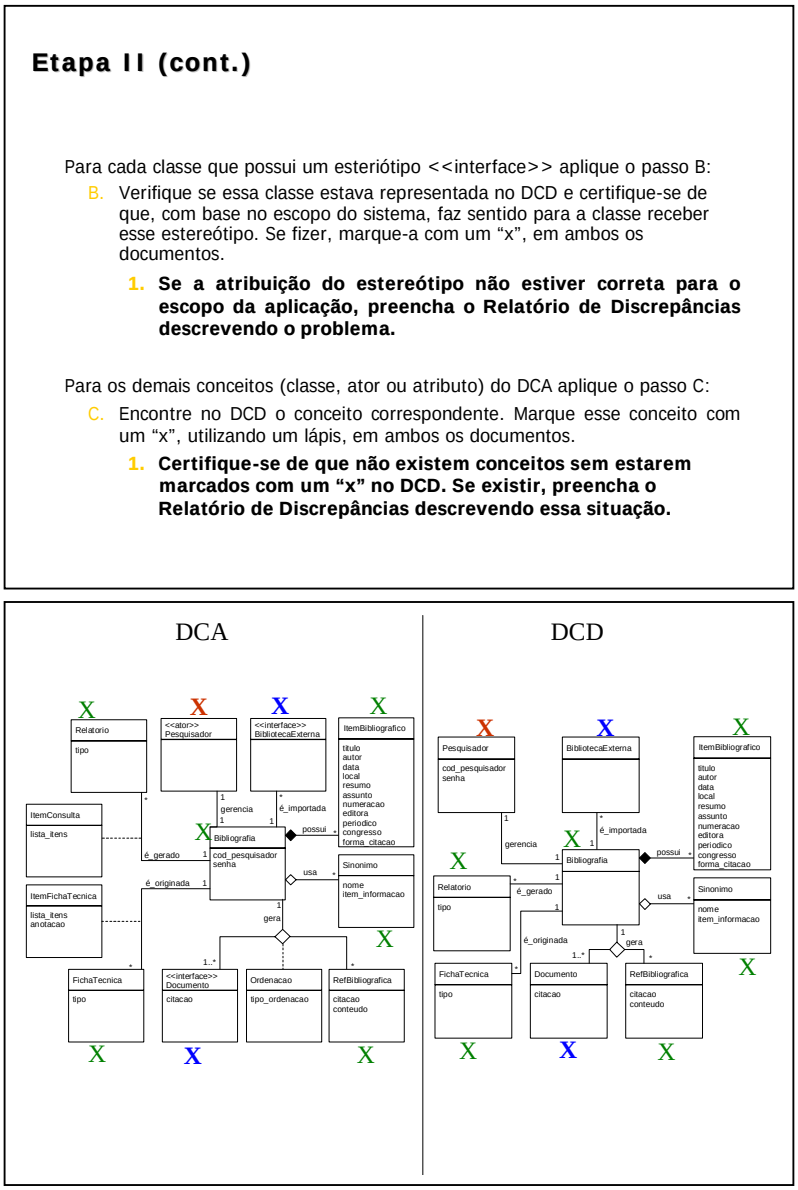


Figura 5.6 - *Slides* de treinamento

Observe que os treinamentos intercalam etapas das técnicas de leitura e *slides* que explicam a aplicação dos passos das mesmas no sistema exemplo, de forma bastante dinâmica, simulando a aplicação detalhadamente.

- Artefatos para a inspeção

Os artefatos para inspeção são os documentos que serão avaliados pela técnica e que portanto, dependem da técnica que está sendo aplicada. Assim, no caso da técnica ER1, por exemplo, que avalia o Documento de Requisitos com o Diagrama e as Especificações dos Casos de Uso, são esses os artefatos que fazem parte do Pacote de Laboratório dessa técnica.

Considerando-se as quatro técnicas, para que elas pudessem ser aplicadas, foram desenvolvidos os seguintes artefatos: o Diagrama de Casos de Uso, as Especificações dos Casos de Uso, o Diagrama de Classes do Domínio e o Diagrama de Classes da Análise. No que diz respeito à técnica P8, faltam ser desenvolvidos o Diagrama de Classes da Análise Refinado e o Diagrama de Visibilidade. Salienta-se que o Documento de Requisitos já estava elaborado, foi utilizado na condução de vários estudos experimentais e foi reutilizado neste trabalho.

Salienta-se que os artefatos que compõem os Pacotes de Laboratório são elaborados com base em outro exemplo, diferente do exemplo utilizado no treinamento. No caso em questão, utilizou-se o sistema PGCS (*Parking Garage Control System*) que vem sendo usado em vários estudos experimentais conduzidos no contexto do Projeto Readers II (MALDONADO et al., 2001). Esse sistema tem a função de controlar e supervisionar as entradas e saídas de um estacionamento. O sistema permite ou rejeita entradas no estacionamento de acordo com o número de vagas disponíveis. O Documento de Requisitos completo do PGCS encontra-se no Apêndice A.

A seguir, na Figura 5.7, apresenta-se um exemplo de artefato usado na técnica ER2, que corresponde ao Diagrama de Classes do Domínio do sistema usado nos Pacotes de Laboratório, isto é, o PGCS.

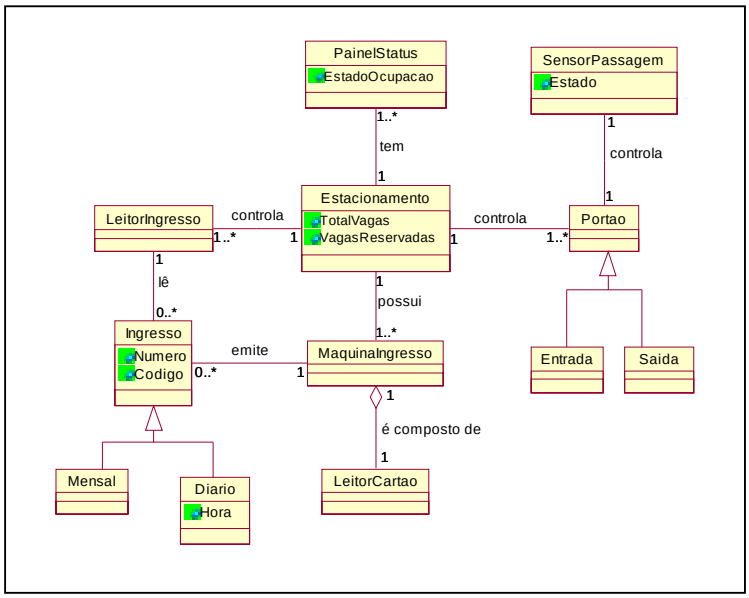


Figura 5.7 - Diagrama de Classes do Domínio

Os Pacotes de Laboratório cujos artefatos foram exemplificados e comentados anteriormente, estão apresentados na íntegra, no Documento de Trabalho (TOMA, 2004). O Apêndice A possui o Pacote de Laboratório da técnica de leitura ER1.

- Relatório de Discrepâncias

É um documento no qual os participantes da inspeção anotam os defeitos encontrados, classificando-os conforme o seu tipo de discrepância (omissão de alguma informação, informação estranha, etc), a sua severidade (que pode ser sem importância, não interferindo no documento avaliado ou muito importante, o que poderia impedir a compreensão do documento) e o seu tipo de conceito (que relaciona a discrepância com o conceito no qual ela ocorre, de forma a facilitar o entendimento do relato). Um outro ponto importante do relatório de discrepâncias é a marcação do tempo gasto para a aplicação da técnica para se obter um maior controle do experimento, como por exemplo, avaliando qual o período em que um inspetor, em média, é mais efetivo e a partir de que momento o seu rendimento começa a cair.

Ressalta-se que a coluna “Comentários” na qual o participante anota o seu ponto de vista em relação à discrepância é muito importante, pois é nela que o inspetor vai relatar a

discrepância encontrada. É fundamental que a redação nessa coluna seja bem feita, pois é analisando esse relato, que será decidido se corresponde a uma real discrepância ou não.

A Figura 5.8 apresenta o relatório de discrepâncias utilizado na técnica de leitura ER1. Observa-se que, como cada técnica avalia tipos de artefatos diferentes, esse relatório pode diferir de uma técnica para outra, pois os conceitos tratados podem não ser os mesmos.

Formulário de Relato de Discrepâncias – ER1

Início da Inspeção: _____ (hora) Data:_____ (data) ID do revisor:_____

Tipo de Conceito:
(AC) ator (AT) atributo (DA) dado
(CO) condição (CR) restrição (EN) entidade
(FU) funcionalidade (RO) papel
(RE) relacionamento (BE) comportamento

Tipo de Discrepância (Tipo Disc.):
(1) funcionalidade ou conceito necessário foi omitido.
(2) o MCU* está incorreto em relação aos requisitos.
(3) como o MCU implementa estes requisitos é ambíguo ou não completamente especificado.
(4) a informação de MCU é estranha, i.e. não mencionada pelos requisitos.
(5) outros tipos de problemas [explique abaixo].

Severidade (Sev.):
(NS) Não é sério. Mas precisa verificar este documento.
(IN) Esta discrepância invalida esta parte do documento. Verifique ambos os documentos.
(SE) Sério. Não é possível continuar a leitura deste documento. Ele deveria ser reorganizado.

Preencha a tabela com as discrepâncias encontradas. Descreva a funcionalidade dos requisitos, utilizando os números da página se possível:

Disc. #	Tipo do Conceito	Nome do Documento	Tipo Disc.	Identificação Requisito	Sev.	Comentários
01						
02						
03						
04						
05						
06						
07						

: : : :

:

(utilize a parte de trás se necessário)

Fim da Inspeção:_____ (hora)

* MCU = Modelo de Caso de Uso = Diagrama de Casos de Uso + Especificação dos Casos de Uso

Figura 5.8 - Relatório de Discrepâncias

5.4.2 Estudo de Caso

Para avaliar a dificuldade de compreensão e a viabilidade da utilização das técnicas de leitura que compõem a estratégia de inspeção do processo ProDeS/UML|pr, planejou-se aplicá-las as em um estudo de caso. No entanto, pelo fato do estudo de caso ser realizado fora do contexto de uma disciplina do curso de Graduação, para que houvesse vários participantes, somente quatro voluntários se prontificaram a participar. Além disso, como essa atividade coincidiu com o final do semestre, após o início do estudo de caso, eles comunicaram que poderiam aplicar somente uma das técnicas. Em vista disso, sem alternativas para condução desse estudo, foi aplicada apenas a técnica ER1.

Antes da aplicação da técnica de leitura ER1, foi feito um treinamento da mesma, juntamente com os conceitos necessários para a sua utilização. A aplicação dessa técnica foi realizada em sala de aula com um tempo limitado de, no máximo, três horas.

Para análise dos resultados, a autora deste trabalho aplicou a técnica ER1 no PGCS, usando o Pacote de Laboratório elaborado, com o intuito de definir uma lista inicial de discrepâncias. Essa lista foi composta de doze discrepâncias e, com base nela, é que os relatórios de discrepâncias dos quatro participantes foram avaliados.

A Tabela 5.3 apresenta o resultado das comparações da lista de discrepâncias original com as discrepâncias localizadas pelos alunos. Cada linha da tabela corresponde a uma discrepância do relatório de discrepâncias original e as colunas representam os participantes do estudo de caso. Nessa tabela estão marcados com um X, só os tipos de discrepâncias que coincidiram com a lista de discrepâncias original:

Tabela 5.3 - Comparações das Discrepâncias da técnica ER1

Discrepância	Participante 1	Participante 2	Participante 3	Participante 4
#1	X	X		
#2				
#3				
#4			X	
#5			X	
#6			X	
#7		X		
#8				
#9	X			
#10				X
#11		X		
#12				

Após a realização do estudo de caso e da análise dos resultados, observou-se que duas discrepâncias identificadas pelos alunos deveriam compor também a lista de discrepâncias. Com isso, a lista final totaliza quatorze discrepâncias.

Considerando que não foi possível aplicar todas as técnicas e que a quantidade de alunos que aplicou a ER1 foi pequena, não se pode fazer nenhuma análise mais aprofundada da utilização dessas técnicas. Por outro lado, pôde-se observar que não houve muita dificuldade em aplicar a ER1 e que, sua aplicação é viável na prática, ajudando a descobrir discrepâncias nos documentos que estão sendo avaliados.

5.5 Considerações Finais

Neste capítulo foram apresentadas duas das três propostas declaradas nos objetivos deste trabalho, ou seja, a revisão e atualização do processo ProDeS/UML, com vistas aos métodos ágeis, gerando uma nova versão desse processo, denominada ProDeS/UML|pr e, conseqüentemente o Pacote de Laboratório relacionada a mesma.

Considerando os pontos positivos do ProDeS/UML, mas ao mesmo tempo, não deixando de lado as iniciativas do “movimento ágil”, procurou-se avaliar esse processo com o objetivo de torná-lo mais ágil e mais atrativo para uma efetiva utilização. De fato, a intenção não foi torná-lo (mais) um processo ágil, mas sim, agilizá-lo, ou seja, torná-lo um processo “prático”, como são consideradas algumas das alternativas encontradas na literatura.

Para isso, foram excluídos do ProDeS/UML os seguintes documentos: Modelo de Operações, Modelo de Ciclo de Vida, Diagrama de Estados da Classe e a Descrição de Classes. Para a retirada desses artefatos, foi realizado um estudo e uma análise dos artefatos elaborados por alguns métodos (puramente) ágeis ou por outras iniciativas que combinaram métodos ágeis com outras abordagens tradicionais.

Feita essa redução no processo ProDeS/UML e gerando-se o processo ProDeS/UML|pr, analisaram-se quais das técnicas de leitura do conjunto OORTs/ProDeS que poderiam ser aplicadas nesse novo processo. Esse novo conjunto é composto das técnicas ER1, ER2, A1, P1, P6 e P8 sendo que as técnicas P1 e P6 pertencem ao conjunto OORTs, que foi proposto por Travassos et al.(2000, 2002b, 2002c). As outras técnicas, isto

é, ER1, ER2, A1 e P8 foram revisadas e atualizadas para que os pacotes de laboratório das mesmas fossem elaborados, como foi apresentado na Seção 5.4.

Nesta seção foram apresentados os documentos que compõem os Pacotes de Laboratório das técnicas de leitura ER1, ER2, A1 e P8, as quais constituem um subconjunto das técnicas de leitura que podem ser aplicadas no ProDeS/UML|pr. Ressalta-se que os pacotes foram elaborados somente para essas técnicas, pois elas foram propostas no trabalho de Marucci (2002), ex-aluna do mesmo programa de pós-graduação. As outras duas técnicas, P1 e P6 foram propostas por Travassos et al.(2000, 2002b, 2002c) e já vêm sendo avaliadas pelo autor que pertence a outra instituição.

Cada um dos documentos que compõem o Pacote de Laboratório foi explicado de forma detalhada, ressaltando a sua importância com relação ao experimento, sendo que alguns documentos servem para a análise de dados e outros para a aplicação da própria técnica de leitura. Salienta-se que todas as técnicas de leitura que permaneceram no ProDeS/UML|pr foram revisadas e atualizadas, sendo que, quanto à técnica P8, ela foi especificada neste trabalho, uma vez que Marucci (2002) não tinha concluído sua especificação.

Depois de montados os Pacotes de Laboratório das técnicas de leitura, que servem para auxiliar na aplicação inicial e na replicação de experimentos que tenham por objetivo avaliá-las e caracterizá-las, tentou-se utilizá-los em um estudo de caso.

Apesar desse planejamento, somente o Pacote de Laboratório da técnica ER1 foi avaliado por um pequeno grupo de alunos voluntários. Mesmo assim, constatou-se que o material do pacote estava bem elaborado, a aplicação da técnica estava fácil de ser entendida e várias discrepâncias foram localizadas pelos alunos.

Independentemente desses resultados iniciais, só serão obtidos resultados concretos quando se utilizar os pacotes em um estudo de caso maior, ou seja, com a participação de mais pessoas aplicando a técnica. Somente dessa forma é que se conseguirá uma melhor avaliação tanto das técnicas como dos Pacotes de Laboratório.

No capítulo seguinte, é apresentada uma estratégia de inspeção para o XP, denominada *Inspeção Ágil*.

CAPÍTULO 6

INSPEÇÃO DE SOFTWARE NO CONTEXTO DE MÉTODO XP

6.1 Considerações Iniciais

Como foi visto no Capítulo 4, o elemento mais representativo do “movimento ágil” e mais utilizado na prática é o método XP. Por isso, este capítulo trata de uma inspeção voltada para o XP.

Trata-se de uma estratégia de inspeção denominada *Inspeção Ágil* que visa melhorar a qualidade do software produzido no XP. Nessa inspeção são incluídas atividades de VV&T com o cuidado de não alterar o andamento das tarefas ao longo do XP.

Na *Inspeção Ágil* teve-se o cuidado de usar somente os artefatos produzidos pelo XP, sem a inclusão de novos documentos para não descaracterizar a agilidade do mesmo.

O capítulo está organizado da seguinte maneira: a Seção 6.2 apresenta a *Inspeção Ágil*, descrevendo todas as etapas em que as atividades de VV&T devem ser inseridas, apresentando também as Técnicas de Leitura utilizadas juntamente com os seus formulários. Na Seção 6.3 são apresentadas as considerações finais.

6.2 Técnica de Inspeção para o XP

Como foi dito anteriormente, o XP é um dos métodos ágeis mais populares. No que diz respeito à atividade de Engenharia de Requisitos, Beck (2000) não apresenta essa etapa de forma concreta e também não cita explicitamente um documento de requisitos. Um motivo que deixa muitas pessoas confusas é o fato do XP não especificar muito bem os possíveis documentos que deveriam ser criados durante o desenvolvimento (AMBLER, 2002).

No método XP os documentos mencionados por Beck (2000) são: Cartões de Estórias e Tarefas do Usuário (mais comumente chamado de Estórias do Usuário), Cartões de Tarefas do Programador (mais comumente chamado de Cartões de Tarefa) e Cartões CRC (*Class Responsibility Collaborator*) (Figuras 6.1, 6.2 e 6.3, respectivamente).

