UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

GERAÇÃO DE PONTOS DE CASOS DE USO NO AMBIENTE COCAR

MARCOS DANILO CHIODI MARTINS

São Carlos
Janeiro/2007

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

M386gp

Martins, Marcos Danilo Chiodi.
Geração de pontos de casos de uso no ambiente Cocar / Marcos Danilo Chiodi Martins. -- São Carlos : UFSCar, 2008.
169f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2007.

1. Métricas de software. 2. Pontos de casos de uso. 3. Modelo de casos de uso. 4. Software – desenvolvimento. 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

"Geração de Pontos de Casos de Uso no Ambiente COCAR"

MARCOS DANILO CHIODI MARTINS

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 P. Ferraz Fabbri
(Orientadora – DC/UFSCar)

Profa. Dra. Rosely Sanches
(ICMC/USP)

Prof. Dr. Auri Marcelo Rizzo Vincenzi
(UNISANTOS)

São Carlos
Janeiro/2007

DEDICATÓRIA

A minha mãe Marli, irmã Mariele, querida noiva Carol
e padrasto Paulo pelo carinho, incentivo
e paciência em todos os momentos.
Amo muito todos vocês.

AGRADECIMENTOS

Acima de tudo a Deus por me proporcionar a sabedoria, a saúde e a coragem necessários para me permitir realizar este trabalho.

A minha orientadora, Profª. Drª. Sandra C.P.F. Fabbri, pela confiança, ensinamentos, motivação e excelente orientação, além da grande paciência e dedicação.

A Profª. Drª Maria da Graça Brasil Rocha pela colaboração com a parte de implementação da ferramenta.

Em especial à minha mãe Marli e irmã Mariele pelo apoio e incentivo, nunca me permitindo o desânimo e sempre me mostrando o melhor caminho.

À minha sempre amorosa e solícita noiva Carol pela colaboração nas revisões, pelo apoio, incentivo, carinho, paciência e compreensão nos meus momentos de ausência ocasionados por este trabalho.

Ao diretor Lúcio da empresa AGX Tecnologia LTDA e à diretora Magda, da empresa VoxAge Teleinformática LTDA, pela compreensão, apoio, incentivo e fornecimento de informações fundamentais para a realização deste trabalho.

Aos companheiros de mestrado e amigos André e Bruno e ao amigo de cruz Pedro, pelos conhecimentos compartilhados e pela enorme amizade que cultivamos. Ao amigo Maurício agradeço as contribuições técnicas para a implementação da ferramenta, sem as quais certamente este trabalho não teria sido concluído.

À secretária da Pós-Graduação, Cristina e à Miriam, pelos seus serviços sempre bem prestados e a toda comissão da PPGCC pela compreensão dos contratempos acontecidos durante este trabalho.

Aos professores membros da banca examinadora por prestigiarem este trabalho com sua presença, críticas e sugestões de melhoria.

SUMÁRIO

SUMÁRIO 4

LISTA DE FIGURAS 7

LISTA DE TABELAS 8

LISTA DE QUADROS 9

ABSTRACT 11

CAPÍTULO 1 – INTRODUÇÃO 1

1.1 Contexto 4

1.2 Motivação e Objetivos 6

1.3 Organização do Trabalho 6

CAPÍTULO 2 - ENGENHARIA DE REQUISITOS 8

2.1 Considerações Iniciais 8

2.2 Engenharia de Requisitos – Principais Conceitos 9

2.3 Considerações Finais 28

CAPÍTULO 3 - MODELAGEM DE REQUISITOS COM CASOS DE USO 30

3.1 Considerações Iniciais 30

3.2 Diagramas de Casos de Uso 31

3.3 Especificação de Casos de Uso 35

3.4 Considerações Finais 37

CAPÍTULO 4 - MÉTRICAS DE TAMANHO DE SOFTWARE 39

4.1 Considerações Iniciais 39

4.2 Pontos de Casos de Uso 40

4.3 Pontos por Função 45

4.4 Considerações Finais 51

CAPÍTULO 5 - TRABALHOS RELACIONADOS 53

5.1 Considerações Iniciais 53

5.2 Trabalhos Teóricos....54

5.3 Ferramentas para Automatização de Métricas de Software....89

5.4 Considerações Finais....94

CAPÍTULO 6 - O AMBIENTE COCAR....95

6.1 Considerações Iniciais....95

6.2 Funcionalidades do Ambiente COCAR....96

6.3 Aspectos de Implementação do Ambiente COCAR....100
6.3.1 Processo de Desenvolvimento....100
6.3.2 Projeto....103
6.3.3 Implementação....107

6.4 Requisitos Funcionais do Ambiente COCAR – Pontos de Caso de Uso....108
6.4.1 Classificação Automática da Complexidade dos Casos de Uso....109
6.4.2 Cálculo da Medida dos Pontos de Caso de Uso....109
6.4.3 Transformação da métrica de PCU em PF....111

6.5 Considerações Finais....111

CAPÍTULO 7 - EXEMPLO DE USO DO AMBIENTE COCAR....113

7.1 Considerações Iniciais....113

7.2 Apresentação do Ambiente COCAR....115

7.3 Criação de Sistema....116

7.4 Cadastramento de Requisitos....117

7.5 Criação do Modelo de Casos de Uso....121
7.5.1 - Actor-Goal Reading Technique....121
7.5.2 - Use Case Reading Technique (UCRT)....125

7.6 Automatização da Técnica PCU e Geração da métrica de PF....130

7.7 Avaliação do Ambiente COCAR....133
7.7.1 Planejamento do Estudo de Caso....133
7.7.2 –Execução do Estudo de Caso....134
7.7.3 –Coleta e Análise de Dados....135

7.8 Considerações Finais....138

CAPÍTULO 8 - CONCLUSÃO....139

8.1 Contribuições e Limitações deste Trabalho....141

8.2 Lições Aprendidas....143

8.3 Trabalhos Futuros....144

REFERÊNCIAS BIBLIOGRÁFICAS....146

APÊNDICE I.....157

APÊNDICE II.....163

LISTA DE FIGURAS

Figura 1– Subáreas da engenharia de requisitos [SWEBOK, 2004] 10
Figura 2 – Representação gráfica de um Caso de Uso 31
Figura 3 – Atores com relacionamento de generalização (adaptado de Boock et al., 2000) 33
Figura 4 – Casos de Uso e seus relacionamentos (adaptado de Boock et al., 2000) 35
Figura 5 – Principais características e diferenças entre APF e PCU 51
Figura 6 – Processo de estimativa de esforço [Carrol, 2005]. 64
Figura 7 – Formulário de Casos de Uso Preliminares [Belgamo, 2004]. 68
Figura 8 - Formulário de Especificação de Casos de Uso 68
Figura 9 - Arquitetura da ferramenta U-EST 84
Figura 10 – Formato de um Requisitos Funcional [Kawai, 2005]. 86
Figura 11 – Diagrama de Classe UML do Padrão DAO[Java, 2006] 105
Figura 12 - Diagrama de Classe UML do Padrão DAO[Java, 2006] 105
Figura 13 - Diagrama de Classes UML do Padrão Transfer Object [Java, 2006]. 106
Figura 14 - Diagrama de Classes e Seqüência UML do Padrão Transfer Object [Java, 2006]. 106
Figura 15 - Diagrama de Classes UML da funcionalidade Cadastrar Sistema. 107
Figura 16 - Prototipação Evolucionária [Sommerville, 2003] 107
Figura 17 – Interface de *help* do sistema COCAR 115
Figura 18 – Escolha de um sistema ou a criação de um novo sistema 116
Figura 19 – Selecionar um sistema já cadastrado como ativo 116
Figura 20 – Cadastramento de um sistema no ambiente COCAR 117
Figura 21 – Cadastramento de Requisitos no ambiente COCAR. 119
Figura 22 – Cadastramento de Entradas no ambiente COCAR 120
Figura 23 – Cadastramento de Atores (primário e secundário) 120
Figura 24 – Cadastramento de Solicitante de Requisitos 120
Figura 25 – Cadastramento de Gerente de Requisito 121
Figura 26 - Processo de marcação de atores, funções e restrições. 123
Figura 27 – Determinação dos atores e objetivos. 124
Figura 28 - Exemplo da interface de resolução de redundância intra-atores e inter-atores. 125
Figura 29 - Exemplo do FAO gerado pela aplicação da AGRT no ambiente COCAR. 125
Figura 30 – Primeira Etapa: Formulário de Casos de Uso Preliminares 127
Figura 31 – Interface para a especificação dos Casos de Uso e Janela com a janela dos requisitos relacionados e janela de especificação de curso alternativo 129
Figura 32 – Classificação de complexidade dos atores 130
Figura 33 – Resultado do Cálculo do PCU não ajustado 131
Figura 34 – Itens de complexidade ambiental e complexidade técnica julgados pelo usuário 132
Figura 35 – Relatório de Pontos de Caso de Uso Ajustado 132

LISTA DE TABELAS

Tabela 1– Forma de classificação dos atores com seus respectivos pesos [Karner, 1993]. 41
Tabela 2 – Forma de classificação dos Casos de Uso com seus respectivos pesos [Karner, 1993] 41
Tabela 3 – Fatores que contribuem para a complexidade do sistema [Medeiros, 2004; Karner, 1993] 42
Tabela 4– Fatores que contribuem para a eficiência [Medeiros, 2004; Karner, 1993] 43
Tabela 5 – Estimativas de especialistas, estimativa por Pontos de Casos de Uso, e Esforço (em horas) [Anda et al., 2001]. 55
Tabela 6 – Resumo dos Pontos de Casos de Uso para projetos incrementais [Mohagheghi et al., 2005]. 59
Tabela 7 - Esforço estimado por projetos 66
Tabela 8 - Esforço de desenvolvimento por Caso de Uso 74
Tabela 9 - Horas Estimadas X Horas Gastas dos projetos das quatro empresas 74
Tabela 10 – Regras para o cálculo da Profundidade das Alterações de um Caso de Uso 76
Tabela 11 – Principais funcionalidades da ferramenta Construx Estimate [Construx, 2005]. . 90
Tabela 12 – Principais funcionalidades da ferramenta Cost Xpert [Costxpert, 2005]. 91
Tabela 13 – Principais funcionalidades da ferramenta COSMOS [Cosmos, 2005]. 91
Tabela 14 – Principais funcionalidades da ferramenta EEUC [EEUC, 2005]. 92
Tabela 15 – Resumo das funcionalidades das ferramentas analisadas 92
Tabela 16– Dados coletados do Estudo de Caso 136

LISTA DE QUADROS

Quadro 1 – Formulário de Especificação de Casos de Uso [Belgamo, 2004]..... 37
Quadro 2 – Complexidade funcional [IFPUG, 1999]. 46
Quadro 3 – Complexidade de EE (IFPUG, 1999). 48
Quadro 4 – Complexidade de SE [IFPUG, 1999]. 48
Quadro 5 – Exemplo de cálculo dos Pontos por Função Não Ajustados [IFPUG, 1999]. 49
Quadro 6 - Características gerais do sistema [Andrade, 2004]. 49
Quadro 7 - Passos para a determinação do VFA [IFPUG, 1999]. 50

RESUMO

Este trabalho teve como objetivo a implementação de uma versão inicial de um ambiente de apoio ao desenvolvimento de software, denominado COCAR, baseado no Modelo de Casos de Uso. A concepção e as funcionalidades desse ambiente são frutos de alguns trabalhos de mestrado, sendo que no contexto deste trabalho deu-se ênfase à métrica Pontos de Caso de Uso (PCU). Essa métrica dá subsídios à aplicação de técnicas de estimativas, as quais são fundamentais para o cálculo do tempo de desenvolvimento de um sistema, o qual está associado a uma das características principais relacionadas à qualidade de um produto de software, que é o atendimento de seu prazo de entrega. Com o advento do paradigma de desenvolvimento orientado a objeto, tem ganhado destaque os Pontos de Caso de Uso, que se baseia no Modelo de Casos de Uso da UML. Porém, dada a falta de formalidade e padronização na especificação e construção desses modelos, a métrica de PCU pode ficar comprometida. Assim, no contexto deste trabalho contribui-se para a construção desse ambiente implementando-se uma técnica denominada TUCCA, que ajuda nessa padronização do modelo, uma funcionalidade que apóia a inserção dos requisitos do sistema no ambiente, e a geração dos PCU, com base no modelo gerado com a aplicação da TUCCA. Para a avaliação do ambiente COCAR foi realizado um estudo de caso informal, no qual uma especificação de software real, de uma empresa de desenvolvimento de software foi submetida ao ambiente COCAR, gerando-se o Modelo de Casos de Uso bem como o PCU e a estimativa de esforço, os quais foram comparados àqueles gerados pela indústria. Os resultados deste pequeno estudo mostraram que, para esta situação específica, os resultados apresentados pelo ambiente COCAR foram bastante semelhantes àqueles definidos pela indústria.

ABSTRACT

The objective of this paper was to implement an initial version of a development support environment named COCAR based on the Use Case Model. Even though the conception and some features of this environment are the outcome of several other master papers, this work emphasises the relevance of the Use Case Point metric (PCU). This metric strengthens the usage of estimates which are of fundamental importance for the calculation of a system development time. Furthermore, such a metric is associated to one of the main drivers of a software product quality, which is the ability to meet delivery time. With the advent of the object oriented development paradigm, the Use Case Point metric based on the Use Case Model has been highlighted. However, given the lack of formality and standardization, specifying and building these models a PCU metric may be jeopardized. In the scope of this work, there have been relevant contributions to building such an environment implementing a technique called TUCCA which helps with the model standardization, a functionality that supports the insertion of system requirements in the environment and the generation of PCU as in the model generated by TUCCA application. In the assessment process of the COCAR environment an informal case study, in which a specification of actual software from a development company, has been carried out, producing Use Case Models, PCU's and effort estimates, both compared against the industry benchmarks. The results of this research showed that for this particular situation the output from the COCAR environment were very close to those defined by the industry.

Capítulo 1
Introdução

Segundo MCT (2003), o setor de produção de software no Brasil tem crescido vertiginosamente nas últimas décadas. O Brasil tem o sétimo maior mercado mundial de software com vendas de US$ 7,7 bilhões em 2001, importa o equivalente a US$ 1 bilhão e exporta em torno de US$ 100 milhões. O mercado brasileiro apresentou um crescimento anual médio de 11% entre 1995 e 2002, cerca de cinco vezes maior do que a expansão do PIB no período. É o segmento que mais cresce dentro da indústria brasileira de Tecnologia da Informação (hardware, serviços e software).

Com tanta concorrência no mercado nacional, no mercado internacional o cenário não é diferente, as empresas estão em busca constante da melhoria de qualidade no desenvolvimento de produtos de software e prestação de serviços.

Como prova disso tem-se o aumento de empresas de software certificadas em *Capability Maturity Model* (CMM), que é um modelo de qualidade de processo de software, saltando de uma empresa certificada em 1997 para 30 certificações em 2003 [MCT, 2005].

Dentro deste contexto, saber o tamanho, a complexidade, custos efetivos, tempo e esforço despendidos na construção de seus produtos pode representar para as empresas de tecnologia de informação um diferencial competitivo muito grande, tanto para aumentar o nível de qualidade do seu processo de desenvolvimento, sempre se certificando de que o cliente irá receber o produto no prazo correto, quanto na melhora em matérias administrativas como contratação de recursos humanos, medidas de produtividade, decisões de projetos, análise de risco, relacionamento cliente fornecedor, entre outros [Andrade, 2004 *apud* Hazan, 1999; Garmus & Herron, 2000; Longstreet, 2002].

Entre as métricas citadas, a de tamanho de um produto de software é, talvez, uma das mais importantes, pois é por meio dela que se faz possível estimar o cronograma, custos, esforços e tempo de desenvolvimento do software [Andrade, 2004 ; Karner, 1993 ; Chen et al., 2004], informações estas imprescindíveis para a construção do plano de desenvolvimento de software [Caldieira et al., 1998].

Contudo, pelos motivos apresentados anteriormente, é necessário então que a medida de tamanho de software seja feita logo nas fases iniciais do processo de desenvolvimento, e que seja atualizada no decorrer do projeto, com todas as informações sobre o software que estiverem disponíveis.

Sendo assim, várias métricas vêm sendo estudadas ao longo dos anos com o intuito de se desenvolver novas tecnologias que garantam medidas de tamanho de software mais precisas, para um melhor gerenciamento do processo de desenvolvimento do software, entre elas pode-se citar: Linhas de Código Fonte, Pontos por Função, *Bang*, *Feature Points,* Pontos de Caso de Uso, *Internet Points*, *Domino Points,*entre outros [Chen et al., 2004 & McPhee, 1999 & Costxpert, 2005].