Cartão de Estória e Tarefa do Usuário

Data:__/__/_____ Tipo de Atividade: Nova:___ Dificuldade:___ Valor:_____

Número da Estória:_____ Prioridade: Usuário:_____ Técnico:_____

Referência Anterior:_____ Risco:_____ Estimativa do Técnico:_____

Descrição da Tarefa:

Notas:

Acompanhamento da Tarefa:

Data	Estado	A ser realizado	Comentário

Figura 6.1 - Modelo de Estória do Usuário (Adaptado BECK, 2000)

Cartão de Tarefa do Programador

Data:__/__/_____

Número da Estória:____ Autor do Software:_____ Estimativa da Tarefa:_____

Descrição da Tarefa:

Notas do Autor do Software:

Acompanhamento da Tarefa:

Data	Realizado	A ser realizado	Comentário

Figura 6.2 - Modelo de Cartão de Tarefa (Adaptado BECK, 2000)

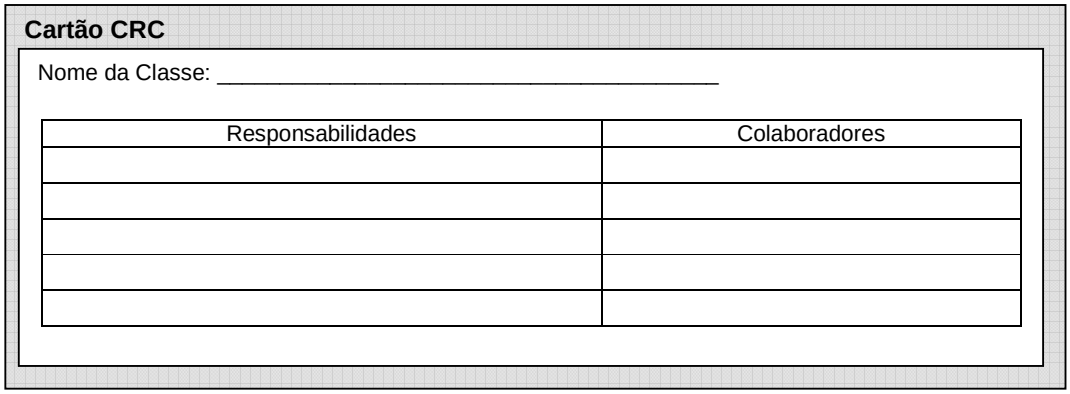


Figura 6.3 - Cartão CRC

As Estórias do Usuário são escritas pelo próprio cliente e, segundo Ambler (2002), elas são uma parte importante do XP, pois são a base dos requisitos do sistema a ser desenvolvido. Assim, nesse tipo de cartão, o cliente descreve uma estória, ou seja, uma tarefa que o sistema deve lhe oferecer. Depois de elaboradas, as Estórias são transformadas em Cartões de Tarefas pelos programadores, sendo que uma Estória do Usuário pode ser divida em vários Cartões de Tarefas. Em geral, o Cartão de Tarefa corresponde a uma subparte da Estória do Usuário, uma vez que a estória completa pode não permitir um desenvolvimento tão rápido como é esperado nesse tipo de método. Assim, esse tipo de cartão é elaborado com o propósito de se obter um melhor entendimento no sentido técnico do desenvolvimento (TEKINERDOGAN, 2003). O Cartão CRC é um documento usado em uma fase mais adiantada do projeto do sistema e serve para explorar a estrutura do código a ser desenvolvido. Ele é dividido em três seções que indicam o nome da classe, as responsabilidades da classe (as coisas que ela conhece, dados, e as coisas que ela faz, comportamentos) e as classes colaboradoras que auxiliam no cumprimento de cada responsabilidade, indicando assim um relacionamento entre as classes (AMBLER, 2002). Fora esses documentos, em XP, antes de implementarem o código, os programadores escrevem Testes de Unidade e o cliente escreve os Testes de Aceitação também conhecidos como Testes Funcionais.

Com o objetivo de melhorar a qualidade em XP, nos trabalhos de Leite (2001) e de Leonardi e Leite (2002), que foram detalhados no Capítulo 4, foi apresentado o processo denominado XR (*Extreme Requirements*) que propõe o uso de cenários no lugar das Estórias do Usuário. Esses cenários melhoram a especificação dos requisitos do cliente porque eles fornecem uma visão focada na tarefa a ser concluída juntamente com uma

comunicação eficaz entre os atores envolvidos no assunto estudado (LEITE, 2001). Assim, ocorre uma melhora na qualidade do XP, pois eles se baseiam na orientação do cliente, na linguagem natural e na facilidade de sua construção e validação.

Um outro trabalho que traz práticas de Engenharia de Requisitos em XP é o de Nawrocki et al. (2002). Eles propõem três modificações no método XP: os documentos de requisitos elaborados num formato mais tradicional devem ser gerenciados por um analista/testador; deve haver a participação de vários clientes representativos e deve ser introduzida uma etapa de Engenharia de Requisitos no começo do projeto.

Por outro lado, de acordo com vários trabalhos da literatura, um dos pontos fortes do XP são as atividades de VV&T (Validação, Verificação e Teste) ao longo do seu processo com o uso dos Testes de Unidade e os Testes de Aceitação, como está destacado (hachurado) nas Figuras 6.4 e 6.6. No entanto, ainda que mantido o levantamento de requisitos como é no XP, outras atividades de VV&T poderiam ser realizadas tanto nessa fase, quanto na fase de projeto, mais próxima da implementação propriamente, de forma a melhorar a qualidade do método.

Assim, baseando-se nas idéias dos trabalhos de Leite (2001), de Leonardi e Leite (2002) e de Nawrocki et al. (2002), comentados com maior detalhe no Capítulo 4, e também pelo fato de se ter observado que o XP ou suas práticas são freqüentemente referenciados em outros trabalhos da literatura, como está evidenciado na Tabela 5.1 do Capítulo 5, este trabalho apresenta uma estratégia de inspeção para o XP focada principalmente nas atividades de levantamento de requisitos, mas considerando os artefatos comuns construídos durante a utilização do método: Estórias do Usuário, Cartões de Tarefa e Cartões CRC. Essa estratégia, denominada *Inspeção Ágil*, propõe três atividades de validação, que podem ser observadas (pelos objetos de desenho pontilhados) nas Figuras 6.4 e 6.5 e que estão focadas na atividade de levantamento de requisitos – *Inspeção Ágil baseada nos Requisitos* – e uma de verificação, que pode ser observada na Figura 6.6, que está focada na fase de projeto – *Inspeção Ágil baseada no Projeto*.

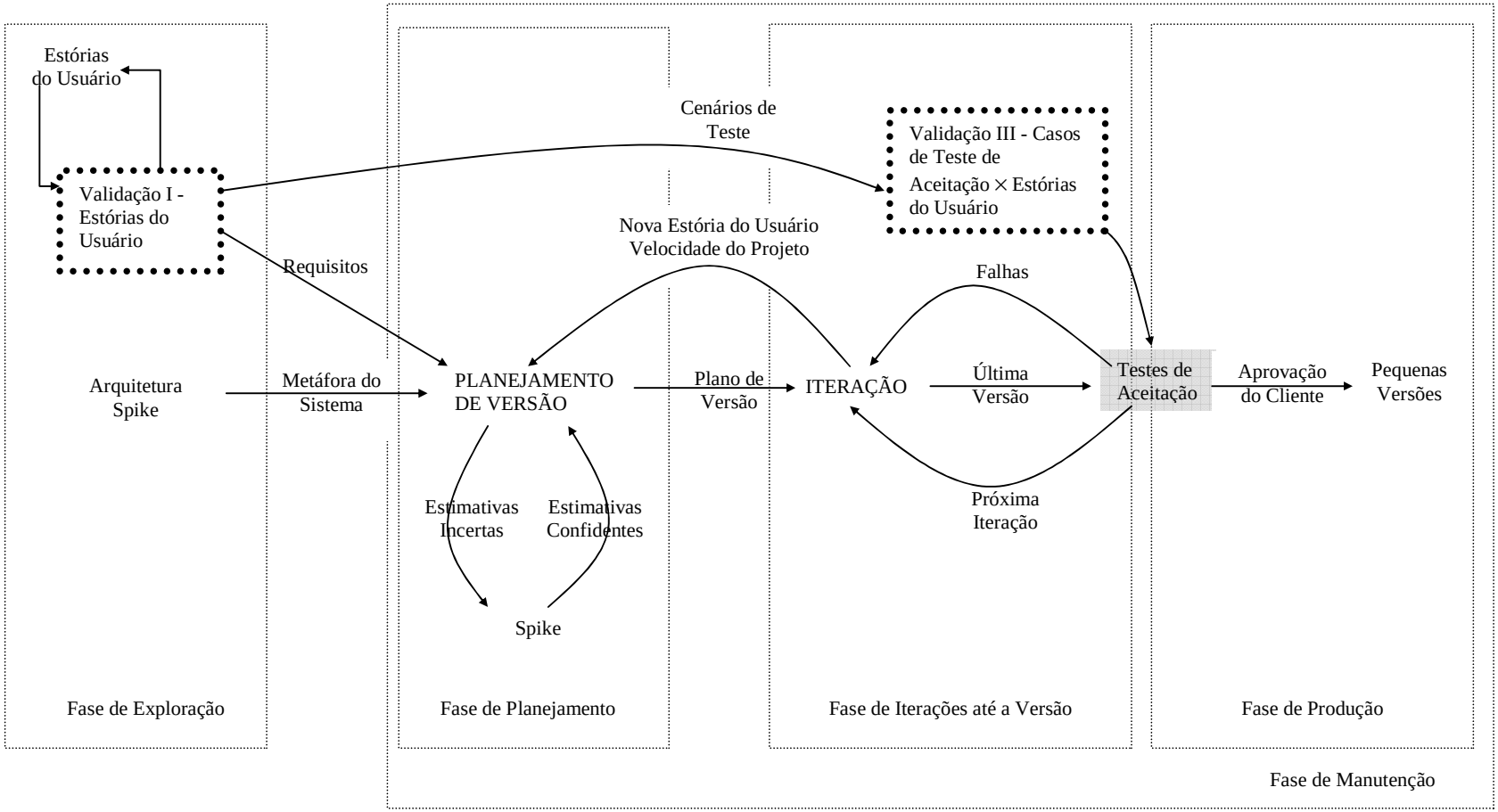


Figura 6.4 - O Ciclo de Vida do Processo XP (Adaptado WELLS, 2003)
(parte da *Inspeção Ágil baseada nos Requisitos*)

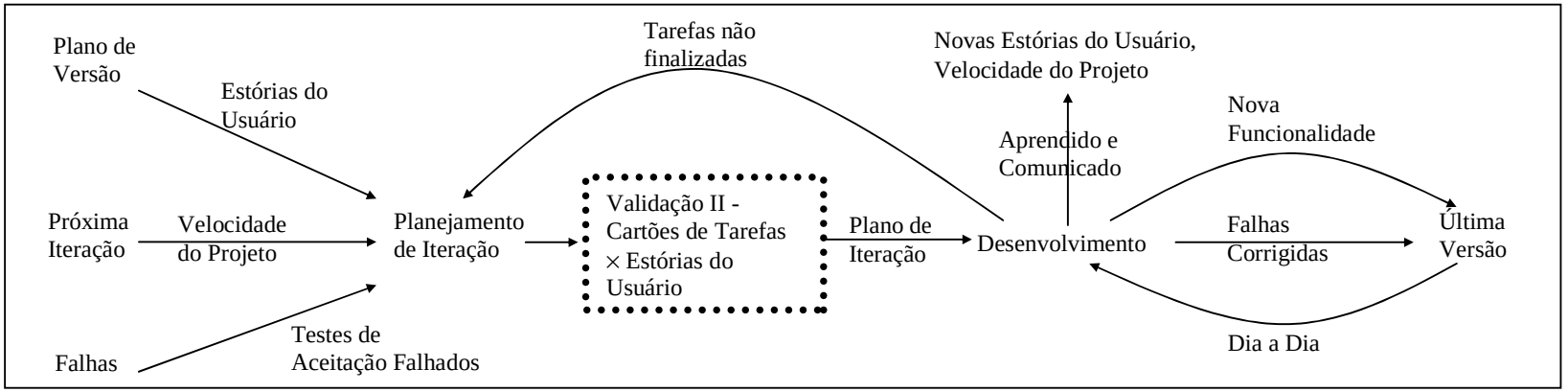


Figura 6.5 - O Ciclo de Vida de uma Iteração XP (Adaptado WELLS, 2003)
(parte da *Inspeção Ágil baseada nos Requisitos*)

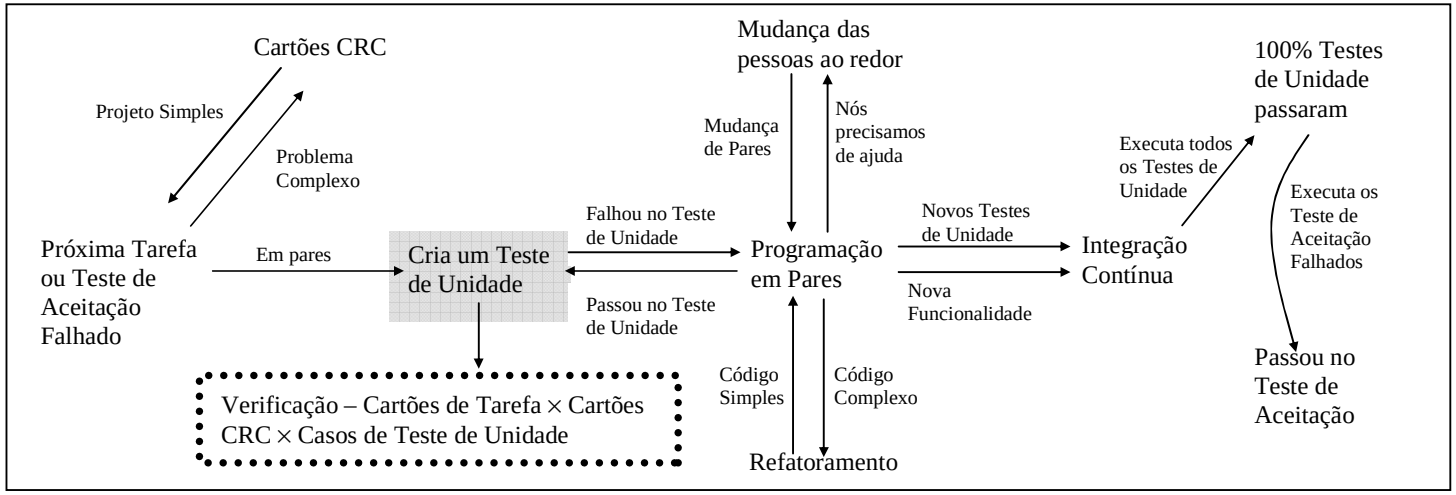


Figura 6.6 - Desenvolvimento do XP (Adaptado WELLS, 2003)
(*Inspeção Ágil baseada no Projeto*)

Observe que as duas validações propostas – Validação I e Validação III – estão inseridas na Figura 6.4, a qual representa o ciclo de vida de um processo XP, segundo Wells (2003). Na Figura 6.5, que detalha a atividade de "Iteração" da Figura 6.4, pode-se notar a terceira atividade de validação proposta – Validação II. Na Figura 6.6, que detalha a atividade de "Desenvolvimento" da Figura 6.5, pode-se observar a atividade de Verificação proposta.

Como pode ser observado pelas Figuras 6.4 e 6.5, as atividades de Validação que, segundo Andriole (1986) são aquelas que manipulam documentos que registram os requisitos do usuário, manipulam, neste caso, as Estórias do Usuário. Essas três atividades estão associadas ao levantamento de requisitos e um processo de validação pode ser abstraído para essa fase, como é apresentado na Figura 6.7.

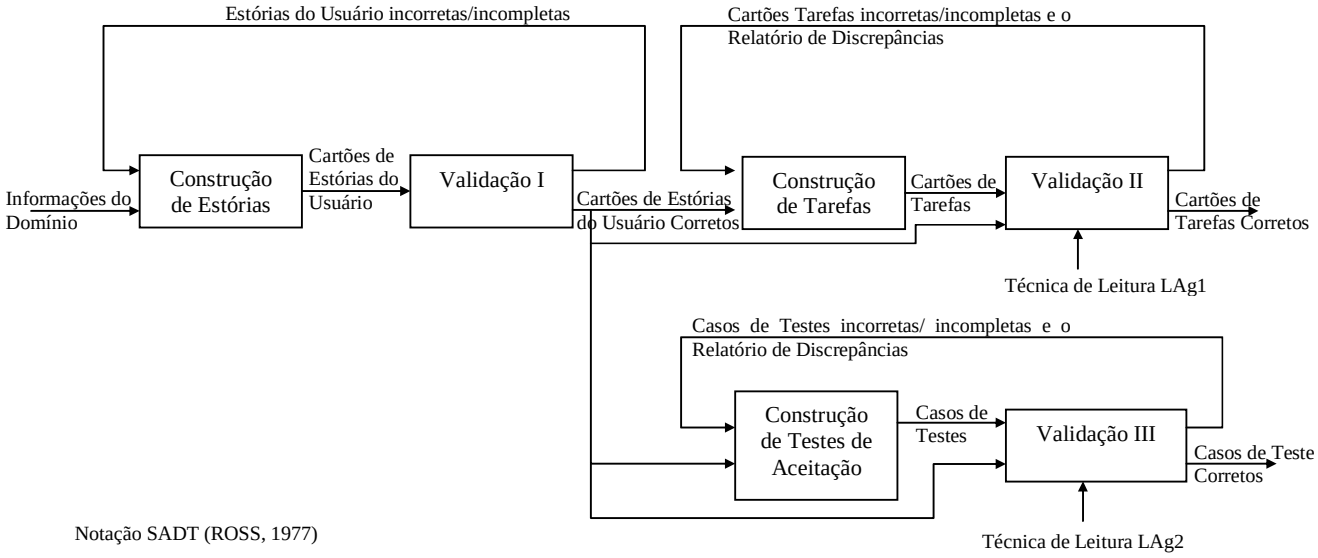


Figura 6.7 - Processo de Inspeção do XP

A seqüência de aplicação das técnicas é um ponto importante, pois como elas foram definidas para apoiar a construção dos artefatos ao longo do XP, elas devem ser aplicadas à medida que estes vão sendo elaborados. Essa sistemática agrega ao processo Atividades de Garantia de Qualidade de Software (PRESSMAN, 2002), uma vez que procura identificar os defeitos nos artefatos, tão logo eles surjam.

A Figura 6.7 representa o processo detalhado referente ao levantamento de requisitos apoiando-se no processo de construção de Cenários proposto por Leite (2001). No entanto, diferentemente de Leite (2001), esse processo utiliza somente os artefatos próprios do XP, mencionados por Beck (2000). Como pode ser observado, as três

Validações trabalham com as Estórias do Usuário que seriam equivalentes ao Documento de Requisitos de um processo de desenvolvimento de software tradicional. Igualmente ao trabalho de Nawrocki et al.(2002), aqui também se propõe que existam os clientes representativos e o cliente sênior que é aquele que estará sempre junto à equipe de desenvolvimento.