Pressman (2006) faz uma divisão entre métricas orientadas a tamanho (como a Linhas de Código Fonte) e métricas orientadas a função (como Pontos por Função e Pontos de Caso de Uso). Contudo, ele afirma também que métricas orientadas a função, ao final do seu cálculo, são usadas de forma análoga às métricas orientadas a tamanho. Ainda, Albrecht (1979) afirma que Pontos por Função medem o tamanho de um software de acordo com as funcionalidades identificadas nos requisitos dos usuários. Portanto, neste trabalho, assim como em Andrade (2004), as métricas orientadas a função serão consideradas como uma forma de medir o tamanho do software e por isso serão tratadas como métricas de tamanho [Medeiros, 2004], mesmo porque já é sabido da literatura que há vários métodos de transformação de métricas orientadas a função para métricas orientadas a tamanho [Sommerville, 2003; Anda et al., 2001; Smith, 1999, Fetcke et al., 1998].

Desde a década de 1970 há duas métricas de tamanho de software que predominaram na indústria nacional e internacional de desenvolvimento: SLOC (Source Lines of Code) e Pontos por Função. No Brasil, em 2001, de 30% das empresas pesquisadas que usavam algum tipo de métrica de tamanho de software, 10,3% utilizavam o SLOC e 18,2% utilizavam Pontos por Função [MCT, 2002].

No entanto, segundo Chen et. al (2004), há algumas desvantagens em usar as métricas de SLOC e Pontos por Função. Chen et al. (2004) diz que a contagem exata do SLOC só acontece depois que a construção do software já foi finalizada quando, na verdade, o interessante seria que essa contagem fosse a mais acurada possível já no início do desenvolvimento do software. Já a métrica de Pontos por Função não é apoiada por nenhuma ferramenta que faça a contagem automaticamente, sendo que esta é uma operação manual e totalmente dependente da experiência do profissional que está fazendo a contagem, tornando a métrica totalmente subjetiva e dependente do ponto de vista desse profissional.

Além disso, a métrica Pontos por Função é baseada no paradigma procedimental, o qual separa dados de função, deixando esse tipo de métrica pouco adequada para os novos desenvolvimentos baseados no paradigma de orientação a objetos, o qual trabalha com dados e funcionalidades de forma combinada [Carbone & Santucci, 2002].

Por último, de acordo com Andrade (2004), a técnica Pontos por Função, apesar de medir o tamanho do software sob o ponto de vista do usuário, usa como base as informações geradas durante todo o processo de desenvolvimento do produto, principalmente as da fase de projeto. Sendo assim, a medida vai ficando mais acurada somente na fase do projeto, o que, embora seja muito mais vantajoso do que o oferecido pelo SLOC, ainda não é tão satisfatório quanto a indústria de desenvolvimento de software necessita.

Com o aumento do uso do paradigma de desenvolvimento orientado a objeto, o Modelo de Caso de Uso tem sido amplamente utilizado para a modelagem de requisitos. No entanto, apesar dele ter surgido junto com o modelo de desenvolvimento de software orientado a objeto, o *Objectory* [Jacobson et al., 1992], o Modelo de Casos de Uso pode ser usado independentemente deste paradigma, dando suporte para outras fases do desenvolvimento [Belgamo, 2004].

O Modelo de Casos de Uso se propõe a capturar e descrever os requisitos funcionais do software, se configurando como uma atividade que poderá ser executada normalmente durante a fase de elicitação e análise de requisitos para a representação com precisão dos requisitos descritos pelos usuários logo no início do processo de desenvolvimento do software.

Pensando na adaptação da técnica Análise de Pontos por Função para atender o novo paradigma de desenvolvimento e ainda tentando prover uma medida de tamanho de software mais precisa logo nas fases iniciais do desenvolvimento do software, Gustav Kerner definiu uma nova métrica para medir tamanho de software chamada Pontos de Caso de Uso (PCU) [Karner, 1993; Damodaran, 2003].

Como o PCU usa um modelo como base para o cálculo da métrica de tamanho, fica viável a construção de ferramentas para a realização do cálculo automático dessa métrica deixando o processo mais barato, rápido e manutenível.

O problema para a construção desse tipo de ferramenta é que um Caso de Uso pode ser especificado de diversas maneiras, a saber: texto estruturado informal, texto estruturado formal, pseudocódigo, fluxograma, redes de Petri ou linguagens de programação e ainda podendo fazer uso de diversos graus de detalhamento [Cockburn, 2001; Boock et al., 2000]. Sendo assim, a maneira de especificação do Caso de Uso e o grau de detalhamento usado podem influenciar muito a contagem dos Pontos de Caso de Uso bem como tornar inviável a automatização do processo. No entanto, embora existam vários trabalhos que discutem e propõem *frameworks*, *templates* e protocolos na tentativa de padronizar a escrita deste modelo [Cockburn, 2001; Ryser & Glinz, 2000; Belgamo 2004], não existe um formato padrão para tanto.

Assim, diferentemente dos Pontos por Função, que segundo Kusumoto et al. (2004) não podem ser automatizados, pois os itens envolvidos na contagem são subjetivos e melhor identificados já na fase de projeto, o Pontos de Caso de Uso podem ser computados logo no início do desenvolvimento desde que adote um padrão de especificação de Caso de Uso.

1.1 Contexto

Dada a importância da fase de Engenharia de Requisitos para o ciclo de desenvolvimento de software, o que é sabido da própria literatura e que ficou caracterizado com alguns dados apresentados anteriormente, alguns trabalhos relacionados com esse tema têm sido orientados na mesma linha de pesquisa que este trabalho está inserido.

Um desses trabalhos foi o mestrado de Belgamo [Belgamo, 2004; Belgamo & Fabbri, 2005], que propôs uma técnica de leitura para dar apoio à construção de Modelos de Casos de Uso (Diagrama e Especificação dos Casos de Uso), de tal forma que à medida que esses modelos

são construídos, faz-se também uma revisão do Documento de Requisitos, uma vez que este pode não ter passado por um processo de inspeção ou que, mesmo que tenha passado, nem todos os defeitos podem ter sido detectados.

Com base nos resultados de alguns estudos experimentais já realizados e publicados por Belgamo [Belgamo & Fabbri, 2004a; Belgamo & Fabbri, 2004b] pode-se dizer que a técnica proposta – TUCCA[1]: *Technique for Use Case Construction and Construction-based Requirements Analysis* – contribui substancialmente para a construção de Modelos de Casos de Uso mais padronizados, de tal forma que a experiência e a subjetividade do projetista não tenham tanta interferência nessa construção. Os Modelos de Casos de Uso gerados nos estudos experimentais realizados por Belgamo foram também avaliados por uma técnica proposta por Anda & Sjoberg (2002) e, de acordo com essa técnica, os modelos satisfazem os requisitos de qualidade que um bom modelo deve apresentar [Belgamo et al., 2005].

Como vários passos da TUCCA são bastante procedimentais, é viável a construção de uma ferramenta que dê apoio à aplicação da técnica. Além disso, um outro trabalho de mestrado [Kawai, 2005] definiu um formato para especificação de requisitos de forma que a aplicação da TUCCA seja facilitada.

Assim, dada a relevância da fase de engenharia de requisitos; dado que os modelos de casos de uso são bastante utilizados na prática; que se tem a técnica TUCCA que ajuda na padronização para geração desses modelos; que se têm diretrizes para especificação dos requisitos de um sistema de forma a facilitar a aplicação da TUCCA; e que a determinação do prazo de entrega de um produto é uma das principais características de qualidade de um processo de desenvolvimento, decidiu-se, no grupo de pesquisa desenvolver um ambiente de apoio ao desenvolvimento de software que pudesse dar suporte a várias atividades, tendo como foco principal o modelo de casos de uso. Outras funcionalidades que podem compor esse ambiente e que já estão em desenvolvimento no grupo de pesquisa são o gerenciamento de requisitos e a geração de casos de teste a partir de casos de uso.

[1] Essa técnica era referenciada anteriormente por GUCCRA-Guidelines for Use Case Construction and Requirements Analysis.

1.2 Motivação e Objetivos

Dado o contexto apresentado anteriormente o principal objetivo deste trabalho foi especificar e automatizar a métrica PCU no ambiente COCAR. Para tanto, foi preciso que se tivesse o modelo de casos de uso do sistema para que outras funcionalidades pudessem ser geradas com base nele. Assim, como a TUCCA [Belgamo, 2004] e as diretrizes para especificação de requisitos [Kawai, 2005] foram definidas em outros trabalhos de mestrado, e não se encontravam implementadas, o objetivo deste trabalho foi implementar essas duas funcionalidades para dar início à construção do ambiente COCAR, de forma que fosse também possível implementar a métrica PCU, que foi o principal alvo de estudo. A implementação da TUCCA e das diretrizes foram compartilhadas com outro mestrando [Di Tommazo, 2007], que desenvolveu a funcionalidade de gerenciamento de requisitos, simultaneamente a este trabalho.

1.3 Organização do Trabalho

O trabalho em questão está dividido em 8 capítulos, sendo que neste capítulo foi apresentado o contexto no qual o trabalho está inserido e os objetivos para sua elaboração.

No Capítulo 2 apresentam-se os principais conceitos relacionados à Engenharia de Requisitos bem como as dificuldades existentes nessa etapa do desenvolvimento de software, com base na visão estabelecida no SWEBOK(2004).

No Capítulo 3 é apresentada a técnica de modelagem de requisitos denominada Casos de Uso, juntamente com a citação de alguns trabalhos que abordam aspectos de padronização da especificação dos Casos de Uso.

No Capítulo 4 são abordadas duas métricas de tamanho de software, Pontos de Caso de Uso, que é alvo do trabalho em questão, e a Análise de Pontos por Função.

No Capítulo 5 é descrito o ambiente COCAR, mostrando a sua estrutura e aspectos de implementação, bem como as funcionalidades deste ambiente implementadas por este trabalho, além de um pequeno exemplo que mostra o funcionamento desse ambiente.

No Capítulo 6 apresenta-se um exemplo de uso do ambiente COCAR com base em um projeto de software retirado da indústria de desenvolvimento de software brasileira.

No Capítulo 7 são apresentadas as conclusões deste trabalho bem como propostas para trabalhos futuros.

No Apêndice I encontram-se partes do Documento de Especificação de Requisitos do ambiente COCAR que segue as diretrizes para a escrita de documentos de requisitos definidas por Kawai (2005).

No Apêndice II são encontradas partes do Modelo de Casos de Uso projetado por meio da técnica TUCCA [Belgamo, 2004] baseando-se no documento de requisitos apresentado no Apêndice I.

Tanto o documento de Especificação de Requisitos quanto o Modelo de Casos de Uso do ambiente COCAR apresentados nos Apêndices I e II ficaram com um volume grande impossibilitando sua inclusão por completo neste trabalho de mestrado.

Capítulo 2
Engenharia de Requisitos

2.1 *Considerações Iniciais*

A Engenharia de Requisitos, segundo [Thayer & Dorfman, 1997], é a primeira etapa de todo o processo da Engenharia de Software, tendo como preocupação principal entender o que os usuários realmente precisam, traduzindo este entendimento num conjunto de especificações de requisitos.

Segundo Sommerville (2003), a definição clássica de qualidade de produto é que o produto desenvolvido deve estar de acordo com as suas especificação. Portanto, entender o que o usuário quer é primordial para o sucesso do software. Dessa forma, segundo o SWEBOK (2004), quando as práticas de Engenharia de Requisitos são realizadas de forma displicente, o projeto pode levar ao desenvolvimento de um produto de baixa qualidade que não atende a aquilo que os *stakeholders* pediram.

Como a estimativa de tamanho de software esta diretamente relacionada com os requisitos especificados para o software em questão [Sommerville, 2003] especificar bem o software é imprescindível para o sucesso da estimativa. Portanto, estudar a teoria associada à Engenharia de Requisitos é de fundamental importância para este trabalho.

Assim, o objetivo principal deste capítulo é apresentar os principais conceitos relacionados à Engenharia de Requisitos, o que está feito nos tópicos na Seção 2.2, segundo a visão do SWEBOK (2004). Em seguida, na Seção 2.3 apresentam-se as considerações finais.

2.2 *Engenharia de Requisitos – Principais Conceitos*

Segundo o SWEBOK (2004) a Engenharia de Requisitos é uma das áreas chaves da Engenharia de Software e é responsável por coletar, analisar, especificar e validar requisitos de software, tendo como principal preocupação traduzir a vontade dos usuários do software em um conjunto de especificação de requisitos de software.

Para melhor explicar a Engenharia de Requisitos, o SWEBOK (2004) a divide em sete subáreas, a saber:

- Fundamentos da Engenharia de Requisitos.
- Processo de Requisitos
- Elicitação de Requisitos.
- Análise de Requisitos.
- Especificação de Requisitos.
- Validação de Requisitos.
- Considerações Práticas.

A Figura 1 mostra cada um dessas subáreas com seus respectivos tópicos. Cada subárea será tratada no decorrer desta seção segundo as definições do próprio SWEBOK (2004).

As duas primeiras subáreas, “Fundamentos da Engenharia de Requisitos” e “Processo de Requisitos’”, trazem definições necessárias para o entendimento das quatro outras subáreas que as seguem: “Elicitação de Requisitos”, “Análise de Requisitos”, “Especificação de Requisitos” e “Validação de Requisitos”, considerados pelo SWEBOK (2004) como sendo as subáreas de maior interesse para a Engenharia de Requisitos em si. Por último há a subárea “Considerações Práticas” a qual mostra algumas práticas importantes que acompanham o processo de Engenharia de Requisitos.

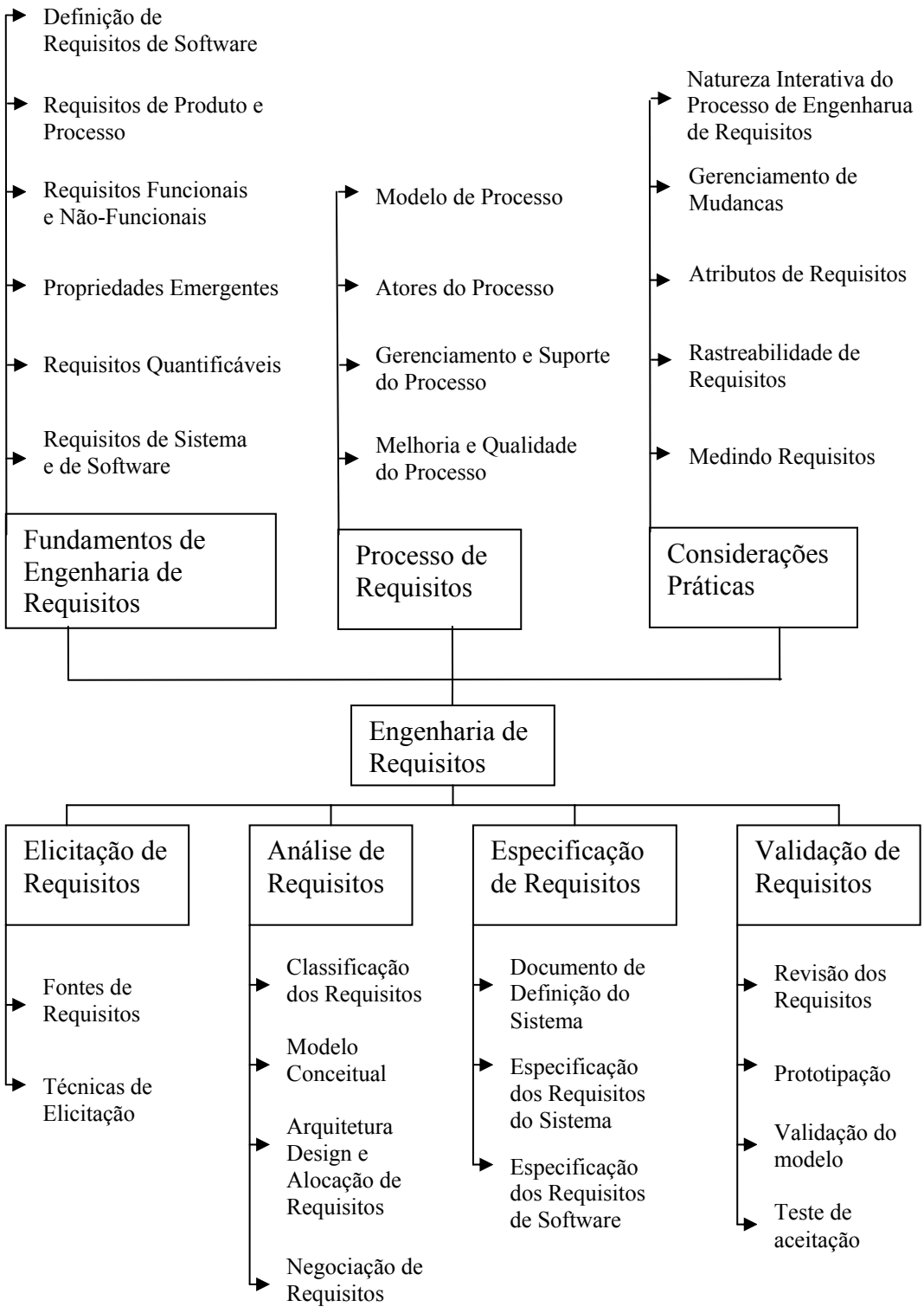

Figura 1– Subáreas da engenharia de requisitos [SWEBOK, 2004]

a) Fundamentos da Engenharia de Requisitos

Nesta subárea são tratadas algumas definições necessárias para o entendimento das próximas subáreas da Engenharia de Requisitos. Aqui se detalha a definição de requisitos, suas propriedades e suas classificações.

Definição de Requisitos de Software

Um requisito de software é uma propriedade que o software deve possuir para resolver um problema específico do mundo real. Dessa forma um requisito de software deve expressar a necessidade e as restrições colocadas em um produto de software.

Normalmente este problema do mundo real não é simples e o requisito de software para resolvê-lo é uma combinação complexa de requisitos conhecidos por diferentes pessoas atuando em diferentes papéis no contexto de onde o software irá operar.

Essas diferentes pessoas, que de alguma forma tem influência direta ou indireta sobre os requisitos do software, denominam-se *stakeholders* [Sommerville, 2005].

Dada a complexidade envolvendo os requisitos de um software uma característica importante que qualquer conjunto de requisitos de software deve possuir é de poderem ser verificados. Mesmo que isso seja difícil e custoso o engenheiro de software deve sempre assegurar que os requisitos sejam verificados para que o produto final possa atender às necessidades dos usuários.

Outras características importantes relacionadas aos requisitos de software são:

- Prioridade: usada para determinar a ordem com a qual os requisitos deverão ser implementados na fase de construção do software;
- Unicamente identificável: cada requisito tem que estar unicamente identificável pelo seu relacionado a um determinado objetivo sustentado por um determinado *stakeholder* ou por um determinado grupo de *stakeholders* que o software deve atender. Isto para possibilitar o controle de configuração de cada requisito e permitir o gerenciamento desse requisito por todo o ciclo de desenvolvimento do software.

Requisitos de Produto e Processo

Há uma diferença bastante sutil entre requisito de produto e requisito de processo.

O requisito de produto esta relacionado com algum atributo do software que esta sendo desenvolvido.

Já um requisito de processo esta relacionado com atributos do processo que será usado para construir o produto de software e podem ser impostos pela organização desenvolvedora do software, pelos clientes ou ainda por um órgão regulamentador.

Requisitos Funcionais e Não Funcionais

Requisitos funcionais são aqueles que descrevem uma função que o software deve executar, ao passo que requisitos não funcionais, ou requisitos de qualidade (como os requisitos não funcionais também são conhecidos) são aqueles que restringem a execução de alguma função.

Os requisitos não funcionais podem se dividir em vários subtipos:

- Requisitos de manutenção;
- Requisitos de desempenho;
- Requisitos de segurança;
- Requisitos de confiabilidade;

Propriedades Emergentes

São propriedades representadas por requisitos que sofrem influência de diferentes componentes do sistema. Como exemplo o SWEBOK (2004) cita o requisito de *throughput* de um *CallCenter* que é dependente de como o sistema telefônico, o sistema de informação e os operadores interagem em situação operacional real.

Requisitos Quantificáveis

Um requisito ser quantificável significa que ele é descrito utilizando-se medidas numéricas e não medidas qualitativas de forma que o requisito possa ser verificado através de testes

objetivos. Assim, ao final da implementação de um software, é possível identificar se o requisito solicitado foi completamente atendido pelo software ou não.

Um exemplo de requisito não quantificável é seguinte: "o software deve ser confiável". A descrição desse requisito permite uma discussão sobre o que é ser "confiável" ou não, ou seja, para a interpretação do desenvolvedor do software a palavra "confiável" pode significar algo completamente diferente da interpretação desta palavra por parte do *stakeholder* que solicitou esse requisito. Simplesmente ser confiável é uma medida qualitativa do software.

O requisito acima poderia ser quantificado da seguinte maneira: "a probabilidade do software gerar um *fatal error* deve ser de 0,00000001". Dessa forma não há dupla interpretação e esse requisito poderá ser verificado na entrega do produto através de testes objetivos.

Requisitos de Sistema e de Software

Para entender o que é um Requisito de Sistema é necessário definir o que é um sistema. Neste tópico, sistema é a combinação interativa de vários elementos para a realização de um objetivo definido. Esses elementos incluem software*,* hardware, *firmware,* pessoas, informações, técnicas serviços e outros.

Dessa forma, Requisitos de Sistema são os requisitos que descrevem as necessidades do mundo real que o sistema em questão irá atender.

Já os Requisito de Software, conforme definido no início desta seção, são os requisitos que descrevem as necessidades do mundo real que o software deve atender. Quando pensamos no software como parte de um sistema maior é dito que Requisitos de Software significam requisitos que devem ser implementados especificamente pelo elemento software para que o sistema possa atender as necessidades do mundo real para as quais ele foi projetado.

b) Processo de Requisitos

A forma como as subáreas da Engenharia de Requisitos foram estruturadas pelo SWEBOK (2004) pode induzir a interpretação do processo de Engenharia de Requisitos como sendo um processo "cascata" no qual as fases devem ser executadas uma após a outra sem possibilidade de retorno para fases já executadas.

Portanto, para evitar essa interpretação errônea do processo de Engenharia de Requisitos, o SWEBOK (2004) propõe a subárea de Processo de Requisitos o qual é tratado nesta seção.

Desta forma, neste tópico é melhor explicado o funcionamento do Modelo de Processo de Engenharia de Requisitos bem como algumas características importantes desse modelo.

Modelos de Processo

O processo de engenharia de requisitos não se configura tão somente como uma fase inicial isolada do ciclo de vida do desenvolvimento do software e sim como um processo que se inicia nas primeiras fases do desenvolvimento e segue sendo refinado conforme o ciclo de vida do software evolui.

Neste contexto, os requisitos do software podem sofrer alterações ao longo do desenvolvimento e por isso é interessante que o processo de Engenharia de Requisitos esteja preparado para tratar essas alterações dando suporte a gerência de configuração desses requisitos alterados.

Atores do Processo

Atores do processo de Engenharia de Requisitos são os papéis por meio dos quais os *stakeholders* "atuam" no processo. Visto que um software possui vários *stakeholders* que possuem necessidades específicas e participam do processo de Engenharia de Requisitos assumindo diferentes posturas. Tipicamente se tem os seguintes *satakeholders:*

- Usuários: grupo formado pelas pessoas que irão operar o software. Normalmente bastante heterogêneo contendo indivíduos com diferentes papéis e solicitando diferentes requisitos;
- Clientes: Grupo composto por pessoas que estão patrocinando a implementação do software ou ainda por pessoas que são o mercado alvo para o qual o software esta sendo construído;
- Analistas de Mercados: software implementado destinado para um mercado de massa (sem um cliente patrocinador específico) necessita do papel do analista de mercado para definir o que este mercado esta demandando.
- Reguladores: há vários domínios de aplicação que necessitam de órgãos reguladores como instituição bancárias e transporte público. Desta maneira se faz necessário que

o software implementado para estes domínios de aplicação sejam regulamentados pelas autoridades competentes.

- Engenheiros de Software: são as pessoas que tem verdadeiro interesse em gerar lucro com a implementação do software.

Dado o grande número de *stakeholders* normalmente envolvidos no processo de Engenharia de Requisitos é simplesmente impossível atender todos os requisitos de todos os *stakeholders*. Por isso, é função do Engenheiro de Requisitos intermediar os interesses dos *stakeholders*, elicitando todos os requisitos, entendendo a natureza da necessidade de cada requisito de cada *stakeholder* e negociando esses requisitos com os principais *stakeholders,* sempre levando em consideração as restrições orçamentárias e técnicas do projeto.

Gerenciamento e Suporte do Processo

O processo de engenharia de requisitos requer alguns recursos gerenciais que sejam capazes de fazer a ligação entre as atividades deste processo e custos, recursos humanos, treinamento e ferramentas. Por isso, é importante levar em consideração, desde o início da Engenharia de Requisitos, as práticas estabelecidas na área chave de Gerenciamento de Engenharia de Software abordadas no SWEBOK (2004).

Melhoria e Qualidade do Processo

Alguns dos principais papéis desenvolvidos pela Engenharia de Requisitos são: melhorar a satisfação do cliente em relação ao produto desenvolvido e gerenciar melhor custos e prazos do projeto. Desta forma é muito importante avaliar a qualidade do processo de Engenharia de Requisitos, analisando a sua eficácia e o quanto este processo esta contribuindo para o papel acima definido.

Portanto, é muito importante que o processo de Engenharia de Requisitos seja regido por padrões de qualidade e melhoria de processo de software e sistemas, principalmente no quesito qualidade do software e estimativas.

c) Elicitação de Requisitos

Segundo SWEBOK (2004), esta é a primeira fase de quatro que compõem o processo de Engenharia de Requisitos de Software, na qual o objetivo é o entendimento sobre o propósito do software que está sendo implementado. O principal interesse na elicitação de requisitos é identificar as fontes nas quais os requisitos de software serão elicitados e definir como o Engenheiro de Software poderá fazer o levantamento de requisitos.

Fontes de Requisitos

Requisitos podem vir de várias fontes e é essencial que todas essas fontes sejam identificadas e os seus impactos sobre os requisitos do software sejam avaliados.

Para ser capaz de identificar e avaliar as fontes de requisitos é importante que os engenheiros de software fiquem atentos aos seguintes itens:

i. Objetivos do Software – este item também é conhecido como fator crítico de sucesso do software e seu propósito é dar uma visão superficial sobre o software, ressaltando seus objetivos gerais e a motivação para a sua construção;
ii. Conhecimento do domínio – Os engenheiros de software devem adquirir conhecimento sobre o domínio da aplicação. Isto permite a eles inferir conhecimento intuitivo que os *stakeholders* não foram capazes de exteriorizar.
iii. *Stakeholders* – É de suma importância que os engenheiros de software não foquem toda a atenção para apenas um grupo de *stakeholders*. O engenheiro de software deve identificar, representar e gerenciar os pontos de vistas de vários e diferentes tipos de *stakeholders* para que o produto de software não seja concebido com uma visão unilateral das necessidades dos *stakeholders*.
iv. Ambiente operacional – Requisitos também serão derivados do ambiente no qual o software será executado e por isso é importante que o engenheiro de software conheça esse ambiente, já que ele pode influenciar bastantes custos e decisões no projeto do software.
v. Ambiente organizacional – Software é construído geralmente para dar suporte a processos de um determinado negócio e podem estar condicionados a uma estrutura, cultura e políticas internas de uma organização. Os engenheiros de software precisam ser sensíveis a esses aspectos, pois software que irão rodar nesses ambientes não podem

implicar (pelo menos não sem um planejamento prévio) em mudanças nas regras de negócio da organização.

Uma vez que as fontes de requisitos tenham sido identificadas e os seus impactos sobre os requisitos do software avaliados, o engenheiro de software pode começar a coletar os requisitos do sistema a partir dessas fontes.

Técnicas de Elicitação

A coleta dos requisitos do sistema é uma atividade muito difícil e o engenheiro de software deve estar ciente de que os usuários apresentam extrema dificuldade em descrever as tarefas do software, podendo deixar passar despercebido informações importantes. Por isso, mesmo com a cooperação dos *stakeholders*, os engenheiros de software têm um trabalho difícil na fase de elicitação dos requisitos.

Segundo Sommerville (2003) esta dificuldade ocorre pois:

i. Normalmente os *stakeholders* são generalistas na definição do que desejam do sistema;
ii. Para os *stakeholders* os requisitos do sistema são quase intuitivos pois, na grande maioria das vezes, expressam conhecimento implícito da própria área de atuação dos envolvidos;
iii. Como os *stakeholders* são formados por um grupo de pessoas grande e multidisciplinar, é freqüente acontecer conflitos entre os requisitos;
iv. Fatores políticos dentro de organização podem afetar diretamente os requisitos do software. Gerentes podem definir certos requisitos apenas para aumentar sua influência na organização;
v. É bastante comum que o ambiente de negócios associado ao sistema seja dinâmico, o que leva os requisitos a sofrerem alterações a todo momento;

Exatamente pela elicitação de requisitos ser um processo extremamente complicado, há várias técnicas para contemplar essa fase da Engenharia de Requisitos. As principais técnicas sugeridas no SWEBOK (2004) são:

i. Entrevistas – meio tradicional de elicitação de requisitos. É importante para o engenheiro de software saber das vantagens e limitações desse método e como conduzi-la bem.
ii. Cenários – meio valioso para prover contexto para a elicitação de requisitos juntamente com o usuário. Esse método é capaz de prover para o engenheiro de software em um conjunto de questões sobre as tarefas do usuário como "o que ... se isso?" ou ainda "como isto é feito ?". O tipo mais comum de cenário é o Casos de Uso.
iii. Protótipos – abordagem eficiente para tornar claro os requisitos ainda não entendidos e avaliar o sistema antes dele ser construído. Essa ferramenta pode atuar de forma similar aos cenários, provendo aos usuários um contexto no qual ele pode entender melhor qual informação ele precisa fornecer, para que o software fique o mais completo possível de acordo com suas necessidades. Há uma gama enorme de técnicas de prototipação que vão desde *mock-up* (técnica de prototipação que usa gráficos, imagens e ilustrações em papel para descrever telas e interação com o usuário) até versões executáveis não oficiais do software (versão beta para teste) que também são usados para a validação dos requisitos.
iv. Reuniões intermediadas – o propósito desta técnica é tentar alcançar um efeito sinérgico, partindo do princípio de que um grupo de pessoas pode contribuir mais para a coleta de requisitos do que cada pessoa individualmente contribuiria. Essa técnica pode trazer idéias por meio de *brainstorms* (reunião na qual várias idéias são enunciadas pelos participantes sem nenhuma restrição e depois as melhores são selecionadas para análise) que dificilmente seriam descobertas por técnicas de entrevistas usuais. Quando funciona bem, essa técnica pode trazer um conjunto de requisitos mais consistente do que outras técnicas.
v. Observação – a importância do contexto do software dentro do ambiente da organização tem conduzido adaptações das técnicas observacionais para a elicitação de requisitos de software. Engenheiros de software aprendem sobre as tarefas do usuário por meio de uma imersão em seu ambiente de trabalho, observando como os usuários interagem com seus software e entre eles. O grande problema dessa técnica é que os usuários tendem a se policiar mais em suas atividades quando tem alguma pessoa observando as suas atividades; sendo assim, o que o engenheiro de software vai observar é uma tarefa que nem sempre é executada da forma que está sendo mostrada.

De posse dos requisitos elicitados por meio dos interessados na construção do sistema, têm-se artefatos suficientes para se avançar para a próxima fase da engenharia de requisitos, a análise dos requisitos elicitados.

d) Análise de Requisitos

Na elaboração dos requisitos de um software deve-se tentar assegurar que estes sempre sejam descritos de forma a possibilitar a sua validação, a verificação de sua implementação e a estimativa de seus custos. Para tanto, modelos conceituais são muito aceitos nesta fase, contudo deve ser claro que, ao contrário da visão tradicional, esta fase não deve se resumir a apenas a elaboração desses modelos conceituais.

Assim, a fase de análise de requisitos também tem por objetivo:

- Detectar e resolver conflitos entre os requisitos.
- Descobrir os limites do software e como ele deve interagir com o seu ambiente.
- Elaborar requisitos de sistema para derivar requisitos de software.

Desta forma, nesta subárea, além dos modelos conceituais, serão tratados também assuntos importantes para garantir os objetivos dessa fase.

Classificação de Requisitos

Para melhor entender o domínio do software e suas funcionalidades, de acordo com o SWEBOK *Guide* 2004 [SWEBOK, 2004], os requisitos devem ser classificados. Existem várias formas para a classificação dos requisitos, por exemplo:

i. Requisito Funcional ou Requisito Não Funcional.
ii. Requisito Derivado ou Emergente: quando um requisito é derivado de um ou mais outros requisitos de alto nível, ou é uma propriedade emergente ou está sendo imposto diretamente por um *stakeholder* ou outra fonte de requisitos.
iii. Requisito de Produto ou Requisito de Processo: requisitos de processo podem restringir a escolha de um contratado ou o processo de engenharia de software a ser escolhido.
iv. Prioridade do requisito: em geral, quanto mais alta a prioridade do requisito mais essencial é o requisito para o software. A prioridade tem sempre que ser balanceada de acordo com o seu custo de desenvolvimento e implementação.

v. Escopo: refere-se à extensão com a qual o requisito afeta o software e seus componentes.
vi. Volatilidade e Estabilidade: Alguns requisitos irão mudar durante o ciclo de vida de desenvolvimento do software. Sendo assim, seria muito interessante classificar os requisitos de acordo com uma probabilidade de mudança que esses requisitos venham a ter. Sinalizar requisitos potencialmente voláteis pode ajudar ao engenheiro de software a estabelecer um projeto mais tolerante a mudanças.

Tendo classificado os requisitos e entendido melhor o escopo do problema pode-se seguir com o desenvolvimento de modelos que é considerado como a chave da análise de requisitos de software, pois esses modelos poderão ser usados para a especificação, validação e verificação de requisitos, projeto e teste do software.