A dinâmica proposta para a realização das atividades desse processo é a seguinte:

Construção de Estórias:

- Objetivo:

O cliente sênior fornece as informações do domínio do problema escrevendo as Estórias do Usuário. No final dessa etapa, um conjunto de versões iniciais de Estórias do Usuário é criado.

- Participante:
  - cliente sênior: porque ele possui a visão global das atividades da empresa.

Validação I:

- Objetivo:

Validar as versões iniciais das Estórias do Usuário com os clientes representativos. Nessa atividade, seguindo a mesma proposta de validação de cenários, através de reuniões, definida no trabalho de Leite (2001), o cliente sênior juntamente com o testador agendam um encontro para conversar individualmente com cada um dos clientes representativos, a fim de validar as Estórias correspondentes a eles. Nesse encontro, o cliente representativo analisa as Estórias que o envolvem e, se for necessário, ele pode acrescentar ou retirar Estórias e corrigir as que achar pertinente. Ao final desse encontro, todos os clientes representativos devem estar de acordo com as suas Estórias e se houver algum conflito de Estórias entre os clientes representativos, o cliente sênior e o testador devem solucioná-lo. Aqui, o testador precisa pensar como um engenheiro de requisitos, descobrindo todas as possíveis fontes de requisitos e encontrar pontos comuns e conflitos, pois segundo Nawrocki et al. (2002), os requisitos e os testes de aceitação estão no mesmo nível de abstração, o que torna o testador a melhor pessoa para fazer o trabalho de analista responsável pela documentação e gerenciamento de requisitos. Se existir alguma Estória que deva ser totalmente alterada, ela deve ser refeita e, se necessário, um novo encontro com o cliente representativo será marcado. Em XP, fala-se na participação constante de um só cliente, mas como Nawrocki et al. (2002) e Leite (2001) dizem, é quase impossível uma

única pessoa saber de todos os assuntos da empresa. Por este fato, deve-se planejar o encontro com cada um dos clientes representativos de cada departamento envolvido.

- Participantes:
  - cliente sênior: pelo motivo dele ter escrito as Estórias;
  - clientes representativos: por possuírem conhecimentos profundos de um determinado departamento da empresa e dos problemas particulares;
  - testador: pelo fato dele ser a melhor pessoa para assumir o papel de Engenheiro de Requisitos nesse caso.

Construção de Tarefas:

- Objetivo:

Dividir as Estórias do Usuário em pequenas tarefas. Isso é realizado depois que as Estórias do Usuário são validadas, pois elas é que vão dar subsídios à elaboração dos Cartões de Tarefa os quais são escritos pelos programadores.

- Participante:
  - programador: porque os Cartões de Tarefa englobam assuntos técnicos.

Validação II:

- Objetivo:

Logo que as Tarefas ficam prontas, elas passam pela Validação II que no caso é uma validação na qual são comparadas as Estórias do Usuário com os Cartões de Tarefa através de uma rápida Técnica de Leitura LAg1 (Técnica de Leitura Ágil 1).

A Técnica de Leitura LAg1 tem como saída um Relatório de Discrepâncias que, caso esteja vazio, significa que todas as Tarefas estão, aparentemente, corretas e de acordo com as Estórias. Se o Relatório de Discrepâncias tiver alguma anotação, significa que algum Cartão de Tarefa tem que ser alterado, ou seja, esse Cartão de Tarefa é refeito e uma nova aplicação da Técnica LAg1 é realizada.

- Participantes:
  - testador: porque ele participou da Validação I e ele também tem o conhecimento técnico de um programador;
  - treinador: porque ele não participou das construções dos artefatos e possui o conhecimento técnico;

- programadores: porque eles criaram os Cartões de Tarefa e além disso, nessa rápida reunião de inspeção todos os programadores ficam sabendo de todas as Estórias do Usuário e dos Cartões de Tarefa do sistema, reforçando desse modo, a prática da Propriedade coletiva do XP.

Construção de Testes de Aceitação:

- Objetivo:

As Estórias do Usuário são a base para a elaboração desses Casos de Teste de Aceitação, os quais são feitos pelo cliente sênior com a ajuda do testador.

- Participantes:
  - cliente sênior: porque participou da Validação I;
  - testador: porque participou da Validação I.

Validação III:

- Objetivo:

Os Casos de Teste de Aceitação são inspecionados na Validação III, por meio de uma Técnica de Leitura LAg2 (Técnica de Leitura Ágil 2) que usa as Estórias do Usuário como referência para avaliar os Casos de Teste. Um Relatório de Discrepâncias poderá ser gerado, se algum Caso de Teste precisar ser refeito, por não estar compatível com as Estórias ou se faltarem Casos de Teste que explorem as Estórias devidamente. Até que o Relatório de Discrepâncias fique vazio, essa Validação III deve ser repetida.

- Participantes:
  - testador: porque participou da validação das Estórias do Usuário e da elaboração dos Casos de Teste de Aceitação;
  - treinador: porque não participou da elaboração desses artefatos e, pelo fato de ter um conhecimento técnico, pode fazer a validação com uma visão diferente dos autores.
  - cliente: porque participou da construção e da validação das Estórias do Usuário e da elaboração dos Casos de Teste de Aceitação.

A partir deste ponto, as Tarefas estão prontas para serem implementadas e os Casos de Teste de Aceitação estão prontos para validarem as funcionalidades do Sistema.

Observe que as duas Técnicas de Leitura LAg1 e LAg2 usam as Estórias do Usuário para a comparação, pois tanto os Cartões Tarefas como os Casos de Teste de Aceitação são originados a partir dessas Estórias.

Além da *Inspeção Ágil baseada nos Requisitos* detalhada até aqui, tem-se também a *Inspeção Ágil baseada no Projeto*, que possui a atividade de Verificação (Figura 5.9) localizada nas proximidades da implementação do código, a qual é descrita a seguir:

Verificação:

- Objetivo:

Faz uma verificação entre os Cartões de Tarefa, os Cartões CRC e os Casos de Teste de Unidade utilizando a LAg3 (Técnica de Leitura Ágil 3). Enquanto o Relatório de Discrepâncias não estiver vazio, a atividade de Verificação deve ser repetida para a correção das discrepâncias anotadas.

Essa técnica deve ser aplicada pelos próprios programadores, em pares. Cada programador aplica a técnica LAg3 separadamente e, depois de finalizada essa aplicação, ocorre uma troca dos Relatórios de Discrepâncias entre os dois programadores. Dessa maneira, cada programador analisa o que o outro escreveu no Relatório de Discrepâncias verificando se essas anotações estão de acordo com o que ele pensou. Se for necessário, o Testador pode auxiliá-los nessa atividade.

- Participantes:
  - pares de programadores: porque foram eles que elaboraram os artefatos utilizados;
  - testador: porque não participou da elaboração desses artefatos e, pelo fato de ter um conhecimento técnico, pode fazer a validação com uma visão diferente dos autores.

Pelo fato do processo XP ser ágil, a aplicação dessas Técnicas de Leitura não é demorada e qualquer um que queira participar dessa inspeção é permitido, pois como o código, a inspeção também é uma propriedade coletiva.

A técnica LAg1 faz uma validação dos Cartões de Tarefa para verificar se os conceitos e as funcionalidades das Estórias do Usuário foram bem representados.

A técnica LAg2 faz uma validação dos Casos de Teste de Aceitação para verificar se as funcionalidades, condições e restrições das Estórias do Usuário foram bem retratadas.

A técnica LAg3 faz uma verificação entre os Cartões de Tarefa, os Cartões CRC e os Casos de Teste de Unidade para investigar se todos os dados estão consistentes entre eles.

A seguir, são apresentadas as técnicas de leitura LAg1, LAg2 e a LAg3, intercaladas com os seus Relatórios de Discrepâncias (Figuras 6.8 a 6.13, respectivamente).

Técnica de Leitura LAg1 – Estórias do Usuário x Cartões de Tarefa (Validação II)

Objetivo: Verificar se os conceitos e funcionalidades que estão descritos nos Cartões de Estória do Usuário foram capturados apropriadamente pelos Cartões de Tarefa.

Entradas para o processo:
1. Um conjunto de Cartões de Estórias do Usuário.
2. Um conjunto de Cartões de Tarefa.

- Para cada Estória do Usuário aplique os passos de 1 a 5:
  1. Leia a Estória para poder entendê-la.
  2. Separe o(s) respectivo(s) Cartão(ões) de Tarefa correspondente(s) à Estória selecionada (veja pelo número da estória que consta no Cartão de Tarefa).
     2.1 Se não encontrar nenhum Cartão de Tarefa, preencha o Relatório de Discrepâncias descrevendo o problema e recomece a Técnica com uma outra Estória.
  3. Verifique se os substantivos importantes (candidatos a classes e atributos) da Estória estão registrados em algum Cartão de Tarefa.
     3.1 Se existir algum substantivo que não é está registrado no(s) Cartão(ões) de Tarefa, verifique se esse conceito é relevante no contexto da Estória do Usuário. Se necessário, preencha o Relatório de Discrepâncias descrevendo o problema.
  4. Verifique se as funcionalidades que compõem uma Estória, as quais estariam representadas por verbos ou descrições de ações, estão registrados em algum Cartão de Tarefa.
     4.1 Se existir algum verbo ou descrição de ações que não está registrado no(s) Cartão(ões) de Tarefa, verifique se essa funcionalidade é relevante no contexto da Estória do Usuário. Se necessário, preencha o Relatório de Discrepâncias descrevendo o problema.
  5. Verifique se as restrições ou condições da Estória estão registradas em algum Cartão de Tarefa;
     5.1 Se existir alguma restrição ou condição da Estória que não está registrada no(s) Cartão(ões) de Tarefa, verifique se esse conceito é relevante no contexto da Estória do Usuário. Se necessário, preencha o Relatório de Discrepâncias descrevendo o problema.
- Se no final sobrar algum Cartão de Tarefa sem a sua Estória correspondente, preencha o Relatório de Discrepâncias.

Figura 6.8 - Técnica de Leitura LAg1

Formulário de Relato de Discrepâncias – LAg1

Início da Inspeção: _____ (hora) Data:_____ (data)

Tipo de Discrepância (Tipo Disc.):
(1) funcionalidade ou conceito necessário foi omitido.
(2) o CT* está incorreto em relação à EU**.
(3) como o CT representa a informação da estória é ambígua ou não completamente especificada.
(4) a informação do CT é estranha, i.e. não mencionada pela EU.
(5) outros tipos de problema [explique abaixo].

Severidade (Sev.):
(NS) Não é sério. Mas precisa verificar este documento.
(IN) Esta discrepância invalida esta parte do documento. Verifique ambos os documentos.
(SE) Sério. Não é possível continuar a leitura deste documento. Ele deveria ser reorganizado.

Preencha a tabela com as discrepâncias encontradas:

Disc. #	Nome do Documento	Tipo Disc.	Número da Estória	Sev.	Comentários
01					
02					
03					
04					
05					
06					
07					
08					
09					
10					
11					
12					
13					
14					
15					
16					

(utilize a parte de trás se necessário)

Fim da Inspeção:_____ (hora)

* CT = Cartão de Tarefa
** EU = Estória do Usuário

Figura 6.9 - Relatório de Discrepâncias LAg1

Técnica de Leitura LAg2 – Estórias do Usuário x Casos de Teste de Aceitação (Validação III)

Objetivo: Verificar se as restrições e as condições que estão descritas nos Casos de Teste de Aceitação foram elaboradas apropriadamente de acordo com os Cartões de Estória do Usuário.

Entradas para o processo:
1. Um conjunto de Cartões de Estórias do Usuário.
2. Um conjunto de Casos de Teste de Aceitação.

- Para cada Estória do Usuário aplique os passos de 1 a 5:
  1. Leia a Estória do Usuário para poder entendê-la.
  2. Identifique o(s) respectivo(s) Caso(s) de Teste de Aceitação correspondente(s) à Estória selecionada e marque-o(s) com o número da Estória.
     2.1 Se não encontrar nenhum Caso de Teste de Aceitação, preencha o Relatório de Discrepâncias descrevendo o problema e recomece a Técnica com uma outra Estória.
  3. Verifique se todas as funcionalidades relacionadas ao contexto da Estória estão exploradas em algum Caso de Teste.
     3.1 Se existir alguma funcionalidade que não é explorada em nenhum Caso de Teste de Aceitação, verifique se essa funcionalidade é realmente relevante no contexto da Estória do Usuário. Se necessário, preencha o Relatório de Discrepâncias apontando a(s) funcionalidade(s) para a qual não foram criados Casos de Teste.
  4. Verifique se todas as restrições relacionadas ao contexto da Estória estão exploradas em algum Caso de Teste.
     4.1 Se existir alguma restrição que não é explorada em nenhum Caso de Teste de Aceitação, verifique se essa restrição é realmente relevante no contexto da Estória do Usuário. Se necessário, preencha o Relatório de Discrepâncias apontando a restrição para a qual não foram criados Casos de Teste.
  5. Verifique se todas as condições relacionadas ao contexto Estória estão exploradas em algum Caso de Teste.
     5.1 Se existir alguma condição que não é representada em nenhum Caso de Teste de Aceitação, verifique se essa condição é realmente relevante no contexto da Estória do Usuário. Se necessário, preencha o Relatório de Discrepâncias apontando a condição para a qual não foram criados Casos de Teste.
- Se no final sobrar algum Caso de Teste de Aceitação sem a sua Estória correspondente, ou seja, sem estar marcado com um número da Estória, preencha o Relatório de Discrepâncias.

Figura 6.10 - Técnica de Leitura LAg2

Formulário de Relato de Discrepâncias – LAg2

Início da Inspeção: _____ (hora) Data:_____ (data)

Tipo de Discrepância (Tipo Disc.):
(1) funcionalidade ou conceito necessário foi omitido.
(2) o CTA* está incorreto em relação à EU**.
(3) como o CTA representa a informação da estória é ambígua ou não completamente especificada.
(4) a informação do CTA é estranha, i.e. não mencionada pela EU.
(5) outros tipos de problema [explique abaixo].

Severidade (Sev.):
(NS) Não é sério. Mas precisa verificar este documento.
(IN) Esta discrepância invalida esta parte do documento. Verifique ambos os documentos.
(SE) Sério. Não é possível continuar a leitura deste documento. Ele deveria ser reorganizado.

Preencha a tabela com as discrepâncias encontradas:

Disc. #	Nome do Documento	Tipo Disc.	Número da Estória	Sev.	Comentários
01					
02					
03					
04					
05					
06					
07					
08					
09					
10					
11					
12					
13					
14					
15					
16					

(utilize a parte de trás se necessário)

Fim da Inspeção:_____ (hora)

* CTA = Caso de Teste de Aceitação
** EU = Estória do Usuário

Figura 6.11 - Relatório de Discrepâncias LAg2

Técnica de Leitura LAg3 – Cartões de Tarefa X Cartões CRC x Casos de Teste de Unidade (Verificação)

Objetivo: Verificar se os conceitos e funcionalidades que estão descritos nos Cartões de Tarefa foram capturados apropriadamente pelos Cartões CRC e pelos Casos de Teste de Unidade.

Entradas para o processo:

1. Um conjunto de Cartões de Tarefa.
2. Um conjunto de Cartões CRC.
3. Um conjunto de Casos de Teste de Unidade.

- Marque cada um dos Cartões CRC utilizando uma numeração seqüencial.
- Para cada Cartão CRC numerado, aplique os passos de 1 a 6:
  1. Leia o nome da classe, suas responsabilidades e seus colaboradores apontados no Cartão CRC.
  2. Identifique o(s) Cartão(ões) de Tarefa que faz referencias à classe descrita no topo do Cartão do CRC marcando-o(s) com o número do Cartão CRC em questão.
     2.1 Se não encontrar nenhum Cartão de Tarefa, preencha o Relatório de Discrepâncias descrevendo o problema e recomece a Técnica com um outro Cartão CRC.
  3. Verifique se cada colaborador possui o seu Cartão CRC correspondente, marcando na frente de seu nome, o número do Cartão CRC correspondente.
     3.1 Se existir algum colaborador sem nenhuma numeração, preencha o Relatório de Discrepâncias descrevendo o problema.
  4. Verifique se as responsabilidades e seus respectivos colaboradores apontados no Cartão CRC estão descritos em algum Cartão de Tarefa identificado no passo 2.
     4.1 Se nem todas as responsabilidades e os colaboradores tiverem sido identificados em algum Cartão de Tarefa, preencha o Relatório de Discrepâncias descrevendo o problema e recomece a Técnica com um outro Cartão CRC.
  5. Identifique o(s) respectivo(s) Caso(s) de Teste de Unidade correspondente(s) à classe lida no passo 1, marcando-o(s) com o número do Cartão CRC em questão.
     5.1 Se não conseguir identificar nenhum Caso de Teste de Unidade para o Cartão CRC, preencha o Relatório de Discrepâncias e recomece a Técnica com um outro Cartão CRC.
  6. Verifique se todas as responsabilidades da classe estão sendo exploradas em algum Caso de Teste de Unidade identificado no passo 5. Para cada responsabilidade encontrada, marque-a com um "×".
     6.1 Se existir uma responsabilidade que não esteja marcada com um "×", preencha o Relatório de Discrepâncias descrevendo o problema.
- Se no final sobrar algum Cartão de Tarefa sem nenhuma numeração associada a algum Cartão CRC, preencha o Relatório de Discrepâncias.
- Se no final sobrar algum Caso de Teste de Unidade sem nenhuma numeração associada o algum Cartão CRC, preencha o Relatório de Discrepâncias.

Figura 6.12 - Técnica de Leitura LAg3

Formulário de Relato de Discrepâncias – LAg3

Início da Inspeção: _____ (hora) Data:_____ (data)

Tipo de Discrepância (Tipo Disc.):
(1) funcionalidade ou conceito necessário foi omitido.
(2) o CRC* está incorreto em relação à CT**.
(3) como o CRC representa a informação do CT é ambígua ou não completamente especificada.
(4) a informação do CRC é estranha, i.e. não mencionada pela CT.
(5) o CTU*** está incorreto em relação à CRC.
(6) como o CTU representa a responsabilidade do CRC é ambígua ou não completamente especificada.
(7) a informação do CTU é estranha, i.e. não mencionada pela CRC.
(8) outros tipos de problema [explique abaixo].

Severidade (Sev.):
(NS) Não é sério. Mas precisa verificar este documento.
(IN) Esta discrepância invalida esta parte do documento. Verifique ambos os documentos.
(SE) Sério. Não é possível continuar a leitura deste documento. Ele deveria ser reorganizado.