Modelo Conceitual

Para efeitos práticos, SWEBOK (2004) define um modelo como sendo uma notação ou um conjunto de notações incluídas em um processo que guia a aplicação dessas notações.

Modelos conceituais devem representar o domínio do problema do mundo real através da modelagem de suas partes e dos relacionamentos e dependências entre essas partes. Portanto o principal objetivo da modelagem não é ser o início do projeto do software, apesar de poder ser usado para tal, mas sim ajudar a compreender melhor o problema do mundo real a ser implementado pelo software.

Segundo Pressman (2006), o modelo de análise deve atingir três objetivos principais: descrever o que o cliente exige; estabelecer a base para a criação de um projeto de software; definir um conjunto de requisitos que possam ser validados quando o software for construído.

Para tanto SWEBOK (2004) sugere vários modelos que podem ser construídos nesta fase, incluindo fluxo de dados, modelos de estado, modelos de rastreamento de eventos, modelos de interação com usuário, modelos de objeto, modelo de dados entre outros.

Para escolher esses modelos há vários fatores que devem ser levados em consideração, podendo-se citar [SEWBOK, 2004]:

i. A natureza do problema: Alguns tipos de software demandam que certos aspectos possam ser analisados particularmente e rigorosamente. Por exemplo, modelos de estado e fluxo de controle parecem ser mais importantes para software de tempo real do que para software de gerenciamento de informações.
ii. A perícia do engenheiro de software: É geralmente mais produtivo adotar um modelo com o qual o engenheiro de software tenha experiência.
iii. O processo de engenharia de requisitos do cliente: Clientes podem impor algum método ou ainda proibir que algum método seja utilizado. Este fator pode entrar em conflito com o fator anterior.
iv. A disponibilidade de métodos e ferramentas: Métodos para os quais não existem ferramentas ou treinamento que o dão suporte podem não ser aceitos mesmo que sejam mais indicados para tipos particulares de problemas.

Arquitetura Design e Alocação de Requisitos

Segundo SWEBOK (2004), o software implementará os requisitos que estão sendo elicitados por meio de vários componentes que serão desenvolvidos.

A arquitetura e modelagem desses componentes são realizadas na fase de Projeto do Software, contudo a identificação desses componentes e a alocação dos requisitos para esses componentes deve ser realizada nesta fase da engenharia de requisitos.

Realizar a alocação de requisitos com componentes significa identificar os componentes do software necessários para a implementação de um determinado requisito. Por isso, é comum dizer que esta fase da Engenharia de Requisitos se sobrepõe com a fase de Projeto de Software, fazendo com que o engenheiro de requisitos atue como um arquiteto de software.

SWEBOK(2004) chama a atenção para a importância da execução dessa atividade na fase de engenharia de requisitos, pois com ela é possível realizar mais uma análise detalhada dos requisitos de software, já que só desta maneira é possível identificar os componentes e os relacionamentos entre os componentes necessários para a sua implementação, ou seja, alocar componentes para requisitos demanda uma nova "rodada" detalhada de análise de cada requisitos.

Porém, o mapeamento de entidades do mundo real para componentes de software não é sempre óbvio e, por isso, o Projeto da Arquitetura é considerado como um tópico separado da

modelagem de requisitos e deve ser continuado em fases posteriores (fase de Projeto de Software) do ciclo de vida de desenvolvimento do software.

Estando os requisitos classificados, modelados e com o projeto da arquitetura do software iniciado, fica mais fácil identificar requisitos conflitantes, perfazendo a atividade de negociação de requisitos, que também é comumente chamada de resolução de conflitos.

Negociação de Requisitos

Essa atividade consiste em resolver problemas com requisitos conflitantes que ocorrem quando dois *stakeholders* requerem concomitantemente características incompatíveis entre requisitos e recursos ou entre requisitos funcionais e não funcionais. Na maioria das vezes não é aconselhável o engenheiro de software tomar uma decisão unilateral, sendo mais conveniente consultar os *stakeholders* para chegar a um consenso. Geralmente, por questões contratuais, é importante que esse tipo de decisão seja levada até o cliente.

Com isso, contemplam-se todas as atividades da fase de análise de requisitos e parte-se para a fase de especificação.

e) Especificação dos Requisitos

Segundo SWEBOK (2004), para muitas outras engenharias o termo especificação significa o ato de se atribuir valores ou limites para os objetivos do projeto que esta sendo desenvolvido.

Contudo, na área de desenvolvimento de software, tipicamente há um número pequeno desses valores, sendo que a ênfase esta na quantificação e no gerenciamento da interação entre o grande número de requisitos existente.

Sendo assim, na engenharia de software, a especificação de requisitos de software geralmente se refere à produção de um documento, ou à sua versão eletrônica equivalente, que possa ser sistematicamente revisto, avaliado e aprovado, buscando um melhor gerenciamento dos requisitos e suas relações.

Para sistemas complexos, geralmente, se desenvolvem três tipos desses documentos, a saber:

- Documento de Definição do Sistema.
- Documento de Requisitos do Sistema.

- Documento de Requisitos de Software.

Documento de Definição do Sistema

O documento de definição do sistema, também conhecido como documento de requisitos do usuário, guarda os requisitos do sistema definindo-os em alto nível segundo uma perspectiva do domínio da aplicação.

Sendo assim, este documento deve listar os requisitos de acordo com uma visão mais abstrata do sistema e deve ser escrito tendo como público alvo o próprio cliente/usuário do software e utilizar termos do escopo da aplicação (usando os termos do domínio para o qual a aplicação será desenvolvida), tendo como objetivo listar os requisitos do sistema, trazendo informações relevantes sobre o ambiente no qual o sistema irá operar, restrições da operação do sistema e ainda requisitos não funcionais.

Este documento também pode incluir alguns modelos conceituais do sistema como modelos de contexto e modelos de cenários.

Documento de Requisitos do Sistema

Para sistemas que reúnem uma grande quantidade de componentes de software e componentes "não-software" (hardware, por exemplo), costuma-se separar a especificação dos requisitos do sistema da especificação dos requisitos de software.

Para esse tipo de sistema o documento de especificação do sistema deverá ser elaborado. Contudo, segundo o SWEBOK(2004), especificar sistemas é uma atividade estritamente da área de engenharia de sistemas e foge do escopo desse trabalho, de qualquer maneira pode-se encontrar informações sobre esse tipo de documento e especificação em IEEE std 1233 (IEEE1234-98).

Documento de Requisitos de Software

O documento de especificação de requisitos do software estabelece as bases para um acordo entre clientes e desenvolvedores definindo o produto de software que irá ser construído.

A especificação de requisitos do software permite que seja feito um julgamento rigoroso dos requisitos antes que a fase de projeto se inicie, reduzindo, desta forma, possíveis re-projetos tardios do software.

Também é possível com este documento prover uma base para a realização de estimativas de custo, risco e cronograma do produto. Ainda algumas indústrias usam esse documento como entrada para se planejar a fase de validação e verificação de forma mais produtiva.

O documento de especificação de requisitos de software deve ser escrito usando linguagens formais ou semi-formais, sendo que a escolha de uma boa notação permite que certos requisitos ou aspectos da arquitetura do software sejam descritos com mais precisão e consistência do que se fossem especificados usando uma linguagem natural. Sendo assim, como regra geral, adotam-se notações sempre que se deseje descrever os requisitos da maneira mais precisa possível.

Por essa característica formal e técnica do documento de especificação de requisitos é comum este vir acompanhado com o documento de definição do sistema para que leitores leigos sejam capazes de compreender melhor o documento.

f) Validação de Requisitos

De acordo com SWEBOK(2004), os documentos de requisitos descritos no tópico "e" servem de entrada para esta fase da Engenharia de Requisitos, já que esses documentos serão aqui avaliados quanto ao grau de entendimento dos requisitos por parte dos engenheiros de software que irão desenvolver o produto, quanto à conformidade com padrões estabelecidos, clareza, consistência e completeza.

Sendo assim, o objetivo principal dessa atividade de validação é analisar os documentos de requisitos especificados na fase anterior para certificar-se de que o software que será construído é o software que o usuário espera ver. Ou seja, é nessa fase que se procura validar se os requisitos especificados realmente definem o sistema que o cliente deseja. [Sommerville, 2005]

Desta forma, considera-se aqui uma vantagem ter os documentos de requisitos escritos com uma linguagem formal, pois isto permite que a consistência e a clareza dos requisitos possam ser testadas.

Ainda nessa fase é importante que diferentes *stakeholders,* incluindo representantes dos clientes e desenvolvedores, revisem o(s) documento(s) de requisitos que ainda servirá como entrada para outras atividades derivadas do ciclo de vida do processo de desenvolvimento do software*,* sendo normal que essas revisões aconteçam mais de uma vez durante o processo de levantamento de requisitos.

Segundo SWEBOK(2004), para se realizar a validação de requisitos várias técnicas podem ser usadas, como por exemplo:

- Revisão de requisitos.
- Prototipação.
- Validação de modelo.
- Planejamento de Testes de aceitação.

Em seguida se explica cada uma delas:

Revisão de Requisitos

Este é o meio de validação mais comum e também conhecido como inspeção de requisitos. Nesta tarefa um grupo é responsabilizado por rever os documentos de requisitos em busca de erros, suposições errôneas, falta de clareza e desvios de padrões estabelecidos.

É aconselhável que o grupo que irá fazer a revisão dos requisitos tenha a presença de um cliente, para que a revisão seja direcionada sob o ponto de vista do usuário, que é um dos principais interessados.

Prototipação

A prototipação é um ótimo meio para validar a interpretação do engenheiro de software em relação aos requisitos coletados, além de configurar-se como uma ferramenta muito poderosa para se elicitar novos requisitos que ainda não foram descobertos.

Contudo, essa técnica apresenta algumas desvantagens. Uma delas é o fato da possibilidade do cliente se desviar das principais funcionalidades do sistema e ficar mais atento a problemas cosméticos ou a problemas de falta de qualidade do protótipo. Por isso, algumas

pessoas recomendam que a prototipação seja feita sem o uso de um software e sim através de *mock-ups* desenhados em papel.

Validação de Modelo

Há a possibilidade de se validar também os modelos conceituais estabelecidos durante a fase de análise. Para tanto existem várias técnicas de validação que buscam analisar a qualidade desses modelos. Se uma notação formal de especificação tiver sido usada é possível utilizar lógica formal para testar certas propriedades dos modelos.

Planejamento de Testes de Aceitação

Uma propriedade fundamental dos requisitos de software é que seja possível validar se o produto final os implementa. Desta forma, uma importante tarefa é planejar como cada requisito será verificado após a sua implementação e, na maioria dos casos, planejar testes de aceitação é uma ótima estratégia para isso.

A grande dificuldade em se planejar testes de aceitação para todos os requisitos encontra-se quando estes são requisitos não funcionais. Sendo assim é importante, antes de realizar o planejamento dos testes de aceitação, ainda na fase de análise, encontrar uma forma de quantificar os requisitos não funcionais, possibilitando o planejamento dos testes de aceitação para esses requisitos.

g) Considerações Práticas

Este último tópico abordado pelo SWEBOK (2004) concentra-se em mostrar alguns comportamentos dos requisitos ou do Processo de Engenharia de Requisitos que precisam ser entendidas na prática.

Assim nesta subárea da Engenharia de Requisitos serão tratados tópicos como a interatividade do processo de Engenharia de Requisitos e a Gerência de Requisitos

A natureza interativa do Processo de Engenharia de Requisitos

É praticamente impossível conceber um processo de Engenharia de Requisitos que seja linear e determinístico no qual os requisitos são elicitados através dos *stakeholders* uma única vez e

depois são armazenados dessa forma durante todo o processo de desenvolvimento do software.

Um ponto que deve ficar claro na Engenharia de Requisitos é que os requisitos de software mudam conforme o desenvolvimento do software avança. Essas mudanças podem ser causadas por alguma falha no momento da elicitação do requisitos, mas frequentemente essas mudanças são reflexos de uma mudança no ambiente no qual o software irá operar como por exemplo mudanças no mercado onde este software será vendido ou ainda mudanças nas regras de negócio do cliente.

Como essas mudanças nos requisitos são inevitáveis é muito importante que o processo de Engenharia de Requisitos não seja considerado como um simples processo que termina assim que a fase de projeto se inicia. Ao invés disso, o processo de Engenharia de Requisitos deve se preocupar em gerenciar os requisitos durante todo o processo de desenvolvimento de software gerenciando as mudanças dos requisitos ocorridas para garantir que cada uma dessas mudanças seja submetida a um processo de aprovação, tenha o seu histórico de alterações armazenado e tenha o seu impacto analisado.

Gerenciamento de Mudanças

Para o SWEBOK (2004), o gerenciamento de mudanças é a atividade central para o gerenciamento de requisitos pois neste tópico é que se descrevem todos os procedimentos necessários e as análises que devem ser feitas para controlar as alterações feitas nos requisitos.

Contudo, o SWEBOK (2004) prefere tratar esse tópico na área chave de Gerência de Configuração de Software.

Atributos dos Requisitos

Todo requisito deve ser composto não apenas pela descrição da especificação do que é requerido pelo software, mas também por informações complementares que ajudam a gerenciar e interpretar melhor os requisitos.

Essas informações adicionais incluem todos os atributos definidos no tópico "classificação de requisitos" e ainda o método ou o plano de teste de aceitação para que o requisito possa ser validado após a sua implementação.

Os requisitos também podem trazer informações relativas a fonte de onde ele foi elicitado ou ainda um histórico contendo todas as suas alterações.

Rastreabilidade de Requisitos

Rastrear um requisito significa conseguir identificar a fonte de onde esse requisito foi elicitado e conseguir predizer os efeitos desse requisito sobre o software. Assim um requisito pode ser rastreado para trás até atingirmos os *stakeholders* ou outros requisitos que o gerou ou ainda rastreado para frente até atingir outros requisitos ou as entidades de projeto construídas para atendê-lo.

Dessa forma, rastrear requisitos é importante para conseguir analisar o impacto desses requisitos sobre o software quando eles sofrem alguma alteração.

Medição de Requisitos

É importante saber qual é o tamanho dos requisitos no software pois com esse número fica possível predizer qual será o tamanho da alteração que um determinado requisito sofreu estimando desta forma o custo do desenvolvimento dessa alteração.

2.3 *Considerações Finais*

Neste capítulo foram apresentados os conceitos que envolvem o processo de Engenharia de Requisitos comentando-se as principais fases desse processo e mostrando as dificuldades e os cuidados que devem ser levados em consideração em cada uma dessas fases, de acordo com a visão do SWEBOK (2004).

A caracterização da fase de Engenharia de Requisitos é relevante para este trabalho, pois é nela que são gerados os principais artefatos tratados no ambiente desenvolvido que são o

documento de requisitos do sistema e o Modelo de Casos de Uso o qual será melhor explicado no próximo capítulo.

Capítulo 3
Modelagem de Requisitos com Casos de Uso

3.1 *Considerações Iniciais*

Neste capítulo serão apresentados os principais conceitos associados à técnica Casos de Uso, uma vez que o ambiente COCAR é essencialmente baseado no Modelo de Casos de Uso de um sistema, para o qual se deseja prover algumas informações, inclusive a funcionalidade de geração dos Pontos de Casos de Uso, que foi o principal objeto de estudo deste trabalho.

O conceito de Casos de Uso foi proposto inicialmente por Ivar Jacobson, em 1992, como parte de seu modelo de processo *Objectory* [Jacobson et al., 1992]. Embora essa técnica tenha sido incorporada à UML [OMG, 2005], como um dos modelos a ser construído, ela não é uma técnica restrita ao paradigma de orientação a objetos, podendo ser utilizada em qualquer contexto.

Por ser uma técnica simples e que facilita o entendimento do sistema que está sendo modelado, ressaltando as funcionalidades e as interações com os atores (usuários) dessas funcionalidades, ela tem sido muito utilizada na prática. Jacobson et al (1992) salientam que o Modelo de Casos de Uso também serve como um modelo central que deve ser utilizado para todos os demais modelos das próximas fases de Análise, Projeto, Implementação, Testes e Manutenção. No entanto, não existem diretrizes associadas à técnica que determinem como um modelo de casos de uso deva ser construído, permitindo com que a experiência e subjetividades tenham grande interferência nessa atividade. Em decorrência disso, o trabalho de Belgamo (2004), desenvolvido anteriormente, propôs a técnica de leitura TUCCA, cujo objetivo principal é a construção desses modelos e corresponde à funcionalidade básica do ambiente COCAR, pois, com base nesses modelos é que outras funcionalidades são oferecidas.

O capítulo está organizado da seguinte forma: na Seção 3.2 apresentam-se os conceitos relacionados com o Diagrama de Casos de Uso, na Seção 3.3 comenta-se sobre a Especificação dos Casos de Uso e na Seção 3.4 apresentam-se as Considerações Finais.

3.2 *Diagramas de Casos de Uso*

Os Modelos de Caso de Uso são representações pictóricas do documento de requisitos, que têm como objetivo mostrar a funcionalidade do sistema, bem como a interação dos usuários com o mesmo [Belgamo, 2004]. Esses modelos são compostos pelo Diagrama de Casos de Uso e pelas Especificações dos Casos de Uso e são considerados uma técnica para capturar os requisitos funcionais de um sistema, descrevendo as interações típicas entre os usuários desse sistema e o próprio sistema, além de fornecer uma narrativa de como o sistema é utilizado [Fowler, 2005].

Conceitualmente, de acordo com Booch et al. (2000), um Caso de Uso é uma descrição de um conjunto de seqüências de ações, incluindo também as variantes dessas ações, que um sistema executa para produzir um resultado de valor observável por um ator. Graficamente, o Caso de Uso é representado como uma elipse.

Sendo assim, um Caso de Uso é um tipo de classificador que representa: uma unidade coerente de funcionalidade provida por sistema ou por um sub-sistema; uma ou mais interações desta unidade com componentes externos (representado pelos atores); ações realizadas pelo próprio sistema [UML 2003].

Todo Caso de Uso deve ter um nome único (uma seqüência de caracteres textuais) que seja capaz de diferenciá-lo dos demais Casos de Uso.

A Figura 2 mostra a representação de um Caso de Uso com o seu nome.

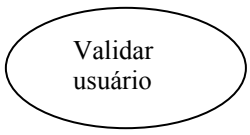

Figura 2 – Representação gráfica de um Caso de Uso

Para mostrar a interação entre o Caso de Uso com entidades externas a ele existe a figura do ator.

Um ator representa um conjunto coerente de papéis que os usuários dos Casos de Uso desempenham quando interagem com esses Casos de Uso. O ator não precisa necessariamente desempenhar apenas o papel de algum usuário humano; ele também pode desempenhar o papel de um dispositivo de hardware ou até de um outro sistema que interage com o sistema em questão. Por isso Fowler (2005) afirma que "papel" seria o termo mais correto para o que hoje a comunidade de Engenharia de Software chama de "ator".

Portanto, pode-se concluir que os Casos de Usos modelam aquilo que o sistema deve fazer enquanto os atores definem a interação de alguma coisa de fora com o sistema [Jacobson et al., 1992].

Schneider & Winters (2001) propõem algumas questões básicas para auxilio à identificação de atores no sistema, a saber:

- Quem usa o sistema?
- Quem instala o sistema?
- Quem inicia o sistema?
- Quem mantém o sistema?
- Quem finaliza o sistema?
- Quais outros sistemas fazem uso deste sistema?
- Quem recebe informações do sistema?
- Quem envia informações para o sistema?
- Alguma coisa acontece automaticamente em um determinado tempo?

A representação de um ator acontece por meio de figuras estilizadas que lembram um ser humano caricato. Esse tipo de representação é o bastante para o ator, pois como ele representa alguma coisa externa ao sistema, não é necessário descrevê-lo em detalhes [Jacobson et al., 1992].

Existe a possibilidade de se definirem grupos gerais de atores que depois poderão ser especializados, como por exemplo, pode-se definir um ator pessoa, que generaliza um grupo

de pessoas qualquer, e depois especializar outros atores como funcionário, professor e aluno, usando o relacionamento de generalização, como é mostrado na Figura 3.

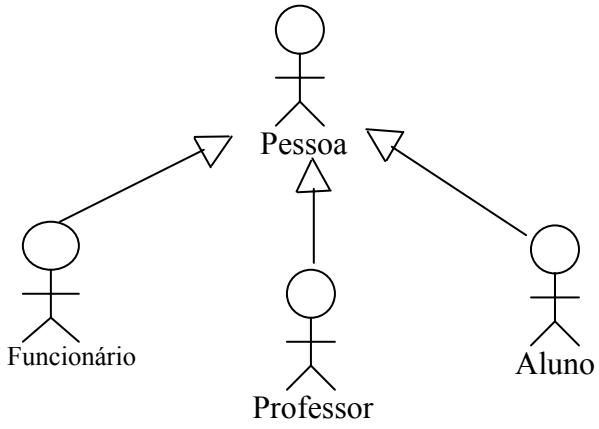

Figura 3 – Atores com relacionamento de generalização (adaptado de Boock et al., 2000)

O relacionamento de generalização é um relacionamento da UML que relaciona itens gerais em tipos mais específicos desses itens, ou seja, há a necessidade das partes que estão se relacionando serem de um mesmo tipo.

Cockburn (2001) inseriu uma forma de se classificar atores, a saber:

- Atores Primários: são aqueles que estão em constante contato com o sistema e por isso irão executar as principais tarefas do sistema. Este ator é freqüentemente aquele que aciona o Caso de Uso
- Atores Secundários: são aqueles que fazem o sistema funcionar através do fornecimento de um serviço para o mesmo. Sendo assim não estão em contato constante com o sistema e não costumam realizar as tarefas principais deixando isso para o ator principal. É importante identificar esses atores, pois a partir deles se identificam as interfaces externas que o sistema usará e os protocolos que cruzam esta interface.

A conexão entre os Casos de Uso e os atores acontece por meio de um relacionamento de associação. Este tipo de relacionamento é usado na UML como um relacionamento estrutural que especifica a conexão de objetos de um item com objetos de outro item. Quando usado para relacionar o Caso de Uso com o ator, essa associação indica que existe uma

comunicação entre esses itens, tendo cada parte a possibilidade de envio e recebimento de mensagens.

Os Casos de Uso podem ser relacionados entre si através de generalizações, inclusões ou extensões [Boock et al., 2000].

Em uma generalização de Casos de Uso o que acontece é que um Caso de Uso filho irá herdar o comportamento e significado do Caso de Uso pai e poderá acrescentar ou sobrescrever algum comportamento.

Para evitar a escrita de um mesmo fluxo de eventos várias vezes, existe a possibilidade da inclusão de Casos de Uso. Dessa forma, um Caso de Uso (que será chamado de Caso de Uso base), poderá incluir, em algum momento do seu fluxo de eventos, um outro Caso de Uso. Esse Caso de Uso incluído nunca poderá aparecer isoladamente sendo que no momento da inclusão o seu comportamento é explicitamente incorporado pelo Caso de Uso base que o está incluindo.

Um relacionamento de inclusão pode ser representado como uma dependência estereotipada com *include.* Dependência é um relacionamento de utilização definido pela UML. Exemplos de inclusão podem ser vistos na Figura 4.

Contudo, se o comportamento a ser incluído em um Caso de Uso não for obrigatório, ou seja, há um fluxo de evento no Caso de Uso base que não passa pelo Caso de Uso que esta sendo incluído, o que acontece então é uma extensão de um Caso de Uso. Sendo assim, o Caso de Uso base poderá aparecer isolado e, sob certas condições, seu comportamento poderá ser estendido pelo comportamento de um outro Caso de Uso. O relacionamento estendido é representado como uma dependência estereotipada como *extend.* Exemplos de extensão podem ser vistos na Figura 4.

Apenas o nome e a interação do Caso de Uso com atores não são suficientes para se descrever o comportamento desse Caso de Uso. É preciso algo que deixe claro o comportamento dos Casos de Uso para qualquer pessoa que precise entender o modelo. Para isso, elaboram-se as Especificações dos Casos de Uso. Na Especificação dos Casos de Uso, existe o que se chama de fluxo de eventos no qual se pode incluir como e quando o Caso de Uso inicia e termina, quando o Caso de Uso interage com os atores, quais objetos são

transferidos e o fluxo básico e alternativo do comportamento, também conhecidos como curso normal e curso alternativo.

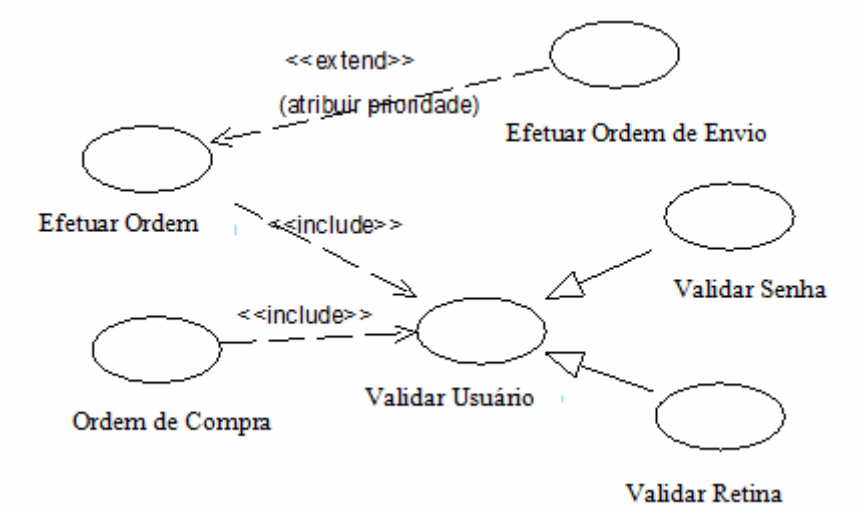

Figura 4 – Casos de Uso e seus relacionamentos (adaptado de Boock et al., 2000)

O fluxo de eventos de um Caso de Uso pode ser especificado de diversas formas: por meio de um texto estruturado informal, texto estruturado formal (com pré e pós condições), pseudocódigo, fluxograma, redes de Petri ou linguagens de programação [Sommerville, 2005].

3.3 *Especificação de Casos de Uso*

Segundo Fowler (2005), apesar da UML adotar os Casos de Uso como um de seus diagramas, ela apresenta uma definição bastante superficial da técnica, se limitando apenas a formalizar o Diagrama de Caso de Uso, sem preocupação em padronizar a Especificação dos mesmos.

Essa falta de padronização sobre a forma de escrita do fluxo de eventos de um Caso de Uso dificulta a determinação de métricas a partir desse modelo [Damodaran, 2003], apesar de várias técnicas usadas atualmente considerarem os Casos de Uso como sendo um indicador de tamanho e complexidade de um sistema [Vinsen et al, 2004].

Por outro lado, Belgamo (2004) mostra que existem alguns autores que sugerem a utilização de *templates* na Especificação de Casos de Uso, como por exemplo, [Kulak & Guiney, 2000; Ryser & Glinz, 2000; Cockburn, 2001, Schneider & Winters, 2001].

No trabalho de Schneider & Winters (2001) são sugeridas algumas questões que auxiliam na identificação dos atores do Modelo de Casos de Uso, bem como algumas diretrizes para a criação de Casos de Uso.

Kulak & Guiney (2000), sugerem um método com quatro interações pré-estabelecidas, com base no qual o Modelo de Casos de Uso evolui gradativamente.

Ryser & Glinz, (1999) desenvolveram um método denominado SCENT (SCENarios-Based Validation and Test of Software) no qual a partir do Modelo de Casos de Uso são elaborados Statecharts que serão utilizados na criação de Casos de Teste. Embora o principal objetivo do trabalho seja o desenvolvimento de Casos de Testes um formulário de especificação de Casos de Uso é apresentado neste trabalho.

Já Cockburn (2001) trabalha com o conceito de objetivos para transformá-los em Casos de Uso. Desta forma, os objetivos de um determinado sistema são classificados em três possíveis níveis: usuário (são objetivos que mostram as necessidades que o ator tem em relação ao sistema), resumo (são os que mostram as necessidades que contexto no qual o sistema esta inserido tem) e sub-funções (são os objetivos necessários para que os objetivos do usuário possam ser executados).

Dos trabalhos acima analisados nenhuma proposta reúne todas as necessidades que a especificação de Casos de Uso exige em um só *template.* Por isso, analisando essas propostas, Belgamo (2004) definiu o *template* de Especificação de Casos de Uso apresentado no Quadro 1, o qual é utilizado no ambiente COCAR, como resultado da aplicação da TUCCA. Assim, com base na maneira em que as informações relacionadas ao Caso de Uso são armazenadas nesse *template*, podem-se calcular os Pontos de Casos de Uso, que é o principal objetivo deste trabalho.

Quadro 1 – Formulário de Especificação de Casos de Uso [Belgamo, 2004]

Especificação do Caso de Uso	Número:	Número do Caso de Uso
Nome do Caso de Uso	Nome do Caso de Uso criado.	
Descrição ou Resumo	Pequena descrição do Caso de Uso.	
Ator Participante	Proprietário da informação.	
Ator Operador	Manuseia o computador.	
Ator Genérico	Ator genérico criado pela análise dos atores participantes.	
Pré-Condição	Condições que devem ser verdadeiras para que o Caso de Uso possa ser realizado.	
Curso Normal	Corresponde a um fluxo de eventos.	
Curso Alternativo	Corresponde a fluxos de eventos, porém mostrando os caminhos menos comuns.	
Evento Disparador	Descreve o critério de entrada para o Caso de Uso sendo especificado.	
Include	Número de Casos de Uso relacionados pelo esteriótipo <<include>>.	
Extend	Número de Casos de Uso relacionados pelo esteriótipo <<extend>>.	
Requisitos Funcionais	Os números de requisitos funcionais associados ao Caso de Uso criado.	
Requisitos Não - Funcionais	Os números de requisitos não funcionais associados ao Caso de Uso criado.	
Autor	A pessoa que criou a especificação.	
Data	A data de criação da especificação	
Versão	Código associado à versão da especificação	

3.4 *Considerações Finais*

Apresentaram-se neste capítulo os principais conceitos sobre o Modelo de Casos de Uso, no que diz respeito tanto ao diagrama como à especificação dos Casos de Uso. Por ser amplamente utilizada, facilitar a comunicação entre desenvolvedores e clientes e por prover

uma representação do sistema que dá apoio a várias outras atividades do desenvolvimento de software, essa técnica é a base da implementação do ambiente COCAR, no qual, a partir da elaboração desses modelos, outras funcionalidades são disponibilizadas, sendo uma delas o cálculo dos Pontos de Casos de Uso, alvo deste trabalho.

Assim, dada uma visão geral sobre os Modelos de Casos de Uso neste capítulo, no próximo capítulo será apresentada a métrica Pontos de Casos de Uso, que pode dar suporte às estimativas sobre o tamanho do software a ser desenvolvido e tempo de desenvolvimento, entre outras informações.

Capítulo 4
Métricas de Tamanho de Software

4.1 *Considerações Iniciais*

Neste capítulo serão comentadas duas métricas bastante conhecidas e utilizadas para medir o tamanho de um software: Pontos por Função [Albrench, 1979] e Pontos de Casos de Uso [Karner, 1993]. Embora o valor numérico produzido por elas represente a complexidade associada ao software que se deseja desenvolver, com base nesse valor de complexidade calculam-se vários outros parâmetros, inclusive o tamanho do software, uma vez que é possível transformar esse valor em linhas de código, por exemplo. Daí dizer-se que tanto Pontos por Função como Pontos de Casos de Uso são consideradas métricas para determinar o tamanho do software. Essa consideração pode ser observada em diversos autores, como por exemplo, Andrade (2004) que afirma existir várias métricas de estimativa de tamanho de software sendo que a maioria é desenvolvida com base nas funções do software como: Análise de Pontos de Função, Bang, Feature Points, *Boeing's* ED *Function Points,* Mark II, Pontos de Caso de Uso e COSMIC *Full Function Point*. Como citado no Capítulo 1, vários outros autores fazem a mesma consideração [Medeiros, 2004, Sommerville, 2003; Anda et al., 2001; Smith, 1999, Fetcke et al., 1998].

Com o crescimento da complexidade dos software desenvolvidos nos dias de hoje, a estimativa de esforço passou a ser uma atividade crucial no planejamento e monitoramento do desenvolvimento, tendo como objetivo entregar o produto no prazo correto e dentro do orçamento previsto.

Nesse contexto, a medida de tamanho do software representa uma das mais importantes características do produto de software, uma vez que ela tem impacto direto no esforço de

desenvolvimento e na gestão do projeto, mostrando-se como um indicador da quantidade de trabalho, sendo possível definir com base nessa medida, o esforço, o prazo e os custos necessários para o desenvolvimento do produto [Costagliola et al., 2006; Andrade, 2004].

Neste capítulo serão então abordados duas técnicas para a estimativa de tamanho de software, as quais foram alvos de estudo neste trabalho e na ferramenta decorrente dele. Assim, na Seção 3.2 apresentam-se os Pontos de Caso de Uso, na Seção 3.3 os Pontos por Função e, na Seção 3.4 as Considerações Finais.

4.2 *Pontos de Casos de Uso*

Como visto no Capítulo 2 os Modelos de Casos de Uso são representações do documento de requisitos que modelam a funcionalidade do sistema, bem como a interação com esse sistema por parte de usuários e outros sistemas externos [Belgamo, 2004].

Como é bastante interessante medir o tamanho do software logo em sua fase inicial para subsidiar o planejamento e negociação de prazos, esforços, recursos e custos com o cliente, as empresas demandam necessidade por técnicas que proporcionem maior precisão nas métricas iniciais de tamanho de software [Andrade, 2004].