Preencha a tabela com as discrepâncias encontradas:

Disc. #	Nome do Documento	Tipo Disc.	Número do Cartão CRC	Sev.	Comentários
01					
02					
03					
04					
05					
06					
07					
08					
09					
10					
11					
12					
13					
14					

(utilize a parte de trás se necessário)

Fim da Inspeção:_____ (hora)

* CRC = Cartão CRC
** CT = Cartão de Tarefa
***CTU = Caso de Teste de Unidade

Figura 6.13 - Relatório de Discrepâncias LAg3

6.3 Considerações Finais

Neste capítulo foi apresentado o último objetivo deste trabalho que era definir uma estratégia de inspeção para o método XP, denominada *Inspeção Ágil*. Essa inspeção é composta por três validações e uma verificação. As validações formam a *Inspeção Ágil baseada nos Requisitos* e a verificação corresponde à *Inspeção Ágil baseada no Projeto*. Essas atividades de validação e verificação seguem uma ordem determinada de aplicação, uma vez que elas são aplicadas à medida que os documentos vão sendo produzidos. Ressalta-se que para apoiar essas atividades, foram definidas técnicas de leituras que auxiliam na revisão dos documentos gerados durante o desenvolvimento apoiado pelo XP, que são: Estórias do Usuário, Cartão de Tarefa, Cartão CRC, Casos de Teste de Aceitação e Casos de Teste de Unidade.

Ainda que o XP possua atividades de VV&T, com o Teste de Unidade e os Testes de Aceitação, a *Inspeção Ágil* pode ser incluída para se obter uma melhora na qualidade do software final, sem alterar muito a estrutura do método XP.

No capítulo seguinte, são apresentadas as conclusões deste trabalho que englobam a criação do processo ProDeS/UML|pr, as técnicas de leitura do OORTs/ProDeS usadas nesse novo processo juntamente com os seus Pacotes de Laboratório e a *Inspeção Ágil* no método XP.

CAPÍTULO 7

CONCLUSÕES

Neste trabalho apresentaram-se duas estratégias de inspeção para artefatos de software no contexto OO: uma delas para dar suporte ao processo ProDeS/UML|pr, o qual também foi proposto neste trabalho e a outra, no contexto dos métodos ágeis, particularmente para o método XP, que é o representante mais evidente desse paradigma de desenvolvimento. Além disso, foram também elaborados neste trabalho os Pacotes de Laboratório para as técnicas de leitura que apóiam a estratégia de inspeção do processo ProDeS/UML|pr. Tanto o processo como as técnicas de leitura foram derivados do processo ProDeS/UML e do conjunto OORTs/ProDeS, respectivamente, propostos anteriormente em outros trabalhos.

Considerando-se as características positivas do processo ProDeS/UML como o desenvolvimento gradativo do software e o uso da notação UML e, considerando-se também, o "movimento ágil" a primeira etapa realizada neste trabalho foi a definição do processo ProDeS/UML|pr.

O processo ProDeS/UML|pr pode ser considerado um processo "prático", isto é, um processo que não é nem tão "pesado" quanto os processos mais tradicionais e nem tão ágil quanto o método XP. Para chegar à sua definição, foi feito um levantamento dos métodos ágeis propostos na literatura bem como de trabalhos que propõem uma combinação desses métodos com métodos mais tradicionais, na tentativa de torná-los mais ágeis. Esse levantamento consistiu em identificar os artefatos de software elaborados por eles, ou as informações que eles registram e o tipo de documentos que eles geram e comparar isso com os artefatos gerados no ProDeS/UML.

Como resultado dessa comparação identificaram-se os documentos do ProDeS/UML que correspondiam aos documentos gerados pelos métodos ou que armazenavam o mesmo tipo de informação, já que vários trabalhos não diziam, explicitamente, os documentos utilizados. Com isso, foram retirados do ProDeS/UML os

seguintes diagramas: Modelo de Operações, Modelo de Ciclo de Vida, Descrição das Classes e Diagrama de Estados da Classe.

Com a eliminação desses documentos, conseqüentemente, algumas técnicas de leitura do conjunto OORTs/ProDeS não podem mais ser aplicadas, permanecendo na estratégia de inspeção do ProDeS/UML|pr as seguintes técnicas, com a seguinte ordem de aplicação: ER1, ER2, A1, P6, P8 e P1. Como o conjunto OORTs/ProDeS foi definido no trabalho de Marucci (2002) e essas técnicas ainda não tinham sido validadas, fez-se uma revisão das que permaneceram no novo processo para que os Pacotes de Laboratório pudessem ser elaborados. Feita essa revisão e as atualizações necessárias, foram elaborados integralmente os Pacotes de Laboratório das técnicas ER1, ER2 e A1 e, parcialmente, o pacote da técnica P8. O pacote é composto dos seguintes documentos: o questionário de caracterização do participante, o relatório de discrepâncias, a técnica de leitura propriamente dita, o treinamento e os artefatos a serem avaliados no experimento. No caso da técnica P8, a elaboração dos artefatos não está concluída.

Depois de concluídos os Pacotes de Laboratório, tentou-se aplicá-los em um estudo de caso, mas somente a técnica ER1 foi aplicada por poucos alunos. Esse fato ocorreu pelo motivo do estudo de caso não ter sido embutido em nenhuma disciplina, contando apenas com alunos voluntários.

No contexto do método XP definiu-se uma estratégia de inspeção ágil composta de quatro atividades, sendo três de validação, por estar fazendo referência ao documento que registra os requisitos do usuário, que no caso do XP é o cartão de Estórias do Usuário e uma de verificação que não envolve esse documento.

As atividades de inspeção aplicam-se aos seguintes artefatos do XP: Estórias do Usuário, Cartões de Tarefa, Cartões CRC, Casos de Teste de Aceitação e Casos de Teste de Unidade, conforme eles vão sendo elaborados. Apesar de um dos pontos mais evidenciados do XP seja o “testar sempre”, percebem-se na literatura alguns trabalhos que apontam uma deficiência, principalmente, no que diz respeito à elicitação de requisitos. Assim, a estratégia proposta insere atividades de VV&T visando prioritariamente essa etapa. Ela foi baseada nas idéias propostas nos trabalhos de Leite (2001), de Leonardi e Leite (2002) e de Nawrocki et al. (2002) que têm como objetivo principal melhorar a qualidade do software desenvolvido com o método XP.

Destaca-se que todas essas técnicas de leitura foram desenvolvidas com o intuito de serem executadas de uma forma muito rápida e simples, como tudo é proposto em XP.

7.1 Contribuições

A seguir relacionam-se algumas contribuições deste trabalho:

- Adequação do processo ProDeS/UML, com base nas informações mantidas pelos métodos ágeis, gerando um processo mais prático denominado ProDeS/UML|pr.
- Revisão e correção das técnicas de leitura OORTs/ProDeS, que permaneceram factíveis de aplicação no processo ProDeS/UML|pr.
- Especificação da técnica de leitura P8, proposta no trabalho de Marucci (2002), para a Fase de Projeto do ProDeS/UML.
- Elaboração dos Pacotes de Laboratório para as técnicas de leitura aplicáveis no ProDeS/UML|pr – ER1, ER2, A1 e P8 – para dar suporte à realização de estudos experimentais que as avaliem. Isso inclui a definição de todos os artefatos necessários para compor os pacotes como, formulários de discrepância, modelos a serem avaliados, treinamento na técnica, etc.
- Definição de uma estratégia de inspeção e das respectivas técnicas de leitura para o método XP, composta por atividades de validação e verificação, introduzindo atividades de Garantia de Qualidade de Software principalmente na etapa de elicitação de requisitos.

7.2 Trabalhos Futuros

A seguir relacionam-se os aspectos que abrem perspectivas de continuidade deste trabalho:

- Aplicar o processo ProDeS/UML|pr e o método XP, com suas respectivas estratégias de inspeção baseadas nas técnicas de leitura, no contexto acadêmico, para comparar essas duas abordagens com o intuito de verificar quão ágil ficou o processo em relação a um método puramente ágil.
- Finalizar o pacote de laboratório para a técnica de leitura P8.
- Realizar estudos experimentais para avaliar as técnicas de leitura aplicáveis ao ProDeS/UML|pr, com o objetivo de fornecer subsídios para a avaliação e o refinamento das mesmas, bem como dos Pacotes de Laboratório criados.

- Realizar estudos experimentais para avaliar as técnicas de leitura definidas para o método XP, verificando-a em relação à eficiência e eficácia na detecção de defeitos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHAMSSON, P.; WARSTA, J.; SIPONEN, M.T. New directions on Agile Methods: a comparative analysis. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 25., 2003, Portland. **Proceedings...** Washington: IEEE Computer Society, 2003. p.244-254. Disponível em: <http://portal.acm.org/ft_gateway.cfm?id=776846&type=pdf&coll=portal&dl=ACM&CFID=15943883&CFTOKEN=21662768>. Acesso em: 28 ago. 2003.

ABRAHAMSSON, P., SALO, O.; RONKAINEN, J.; WARSTA, J. Agile software development methods: reviews and analysis. Espoo: VTT Publications, 2002. Disponível em: < http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf>. Acesso em: 25 abr. 2004.

AMARAL, E.A.G.G.; TRAVASSOS, G.H. Em busca de uma abordagem para empacotamento de experimentos em engenharia de software. In: JIISIC - JORNADA IBERO-AMERICANA DE ENGENHARIA DE SOFTWARE E ENGENHARIA DO CONHECIMENTO, 2., 2002, Salvador. **Anais...** Salvador: UNIFACS; São Carlos: ICMC-USP, 2002. 1 CD-ROM.

AMBLER, S.W. **Agile modeling:** effective practices for eXtreme programming and unified process. New York: John Wiley & Sons, 2002.

ANDRIOLE, S.J. **Software validation, verifications, testing and documentation**. Princeton: Petrocelli Books, 1986. 389 p.

ARMANO, G.; MARCHESI, M. A rapid development process with UML. **ACM SIGAPP Applied Computing Review**, New York, v.8, n.1, p.4-11, Apr. 2000.

AURUM, A.; PETERSSON, H.; WOHLIN, C. State-of-the-art: software inspections after 25 years. **Software Testing, Verification and Reliability,** Chichester, v.12, n.3, p. 133-154, Sep. 2002.

BASILI, V.R. Evolving and packaging reading technologies. **Journal of Systems and Software,** New York, v.38, n.1, p.3-12, Jul. 1997.

BASILI, V.R.; CALDIERA, G.; LANUBILE, F.; SHULL, F. Studies on reading techniques. In: ANNUAL SOFTWARE ENGINEERING WORKSHOP, 21., 1996, Greenbelt. **Proceedings...** Greenbelt: The Software Engineering Laboratory, 1996a. p. 59-65. Disponível em: <http://sel.gsfc.nasa.gov/website/sew/1996/topics/basili_p.pdf>. Acesso em: 15 jan. 2003.

BASILI, V. R.; GREEN, S.; LAITENBERGER, O.; LANUBILE, F.; SHULL, F.; SORUMGARD, S.; ZELKOWITZ, M. Packging researcher experience to assist replication of experiments. In: ISERN MEETING, 1996, Sydney. **Proceedings...** Sydney: The University of New South Wales : The CSIRO-Macquarie University Joint Research Center for Advanced System Engineerin, 1996b. Disponível em: <http://www.cs.umd.edu/ projects/SoftEng/ESEG/manual/pbr_package/docs/ISERN96.ps>. Acesso em: 20 out. 2003.

BECK, K. **Extreme programming explained:** embrace change. Reading: Addison Wesley Longman, 2000.

BECK, K.; BEEDLE, M.; BENNEKUM, A.; COCKBURN, A.; CUNNINGHAM, W. et al. Manifesto for Agile Software development. 2001. Disponível em: <http://www.agilemanifesto.org>. Acesso em 20 abr. 2004.

BOOCH, G. **Object-oriented analysis and design with applications.** 2. ed. Reading: Addison-Wesley Longman, 1994.

BOOCH, G.; MARTIN, R.C.; NEWKIRK, J. The process. In: _____. **Object oriented analysis and design with applications**. 2. ed. Reading: Addison Wesley, 1998. cap. 4, p. 93-108.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML:** guia do usuário. Rio de Janeiro: Campus, 2000. 472 p.

CHAU, T.; MAURER, F.; MELNIK, G. Knowledge sharing: Agile Methods vs. Tayloristic Methods. In: INTERNATIONAL WORKSHOP ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES, 12., 2003, Linz. **Proceedings...** Washington**:** IEEE Computer Society, 2003. Disponível em: <http://csdl.computer.org/dl/proceedings/wetice/2003/1963/00/ 19630302.pdf>. Acesso em: 5 mar. 2004.

COAD, P.; LEFEBVRE, E.; DE LUCA, J. **Java modeling in color with UML:** enterprise components and process. Upper Saddle River: Prentice Hall, 1999.

COCKBURN, A. **Agile software development.** Boston: Addison Wesley, 2002.

COHEN, D.; LINDVALL, M.; COSTA, P. A State of the Art Report: Agile Software development. Rome: DACS, 2003. Disponível em: <http://www.dacs.dtic.mil/ techs/agile/agile.pdf>. Acesso em: 28 out. 2003.

COHN, M.; FORD, D. Introducing an Agile process to an organization. **Computer,** Long Beach, v.36, n.6, p. 74-78, Jun. 2003.

COLANZI, T.E. **Uma abordagem integrada de desenvolvimento e teste de software baseada na UML**. 1999. 143 f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 1999.

COLEMAN, D.; ARNOLD, P.; BODOFF, S.;DOLLIN, C.; GILCHRIST, H.; HAYES, F.; JEREMAES, P. **Object-oriented development**: the fusion method. Englewood Cliffs: Prentice Hall, 1994.

EVANS, G. A simplified approach to RUP. 2003. Disponível em: <http://www-106.ibm.com/developerworks/rational/library/354.html#author1>. Acesso em: 9 out. 2003.

FAGAN, M.E. Design and code inspections to reduce errors in program development. **IBM Systems Journal,** Riverton, v.15, n.7, p.182-211, 1976.

FAGAN, M.E. Advances in software inspections. **IEEE Transactions on software Engineering**, New York, v.12, n.7, p. 744-751, Jul. 1986.

FURLAN, J.D. **Modelagem de objetos através da UML –** The unified modeling language**.** São Paulo: Makron Books, 1998. 329 p.

FUSARO, P.; LANUBILE, F.; VISAGGIO, G. A replicated experiment to assess requirements inspection techniques: **Empirical Software Engineering Journal**, v.2, n.1, p. 39-57, 1997.

GRÜENBACHER, P. Session on Agile Methods. In: EUROMICRO CONFERENCE "NEW WAVES IN SYSTEM ARCHITECTURE, 29., 2003, Belek-Antalya. **Proceedings...** Washington: IEEE Computer Society, 2003. Disponível em: <http://csdl.computer.org/comp/proceedings/euromicro/2003/1996/00/19960258.pdf>. Acesso em: 20 fev. 2004.

HIGHSMITH, J. A. **Adaptive Software Development**: A Collaborative Approach to Managing Complex Systems. New York, NY, Dorset House Publishing, 2000.

HIRSCH, M. Making RUP Agile. In: OBJECT ORIENTED PROGRAMMING SYSTEMS LANGUAGES AND APPLICATIONS - OOPSLA, 2002, Seattle. **Conference on ...** New York: ACM Press, 2002. p.1-8.

JACOBSON, I.; CHRISTERSON, M.; JONSSON, P.; OVERGAARD, G. **Object-oriented software engineering**: a use case driven approach. Wokingham: Addison-Wesley, 1992.

KÄÄRIÄINEN, J.; KOSKELA, J.; TAKALO, J.; ABRAHAMSSON, P., KOLEHMAINEN, K. Supporting requirements engineering in extreme programming: managing user stories. In: JOURNÉES INTERNATIONALES "GÉNIE LOGICIEL & INGÉNIERIE DE SYSTÈMES ET LEURS APPLICATIONS, 16., 2003. **Annales** ... Paris: CNAM-CMSL, 2003. p. 1-8. Disponível em: <http://www.vtt.fi/ele /research/soh/projects/moose/docs/icssea_2003_xp_rm.pdf>. Acesso em: 26 mar. 2004.

KÄHKÖNEN, T.; ABRAHAMSSON, P. Digging into the fundamentals of Extreme Programming: building the theoretical base for Agile methods. In: EUROMICRO CONFERENCE "NEW WAVES IN SYSTEM ARCHITECTURE", 29., 2003, Antalya. **Proceedings…** Washington: IEEE Computer Society, 2003. p. 273-280. Disponível em: <http://csdl.computer.org/dl/proceedings/euromicro/2003/1996/00/19960273.pdf>. Acesso em: 25 abr. 2004.

KRUCHTEN, P. Agility with the RUP. **Cutter IT Journal**, Arlington, v.14, n.12, p. 27-33, Dec. 2001.

KRUTCHEN, P. **The Rational Unified Process**: An Introduction. Adisson-Wesley, Reading, Mass., 1999.

KUTSCHERA, P.; SCHÄFER, S. Applying Agile methods in rapidly changing environments. 2002. Disponível em: <http://jeckstein.com/papers/Agile%20Methods %20-%20Steffen%20Schaefer%20&%20Peter%20Kutschera.pdf>. Acesso em: 22 ago. 2003.

LAITENBERGER, O. A survey of software inspection technologies. In: CHANG, S.K. (Ed.). **Handbook of software engineering and knowledge engineering.** River Edge: World Scientific, 2001. v.2. Disponível em: <ftp://cs.pitt.edu/chang/handbook/ 61b.pdf>. Acesso em: 16 jan. 2003.

LAITENBERGER, O.; ATKINSON, C.; SCHLICH, M.; EL EMAM, K. **An experimental comparison of reading techniques for defect detection in UML design documents**. Ottawa: Institute for Information Technology, National Research Council of Canada, 1999. Disponível em: < http://www.iese.fhg.de/network/ISERN/pub/ technical_reports/isern-00-01.pdf>. Acesso em: 9 abr. 2003.

LEITE, J.C.S.P. Extreme requirements (XR). In: JORNADAS DE INGENIERÍA DE REQUISITOS APLICADA, 2001, Sevilla. **Actas ...** Sevilla: Facultad de Informática y Estadística de Sevilla, 2001. p.1-13. Disponível em: <http://www.lsi.us.es/ ~amador/JIRA/Ponencias/JIRA_Leite.pdf>. Acesso em: 25 mar. 2004.

LEONARDI, M.C.; LEITE, J.C.S.P. Using business rules in EXtreme requirements. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 14., 2002, Toronto. **Proceedings...** Berlin: Springer-Verlag, 2002. p. 420-435. Disponível em: <http://www-di.inf.puc-rio.br/~julio/Slct-pub/carmen-caise.pdf>. Acesso em: 23 mar. 2004.