Pensando no processo de desenvolvimento de software estabelecido por Jacobson et al (1992), chamado *Objectory,* Karner (1993) propõe a métrica de tamanho de software Pontos de Casos de Uso (PCU)*,* que é baseada na técnica proposta por Albrecht A. J. [Albrecht , 1979], os Pontos por Função*,* que foi vista na seção anterior.

Segundo Karner (1993), os Pontos de Casos de Uso é capaz de predizer o total de recursos necessários para a construção do software, na fase inicial do processo de desenvolvimento de software, usando como artefato de entrada o Modelo de Casos de Uso.

O modelo sugerido por Karner (1993) começa propondo a contagem de um fator chamado Pontos de Casos de Uso Não Ajustados (PCUNA). Para computar esse fator é preciso, primeiramente, classificar cada ator como simples, médio ou complexo de acordo com os critérios definidos na Tabela 1.

Tabela 1– Forma de classificação dos atores com seus respectivos pesos [Karner, 1993].

Complexidade	Definição	Peso
SIMPLES	Um ator é considerado simples se ele representa um outro sistema com uma interface programável de aplicação definida.	1
MÉDIO	Um ator é considerado como médio se: 1. Ele interage com um outro sistema usando para isso um protocolo. 2. Ele interage com o sistema através de linhas de comando.	2
COMPLEXO	Um ator é considerado complexo se ele interage com o sistema através de uma interface gráfica com o usuário.	3

Posteriormente, é necessário classificar cada Caso de Uso também como simples, médio ou complexo utilizando para isso o critério proposto na Tabela 2.

Tabela 2 – Forma de classificação dos Casos de Uso com seus respectivos pesos [Karner, 1993]

Complexidade	Definição	Peso
SIMPLES	Um Caso de Uso é simples se ele tem 3 ou menos transações incluindo aquelas do curso alternativo. Ou seja, deve ser possível implementar o Caso de Uso usando para isso menos que 5 objetos lógicos.	5
MÉDIO	Um Caso de Uso é médio se ele tem de 3 a 7 transações incluindo aquelas do curso alternativo. Ou seja, deve ser possível implementar o Caso de Uso usando para isso de 5 a 10 objetos lógicos.	10
COMPLEXO	Um Caso de Uso é complexo se ele tem mais do que 7 transações incluindo aquelas do curso alternativo. Ou seja, deve ser possível implementar o Caso de Uso usando para isso mais que 10 objetos lógicos.	15

Após isso, é preciso calcular a soma dos pesos dos atores e Caso de Usos, de acordo com a classificação feita anteriormente e utilizando a Equação 1, para se chegar ao fator PCUNA, na qual n_i é o número de itens de variedade i e Pi é o peso dado a variedade de i.

$$\text{PCUNA} = \sum_{i=1}^{6} n_i * P_i \qquad \textbf{Equação 1}$$

Caso os Diagramas de Casos de Uso sejam as únicas informações disponíveis sobre o sistema, o PCUNA já pode ser usado para a realização da medida do tamanho de software. Contudo, se existirem informações sobre o ambiente de implementação do projeto e/ou informações sobre a complexidade técnica de se implementar o projeto, outros fatores como o Fator de Complexidade Técnica (FCT) e o Fator Ambiental (FA) devem ser calculados para se alcançar o índice de Pontos de Casos de Uso.

O FCT é um fator usado para balancear o PCUNA ajustando esse último de acordo com a dificuldade técnica de se construir o sistema. Esse fator é muito parecido com aquele utilizado na técnica Pontos por Função, com a diferença da adição de alguns fatores e a remoção de outros, além de uma pequena alteração no peso dado a cada fator.

Tabela 3 – Fatores que contribuem para a complexidade do sistema [Medeiros, 2004; Karner, 1993]

F_i	Fatores que contribuem para a complexidade do sistema	P_i
F_1	Distribuição do sistema.	2
F_2	Resposta aos objetivos de desempenho	1
F_3	Eficiência do usuário final.	1
F_4	Complexidade do processamento interno.	1
F_5	Código deve ser reutilizado	1
F_6	Facilidade de instalação	0.5
F_7	Facilidade de uso	0.5
F_8	Portabilidade	2
F_9	Fácil de alterar	1
F_{10}	Concorrência	1
F_{11}	*Feature* de segurança	1
F_{12}	Acesso direto a dispositivos de parceiros	1
F_{13}	Treinamento especial aos usuários	1

$$FCT = C_1 + C_2 * \sum_{i=1}^{13} F_i * P_i \qquad \textbf{Equação 2}$$

Onde: $C_1 = 0.6$ e $C_2 = 0.01$.

Para se calcular o FCT é necessário atribuir uma nota de 0 a 5 para cada um dos fatores presentes na Tabela 3, considerando 0 para quando o fator for irrelevante para o sistema e 5 para quando o fator for essencial para o sistema, e aplicar o resultado na Equação 2.

Se o fator de complexidade técnica for irrelevante para o sistema assume-se o valor 3 para todos os F_i resultando em um valor de aproximadamente 1 para o FCT.

Já o Fator Ambiental é um outro fator usado também para balancear o PCUNA e o seu cálculo é muito parecido com do FCT, sendo também necessário atribuir uma nota de 0 a 5 para cada um dos fatores presentes na Tabela 4, considerando 0 para quando o fator for irrelevante para o sistema e 5 para quando o fator for essencial para o sistema, e aplicar o resultado na Equação 3.

Tabela 4– Fatores que contribuem para a eficiência [Medeiros, 2004; Karner, 1993]

F_i	Fatores que contribuem para a eficiência	P_i
F_1	Familiaridade com o Processo de Interativo Unificado	1.5
F_2	Experiência na Aplicação	0.5
F_3	Experiência com orientação a objeto	1
F_4	Capacidade de Liderança de Análise	0.5
F_5	Motivação	1
F_6	Estabilidade de Requisitos	2
F_7	Consultores *Part-Time*	-1
F_8	Dificuldade de Programação na Linguagem	-1

$$FA = C_1 + C_2 * \sum_{i=1}^{8} F_i * P_i \qquad \textbf{Equação 3}$$

Onde: $C_1 = 0.6$ e $C_2 = 0.01$.

Se o fator de ambiente for irrelevante para o sistema, assume-se o valor 3 para todos os F_i resultando em um valor de aproximadamente 1 para ele.

Após o cálculo do PCUNA, do FCT e do FA, finalmente pode-se chegar ao valor dos Pontos de Casos de Uso (PCU) pela Equação 4.

$$PCU = PCUNA * FCT * FA \qquad \textbf{Equação 4}$$

Com o valor dos Pontos de Casos de Uso calculado é possível realizar a estimativa da quantidade de esforço necessária para a construção do sistema em questão.

Segundo Karner(1993) *apud* Anda (2002), Karner propõem um fator de 20 horas homens para o desenvolvimento de cada Ponto de Casos de Uso, já Scheneider and Winters recomendam que esse valor deve ser definido de acordo com o valor do Fator Ambiental do projeto da seguinte forma:

- Dentre os fatores ambientais de 1 a 6, conte o número de vezes que aparece a classificação menor do que três;
- Dentre os fatores ambientais de 7 a 8, conte o número de vezes que aparece a classificação maior do que três;
- Some os valores encontrados nos dois itens acima:
    - Se o total for 2 ou menos que 2, então use o fator de 20 homens/hora por Ponto de Casos de Uso para o cálculo da estimativa de esforço;
    - Se o total for 3 ou 4, então use o fator de 28 horas homens por Ponto de Casos de Uso para o cálculo da estimativa de esforço;
    - Se o total for maior do que quatro então o autor recomenda que alterações sejam feitas no projeto de forma que esse número possa ser ajustado ou então que se use o fator de 36 horas homens por Ponto de Casos de Uso para o cálculo da estimativa de esforço.

Também, da mesma forma que o PF, o PCU pode ser usado como dado de entrada para algum método de estimativa como o COCOMO [Boehm et. al, 1995], permitindo a geração de várias informações relevantes para o plano de desenvolvimento de software.

4.3 *Pontos por Função*

A métrica Análise de Pontos por Função tem por objetivo caracterizar a complexidade da funcionalidade do software a ser desenvolvido, tendo como base alguns itens que estão geralmente presentes nos produtos de software, como arquivos, transações de entrada, de saída, etc. e algumas questões que procuram caracterizar outros aspectos que não ligados diretamente à funcionalidade do produto, mas que podem interferir na complexidade da implementação do produto como exigência de disponibilidade de recursos computacionais, processamento de dados distribuído, entre outros.

O processo de contagem da métrica de Análise de Pontos por Função, que é composta dos Pontos por Função Bruto e dos Pontos por Função Ajustados, é dividido em 7 etapas, a saber (IFPUG, 1999):

i. **Determinar o tipo de contagem** – A Análise de Pontos por Função pode ser aplicada para calcular 3 tipos de contagem dependendo das seguintes situações: projetos em desenvolvimento, aplicações em uso ou projetos de manutenção.
ii. **Determinar o escopo da contagem e as fronteiras da aplicação** – Para determinar o escopo de aplicação da métrica, é fundamental que fiquem claramente definidas as fronteiras da aplicação em relação às aplicações externas e ao domínio do usuário. Esse limite deverá levar em consideração o propósito para o qual a Análise de Pontos por Função será usada; como e quais os sistemas irão manter os dados; e a identificação das regras de negócio que darão suporte à aplicação, [Longstreet, 2004].
iii. **Contar as funções de dados –** Funções de dados são funcionalidades solicitadas pelo usuário e que representam requisitos de dados internos e externos. As funções de dados são divididas em arquivos lógicos internos (ALI) e arquivos de interface externa (AIE). É importante deixar claro que o termo *arquivo* não deve ser interpretado no seu sentido tradicional de processamento de dados e sim como sendo apenas um conjunto de dados lógicos agrupados, mas não necessariamente em uma implementação física.

Para contar as funções de dados é necessário identificar os ALI e os AIE; e determinar suas complexidades. Um ALI é um grupo lógico de informações de controle ou dados identificado pelo usuário e mantido dentro dos limites da aplicação. Já um AIE é um grupo lógico de

informações de controle ou de dados identificados pelo usuário e relacionado com a aplicação que está sendo contada, mas atualizados e mantidos dentro dos limites de uma outra aplicação. Um AIE de uma aplicação sempre será contado como um ALI em uma outra aplicação [Andrade, 2004].

A complexidade de cada ALI/AIE é baseada no número de itens de dados e de registros lógicos identificados e associados a cada ALI/AIE, sendo que um item de dado é definido como um campo não repetido, único e identificável pelo usuário. E um registro lógico é um subgrupo de elementos de dados de um ALI/AIE reconhecido pelo usuário, podendo existir dois tipos de subgrupos: opcional e obrigatório.

Com a identificação e contagem de cada item de dados e de seus registros lógicos, pode-se classificar um ALI, de acordo com o Quadro 2, como sendo de complexidade baixa, média ou alta o que contribui, respectivamente, com 7, 10 ou 15 Pontos por Função Não Ajustados para a contagem final dos Pontos por Função Não Ajustados.

Um AIE também é classificado como sendo baixo, médio ou alto de acordo com o número de itens de dados e registros lógicos identificados contribuindo respectivamente com 5,7e 10 Pontos por Função Não Ajustados como também mostra o Quadro 2.

Quadro 2 – Complexidade funcional [IFPUG, 1999].

	Itens de dados		
Registros Lógicos	1 a 19	20 a 50	51 ou mais
1	BAIXA	BAIXA	MÉDIA
2 a 5	BAIXA	MÉDIA	ALTA
6 ou mais	MÉDIA	ALTA	ALTA

iv. **Contar as funções transacionais –** funções transacionais são as funcionalidades de processamento dos dados providas para o usuário através da aplicação. Essas funções são definidas como entradas externas (EE), saídas externas (SE) e consultas externas (CE).

Para entender a contagem das funções transacionais é necessário definir o que é processo elementar. Desta maneira, processo elementar é a menor unidade funcional que possui significado no software que se está desenvolvendo.

Assim, para a identificação dos processos elementares devem-se observar as atividades realizadas pelos usuários na aplicação e que obedeçam as duas regras abaixo estabelecidas.

a) O processo é a menor atividade possível e que ainda assim tenha significado para o usuário.
b) O processo é auto contido e deixa as regras de negócio do sistema em um estado consistente.

Uma EE é um processo elementar que processa dados ou entradas de controle que vêm de fora da fronteira da aplicação. O primeiro objetivo de uma EE é manter uma ou mais ALI e/ou alterar o comportamento do sistema.

Uma SE é um processo elementar que envia dados ou informações de controle para fora da fronteira da aplicação. O objetivo principal da SE é apresentar informações para o usuário através do processamento de outras informações ou ainda apresentar a resposta de um dado ou informação de controle.

Uma CE é um processo elementar que envia dados ou informações de controle para fora da fronteira da aplicação. O objetivo maior de uma CE é apresentar informações para o usuário através do retorno de um dado ou de uma informação de controle que venha de uma ALI ou de uma AIE.

Como se pode perceber a principal diferença entre as funções transacionais são o que elas trazem como objetivo principal. O número de CE, SE e EE e suas complexidades funcionais relativas determinam a contribuição das funções transacionais para o cálculo da contagem dos Pontos por Função Não Ajustados. Portanto, deve-se atribuir um número de complexidade em todos os CE, SE e EE identificados baseando-se no número de arquivos referenciados e itens de dados.

A atribuição de complexidade para uma EE é diferente da atribuição de complexidade de uma SE que por sua vez é semelhante a uma atribuição de complexidade de uma CE.

Desta forma, após contar os itens de dados e os arquivos referenciados pela EE deve-se determinar a sua contribuição para a contagem final dos Pontos por Função Não Ajustados como 3, 4 ou 6 Pontos por Função Não Ajustados respectivamente a uma complexidade baixa, média ou alta. Para realizar a classificação da complexidade de uma EE usa-se o Quadro 3.

Quadro 3 – Complexidade de EE (IFPUG, 1999).

Arquivos Referenciados	Itens de dados		
	1 – 4	5 – 15	16 - mais
0 – 1	Baixa	Baixa	Média
2	Baixa	Média	Alta
3 – mais	Média	Alta	Alta

Depois de concluída a contagem dos itens de dados e dos arquivos referenciados nas SEs e nas CEs deve-se usar o Quadro 4 para classificar todos os SE e todos os CE como sendo de complexidade baixa, média e alta. A contribuição de Pontos por Função Não Ajustados para a SE deve ser de 4 para uma complexidade baixa, 5 para a media e 6 para a alta. Mas para uma CE a contribuição de Pontos por Função Não Ajustados deve ser de 3 para a complexidade baixa, 4 para a média e 6 para a alta.

v. **Determinar o ponto de função não ajustado –** o ponto de função não ajustado (PFNA) reflete a contagem especifica das funcionalidades do sistema providas para o usuário. Essa contagem é baseada "no que" será entregue para o usuário, e não em "como" isto será entregue. Sendo assim, só estão sendo levados em consideração nesta contagem os requisitos dos usuários.

Quadro 4 – Complexidade de SE [IFPUG, 1999].

Arquivos Referenciados	Itens de dados		
	1 – 5	6 – 19	20 - mais
0 – 1	Baixa	Baixa	Média
2 – 3	Baixa	Média	Alta
4 – mais	Média	Alta	Alta

Para o cálculo do PFNA deve-se multiplicar o total de ALI, AIE, EE, SE e CE por suas respectivas contribuições, adquiridas através da atribuição da complexidade. Esse total é o valor dos Pontos por Função Não Ajustados. O Quadro 5 exemplifica uma contagem de Pontos por Função não ajustado no qual se tem um ALI com complexidade alta, dois AIE com complexidade média e um AIE com complexidade alta, uma EE com complexidade baixa, uma SE com complexidade média e duas CE com complexidade alta.

Quadro 5 – Exemplo de cálculo dos Pontos por Função Não Ajustados [IFPUG, 1999].

Tipo de função	Qdde	Complexidade	Contribuição	Multiplicação	Total	Tipo de função	Qdde	Complexidade	Peso	Multiplicação	Total
ALI	0	BAIXA	7	0		**SE**	0	BAIXA	4	0	
	0	MÉDIA	10	0			1	MÉDIA	5	5	
	1	ALTA	15	15	15		0	ALTA	6	0	5
AIE	0	BAIXA	5	0		**CE**	0	BAIXA	3	0	
	2	MÉDIA	7	14			0	MÉDIA	4	0	
	1	ALTA	10	10	24		2	ALTA	6	12	12
EE	1	BAIXA	3	3							
	0	MÉDIA	4	0		**Total**				**PFNA = 59**	
	0	ALTA	6	0	3						

vi. **Determinar o fator de ajuste –** O valor do fator de ajuste (VFA) é baseado em 14 características gerais do sistema que pontuam as funcionalidades do sistema que esta sendo contado. Cada uma dessas características é associada a descrições que auxiliam na determinação do grau de influência dessas características no sistema. O grau de influência dessas características varia em uma escala de zero a cinco, significando não influencia para o 0 até fortemente influente para o 5.