LEONARDI, M.C.; LEITE, J.C.S.P. Un proceso para XR basado en reglas de negocio. In: WORKSHOP ON REQUIREMENTS ENGINEERING, 4., 2001, Buenos Aires. **Anais...** Buenos Aires: Universidad Tecnológica Nacional, 2001. p. 267-282. Disponível em: <http://www.inf.puc-rio.br/wer01/Eli-Req-2.pdf>. Acesso em: 10 mar. 2004.

MALDONADO, J.C., MARTIMIANO, L.A.F., DÓRIA, E.S., FABBRI, S.C.P. F, MENDONÇA NETO, M.G. Readers Project: replication of experiments: a case study using requirements documents. In: PROTEM INTERNATIONAL COOPERATION PROJECTS EVALUATION WORKSHOP**,** 2001. **Proceedings...** Rio de Janeiro: IME, 2001. Disponível em: <http://www.nuperc.unifacs.br/RT-NUPERC-2001-8e.pdf>. Acesso em: 6 abr. 2004.

MARUCCI, R.A. **Definição de uma estratégia de inspeção para um processo de desenvolvimento de software orientado a objetos.** 2002. 125 f. Dissertação (Mestrado em Ciência da Computação) – Centro de Ciências Exatas e de Tecnologia, Universidade Federal de São Carlos, São Carlos, 2002.

MARUCCI, R.A.; FABBRI, S.C.P.F.; MALDONADO, J.C.; TRAVASSOS, G.H. OORTs/ProDeS: Definição de Técnicas de Leitura para um Processo de Software

Orientado a Objetos. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 1., 2002, Gramado. **Anais...** Gramado: SBC, 2002.

NAWROCKI, J.; JASINSKI, M.; WALTER, B.; WOJCIECHOWSKI, A. Extreme programming modified: embrace requirements engineering practices. In: IEEE JOINT INTERNATIONAL CONFERENCE ON REQUIREMENTS ENGINEERING, 10., 2002, Essen. **Proceedings...** Essen: University of Essen, 2002. Disponível em: <http://www.cs.put.poznan.pl/jnawrocki/publica/re02-essen.doc>. Acesso em: 12 abr. 2004.

OMG. **Unified Modeling Language specification**: version 1.5. Needham: Object Management Group – OMG, 2003. Disponível em: <http://www.omg.org/ docs/formal/03-03-01.pdf>. Acesso em 1 maio 2003.

PAIGE, R.; OSTROFF, J. A proposal for a lightweight rigorous UML – based development method for reliable systems. In: PUML WORKSHOP, GI-Series 7, Lecture Notes in Informatics, 2001. Disponível em: <http://citeseer.nj.nec.com/cache/papers/cs/25140/http:zSzzSzwww-users.cs.york.ac. ukzSz~paigezSzWritingzSzuml3.pdf/paige01proposal.pdf>. Acesso em: 11 set. 2003.

POLLICE, G. Using the RUP for small projects: expanding upon extreme programming. 2003. Disponível em: <http://www-106.ibm.com/developerworks/ rational/library/409.html>. Acesso em: 9 out. 2003.

PRESSMAN, R.S. **Engenharia de software**. 5.ed. Rio de Janeiro: McGraw-Hill, 2002 843p.

QUATRANI, T. **Introduction to the Unified Modeling Language**. Cupertino: Rational Software, 2001. Disponível em: <http://www.rational.com/media/uml/intro_rdn.pdf>. Acesso em: 16 dez. 2002.

RATIONAL SOFTWARE CORPORATION. RUP - Rational Unified Process: Versão 2002.05.00.

ROSENBERG, D.; SCOTT, K. **Applying use case driven object modeling with UML**: an anotated e-commerce example. Boston: Addison-Wesley, 2001.

ROSS, D.T. Structured analysis (SA): a language for comunicating ideas. **IEEE Transactions on software Engineering**, New York, v.3, n.1, p.16-34, 1977.

RUMBAUGH, J. BLAHA, M., PREMERLANI, W., EDDY, F., LORENSON, W. Object-Oriented Modeling and Design. Prentice-Hall, 1992

SANZ, L.F. Presentation: eXtreme Programming. **Upgrade**, v.3, n.2, p. 2-3, Apr. 2002. Disponível em: <http://www.upgrade-cepis.org/issues/2002/2/up3-2Present.pdf>. Acesso em: 23 jan. 2004.

SCHWABER, K.; BEEDLE M., **Agile Software Development with SCRUM.** Prentice-Hall, 2002.

SHULL, F.; BASILI, V. R.; CARVER, J., MALDONADO; J. C., TRAVASSOS, G. H.; MENDONÇA; M.; FABBRI, S.C.P.F. Replicating software engineering experiments: addressing the tacit knowledge problem. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING, 2002. **Proceedings....** Disponível em: <http://www.cebase.org:444/fc-md/proposals/isese2002/isese2002.pdf>. Acesso em: 2 maio 2003.

SILVA, D.M. **UML:** guia de consulta rápida. São Paulo: Novatec, 2001.

SOMMERVILLE, I. **Engenharia de software.** 6.ed. São Paulo: Addison Wesley, 2003.

STAPLETON, J., **DSDM: The Method in Practice.** Addison Wesley Longman, 1997.

TEKINERDOGAN, B. Formalizing agile software development methods. In: IMPACT OF SOFTWARE PROCESS ON QUALITY, 2003, Ankara. **Workshop...** Ankara: Bilkent University, 2003. Disponível em: <http://www.cs.bilkent.edu.tr/ improq03/Papers/Tekinerdogan.pdf>. Acesso em: 25 jun. 2004.

TOMA, K.M. **Pacotes de laboratório das técnicas de leitura para o ProDeS/UML|ag**. São Carlos: UFSCar, 2004. (Documento de Trabalho)

TRAVASSOS, G.H.; GUROV, D.; AMARAL, E.A.G. **Introdução à engenharia de software experimental.** Rio de Janeiro: COPPE/UFRJ, 2002a. 52 p. Relatório técnico RT-ES-590/02. Disponível em: <http://www.cos.ufrj.br/publicacoes/reltec/es59002.pdf >. Acesso em: 15 dez. 2002

TRAVASSOS, G.H.; SHULL, F.; CARVER, J. A family of reading techniques for OO design inspections. In: WORKSHOP DE QUALIDADE DE SOFTWARE (WQS2000); BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 14., 2000, João Pessoa. **Proceedings...** Disponível em: <http://www.cs.umd.edu/projects/SoftEng/ESEG/ papers/pdf/wqs00.pdf>. Acesso em 9 abr. 2003.

TRAVASSOS, G.H.; SHULL, F.; CARVER, J. Working with UML: a software design process based on inspections for the Unified Modeling Language. **Advances in Computers**, San Diego, v.54, p.35-98, 2001. Disponível em: <http://www.cos.ufrj.br/ ~ght/umldraft.pdf>. Acesso em: 12 dez. 2002.

TRAVASSOS, G.H.; SHULL, F.; CARVER, J.; BASILI, V.R. **Reading techniques for OO design inspections**. Rio de Janeiro: COPPE/UFRJ, 2002b. 56p. (Technical Report 575/02). Disponível em: <http://www.cos.ufrj.br/publicacoes/reltec/es57502.pdf>. Acesso em: 28 nov. 2002.

TRAVASSOS, G.H.; SHULL, F.; CARVER, J.; BASILI, V.R. **Reading techniques for OO design inspections**. 2002c. 10p. Disponível em: <http://sel.gsfc.nasa.gov/website/ sew/1999/topics/travassos_SEW99paper.pdf>. Acesso em: 28 nov. 2002.

TUFFS, D.; STAPLETON, J.; WEST, D.; EASON, Z. Inter-operability of DSDM with the Rational Unified Process. 1999. p. 1-29 Disponível em: <http://www.agilealliance.org/articles/articles/DSDM_and_RUP.pdf>. Acesso em: 22 ago. 2003.

VISAGGIO, G. **Knowledge transfer through experience packages**. Chicago: International Software Engineering Research Network, 2002. (Technical Reports 2002 - ISERN-02-02). Disponível em: <http:www.iese.fhg.de/network/ISERN/pub/technical_reports/isern-02-04.pdf>. Acesso em 12 dez. 2002.

WELLS, D. Extreme programming: a gentle introduction. 2003. Disponível em: <http://www.extremeprogramming.org>. Acesso em: 15 abr. 2004.

WILLIAMS, L.; COCKBURN, A. Guest editors' introduction: Agile software development: It's about feeback and change. **IEEE Computer Society,** Long Beach, v.36, n.6, p.39-43, Jun. 2003.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering:** an introduction. Boston: Kluwer Academic Publishers, 2000. 204 p.

ZELKOWITZ, M.V.; WALLACE, D**.** Experimental validation in software engineering**. Information and Software Technology**, Amsterdam, v. 39, n.11, p. 735-743, Nov. 1997.

APÊNDICE A

Pacote de Laboratório ER1

- Questionário de Caracterização
- Técnica de Leitura ER1
- Treinamento ER1
- Documento de Requisitos
- Diagrama de Casos de Uso
- Especificações dos Casos de Uso
- Relatório de Discrepâncias

Caracterização do Participante

ID do Revisor: _____ Data: ___/___/___

Este formulário possui algumas questões sobre seu conhecimento e experiência

Conhecimento Geral

1. Qual é sua experiência anterior com desenvolvimento de software na prática? (Marque o item mais apropriado)

Eu nunca desenvolvi software.	Eu desenvolvi software por minha própria conta.	Eu desenvolvi software como parte de uma equipe, como parte de um curso.	Eu desenvolvi software como parte de uma equipe, na indústria.

Quantos anos / meses de experiência você tem nesta prática? _____

Experiência em Desenvolvimento de Software

Experiência com Requisitos

2. Qual é a sua experiência em escrever requisitos?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

3. Qual é a sua experiência na elaboração Casos de Uso?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

4. Qual é a sua experiência em revisão de requisitos?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

5. Qual é a sua experiência em revisão de Casos de Uso?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

6. Qual é a sua experiência em alterar requisitos como decorrência de manutenção?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

Experiência em Projeto

7. Qual é a sua experiência em projeto de sistemas?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

( ) Análise Estruturada: _____ anos / meses.
( ) Orientado a Objeto: _____ anos / meses.
( ) Outro: _____

8. Qual é a sua experiência em alterar projetos como decorrência de manutenção?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

( ) Análise Estruturada: _____ anos / meses.
( ) Orientado a Objeto: _____ anos / meses.
( ) Outro: _____

9. Qual é a sua experiência na elaboração de documentos de projetos?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

( ) Análise Estruturada: _____ anos / meses.
( ) Orientado a Objeto: _____ anos / meses.
( ) Outro: _____

10. Qual é a sua experiência em ler (avaliar, inspecionar) documentos de projetos?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

( ) Análise Estruturada: _____ anos / meses.
( ) Orientado a Objeto: _____ anos / meses.
( ) Outro: _____

11. Qual é a sua experiência com Unified Modeling Language (UML)?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

Quantos anos / meses de experiência você tem com a UML?_____

12. Qual é a sua experiência em elaborar projeto de sistemas a partir de Requisitos / Caso de Uso?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

( ) A partir de requisitos: _____ anos / meses.
( ) A partir de Caso de Uso: _____ anos / meses.

Outras Experiências

13. Qual é a sua experiência com inspeção de software?

Nenhuma	Estudou em Aula ou em livro	Praticou em um projeto de aula	Usou em um projeto na Indústria	Usou em vários projetos na Indústria

Quantos anos / meses de experiência você tem? _____

Sobre esta atividade

14. Quanto você espera "ganhar" (em termos de aprendizado) com esta atividade?

Nada	Pouco	Moderado	Bastante

Comentários:

Técnica de Leitura ER1

Leitura ER1 - Diagrama de Casos de Uso x Especificação dos Casos de Uso x Documento de Requisitos

Objetivo: Verificar se os conceitos e funcionalidades que estão descritos no Documento de Requisitos foram capturados apropriadamente pelo Diagrama de Casos de Uso e se as Especificações dos Casos de Uso estão descritas de forma precisa.

Entradas para o processo:
1. Um conjunto de Requisitos Funcionais e Não Funcionais.
2. Um Diagrama de Casos de Uso.
3. As Especificações dos Casos de Uso.

I. Leia o Documento de Requisitos para entender a funcionalidade descrita.

ENTRADAS: Um conjunto de Requisitos Funcionais (RFs) e Não Funcional (RNFs).

SAÍDAS: Substantivos candidatos a atores, entidades que serão mantidas (ou geradas) no sistema ou atributos das entidades (marcados com um traço nos RFs);
Verbos ou descrições de ações candidatos a funcionalidades do sistema (marcados com um círculo nos RFs);
Restrições ou condições associadas aos substantivos ou verbos (marcadas com um retângulo nos RNFs).

Para cada Requisito Funcional do Documento de Requisitos faça:

A. Leia o Requisito para entender a funcionalidade que ele descreve.

B. Encontre os substantivos que constam desse Requisito e que sejam candidatos a tornarem-se atores ou entidades que serão mantidas (ou geradas) no sistema, ou atributos dessas entidades. Sublinhe com um traço esses substantivos.

C. Encontre os verbos ou descrições de ações, os quais sejam candidatos a funcionalidades realizadas pelo sistema. Marque esses verbos ou descrições de ações com um círculo.

D. Procure descrições de restrições ou condições relacionadas aos substantivos e verbos que você identificou nos dois passos anteriores. Especialmente preste atenção para os requisitos não funcionais, que tipicamente contêm restrições e condições sobre a funcionalidade do sistema. Verifique se há limitações ou restrições explícitas na forma como as ações são executadas. Tente observar se foram especificadas quantidades bem definidas em qualquer ponto dos requisitos. Marque essas condições e restrições com um retângulo.

II. Identifique a funcionalidade descrita pela Especificação do Caso de Uso e os conceitos importantes do sistema que são necessários para alcançar essa funcionalidade.

ENTRADAS: Diagrama de Casos de Uso (DCU);
Especificação dos Casos de Uso (ECU).

SAÍDAS: Substantivos candidatos a atores, entidades que serão mantidas (ou geradas) no sistema ou atributos das entidades (marcados com um traço na ECU);
Verbos ou descrições de ações candidatos a funcionalidades do sistema (marcados com um círculo na ECU);
Restrições ou condições associadas aos substantivos ou verbos (marcadas com um retângulo na ECU);
Relatório de Discrepâncias.

Para cada caso de uso do Diagrama de Casos de Uso faça:

A. Encontre a Especificação do Caso de Uso correspondente.

1. **Caso ela não seja encontrada, então existe um caso de uso no Diagrama de Casos de Uso que não foi especificado. Relate esse fato no Relatório de Discrepâncias.**

B. Leia a Especificação do Caso de Uso identificada no passo anterior para entender a funcionalidade que ela descreve.

C. Encontre os substantivos que aparecem na Especificação do Caso de Uso; eles descrevem os atores ou as entidades que serão mantidas (ou geradas) no sistema ou os atributos dessas entidades. Sublinhe com um traço cada substantivo à medida que for encontrado.

D. Para cada substantivo identifique os verbos ou descrições de ações que descrevem as funcionalidades do sistema e marque-as com um círculo.

E. Procure restrições ou condições que sejam necessárias para que as funcionalidades identificadas no passo anterior, possam ser alcançadas. Marque essas condições e restrições com um retângulo.

III. Compare os documentos para verificar se os conceitos descritos no Documento de Requisitos foram corretamente capturados pelo Diagrama e Especificação dos Casos de Uso.

ENTRADAS: Diagrama de Casos de Uso (DCU);
Especificação dos Casos de Uso com substantivos, verbos ou descrições de ações, restrições ou condições marcados;
Requisitos Funcionais e Não Funcionais com substantivos, verbos ou descrições de ações, restrições ou condições marcados.

SAÍDAS: Relatório de Discrepâncias.

Para cada Requisito Funcional do Documento de Requisitos, que ainda não esteja rotulado com um "número" de alguma Especificação de Caso de Uso, com um (*) ou com um (X), faça:

A. Leia o Requisito Funcional e verifique se ele está apenas mencionando as principais funcionalidades do sistema.

1. **Caso esteja, marque-o com um (X) e preencha o Relatório de Discrepâncias, pois esse tipo de informação deve estar em uma seção do Documento de Requisitos que descreve o Sistema de forma mais genérica. Nesse caso, os passos subseqüentes não devem ser realizados.**

B. Localize no Diagrama de Casos de Uso o caso de uso que está tratando desse Requisito Funcional e em seguida separe a respectiva Especificação do Caso de Uso.

1. **Caso você não encontre o caso de uso que esteja tratando desse Requisito Funcional, preencha o Relatório de Discrepâncias, pois alguma funcionalidade foi omitida do Diagrama e das Especificações dos Casos de Uso. Marque esse Requisito Funcional com um (*).**

C. Caso encontre o Caso de Uso, leia a Especificação selecionada no item anterior e verifique se ela trata de outras funcionalidades relacionadas àquela que consta do Requisito Funcional selecionado no passo A. Caso ela trate, identifique os Requisitos Funcionais associados com elas e a partir deste ponto, considere esse Conjunto de Requisitos Funcionais (CRF) para utilizar nos passos seguintes. Marque esses Requisitos Funcionais com o número correspondente à Especificação em questão.

1. **Certifique-se de que cada Requisito Funcional esteja descrevendo a funcionalidade do caso de uso selecionado no passo B. Se não estiver, preencha o Relatório de Discrepâncias descrevendo o problema.**

C1. Para cada substantivo marcado com um traço nos Requisitos Funcionais, verifique se existe esse mesmo substantivo, também marcado com um traço, na Especificação dos Casos de Uso. Caso o termo utilizado não seja o mesmo, usando seu conhecimento sobre o domínio da aplicação, verifique se algum outro substantivo pode corresponder a esse conceito. Se os conceitos entre os documentos forem correspondentes, faça uma marcação tanto nos Requisitos Funcionais como na Especificação dos Casos de Uso com um (*).

1. **Certifique-se de que não existem substantivos nos Requisitos Funcionais que não foram marcados com (*). Caso isso ocorra, significa que um conceito que consta nos Requisitos, não está representado na Especificação dos Casos de Uso. Verifique se esse conceito é necessário nessa fase do**

desenvolvimento e se corresponde a um nível de detalhe que é necessário nesse momento. Se a representação desse conceito não é relevante nesse momento, então certifique-se de que esse conceito é importante considerando o domínio da aplicação. Se necessário, preencha o Relatório de Discrepâncias descrevendo o problema.

2. **Certifique-se de que não existem substantivos na Especificação dos Casos de Uso que não foram marcados com (*). Caso isso ocorra, significa que existem conceitos na Especificação dos Casos de Uso que não estão nos Requisitos. Usando seu conhecimento sobre o domínio da aplicação, verifique se essa informação é estranha ou se um fato incorreto que está presente nos Requisitos Funcionais pode ter sido responsável por essa incorreção entre os documentos. Preencha o Relatório de Discrepâncias descrevendo essa situação.**
3. **Se, com base na sua interpretação da funcionalidade realizada pela Especificação dos Casos de Uso e o conjunto de Requisitos Funcionais associado, você não puder concluir quais atributos são necessários para que ela ocorra, relate o fato no Relatório de Discrepâncias.**

C2. Para cada verbo ou descrição de ação marcado com um círculo nos Requisitos Funcionais, verifique se existe esse mesmo verbo ou descrição de ação, também marcado com um círculo, na Especificação dos Casos de Uso. Caso o termo utilizado não seja o mesmo, usando seu conhecimento sobre o domínio da aplicação, verifique se algum outro verbo ou descrição de ação pode corresponder a esse verbo ou descrição de ação. Se os verbos ou descrições de ações entre os documentos forem correspondentes, faça uma marcação tanto nos Requisitos Funcionais como na Especificação do Caso de Uso com um (*).

1. **Certifique-se de que não existem verbos ou descrições de ações nos Requisitos Funcionais que não foram marcados com (*). Caso isso ocorra, significa que existem verbos ou descrições de ações que foram usados para descrever uma funcionalidade nos Requisitos, mas não foram descritos na Especificação dos Casos de Uso. Verifique se essa informação é necessária para descrever a funcionalidade nessa fase do desenvolvimento e se corresponde a um nível de detalhe que é necessário nesse momento. Se a representação dessa informação não é relevante nesse momento para a execução da funcionalidade, então certifique-se de que essa informação é importante considerando o domínio da aplicação. Se necessário, preencha o Relatório de Discrepâncias descrevendo o problema.**
2. **Certifique-se de que não existem verbos ou descrições de ações na Especificação dos Casos de Uso que não foram marcadas com (*). Caso isso ocorra, significa que existem verbos ou descrições de ações que foram usados para descrever uma funcionalidade na Especificação dos Casos de Uso que não estão nos Requisitos Funcionais. Usando seu conhecimento sobre o domínio da aplicação, verifique se essa informação é estranha ou se um fato incorreto que está presente nos Requisitos Funcionais pode ter sido responsável por essa incorreção entre os documentos. Preencha o Relatório de Discrepâncias descrevendo essa situação.**

C3. Para cada restrição ou condição marcada com um retângulo nos Requisitos Funcionais, verifique se existe essa mesma restrição ou condição, também marcada com um retângulo, na Especificação dos Casos de Uso. Caso o termo utilizado não seja o mesmo, usando seu conhecimento sobre o domínio da aplicação, verifique se alguma outra restrição ou condição pode corresponder a essa restrição ou condição. Se as restrições ou condições entre os documentos forem correspondentes, faça uma marcação tanto nos Requisitos Funcionais como na Especificação dos Casos de Uso com um (*).

1. **Certifique-se de que não existem condições ou restrições nos Requisitos Funcionais que não foram marcadas com (*). Caso isso ocorra, significa que elas foram descritas nos Requisitos Funcionais, mas não estão representadas na Especificação dos Casos de Uso. Verifique se a representação delas é necessária para descrever a funcionalidade nessa fase do desenvolvimento e se corresponde a um nível de detalhe que é necessário nesse momento. Se a representação dessas restrições ou condições não é necessária nesse momento, então certifique-se de que elas são importantes considerando o domínio da aplicação. Se necessário, preencha o Relatório de Discrepâncias descrevendo o problema.**
2. **Certifique-se de que não existem condições ou restrições na Especificação dos Casos de Uso que não foram marcadas com (*). Caso isso ocorra, significa que elas estão representadas na Especificação dos Casos de Uso, mas não foram descritas nos Requisitos. Usando seu conhecimento sobre o domínio da aplicação, verifique se essa restrição ou condição é estranha ou se um fato incorreto que está presente nos requisitos funcionais pode ter sido responsável por essa incorreção entre os documentos. Preencha o Relatório de Discrepâncias descrevendo essa situação.**

D. Concluídos os passos A, B e C, marque no Diagrama de Casos de Uso o caso de uso selecionado no passo B com o "número" da respectiva Especificação de Caso de Uso.

IV. Considerando as marcações realizadas anteriormente, verifique a seguinte situação:

ENTRADAS: Especificações relacionadas aos Casos de Uso que não foram marcadas com um número;
Requisitos Funcionais e Não Funcionais marcados com um (X).

SAÍDAS: Relatório de Discrepâncias.

Para cada Requisito Funcional do Documento de Requisitos marcado somente com um (X) faça:

A. Leia a(s) Especificação(ões) do(s) Caso(s) de Uso relacionadas aos casos de uso não marcado(s) com um número no Diagrama de Casos de Uso e analise se alguma dessa(s) Especificação(ões) corresponde ao Requisito Funcional.

A1. Se existir alguma Especificação que se relaciona com o Requisito Funcional, marque esse Requisito com o número correspondente à Especificação em questão e execute os passos C1, C2, C3 e D da etapa III.

V. Considerando as marcações realizadas anteriormente, verifique a seguinte discrepância:

ENTRADAS: Casos de Uso do Diagrama de Casos de Uso não marcados com um número.

SAÍDAS: Relatório de Discrepâncias.

A. Caso exista algum caso de uso no Diagrama de Casos de Uso não marcado com um número, preencha o Relatório de Discrepâncias, pois a funcionalidade desse Caso de Uso não está representada no Documento de Requisitos.

Treinamento ER1

Treinamento

Leitura ER1 - Diagrama de Casos de Uso x Especificação dos Casos de Uso x Documento de Requisitos

Definição dos termos utilizados na Leitura ER1

- **Funcionalidade:** descreve o comportamento do sistema. Tipicamente, a funcionalidade é descrita sob o ponto de vista do usuário.
- **Condição:** descreve aquilo que deve ser verdadeiro para que a funcionalidade possa ser executada.
- **Restrição:** deve ser sempre verdadeira para qualquer funcionalidade do sistema.

Exemplo: Condições e Restrições

Freqüentemente os requisitos descrevem *condições* e *restrições* sobre a funcionalidade.

- O sistema deve, no caso de ocorrer a tentativa de inserção de um *item bibliográfico* já existente, comunicar ao *pesquisador* a existência deste *item bibliográfico* na *bibliografia*. Se, neste caso, o *pesquisador* confirmar a operação de inserção, o sistema deve informar que tal operação irá alterar o *item bibliográfico* existente.
- O tempo de processamento de uma operação de consulta não deve exceder três segundos para uma quantidade inferior a 10 *itens bibliográficos*.

Exemplo de Projeto: Sistema de Apoio à Escrita (SAPES)

- **Descrição do sistema:**
  - "O SAPES tem como objetivo principal auxiliar a pesquisa bibliográfica. Os *usuários* deste sistema são, principalmente, *pesquisadores* que durante a sua pesquisa bibliográfica podem ler *publicações* (por exemplo: *artigos*, livros e periódicos) e armazená-las no sistema através de *itens bibliográficos*, construindo, assim, a sua *bibliografia* pessoal. Esta *bibliografia* pode ser alterada e consultada conforme a necessidade do *pesquisador*, além da possibilidade de fornecer diferentes tipos de relatório. O *pesquisador* pode também utilizar o sistema durante a redação de textos científicos. Através do *documento* produzido pelo *pesquisador*, o sistema reconhece as *citações* e gera automaticamente a *referência bibliográfica*."

Leitura ER1 - Diagrama de Casos de Uso x Especificação dos Casos de Uso x Documento de Requisitos

- **Os artefatos:**
  - Um *conjunto de Requisitos Funcionais e Não Funcionais (RFs e RNFs)*.
  - Um *Diagrama de Casos de Uso (DCU)*.
  - As *Especificações dos Casos de Uso (ECU)*.
- **O objetivo:**
  - Verificar se os conceitos e funcionalidades que estão descritos pelo Documento de Requisitos foram capturados de forma apropriada pelo Diagrama de Casos de Uso e se as Especificações dos Casos de Uso estão descritas de forma precisa.

Etapa I – Ler o Documento de Requisitos para entender a funcionalidade descrita

- **Entradas:** Um conjunto de Requisitos Funcionais (RFs) e Não Funcionais (RNFs).
- **Saídas:** Substantivos (atores ou entidades ou atributos); Verbos ou descrições de ações (funcionalidades do sistema); Restrições ou condições.
- **Instruções Gerais:** Leia o Documento de Requisitos para entender a funcionalidade descrita.
- **Instruções Específicas:**

  Para cada Requisito Funcional do Documento de Requisitos faça:

  A. Leia o Requisito para entender a funcionalidade
  B. Encontre os substantivos que constam nesse Requisito (atores ou entidades ou atributos) – marcar com um traço;
  C. Encontre os verbos ou descrições de ações (funcionalidades) – marcar com um círculo;
  D. Procure descrições de restrições ou condições – marcar com um retângulo.

Exemplo: Requisitos Funcionais do SAPES

RF1. O sistema deve permitir a inserção, alteração e exclusão de *itens bibliográficos*, mantendo uma *bibliografia*.

:

:

RF3. O sistema deve fornecer mensagens de erro quando *itens bibliográficos* incompletos forem inseridos. Tais mensagens interrogam o *pesquisador* se deseja cancelar a operação de inserção, completar as informações incompletas ou concluir a inserção assim mesmo.

RF4. O sistema deve, no caso de ocorrer a tentativa de inserção de um *item bibliográfico* já existente, comunicar ao *pesquisador* a existência deste *item bibliográfico* na *bibliografia*. Se, neste caso, o *pesquisador* confirmar a operação de inserção, o sistema deve informar que tal operação irá alterar o *item bibliográfico* existente.

Exemplo: Requisitos Funcionais (Substantivos, Verbos e condições ou restrições marcados)

RF1. O sistema deve permitir a inserção, alteração e exclusão de *itens bibliográficos*, mantendo uma *bibliografia*.

:

:

RF3. O sistema deve fornecer mensagens de erro quando *itens bibliográficos* incompletos forem inseridos. Tais mensagens interrogam o *pesquisador* se deseja cancelar a operação de inserção, completar as informações incompletas ou concluir a inserção assim mesmo.

RF4. O sistema deve, no caso de ocorrer a tentativa de inserção de um *item bibliográfico* já existente, comunicar ao *pesquisador* a existência deste *item bibliográfico* na *bibliografia*. Se, neste caso, o *pesquisador* confirmar a operação de inserção, o sistema deve informar que tal operação irá alterar o *item bibliográfico* existente.

Etapa II – Identificar a funcionalidade descrita pela Especificação do Caso de Uso e os seus conceitos importantes

- **Entradas:** Diagrama de Casos de Uso; Especificação dos Casos de Uso.
- **Saídas:** Substantivos (atores ou entidades ou atributos); Verbos ou descrições de ações (funcionalidades); Restrições ou condições; Relatório de Discrepâncias.
- **Instruções Gerais:** Identifique a funcionalidade descrita pela Especificação do Caso de Uso e os conceitos importantes do sistema que são necessários para alcançar essa funcionalidade.
- **Instruções Específicas:**

Para cada caso de uso do DCU faça:

A. Encontre a Especificação do Caso de Uso correspondente;

Se não encontrar => Relatório de Discrepâncias.

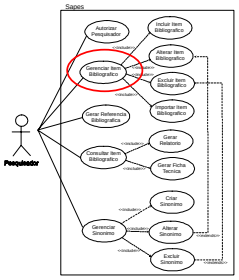
B. Leia a ECU para entender a funcionalidade que ela descreve;

C. Encontre os substantivos (atores ou entidades ou atributos) – marque com um traço;

D. Para cada substantivo identifique os verbos ou descrições de ações (funcionalidades) – marque com um círculo;

E. Procure restrições ou condições – marque com um retângulo.

Diagrama de Casos de Uso



Exemplo: Diagrama e Especificação dos Casos de Uso

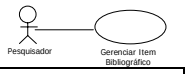
No. Caso de Uso:	**2**
Caso de Uso:	**Gerenciar Item Bibliográfico**
Atores:	Pesquisador
Propósito:	Gerenciar os itens bibliográficos que são utilizados pelo pesquisador.
Descrição:	O pesquisador pode inserir, alterar, excluir ou importar itens bibliográficos.
Tipo:	Primário e Essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O pesquisador determina qual ação ele deseja executar: inserir, excluir, alterar ou importar um item bibliográfico.	2. Se o pesquisador escolher *inserir*, o caso de uso 3 será ativado. Se o pesquisador escolher *alterar*, o caso de uso 4 será ativado. Se o pesquisador escolher *excluir*, o caso de uso 5 será ativado. Se o pesquisador escolher *importar*, o caso de uso 6 será ativado.

Curso Alternativo

Exemplo: Especificação dos Casos de Uso (Substantivos, verbos e condições ou restrições marcados)



No. Caso de Uso:	**2**
Caso de Uso:	**Gerenciar Item Bibliográfico**
Atores:	Pesquisador
Propósito:	Gerenciar os itens bibliográficos que são utilizados pelo pesquisador.
Descrição:	O pesquisador pode inserir, alterar, excluir ou importar itens bibliográficos.
Tipo:	Primário e Essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O pesquisador determina qual ação ele deseja executar: inserir, excluir, alterar ou importar um item bibliográfico.	2. Se o pesquisador escolher *inserir*, o caso de uso 3 será ativado. Se o pesquisador escolher *alterar*, o caso de uso 4 será ativado. Se o pesquisador escolher *excluir*, o caso de uso 5 será ativado. Se o pesquisador escolher *importar*, o caso de uso 6 será ativado.

Etapa III – Comparar os conceitos do Documento de Requisitos com o Diagrama e Especificação dos Casos de Uso

- **Entradas:** Diagrama de Casos de Uso;Especificação dos Casos de Uso com conceitos marcados; Requisitos Funcionais com conceitos marcados.
- **Saídas:** Relatório de Discrepâncias.
- **Instruções Gerais:** Compare ambos os documentos para verificar se os conceitos descritos no Documento de Requisitos foram corretamente capturados pelo Diagrama e Especificação dos Casos de Uso.
- **Instruções Específicas:**

Para cada RF, ainda não marcado com um número, com um (*) ou um (X):

A. Leia o RF e verifique se ele está apenas mencionando as principais funcionalidades.

1. **Caso esteja, marque-o com um (X) e preencha o Relatório de Discrepâncias. Nesse caso os passos subseqüentes não devem ser realizados.**

B. Localize no DCU o caso de uso que está tratando desse RF e em seguida separe a respectiva ECU.

1. **Caso você não encontre o caso de uso que esteja tratando desse RF, preencha o Relatório de Discrepâncias, pois a funcionalidade foi omitida do Diagrama e das Especificações dos Casos de Uso. Marque esse RF com um (*).**

Documento de Requisitos

RF1. O sistema deve permitir a inserção, alteração e exclusão de *itens bibliográficos*, mantendo uma *bibliografia*.

X

:

:

RF3. O sistema deve fornecer mensagens de erro quando *itens bibliográficos* incompletos forem inseridos. Tais mensagens interrogam o *pesquisador* se deseja cancelar a operação de inserção, completar as informações incompletas ou concluir a inserção assim mesmo.

RF4. O sistema deve, no caso de ocorrer a tentativa de inserção de um *item bibliográfico* já existente, comunicar ao *pesquisador* a existência deste *item bibliográfico* na *bibliografia*. Se, neste caso, o *pesquisador* confirmar a operação de inserção, o sistema deve informar que tal operação irá alterar o *item bibliográfico* existente.

RF8. O sistema deve permitir a exclusão de um *item bibliográfico* se esse item existe na *bibliografia*.

5 **O *pesquisador* pode acessar/recuperar um *item bibliográfico* a ser excluído pelos *itens de informação*: autor, título e pelos *sinônimos* de autor e título, respectivamente.**

:

:

Nº. Caso de Uso:	**5**
Caso de Uso:	**Excluir Item Bibliográfico**
Atores:	Pesquisador
Propósito:	Excluir um item bibliográfico da bibliografia.
Descrição:	O pesquisador fornece informação sobre o item bibliográfico que deseja excluir e este é excluído da bibliografia.
Tipo:	Primário e essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O pesquisador recupera o item bibliográfico por meio dos itens de informação: autor ou sinônimo de autor e título ou sinônimo de título.	**2.** O sistema mostra os itens de informação do item bibliográfico e pede que o pesquisador confirme a exclusão apresentando a seguinte mensagem: *"Confirma a exclusão do item bibliográfico?"*
3. O pesquisador confirma a exclusão.	**4.** O item bibliográfico é excluído da bibliografia e a seguinte mensagem é mostrada: *"Item bibliográfico excluído."*

Curso Alternativo 1

2. O item bibliográfico não é encontrado na bibliografia, então a mensagem *("Item bibliográfico não encontrado")* é mostrada ao pesquisador e a exclusão é cancelada.

Curso Alternativo 2

3. O pesquisador não confirma a exclusão do item bibliográfico.

4. A exclusão não é efetuada.

Etapa III (Cont.)

C. Caso encontre o caso de uso, leia a Especificação selecionada no item anterior e verifique se ela não trata de outras funcionalidades relacionadas àquela que consta do RF selecionado no passo A. Caso ela trate, identifique os RF associados com elas e a partir deste ponto, considere esse Conjunto de Requisitos Funcionais (CRF) para utilizar nos passos seguintes.

1. **Certifique-se de que cada Requisito Funcional esteja descrevendo a funcionalidade do Caso de Uso selecionado no passo B. Se não estiver, preencha o Relatório de Discrepâncias descrevendo o problema.**

Etapa III (cont.)

C1. Verifique a correspondência entre os substantivos marcados nos Requisitos e na Especificação dos Casos de Uso. Faça uma marcação nos dois documentos com (*).

- **substantivos sem estarem marcados:**
  - **Requisitos – (conceito relevante; corresponde a um nível de detalhe que é necessário no momento) => Relatório de discrepâncias**
  - **Especificação dos Casos de Uso - (informação é estranha ou fato incorreto) => Relatório de discrepâncias**

C2. Verifique correspondência entre os verbos ou descrições de ações marcadas nos Requisitos Funcionais e na Especificação dos Casos de Uso. Faça uma marcação nos dois documentos com (*).

- **verbos ou descrições de ações sem estarem marcados:**
  - **Requisitos => Relatório de discrepâncias**
  - **Especificação dos Casos de Uso => Relatório de discrepâncias**

Etapa III (cont.)

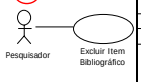
C3. Verifique a correspondência entre as restrições ou condições marcadas nos Requisitos Funcionais e na Especificação dos Casos de Uso. Faça uma marcação nos dois documentos com (*).