As características gerais do sistema, que estão mostradas no Quadro 6, são 14 itens que avaliam, de maneira geral, o grau de complexidade do sistema.

Quadro 6 - Características gerais do sistema [Andrade, 2004].

Características Gerais do Sistema	
Comunicação de dados	Atualização on-line
Processamento distribuído	Processamento complexo
Desempenho	Reutilização de código
Configuração altamente utilizada	Facilidade de implantação
Volume de transações	Facilidade operacional
Entrada de dados on-line	Múltiplos locais
Eficiência do usuário final	Facilidade de mudanças

Juntamente com o seu grau de influência, as características gerais do sistema, são responsáveis pelo cálculo do valor do fator de ajuste que é capaz de ajustar os Pontos por Função Não Ajustados em +/- 35%, produzindo o valor dos Pontos por Função Ajustado.

O Quadro 7 mostra os três passos necessários para se determinar o VFA.

Quadro 7 - Passos para a determinação do VFA [IFPUG, 1999].

Passo	Ações
1	Avaliar cada uma das 14 características gerais do sistema em uma escala de zero a cinco correspondendo a não influência até a fortemente influente respectivamente, determinando o grau de influência.
2	Somar os graus de influência encontrados para gerar o total de grau de influência (TGI).
3	Usar a seguinte formula para determinar o VFA: $$VFA = (TGI * 0.01) + 0.65$$

vii. **Calcular os Pontos por Função Ajustados –** Este é o último passo para a conclusão da contagem dos Pontos por Função. Como visto na etapa i, há três tipos de projetos que podem ser contados, a saber: projetos de desenvolvimento, aplicações em uso ou projetos de manutenção. Sendo assim, têm-se três fórmulas para o cálculo dos Pontos por Função ajustado, uma para cada tipo de projeto. Contudo neste trabalho iremos considerar apenas a contagem dos Pontos por Função para projetos de desenvolvimento.

O cálculo dos Pontos por Função para projetos de desenvolvimento se dá pela seguinte fórmula:

$$PF\_Desenvolvimento = PFNA * Fator\_de\_Ajuste$$

O valor dos Pontos por Função é uma medida que acusa o tamanho do software que será desenvolvido. Através de técnicas de estimativa como o COCOMO [Boehm et al., 1995] é possível calcular algumas informações sobre o desenvolvimento como, por exemplo, quanto tempo o software demandará para sua construção, tendo como base o valor dos Pontos por Função calculado.

Segundo Andrade (2004), ainda há muitas críticas e diferenças em relação ao uso das duas métricas PCU e PF em projetos OO, porém, apesar das diferenças ressaltadas na Figura 5, estas duas métricas continuam sendo utilizadas para estimar projetos deste tipo.

APF	PCU
Métrica mais antiga e mais utilizada no mundo	Métrica relativamente nova e pouca utilizada
Padrão internacional desde 2002	Ainda não alcançou o nível de padronização e nem foi incorporada em ferramentas populares;
Não requer o uso de notação padrão, mas é baseada no modelo funcional e independente de tecnologia	Baseada no modelo de casos de uso
Largamente discutida na literatura	Tem aumentado o uso e a publicação de estudos na literatura
É suportada pelo IFPUG e diversos grupos nacionais de usuários e bases históricas de contagens realizadas	O Modelo de casos de uso ainda não possui bons históricos de produtividade
Possui regras de contagem padronizadas	Não há padrões para descrever casos de uso, há dúvidas na contagem de casos de uso incluídos e estendidos e para determinar o nível apropriado de detalhe para cada transação do caso de uso
È mais utilizada no final das fases de análise e projeto	Utilizado na fase inicial do projeto
Visão do usuário	Visão do usuário
Alto nível de maturidade	Em fase de amadurecimento
É subjetiva e possui diferença entre contadores	É subjetiva e possui diferença entre contadores
Oferece treinamento e certificação	Ainda não oferece treinamentos e certificação

Figura 5 – Principais características e diferenças entre APF e PCU

4.4 *Considerações Finais*

Neste capítulo apresentaram-se as técnicas de tamanho de software Pontos de Caso de Uso [Karner, 1993] e Pontos por Função [Albrech, 1979], que foram o principal alvo de estudo deste trabalho e também a contribuição específica para o Ambiente COCAR.

A técnica Pontos de Casos de Uso é mais recente que Pontos por Função e foi definida para dar suporte ao cálculo de estima de tamanho de software quando se usa o paradigma orientado a objetos, muito embora a técnica de modelagem baseada em Casos de Uso seja independente de paradigmas de desenvolvimento, apesar de ter se popularizado com a linguagem UML.

Já a técnica Pontos por Função é bem anterior à PCU e, em decorrência disso, mais usada no mercado, tendo várias ferramentas de estimativa baseadas nessa técnica.

No capítulo que segue serão apresentados trabalhos relacionados a essas duas técnicas mostrando o que a academia e a indústria já desenvolveram e testaram em relação à métrica de Pontos por Função e Pontos de Caso de Uso.

Capítulo 5
Trabalhos Relacionados

5.1 *Considerações Iniciais*

Este capítulo resume, primeiramente, na Seção 5.2, alguns trabalhos teóricos da literatura relacionados ao contexto das métricas apresentadas no Capítulo 4, que são interessantes para retratar o que a academia e a indústria estão estudando, usando e necessitando em relação ao assunto tratado nesta dissertação.

Esses artigos foram provenientes de um levantamento bibliográfico *ad-hoc* e também de um levantamento bibliográfico por meio da técnica de Revisão Sistemática baseada nos trabalhos de Kitchenham (2004) e Travassos & Mafra (2006). Contudo, como foi tomado conhecimento da técnica de Revisão Sistemática tardiamente, foi realizado apenas um ensaio com esta técnica não havendo a sua aplicação efetiva, ao invés disso, esta técnica foi aplicada em partes, enfatizando, principalmente, o que diz respeito à sistematização de busca de artigos. Do conjunto de artigos da revisão sistemática que foram aceitos, apenas os que tiveram uma maior influência neste trabalho é que serão apresentados.

Em seguida, ainda seção 5.2, os artigos apresentados são discutidos enfatizando a necessidade e possibilidade de uma ferramenta que seja capaz de automatizar o cálculo da métrica de PCU.

A Seção 5.3 mostra algumas ferramentas disponíveis no mercado que, de alguma forma, automatizam o cálculo de métricas de tamanho de software baseadas na técnica de PCU e PF.

Por fim, na Seção 5.4, são apresentadas as considerações finais deste capítulo.

5.2 *Trabalhos Teóricos*

Os trabalhos discutidos nesta seção foram importantes para nortear e dar subsídios à proposta desta dissertação, já que com eles foi possível constatar o atual estado da arte das métricas de tamanho de software, provendo referencial teórico e requisitos para a implementação de uma ferramenta que automatize a técnica de PCU, que é um dos objetivos principais deste trabalho.

A seguir, cada artigo é discutido individualmente:

1) *Estimating* Software *Development Effort Based on Use Cases – Experiences from industry* [Anda et al, 2001].

Nesse artigo, Anda et al. deixam clara a potencialidade da métrica de Pontos de Casos de Uso na medida de tamanho de software.

A métrica foi comparada com estimativas de especialistas para três projetos de desenvolvimento de software industrial nas áreas de *e-commerce*, *call-centers* e instituições bancárias e financeiras. Esses projetos tiveram a duração de 3 a 7 meses de desenvolvimento com um total de esforço de 3000 ou 4000 pessoas hora. Os resultados da comparação podem ser vistos na Tabela 5.

Anda et al. não deixam claro como se chegou à estimativa de esforço em horas utilizado a métrica de tamanho de software. No entanto, quando o artigo aborda os conceitos da técnica de Pontos de Casos de Uso, é enfatizado o uso de uma técnica linear de estimativa de esforço de software, por meio da qual esse valor é obtido multiplicando-se a métrica de tamanho de software por uma constante fixa. Essa constante representa o esforço em homens/hora necessários para se implementar uma unidade da métrica.

Anda *et al* afirmam, baseando-se na Tabela 5, que apesar de não ter havido nenhuma adaptação da métrica de Pontos de Casos de Uso para as empresas em questão, as estimativas realizadas com base nessa métrica, ficaram muito perto das estimativas realizadas pelos especialistas das empresas.

Esse resultado é considerado promissor pelos autores, pois os especialistas que estimaram os software são pessoas muito competentes, com ótimo conhecimento tanto na tecnologia usada quanto no domínio da aplicação.

Tabela 5 – Estimativas de especialistas, estimativa por Pontos de Casos de Uso, e Esforço (em horas) [Anda et al., 2001].

Projeto	Estimativa de Esforço (horas)			
	Especialistas	Pontos de Casos de Uso		Real
A	2730	2550		3670
B	2340	3320	2730	2860
C	2100	2080		2740

O caso do projeto B, em sua primeira contagem (3320), ficou extremamente longe da contagem dos especialistas. Isso se deu porque a primeira contagem foi realizada baseada em informações cedidas por desenvolvedores *seniores* do projeto, levando em consideração atores simples como *fax* e impressoras e alguns Casos de Uso estendidos (do tipo <<*extends*>>). Contudo, uma nova contagem foi realizada baseada no diagrama de Casos de Uso, gerando um número muito próximo do esforço real.

Como foi mostrado no artigo, o erro das medidas, quando comparado com o esforço real de desenvolvimento, foi de 4,5% - 30%, o que é, segundo os autores, perfeitamente aceitável quando se consideram outras métricas de medida de tamanho de software, mostrando que a métrica Pontos de Casos de Uso tem grande potencial prático.

Apesar do método original proposto por Karner (1993) não ter sido alterado nesse experimento, os autores observaram que alguns outros aspectos não considerados por Karner (1993) têm impacto na métrica do tamanho do software:

1. Quanto mais generalizações em atores comuns, mais precisa fica a métrica. Nessa experiência os atores observaram que se a descrição entre dois ou mais atores tem muito em comum então a precisão da estimativa aumenta se for criado um "super-ator" contendo as semelhanças que serão herdadas pelos outros atores, agora mais simples, contendo apenas as diferenças. Dessa forma o que é comum entre os atores contribui apenas uma vez para a contagem.

2. Não é apropriado para a métrica sempre descartar Casos de Uso incluídos ou estendidos (<<*includes*>>,<<*extends*>>), como afirma Karner (1993). Segundo Anda et al. uma análise mais refinada deve ser feita nesses Casos de Uso para verificar se neles estão contidas partes essenciais das funcionalidades do sistema e que devem ser levadas em consideração no momento da contagem dos Pontos de Casos de Uso.
3. O nível de detalhamento dos Casos de Uso influência diretamente nos resultados da estimativa, pois não há uma classificação maior do que "complexo" para um Caso de Uso. Ainda há o fato de não haver uma formalização para o nível de detalhamento das transações de um Caso de Uso. Dessa forma um mesmo Caso de Uso pode ser considerado complexo ou simples dependendo de quem o especifica.

Como foi observado, a forma de se estruturar o diagrama de Casos de Uso e o grau de detalhamento que cada Caso de Uso deve conter, influencia diretamente o valor da contagem dos Pontos de Casos de Uso. Contudo, não há ainda uma forma consolidada do melhor modo de se escrever um Caso de Uso ou um padrão formal que o especifique [Kulak & Guiney 2000]. Porém, alguns trabalhos têm tentado estabelecer pelo menos um *template* de escrita dos Casos de Uso [Belgamo, 2004; Cockburn, 2001].

Uma outra observação cabe quanto à complexidade associada a um Caso de Uso. O artigo mostra que quase todos os Casos de Uso do projeto A foram classificados como complexos tendo, em média, 14.2 transações cada Caso de Uso. Como já discutido, por não haver uma forma padronizada de escrita, os Casos de Uso podem ter sido construídos de forma muito detalhada. Contudo, segundo Anda et al., a métrica Pontos de Caso de Uso deveria admitir mais um nível de complexidade dos Casos de Uso para aqueles que contivessem 15 ou mais transações.

2) *Comparing Effort Estimates Based on Use Case Points with Expert Estimates* [Anda, 2002].

Esse artigo ratifica os resultados do artigo comentado anteriormente em Anda et al. (2001). Nele é apresentado um estudo comparativo entre a estimativa calculada com base na técnica de Pontos de Casos de Uso e a estimativa realizada por 37 especialistas experientes em desenvolvimento, análise de negócios e gerência de projetos. Contudo, os especialistas não tinham experiência no domínio do software em questão e alguns tinham conhecimento da métrica Pontos por Função, mas nenhum tinha conhecimento sobre Pontos de Casos de Uso. Sendo assim, Anda quis mostrar nesse estudo como a métrica Pontos de Casos de Uso pode

ajudar especialistas em estimativas a fazer estimativas de software, principalmente em áreas de aplicação não conhecidas por eles.

O estudo foi aplicado em três cursos sobre Casos de Uso lecionados em grandes empresas de tecnologia da informação, os quais tinham como alunos os especialistas acima citados. Nesses cursos os alunos foram divididos em grupos de 3 a 4 pessoas, perfazendo um total de 11 grupos, e a eles foi dada a missão de estimar a quantidade de pessoas horas para o desenvolvimento de um software que tinha como especificação um breve resumo sobre o problema a ser resolvido e um diagrama detalhado de Casos de Uso. A forma de cálculo da estimativa ficava a mercê da escolha dos integrantes do grupo.

Após os grupos terem calculado a estimativa de tamanho de time de desenvolvimento, tempo de desenvolvimento em meses e esforço em pessoas por mês para o desenvolvimento do software, Anda realizou o cálculo da estimativa de esforço de pessoas hora e pessoas mês a partir da métrica Pontos de Casos de Uso.

Os resultados mais uma vez se mostraram satisfatórios para o uso da técnica Pontos de Casos de Uso, produzindo, estimativas razoavelmente acuradas, sendo que 8 das 11 estimativas produzidas pelos alunos do curso estavam menos precisas do que aquela produzida pelos Pontos de Casos de Uso.

Com este estudo Anda mostrou que o cálculo da estimativa de esforço baseado em modelos pode ajudar especialistas na tarefa de estimativa de projeto de software, principalmente quando o domínio do software foge do conhecimento deles.

Além disso, nesse artigo, Anda deixou claro que mesmo sem enrijecer a forma de escrita dos Casos de Uso, a métrica Pontos de Casos de Uso consegue ser bastante útil para as empresas, permitindo que elas se baseiem no cálculo de estimativas de esforço, quando este é feito com base nos PCU.

3) *Effort Estimation of Use Cases for Incremental Large-Scale* Software *Development* [Mohagheghi *et al.*, 2005].

Segundo Mohagheghi et al., a métrica Pontos de Casos de Uso foi testada somente em projetos pequenos aplicada sempre no início do projeto, ou seja, não há relatos sobre o uso

dessa técnica em grandes projetos e/ou projetos que já foram iniciados e estão sendo modificados ou terminados.

Um dos grandes problemas na indústria de construção de software é a volatilidade dos requisitos. Apesar da aplicação de várias técnicas de análise de requisitos serem usadas na concepção do software, existem vários fatores externos, que vão desde o escopo de atuação do software até o não entendimento das funcionalidades por parte do próprio cliente, que fazem com que os requisitos do software se alterem a todo instante.

O processo de desenvolvimento de software incremental foi concebido pensando nessa característica de volatilidade dos requisitos do software. Esse modelo combina elementos do modelo seqüencial linear (cascata) com a filosofia iterativa da prototipagem [Pressman 2002].

Tentando atender a construção de sistemas de grande porte e que sejam concebidos em processos de desenvolvimento de software incremental, nos quais os requisitos são alterados a cada fase de incremento, Mohagheghi et al. adaptaram a métrica Pontos de Casos de Uso gerando o processo de seis passos, descrito abaixo, e sumarizado na Tabela 6.

No primeiro passo, Mohagheghi et al. (2005) propõem que os atores sejam contados e, desde que esse número seja de pouco impacto (apesar dos autores não definirem o que é ser de pouco impacto) na contagem final, pode-se assumir que sejam de grau de complexidade média, ou seja, PF = 2. Ainda nesse passo uma nova regra é acrescida: se algum ator tiver sido modificado da versão anterior para a atual, um novo fator, chamado de Peso dos Atores Modificados Não Ajustado (PAMNA) deve ser calculado, contabilizando o número de atores novos ou alterados.

O passo dois dos Pontos de Casos de Uso Modificado é composto por seis subpassos. Primeiramente o Peso do Caso de Uso Não Ajustado PCUNA de uma primeira versão deve ser computado; para isso usam-se as regras descritas pela técnica original definida por Karner (1993).

Contudo, como um mesmo Caso de Uso pode ser escrito com diferentes níveis de rigor e de detalhes técnicos [Cockburn, 2001], Mohagheghi et al. sugerem que uma regra de simplificação de Casos de Uso seja usada apenas para efeito da contagem dos Pontos de Casos de Uso, ou seja, os autores não aplicaram a regra para reescrever os Casos de Uso, e

sim simplesmente aplicaram a regra para contar os Pontos de Caso de Uso. Essas regras estão escritas nos passos 2.1 até o passo 2.5 da Tabela 6.