- **restrições ou condições sem estarem marcadas:**
  - **Requisitos - (conceito relevante; corresponde a um nível de detalhe que é necessário no momento) => Relatório de discrepâncias**
  - **Especificação dos Casos de Uso - (informação é estranha ou fato incorreto) => Relatório de discrepâncias**

D. Concluídos os passos A,B e C, marque no DCU o caso de uso selecionado no passo B com o "número" da respectiva ECU.

RF8. O sistema deve permitir a exclusão de um *item bibliográfico* se esse item existe na *bibliografia*.
O *pesquisador* pode acessar / recuperar um *item bibliográfico* a ser excluído pelos *itens de informação*: autor, título e pelos *sinônimos* de autor e título, respectivamente.

5



N°. Caso de Uso:	**5**
Caso de Uso:	**Excluir Item Bibliográfico**
Atores:	Pesquisador
Propósito:	Excluir um item bibliográfico da bibliografia.
Descrição:	O pesquisador fornece informação sobre o item bibliográfico que deseja excluir e este é excluído da bibliografia.
Tipo:	Primário e essencial
Curso Típico dos Eventos	
Ação do Ator	**Resposta do Sistema**
1. O pesquisador recupera o item bibliográfico por meio dos itens de informação: autor ou sinônimo de autor e título ou sinônimo de título.	**2.** O sistema mostra os itens de informação do item bibliográfico e pede que o pesquisador confirme a exclusão apresentando a seguinte mensagem: *"Confirma a exclusão do item bibliográfico?"*
3. O pesquisador confirma a exclusão.	**4.** O item bibliográfico é excluído da bibliografia e a seguinte mensagem é mostrada: *"Item bibliográfico excluído."*
Curso Alternativo 1 **2.** O item bibliográfico não é encontrado na bibliografia, então a mensagem (*"Item bibliográfico não encontrado"*) é mostrada ao pesquisador e a exclusão é cancelada. **Curso Alternativo 2** **3.** O pesquisador não confirma a exclusão do item bibliográfico. **4.** A exclusão não é efetuada.	

Etapa IV – Considerando as marcações realizadas anteriormente, verifique a seguinte situação:

- **Entradas:** Especificações relacionadas aos Casos de Uso que não foram marcadas com um número; Requisitos Funcionais e Não Funcionais marcados com um (X).
- **Saídas:** Relatório de Discrepâncias.
- **Instruções Específicas:**
  Para cada RF do Documento de Requisitos marcado somente com um (X) faça:
  A. Leia a(s) Especificações dos Casos de Uso relacionadas aos casos de uso não marcado(s) com um número no DCU e analise se alguma dessa(s) Especificação(ões) corresponde ao RF.
  1. **Se existir alguma Especificação que se relaciona com o RF, marque esse Requisito com o número correspondente à Especificação em questão e execute os passos C1, C2, C3 e D da etapa III.**

No. Caso de Uso:	**2**
Caso de Uso:	**Gerenciar Item Bibliográfico**
Atores:	Pesquisador
Propósito:	Gerencia os itens bibliográficos que são utilizados pelo pesquisador.
Descrição:	O pesquisador pode inserir, alterar, excluir ou importar itens bibliográficos.
Tipo:	Primário e Essencial
Curso Típico dos Eventos	
Ação do Ator	**Resposta do Sistema**
1. O pesquisador determina qual ação ele deseja executar (insere, exclui, altera ou importa um item bibliográfico.	2. Se o pesquisador escolher *inserir*, o caso de uso 3 será ativado. Se o pesquisador escolher *alterar*, o caso de uso 4 será ativado. Se o pesquisador escolher *excluir*, o caso de uso 5 será ativado. Se o pesquisador escolher *importar*, o caso de uso 6 será ativado.

RF1. O sistema deve permitir a inserção, alteração e exclusão de *itens bibliográficos*, mantendo uma *bibliografia*. X 2

RF3. O sistema deve fornecer mensagens de erro quando *itens bibliográficos* incompletos forem inseridos. Tais mensagens interrogam o *pesquisador* se deseja cancelar a operação de inserção, completar as informações incompletas ou concluir a inserção assim mesmo.

RF4. O sistema deve, no caso de ocorrer a tentativa de inserção de um *item bibliográfico* já existente, comunicar ao *pesquisador* a existência deste *item bibliográfico* na *bibliografia*. Se, neste caso, o *pesquisador* confirmar a operação de inserção, o sistema deve informar que tal operação irá alterar o *item bibliográfico* existente.

Etapa V – Considerando as marcações realizadas anteriormente, verifique a seguinte discrepância:

- **Entradas:** Casos de Uso do DCU não marcados com um número.
- **Saídas:** Relatório de Discrepâncias.
- **Instruções Específicas:**
  Caso exista algum caso de uso no DCU não marcado com um número, preencha o Relatório de Discrepâncias, pois a funcionalidade desse Caso de Uso não está representada no Documento de Requisitos.

Tipo de Discrepância

Tipos de Disc.	Descrição Geral
1	**Funcionalidade ou conceito necessário foi omitido.**
2	**O MCU está incorreto em relação aos requisitos.**
3	**Como o MCU implementa estes requisitos é ambíguo ou não completamente especificado.**
4	**A informação do MCU é estranha, i.e. não mencionada pelos requisitos.**
5	**Outros tipos de problema.**

MCU = Modelo de Caso de Uso = DCU + ECU

Formulário de Relato de Discrepâncias –ER1

Início da Inspeção: _____ (Hora) Data:_____ ID do revisor:_____

Tipo de Conceito:

(AC) ator	(AT) atributo	(BE) comportamento	(CA) cardinalidade
(CO) condição	(CR) restrição	(DA) dado	(EN) entidade
(FU) funcionalidade	(IN) herança	(ME) mensagem	(OB) objeto/classe
(RE) relacionamento	(RO) papel		

Tipo de Discrepância (Tipo Disc.):
(1) funcionalidade ou conceito necessário foi omitido
(2) o MCU está incorreto em relação aos requisitos
(3) como o MCU implementa estes requisitos é ambíguo ou não completamente especificado
(4) a informação do MCU é estranha, i.e. não mencionada pelos requisitos
(5) outros tipos de problema [explique abaixo]
Severidade (Sev.):
(NS) Não é sério. Mas precisa verificar este documento.
(IN) Esta discrepância invalida esta parte do documento. Verifique ambos os documentos.
(SE) Sério. Não é possível continuar a leitura deste documento. Ele deveria ser reorganizado.

Preencha a tabela com as discrepâncias encontradas. Descreva a funcionalidade dos requisitos, utilizando os números da página se possível:

Disc. #	Tipo do Conceito	Nome do Documento	Tipo Disc.	Identificação Requisito	Sev.	Comentários
01	DA	ECU no. 5	1	R.F. 8	NS	O item de informação título não foi mencionado na ECU
02						

Documento de Requisitos

Laboratório de Engenharia de Software **UFSCar – 1º. Semestre 2004**
Responsável pela atividade: Sandra Fabbri

Engenharia de Software Experimental

Documento de Requisitos para um Sistema de Controle de Estacionamento[1]

1 Introdução

1.1 Propósito

Este documento descreve os requisitos de software para um sistema de controle de estacionamento (PGCS). Esta especificação está planejada para o projetista, para o desenvolvedor e para o responsável pela manutenção do PGCS.

1.2 Escopo

A função do PGCS é controlar e supervisionar as entradas e saídas de um estacionamento. O sistema permite ou rejeita entradas no estacionamento de acordo com o número de vagas disponíveis.

1.3 Organização do Documento

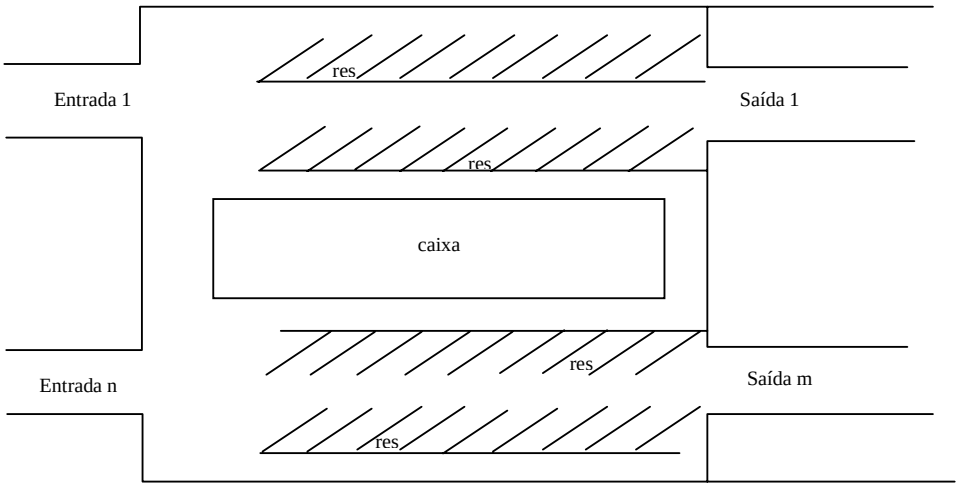
O resto deste documento é organizado como segue: haverá algumas definições importantes na próxima subseção. O capítulo 2 contém uma descrição geral do PGCS. O capítulo 3 identifica os requisitos funcionais específicos, as interfaces externas e os requisitos de desempenho do PGCS.

[1] Tradução disponibilizada pelo Prof. Dr. Guilherme Horta Travassos, referente ao Documento de Requisitos originalmente produzido pelo Experimental Software Engineering Group – University of Maryland.

1.4 Definições

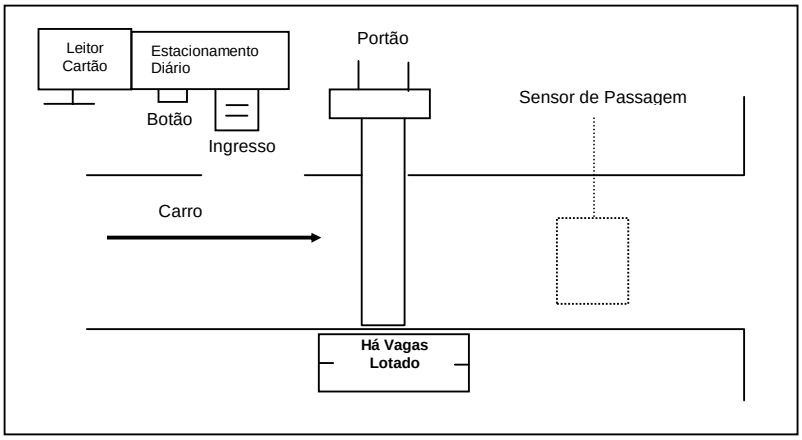
- Estacionamento

Consiste de **n** entradas e **m** saídas. Existem **k** vagas para estacionamento, das quais **r** são reservadas. O número máximo de vagas é 1000.



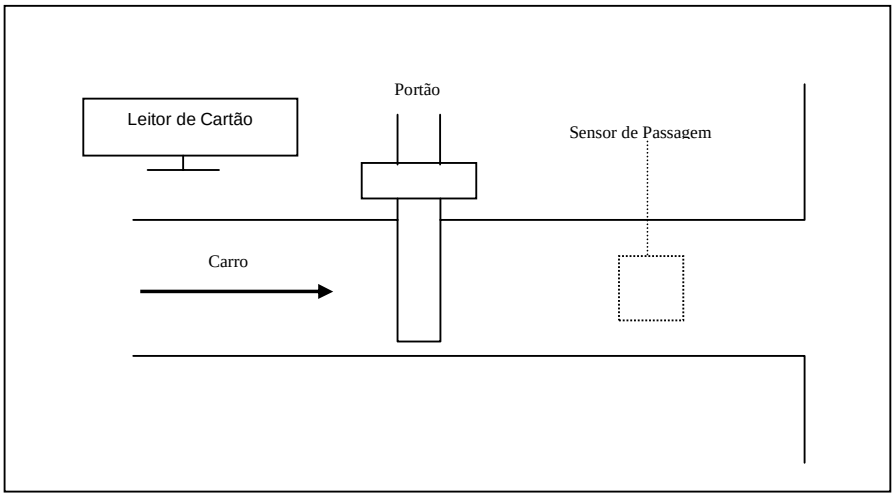
- Entrada

Uma entrada consiste em um portão, um painel de *status* que indica se existem vagas disponíveis, uma máquina de ingresso e um sensor de passagem. A máquina de ingresso consiste de um botão de pedido, uma unidade para a produção dos ingressos e um leitor de cartão.



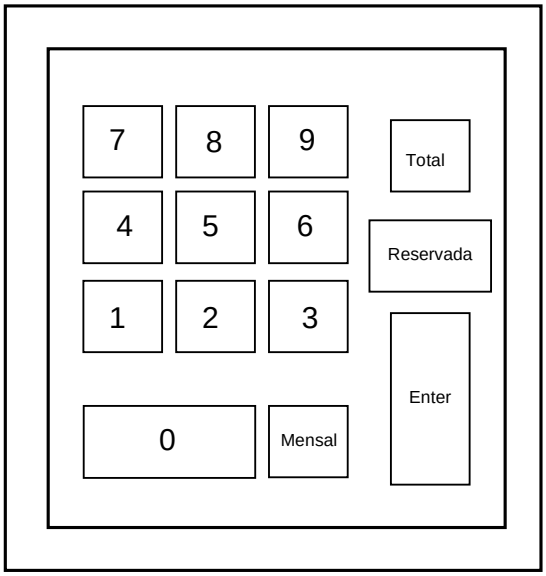
- Saída

A saída consiste em um portão, um leitor de cartão e um sensor de passagem que está atrás do portão.



- Unidade de Controle

A unidade de controle consiste de um teclado numérico.



Descrição geral

2.1 Descrição

Os seguintes cenários apresentam uma descrição sucinta da funcionalidade do PGCS:

- Entrada de motorista sem uma vaga de estacionamento reservada:
  1. Um motorista pressiona o botão da máquina de ingresso;
  2. Se o estacionamento estiver lotado, nada acontece (painel de status indica "lotado");
  3. Caso contrário, a máquina de ingresso escreve o horário no ingresso e o entrega para o motorista;
  4. O portão abre quando o motorista pegar o ingresso;
  5. O motorista entra no estacionamento;
  6. Depois da passagem do carro pelo loop de indução, o portão é fechado;
  7. O motorista estaciona o carro e deixa o estacionamento.

- Motorista com uma vaga reservada:
  1. O motorista insere o seu ingresso mensal no leitor de cartão da máquina de ingresso;
  2. A máquina de ingresso confere se é um ingresso mensal válido;
  3. Se for um ingresso mensal válido, o portão abre e o motorista entra no estacionamento;
  4. Depois da passagem do carro pelo sensor de passagem, o portão é fechado;
  5. O motorista estaciona o carro em uma vaga e deixa o estacionamento.

- Pagamento do estacionamento por motoristas sem uma vaga reservada:
  1. O motorista paga a taxa no caixa;
  2. O caixa notifica o sistema do número do ingresso que foi pago.

- Saída
  1. O motorista volta ao carro e vai até uma estação de saída;
  2. O motorista insere o ingresso no leitor de cartão;
  3. O leitor confere o ingresso através de seu número e verifica se a taxa foi paga nos últimos 15 minutos ou se foi inserido um ingresso mensal válido;
  4. Em caso negativo, nada acontece. O motorista tem que retornar ao caixa;
  5. Senão, o portão abre;
  6. Depois da passagem de carro, o portão é fechado.

- Manutenção e teste do sistema
  1. Para testar e manter o sistema é possível entrar o número total de vagas e o número de vagas reservadas com a ajuda da unidade de controle.

- Ingressos
  1. Os ingressos mensais e diários serão identificados por números e um código que o leitor de ingresso possa interpretar para determinar se o ingresso é mensal ou diário. Esta funcionalidade está fora do contexto do sistema. O sistema receberá uma mensagem do leitor de ingresso que notifica se um ingresso mensal ou diário foi inserido e o número do ingresso.
  2. Não é necessário que os números de ingresso sejam seqüenciais.

2.2 Perspectiva de produto

O software é um sistema embarcado. Serão descritas as características dos dispositivos. Em cada entrada, o sistema de software deve controlar:

- uma máquina de ingresso
- um portão
- um leitor de cartão
- um sensor de passagem
- um painel de status

Para cada saída:

- um leitor de ingresso
- um portão
- um sensor de passagem
- uma unidade de controle
- um teclado complementar numérico

A seguir, descrevemos o comportamento do sistema. O termo “automaticamente” descreve o comportamento de um dispositivo que é realizado sem controle do sistema de software. A máquina de ingresso imprime automaticamente o horário no ingresso se um ingresso é fornecido. O leitor de cartão lê automaticamente um cartão inserido. O leitor de ingresso identifica automaticamente os ingressos mensais.

O loop de indução pode estar em dois estados: um carro está presente no loop ou não. Após as mudanças de estado, um sinal será enviado ao PGCS. Os portões têm dois estados: aberto e fechado. A mudança de estado leva algum tempo. O painel de status mostra dois estados: lotado e com vagas, de acordo com o número de vagas disponíveis no estacionamento. O painel de status é irrelevante para os motoristas que possuem um ingresso mensal.

Com o teclado numérico da unidade de controle há a possibilidade de entrar o número de total de vagas reservadas. O caixa não é controlado pelo sistema de software, mas notifica o sistema do número de um ingresso pago.

2.3 Funções do produto

O sistema de software deve controlar o painel de status, os portões, as máquinas de ingresso e os leitores de ingresso. Se um ingresso mensal válido é inserido no leitor de cartão de uma máquina de ingresso, o portão deve abrir. Se houver vagas disponíveis e o botão de pedido for pressionado, o motorista deve adquirir um ingresso e o portão deve ser aberto.

A inserção de um ingresso não reservado em um leitor de cartão deve abrir o portão de saída se a taxa de estacionamento foi paga nos últimos 15 minutos. A inserção de um ingresso mensal válido no leitor de cartão deve abrir o portão de saída. Os portões devem ser fechados depois que o carro passar pelo sensor de indução.

O painel de status deve mostrar o estado atual de ocupação do estacionamento. Para testes e manutenção do sistema, há a possibilidade de entrar o número de total vagas e vagas reservadas com ajuda da unidade de controle. Ingressos mensais podem ser reservados no caixa. O número de vagas reservadas não deve ser superior a 40% de k. O caixa não é parte do sistema de software.

2.4 Características do usuário

Os usuários do sistema (os motoristas) não devem requerer treinamento especial.

2.5 Suposições sobre o estacionamento

- Toda vaga do estacionamento pode ser alcançada de qualquer entrada;
- Toda saída pode ser alcançada de cada vaga de estacionamento;
- Nenhuma entrada é conversível a saída e vice-versa;
- As vagas não são numeradas;
- Existem ingressos mensais para reservar vagas de estacionamento;
- Situações de emergência (por exemplo, fogo) não serão consideradas;
- Os ingressos mensais para as vagas reservadas estão disponíveis no caixa.
- O caixa controla o número de ingressos mensais para vagas reservadas.
- Quando um ingresso mensal for vendido, o caixa notifica o sistema do número do ingresso. O sistema reduzirá de um o número de vagas não reservadas e será responsável por aumentar o número de vagas não reservadas quando o ingresso expirar (em 30 dias).

3. Requisitos

3.1 Requisitos funcionais

Esta é uma lista de requisitos funcionais que o sistema deve satisfazer. Os requisitos funcionais são apresentados do seguinte modo:

- Descrição: uma descrição da exigência específica (opcional);
- Entradas: uma descrição das entradas que o sistema de software recebe;
- Processamento: uma descrição do que o sistema de software deve fazer com as entradas;
- Saídas: uma descrição da resposta ou novo estado do sistema de software.

Exigência Funcional 1: Objetos de Dados
No software, existem os seguintes objetos de dados:
k: número máximo de vagas disponíveis no estacionamento;
r: número de vagas reservadas no estacionamento;
a = k – r: número de vagas não reservadas e disponíveis;
o: número de vagas não reservadas e ocupadas.

3.1.1 Requisitos gerais

Requisito Funcional 1

- Descrição: O PGCS deve controlar as entradas e saídas de um estacionamento.

Requisito Funcional 2

- Descrição: O PGCS tem que garantir que não mais que **k** carros estão no estacionamento.

Requisito Funcional 3

- Descrição: O valor *default* para **k** é 1000.

Requisito Funcional 4

- Descrição: **k** é dividido em **r** vagas reservadas e **a** vagas não reservadas.

Requisito Funcional 5

- Descrição: o PGCS deve apoiar **n** entradas e **m** saídas, eventualmente simultâneas.

Requisito Funcional 6

- Descrição: Se o sensor de passagem é cruzado, o portão deve fechar.
- Entradas: sensor de passagem vai do estado “carro presente” para o estado “carro não presente”
- Saída: o portão fecha

Requisito Funcional 7

- Descrição: ingressos mensais são válidos durante 30 dias.

3.1.2 Requisitos de atualização

Requisito Funcional 8

- Descrição: a notificação do caixa sobre a compra de um novo ingresso mensal mantém histórico deste ingresso pelo seu número e aumenta de um o valor de **r**.
- Entrada: o caixa entra o número de ingresso mensal e pressiona o botão “mensal” no painel de controle.
- Processamento: Verifique se ‘**r** + 1 0.4 * **k’** e se ‘**a** - **o** 1’. Nesse caso, decremente o valor de **r** por 1 e armazene o novo ingresso mensal. Caso contrário, produza uma mensagem de erro.
- Saídas: novo valor de **r** e de **a**, ou uma mensagem de erro

Requisito Funcional 9

- Descrição: a notificação de caixa para o sistema de que um ingresso foi pago.
- Entradas: caixa entra o número de ingresso e pressiona a tecla “Enter”
- Processamento: o sistema mantém histórico do número de ingresso e tempo pago, conferindo as saídas.

Requisito Funcional 10

- Descrição: a unidade de controle pode fixar um valor novo de **r**.
- Entrada: entrada de um número e pressão do botão “Reservado" e "Enter" na unidade de controle
- Processamento: fixe o valor novo de **r**
- Saída: novo valor de **r**

Requisito Funcional 11

- Descrição: o número total de vagas pode ser escrito na unidade de controle
- Entradas: entrada de um número e pressão do botão "Total" e "Enter" na unidade de controle
- Processamento: verifique se **k** é maior que ‘**r** + **o**’ e aquele ‘**k** * 0.4 **r**’. Assume o novo valor de **k**.
- Saídas: novo valor de **k**

Requisito Funcional 12

- Descrição: sistema deve liberar as vagas reservadas quando os ingressos mensais expiram.
- Processamento: às 12:01 AM o sistema deve conferir quantos ingressos foram comprados a 31 dias atrás e reduzir o valor de **r** por este número.
- Saída: novos valores para **r** e **a**.

3.1.3 Requisitos de entrada

Estes requisitos caracterizam a exigência para uma entrada.

Requisito Funcional 13

- Descrição: o painel de status deve representar o estado de ocupação das vagas não reservadas. Deve exibir ‘vagas' se houver uma vaga não reservada disponível naquele momento. Deveria exibir ‘lotado’, se não há nenhuma vaga não reservada disponível naquele momento.

Requisito Funcional 14

- Descrição: todo motorista com um ingresso mensal deverá inserir o ingresso no leitor de ingresso. Se este for válido, o portão abrirá. Um ingresso reservado válido sempre permitirá entrada.
- Entrada: motorista insere ingresso mensal.
- Processamento: o número de ingresso será analisado pelo sistema para determinar se a data de sua compra foi há 30 dias ou menos antes da data atual. Em caso afirmativo, o portão abrirá.
- Saídas: o portão é aberto.

Requisito Funcional 15

- Descrição: todo motorista que usa uma vaga não reservada, só deve adquirir um ingresso se houver pelo menos uma vaga disponível.
- Entrada: motorista aperta o botão uma vez.
- Processamento: verifique se há uma vaga disponível, isto é, se ‘**a** - **o** > 0’.
- Saída: forneça um ingresso ao motorista se houver uma vaga disponível e abre o portão.

Requisito Funcional 16

- Descrição: diminua o número de vagas disponíveis se o motorista entrar no estacionamento
- Entradas: conclusão com sucesso da exigência 16 e carro atravessa o loop de indução
- Processamento: incrementa o valor de **o** por 1.

Requisito Funcional 17

- Descrição: será dado no máximo um ingresso de entrada a cada motorista
- Entradas: pressão do botão enquanto o portão estiver aberto.
- Processamento e Saídas: Pedidos de ingresso enquanto o portão estiver aberto serão ignorados.

Requisito Funcional 18

- Descrição: se botão é pressionado e não há nenhuma vaga não reservada, nada acontece.
- Entrada: motorista aperta o botão e ‘a–o = 0’.
- Processamento: nada acontece

Requisito Funcional 19

- Descrição: se mais de um carro quer entrar no estacionamento por entradas diferentes, o PGCS tem controlar todos os eventos na ordem eles acontecem.
- Entrada: vários motoristas apertam o botão de pedido de ingresso antes que qualquer ingresso de entrada seja emitido.
- Processamento: os eventos devem ser controlados na ordem em que eles acontecem e só os motoristas com vagas disponíveis terão entrada permitida.
- Saída: habilite entrada aos vários motoristas

3.1.4 Requisitos de saída

Estes requisitos caracterizam uma saída.

Requisito Funcional 20

- Descrição: um carro chega à saída e o motorista insere um ingresso mensal no leitor de cartão. Se o ingresso é válido, o portão abre.
- Entrada: motorista põe um ingresso mensal no leitor de cartão.
- Processamento: verifique o ingresso no sistema e determine se a data de compra foi 30 dias ou menos antes da data atual. Em caso afirmativo, o ingresso é válido: abra o portão.
- Saídas: se o ingresso for válido, portão abre. Senão, nada acontece.

Requisito Funcional 21

- Descrição: Motorista com um ingresso não reservado chega à saída e põe o ingresso no leitor de cartão. Se o ingresso for válido, o portão abre.
- Entrada: Motorista põe um ingresso no leitor de ingresso.
- Processamento: o sistema verifica o ingresso através de seu número. Se o ingresso foi pago nos últimos 15 minutos então é válido. Se o ingresso é válido, o portão abre. Uma vez o carro atravessou o sensor de passagem, o valor de **o** é reduzido de um. Se o ingresso é inválido, nada acontece.
- Saída: se o ingresso é válido, o portão abre. Senão, nada acontece.

Requisito Funcional 22

- Descrição: se vários carros deixam o estacionamento ao mesmo tempo, o PGCS tem que controlar os eventos em ordem.

3.2 Requisitos de interface externas

O PGCS tem que prover uma interface para obter mensagens do teclado numérico usado pelo caixa.

3.2.1 Interface com o usuário

Além da unidade de controle, não há nenhuma necessidade por uma interface de usuário.

3.2.2 Interfaces de hardware

Devem existir interfaces de hardware com as máquinas de ingresso, os leitores de cartão, os portões, os sensores de passagem e a unidade de controle. O PGCS receberá e enviará sinais para estes dispositivos.

3.3 Requisitos de desempenho

Requisito de Desempenho 1

- Descrição: depois que um carro passa o sensor de passagem, o portão tem que fechar dentro de 5 segundos.

Requisito de Desempenho 2

- Descrição: se um motorista pede um ingresso e existem vagas disponíveis, ele deve adquirir o ingresso dentro de 3 segundos.

Requisito de Desempenho 3

- Descrição: se um portão abrir, deve permanecer aberto no máximo 20 segundos, a menos que um carro esteja no sensor de passagem.

Requisito de Desempenho 4

- Descrição: só um carro deve atravessar o portão a cada vez.

Requisito de Desempenho 5

- Descrição: a compra de um ingresso mensal altera os valores de **a** e **r** dentro de 15 segundos.

Requisito de Desempenho 6

- Descrição: alterações em variáveis de estado (entradas ou saídas) devem acontecer dentro de 5 segundos.

Requisito de Desempenho 7

- Descrição: para cada carro que entra no estacionamento há uma vaga disponível.

3.4 Atributos

3.4.1 Disponibilidade

O sistema tem que estar disponível 24 h/dia. O estacionamento nunca estará fechado.

3.4.2 Segurança

Nenhum ingresso diferente dos ingressos deste estacionamento deve ser aceito pelo leitor de ingresso.

3.4.3 Manutenção

Não Aplicável

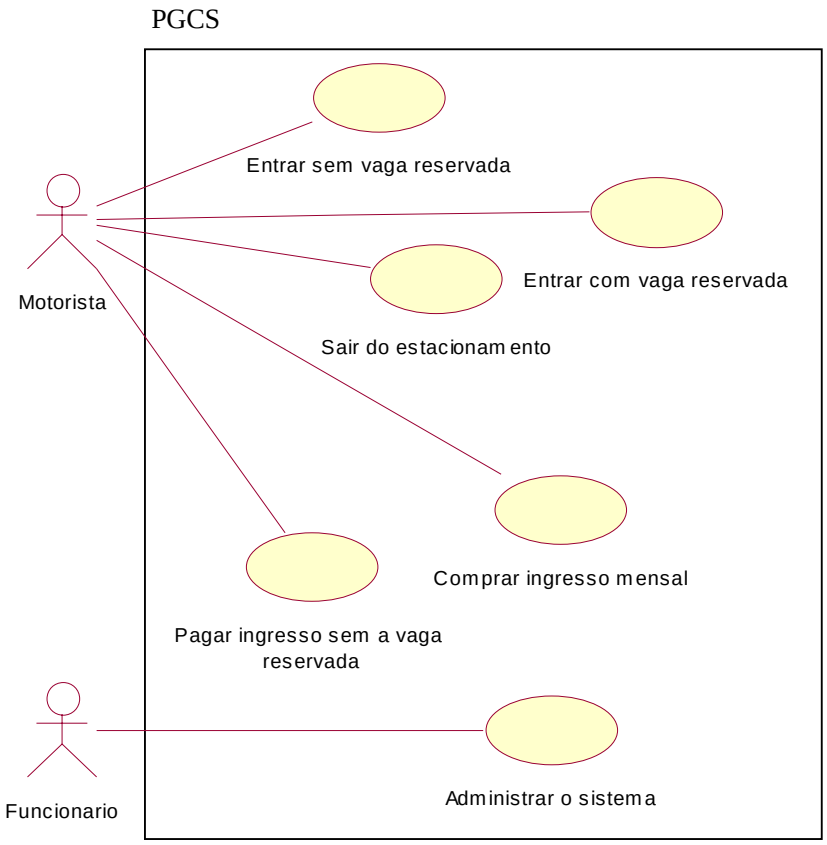
3.4.4 Conversões

Não Aplicável

3.4.5 Precaução

Não Aplicável

Diagrama de Casos de Uso


PGCS
Entrar sem vaga reservada
Entrar com vaga reservada
Sair do estacionamento
Comprar ingresso mensal
Pagar ingresso sem a vaga reservada
Administrar o sistema
Motorista
Funcionario

Especificações dos Casos de Uso

Nº. Caso de Uso:	**1**
Caso de Uso:	**Entrar sem vaga reservada**
Atores:	Motorista
Propósito:	Entrar no estacionamento sem ter uma vaga reservada.
Descrição:	O motorista pressiona a máquina de ingresso. Ele pega o ingresso e entra no estacionamento.
Tipo:	Primário e essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O motorista pressiona o botão da máquina de ingresso.	2. A máquina de ingresso escreve o horário no ingresso e o entrega para o motorista.
3. O motorista pega o ingresso.	4. O portão abre.
5. O motorista entra no estacionamento.	6. Depois da passagem do carro pelo loop de indução, o portão é fechado e o painel de status é atualizado.

Curso Alternativo 1

1. Vários motoristas pressionam o botão da máquina de ingresso ao mesmo tempo antes que qualquer ingresso de entrada seja emitido.
1.1 Os eventos devem ser controlados na ordem em que eles acontecem.

Curso Alternativo 2

2. Se o estacionamento estiver lotado, nada acontece e o painel de status indica ***"Lotado".***

Nº. Caso de Uso:	**2**
Caso de Uso:	**Entrar com uma vaga reservada**
Atores:	Motorista
Propósito:	Entrar no estacionamento com uma vaga reservada.
Descrição:	O motorista insere o seu ingresso mensal no leitor de cartão da máquina de ingresso e entra no estacionamento.
Tipo:	Primário e essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O motorista insere o seu ingresso mensal no leitor de cartão da máquina de ingresso.	2. A máquina de ingresso confere o ingresso mensal válido, ou seja, se a compra foi feita há 30 dias ou menos antes da data atual. O portão é aberto.
3. O motorista entra no estacionamento.	4. Depois da passagem de carro pelo sensor de passagem, o portão é fechado.

Curso Alternativo

2. A máquina de ingresso verifica que o ingresso mensal é inválido, nada acontece.

Nº. Caso de Uso:	**3**
Caso de Uso:	**Sair do Estacionamento**
Atores:	Motorista
Propósito:	Sair do estacionamento.
Descrição:	O motorista pega o carro e insere o ingresso no leitor de cartão e sai do estacionamento.
Tipo:	Primário e essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O motorista volta ao carro e vai até uma estação de saída para inserir o ingresso no leitor de cartão.	2. O leitor confere o número do ingresso e verifica que a taxa foi paga nos últimos 15 minutos ou que foi inserido um ingresso mensal válido. O portão é aberto.

3. O motorista sai do estacionamento. 4. Depois da passagem do carro, o portão é fechado.

Curso Alternativo 1

1. Vários motoristas voltam ao carro e vão sair ao mesmo tempo, o sistema tem que controlar os eventos em ordem.

Curso Alternativo 2

2. O leitor verifica que a taxa não foi paga nos últimos 15 minutos ou que o ingresso mensal é inválido.
3. Nada acontece e o motorista tem que retornar ao caixa.

Nº. Caso de Uso:	**4**
Caso de Uso:	**Comprar ingresso mensal**
Atores:	Motorista
Propósito:	Comprar um ingresso mensal
Descrição:	O motorista compra um ingresso mensal.
Tipo:	Primário e essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O motorista compra um ingresso mensal.	2. O caixa notifica o sistema da compra do ingresso e este mantém o histórico através de seu número. 3. O valor de **r** é alterado.

Curso Alternativo

2. O sistema produz uma mensagem de erro quando não há nenhuma vaga reservada.

Nº. Caso de Uso:	**5**
Caso de Uso:	**Pagar ingresso sem a vaga reservada**
Atores:	Motorista
Propósito:	Pagar um ingresso sem ter uma vaga reservada
Descrição:	O motorista sem vaga reservada paga o ingresso.
Tipo:	Primário e essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O motorista paga a taxa no caixa.	2. O caixa notifica o sistema do número do ingresso que foi pago.

Nº. Caso de Uso:	**6**
Caso de Uso:	**Administrar o sistema**
Atores:	Funcionário
Propósito:	Administrar o sistema de estacionamento.
Descrição:	O funcionário entra com o número total de vagas e/ou o número de vagas reservadas do estacionamento com a ajuda da unidade de controle.
Tipo:	Secundário e essencial

Curso Típico dos Eventos

Ação do Ator	Resposta do Sistema
1. O funcionário entra com um novo valor para **r** (número de vagas reservadas) e/ou entra com um novo valor para **k** (número máximo de vagas disponíveis no estacionamento).	2. Para fixar um valor para **r**, digita-se o número e pressiona-se o botão "Reservado" e "Enter". Para fixar um valor para **k**, digita-se o número e pressiona-se o botão "Total" e "Enter".

Relatório de Discrepâncias

Formulário de Relato de Discrepâncias – ER1

Início da Inspeção: _____ (hora) **Data:_____ (data)** **ID do revisor:_____**

Tipo de Conceito:

(AC) ator	(AT) atributo	(DA) dado
(CO) condição	(CR) restrição	(EN) entidade
(FU) funcionalidade	(RO) papel	
(RE) relacionamento	(BE) comportamento	

Tipo de Discrepância (Tipo Disc.):
(1) funcionalidade ou conceito necessário foi omitido.
(2) o MCU* está incorreto em relação aos requisitos.
(3) como o MCU implementa estes requisitos é ambíguo ou não completamente especificado.
(4) a informação de MCU é estranha, i.e. não mencionada pelos requisitos.
(5) outros tipos de problemas [explique abaixo].

Severidade (Sev.):
(NS) Não é sério. Mas precisa verificar este documento.
(IN) Esta discrepância invalida esta parte do documento. Verifique ambos os documentos.
(SE) Sério. Não é possível continuar a leitura deste documento. Ele deveria ser reorganizado.

Preencha a tabela com as discrepâncias encontradas. Descreva a funcionalidade dos requisitos, utilizando os números da página se possível:

Disc. #	Tipo do Conceito	Nome do Documento	Tipo Disc.	Identificação Requisito	Sev.	Comentários
01						
02						
03						
04						
05						
06						
07						
08						
09						
10						
11						
12						
13						
14						
15						
16						

(utilize a parte de trás se necessário)

Fim da Inspeção:_____ (hora)

* MCU = Modelo de Caso de Uso = Diagrama de Casos de Uso + Especificação dos Casos de Uso