Tabela 6 – Resumo dos Pontos de Casos de Uso para projetos incrementais [Mohagheghi et al., 2005].

Passo	**Regra**	**Saída**
1	1.1 Classificar todos os atores como médios. PF = 2	PANA = atores * 2
	1.2 Contar o número de atores novos/modificados	PAMNA = atores novos ou modificados * 2
2	2.1. Se cada transação no curso normal for composta por uma ou mais *transações* deve-se contar cada transação como sendo um Caso de Uso. 2.2. Contar cada curso alternativo como sendo um Caso de Uso. 2.3. Cursos excepcionais, parâmetros e eventos são classificados como peso 2 e na soma balanceada são limitados a 15 (um Caso de Uso complexo). 2.4. Casos de Uso incluídos ou estendidos são considerados como Casos de Uso base. 2.5. Classificar os Casos de Uso como: a) Simples- 2 ou menos transações, PF = 5; b) Médios- 3 to 4 transações, PF = 10; c) Complexos – mais do que 4 transações, PF = 15;	PCUNA = ∑(Casos de Uso * PF) + ∑ (pontos atribuídos para cursos excepcionais e parâmetros).
	2.6. Contar pontos para os Casos de Uso modificados de acordo com as regras 2.1-2.5 para calcular o PCUMNA.	PCUMNA = ∑(Casos de Uso novos/modificados *WF) + ∑ (pontos atribuídos para cursos excepcionais e parâmetros novos/modificados).
3	3.1. Calcular PCUNA para todo o software	PCUNA = PANA + PCUNA
	3.2. Calcular o PCUMNA.	PCUMNA = PAMNA + PCUMNA
4	4.1. Assumir o projeto como médio	FTC = FA = 1
5	5.1.Calcular o PCU	PCU = PCUNA
	5.2.Calcular PCUM	PCUM = PCUMNA
6	6.1.Estimar o esforço para os Casos de Uso novos/modificados	*E_primary* = PCUM * PHperPCU
	6.2.Estimar esforços para mudanças secundárias do software	*E_secondary* = (PCU- PCUM) * EMF *PHperPCU
	6.2.3.Estimar o total de esforço	E = *E_primary* + *E_secondary*

Ainda no passo dois, um novo fator é inserido no cálculo dos Pontos de Casos de Uso*:* o Peso de Caso de Uso Modificado Não Ajustado (PCUMNA). Esse novo fator serve para contar os Pesos dos Casos de Uso Não Ajustados (PCUNA) das transações ou parâmetros que sofram alteração de uma versão para outra. A fórmula do PCUMNA está descrita no passo 2.6 da Tabela 6.

Tendo calculado o PANA, o PAMNA, o PCUNA e o PCUMNA, o método propõe o cálculo dos Pontos de Caso de Uso Não Ajustados *(*PCUNA*),* através das regras originais propostas por Karner (2003), e propõe o cálculo dos Pontos de Caso de Uso Modificados Não Ajustados MUUCP cuja fórmula está especificada no passo 3.2 da Tabela 6.

Segundo Anda et al. (2001), a avaliação do Fator de Complexidade Técnica e do Fator Ambiental é geralmente realizada por uma pessoa sênior ou um líder de projetos sendo que o FCT tem um impacto muito pequeno e não abrange todos os requisitos não funcionais. Por esse motivo, no passo 4, é proposto que o valor do FTC seja 1. Contudo, apesar dos autores assumirem que em projetos incrementais a variação do ambiente não muda de uma versão para outra, e de atribuir 1 também para o FA, Mohagheghi et al. (2005) frisa que o FA é muito relevante para o cálculo do esforço total e que no passo 6 haverá uma compensação do FA através da escolha do maior PHperPCU disponível na literatura.

Como o FCT e o FA assumem valor igual a 1, no passo 5 calcula-se o valor dos Pontos de Caso de Uso Ajustados (PCUA) e dos Pontos de Caso de Uso Modificados Ajustados (PCUMA) como sendo iguais ao PCUNA e ao PCUMNA, respectivamente.

Finalmente, o último passo da modificação proposta por Mohagheghi et al. sugerem que três tipos de esforços sejam estimados, a saber: *E_Primary, E_Secundary* e o esforço total que é resultante da soma dos dois esforços anteriores.

Mohagheghi et al. afirmam que dois tipos de alterações acontecem na mudança de uma versão para outra em processos incrementais:

1. Novos Casos de Uso são criados ou Casos de Uso são alterados configurando o esforço chamado de *E_primary;*
2. Alterações secundárias que configuram o esforço chamado *E_secundary;*

Além de alterações geradas por mudança de requisitos ou adição de novos requisitos ao sistema, os software também podem ser alterados por razões secundárias, a saber: alterações para atender melhorias de funcionalidades já estabelecidas, modificações para atender melhorias na qualidade do software, mudanças para atender a correção de funcionalidades implementadas e mudanças preventivas como alteração de projeto ou modularização.

Por último, a fórmula descrita no passo 6 da Tabela 6 é usada para estimar o esforço total, sempre considerando o valor de PHperPCU como sendo o maior possível para compensar o valor 1 do FA. Sendo assim, o valor assumido pelos autores foi de 36 PHperPCU. [Karner, 1993].

Um ponto muito interessante nesse artigo não é apenas o fato da adaptação da métrica Pontos de Caso de Uso para desenvolvimentos incrementais, mas sim a necessidade encontrada por Mohagheghi et al. de reescrever, para efeitos de contagem, os Casos de Uso que já estavam estabelecidos.

A métrica Pontos de Caso de Uso classifica um Caso de Uso de acordo com a quantidade de transações que ele apresenta em sua descrição. Sendo assim, se um Caso de Uso tiver 8 ou 20 transações ele é considerado complexo e vai contribuir da mesma forma para o cálculo dos Pontos de Caso de Uso.

Os Casos de Uso descritos no projeto eram demasiadamente complexos, sugerindo a Mohagheghi et al. que os dividissem. Dessa forma, um Caso de Uso considerado muito complexo (por ter muitas transações) foi quebrado em vários outros com complexidades menores (de acordo com as regras estabelecidas no passo 2 da Tabela 6), fazendo com que a contribuição do Caso de Uso original na contagem dos Pontos de Caso de Uso fosse maior do que apenas 15 Pontos de Caso de Uso por Caso de Uso complexo.

Isso mostra a necessidade na padronização de escrita dos Casos de Uso, pois como se verificou nesse artigo, ela influencia muito no resultado final da contagem dos Pontos de Caso de Uso [Anda et al., 2001].

4) *Estimating* Software *Based on Use Cases Points* [Carroll, 2005].

Esse artigo descreve um modelo através do qual, segundo o autor, uma grande indústria de desenvolvimento de software, com múltiplas equipes, consegue estimar com precisão o custo

do desenvolvimento de software se baseando em Pontos de Caso de Uso, obtendo este valor em fases iniciais do ciclo de desenvolvimento.

Carroll defende o fato de que todas as principais estimativas de custo de desenvolvimento de software se apóiam fortemente nas informações de projetos passados consolidadas em bases históricas e propõe um novo processo de estimativa de esforço de software. Esse novo processo se ajusta, ficando cada vez mais refinado e preciso, na medida em que os dados da base histórica de projetos passados ficam mais completos e consistentes. Ou seja, o modelo apresentado por Carroll prevê uma fase de "análise e melhorias" na qual os dados históricos de projetos passados são analisados com a intenção de se ajustar o processo de estimativa, assim como mostra a Figura 6.

A explicação de Carroll se divide em duas partes:

a) Descrição de um processo de estimativa de custo de desenvolvimento de software baseado em Pontos de Caso de Uso;

b) Descrição do processo de análise dos dados históricos para a definição de melhorias no processo de estimativa do tamanho de software;

A técnica de estimativa de software descrita nesse artigo se baseia na métrica de PCU. Porém, após 200 projetos implementados em um período de 5 anos, Carroll, por meio da aplicação do seu modelo de melhoria de processo de estimativa de software, adaptou a técnica de PCU, a saber:

a) Contagem dos Atores: Na determinação dos pontos dos atores, devem-se usar os dados históricos para ajustar o peso atribuído a um ator simples, médio e complexo, ou seja, não se deve ficar preso à medida de 1, 2 e 3, respectivamente para atores simples, médio e complexo descrita por Karner (1993). Além disso, se for preciso, devem-se estabelecer novos pesos para os atores simples, médio e complexo de acordo com as bases de dados históricos;

b) Contagem dos Casos de Uso: Se no momento da contagem das transações de um determinado Caso de Uso já forem conhecidas as classes que implementarão esse Caso de Uso, então é preferível usar essa informação para a classificação de

complexidade do Caso de Uso a utilizar as transações contidas no próprio Caso de Uso;

c) Cálculo de homens-horas por Pontos de Caso de Uso: Para estimar a quantidade de homens/horas necessárias para a implementação do projeto, normalmente basta multiplicar a métrica de Pontos de Caso de Uso pela quantidade de homens-hora para se implementar uma unidade de ponto de caso de uso (taxa de esforço).

Contudo, o modelo aqui defendido considera o fato de existirem projetos simples e complexos, e sendo assim, são adotadas duas taxas de esforço diferentes: para projetos simples que determina 20 homens-hora por Pontos de Caso de Uso e para projetos complexos que determina 28 homens-horas por Pontos de Caso de Uso.

A definição de projetos simples e complexos é obtida pela análise das respostas às perguntas dos Fatores Ambientais, como definem Schneider e Winters [Schneider & Winters, 2001] que consideram que um projeto seja simples caso a soma da quantidade de respostas aos itens 1-6 do fator ambiental que são menores que três com a quantidade de respostas aos itens 7-8 que são maiores do que três, seja menor do que 2; caso contrário, o projeto deve ser considerado complexo.

d) Fator de ajuste de risco: Além do fator de ajuste técnico e do fator de ajuste ambiental, também é definido nesse modelo um Fator de Ajuste de Risco, que é aplicado na taxa de esforço de homens-hora por Pontos de Caso de Uso, gerando a taxa de esforço de homens-hora por Pontos de Caso de Uso Ajustado. Esse fator é calculado baseado na possibilidade do projeto exigir treinamento especial, foco no desenvolvimento de componentes reutilizáveis ou integração especial com outros sistemas.

e) Estimativa de Relatórios: Todas as estimativas de relatórios são feitas à parte e usando diretamente medidas de horas retiradas da base histórica ao invés de aplicar Pontos de Caso de Uso.

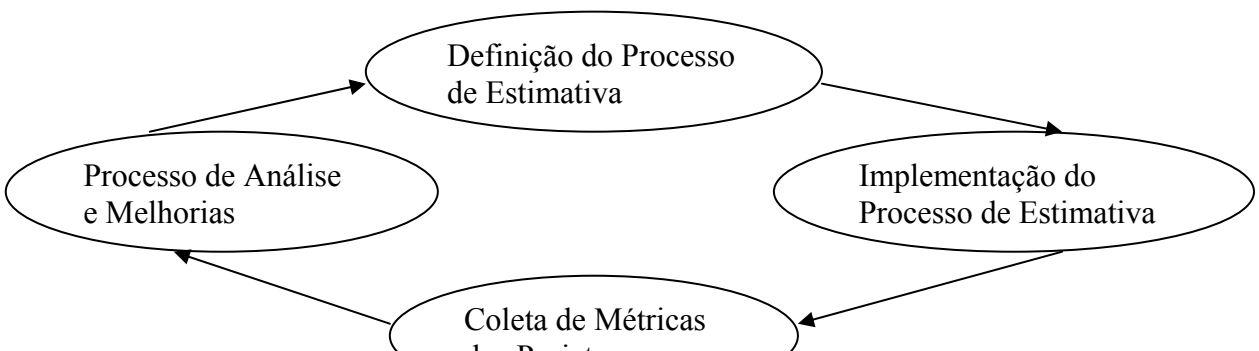

Figura 6 – Processo de estimativa de esforço [Carrol, 2005].

Para sempre melhorar o processo de estimativa de software, Carroll define um processo de análise de bases históricas dividido em 6 partes, que sempre busca apontar grandes desvios da curva normal:

a) Análise de *Performance*: são analisados os desvios existentes entre o que foi planejado e o que foi executado, como por exemplo, data de início, data de fim e duração;
b) Análise de *Releases* Entregues: segundo o autor, essa análise é capaz de mostrar quando e onde problemas podem existir em um projeto através da coleta e análise das datas dos *releases* e dos *releases* entregues confrontadas com normas de desempenho desses *releases*;
c) Análise de Cronograma: essa análise é uma verificação macro das atividades que foram desenvolvidas. Deve-se confrontar a data em que as atividades foram entregues com as datas que elas foram planejadas;
d) Análise de Esforço: essa análise confronta o esforço de desenvolvimento gasto em cada fase do ciclo de desenvolvimento do software. Mais uma vez o executado deve ser confrontado com o planejado;
e) Análise de Defeitos: identificar as fases em que defeitos foram introduzidos no software (especificação, projeto, implementação, entre outros) e confrontar com as fases nas quais esses defeitos foram encontrados;
f) Análise da Causa: é realizada pelo líder da equipe e consiste em permitir que a equipe explique os desvios encontrados nas análises feitas nos itens anteriores, com o intuito de identificar as causas básicas desses desvios.

Carroll deixa claro que o alto nível de precisão atingido em seu modelo de estimativa de software só foi possível devido ao processo de melhoria deste modelo, baseando-se na análise dos dados das bases históricas de projetos já desenvolvidos. Dessa forma o autor defende que não é suficiente apenas coletar métricas, mas sim analisar essas métricas, de

forma a encontrar um modo de melhorar o processo de estimativa de acordo com as características dos projetos e dos desenvolvedores da empresa em questão.

5) *Estudo de Caso da Aplicação da Métrica de Pontos de Caso de Uso numa Empresa de Software* [Heimberg & Grahl, 2005]. Viviane Heimberg, Everaldo Artur Grahl, 2005.

No estudo de caso apresentado nesse artigo, a métrica de Pontos de Caso de Uso foi utilizada em uma empresa para aumentar a precisão das estimativas de tempo de desenvolvimento em três projetos de software *WEB*.

A empresa em questão é uma empresa desenvolvedora de software corporativo com 300 funcionários e com um faturamento de cerca de R$ 30 milhões, que está migrando os seus sistemas para o ambiente *WEB* e, para tanto, está fazendo uso de uma metodologia de desenvolvimento de software baseada no processo unificado de desenvolvimento.

Foi escolhida como métrica de tamanho de software os Pontos de Caso de Uso, pois ela é baseada fortemente na UML, que a empresa em questão estava começando a utilizar.

O estudo de caso analisou o diagrama de casos de uso de três software, a saber: Sistema de cálculo de folha de pagamento (Projeto 1), Sistema contábil (Projeto 2), e o Sistema de cartão de ponto (Projeto 3).

O índice de complexidade ambiental obteve o mesmo valor para os três projetos, já que os desenvolvedores envolvidos possuíam as mesmas características, sendo que todos receberam o mesmo treinamento padronizado antes do início dos projetos.

O índice de complexidade técnica variou em alguns pontos entre os projetos, pois eles eram diferentes em algumas particularidades, como complexidade de processamento, concorrência e acesso direto a terceiros, mas o resultado final desse índice teve apenas uma pequena diferença entre cada um dos projetos.

A Tabela 7 - **Esforço estimado por projetos**, nas colunas de "Valor Inicial", mostra uma primeira estimativa realizada. Contudo, os coordenadores de projetos, juntamente com as equipes de desenvolvimento, julgaram ser muito alto o valor das estimativas de esforço obtidas utilizando o fator de 20 homens-hora por unidade de Pontos de Caso de Uso. De

acordo com a percepção e experiência desses profissionais, o tempo de desenvolvido estaria em cerca de 70% do tempo estimado.

Tabela 7 - Esforço estimado por projetos

Projetos	Atores	Caso de Uso	FCT	FCA	PCUNA		PCUA		Horas Estimadas	
					Valor Inicial	Valor Alterado	Valor Inicial	Valor Alterado	Valor Inicial	Valor Alterado
1	4	5	1,00	0,81	72	54	58,68	43,73	1173,60	437,40
2	6	8	1,02	0,81	87	76	72,32	61,50	1446,46	615,60
3	7	4	1,03	0,81	54	57	53,72	46,17	1074,50	461,70

Desta forma, foi decido realizar mais uma estimativa nos três projetos, porém agora levando em consideração algumas alterações no método:

1) Foram ajustados os pesos dos casos de uso para: simples = 5; médio = 10; complexo = 25;
2) A quantidade de homens-hora para uma unidade de Pontos de Caso de Uso foi alterada de 20 para 10;

Com essas alterações no processo foram obtidos os resultados mostrados na Tabela 7, na coluna "Valor Alterado".

Com a nova estimativa, os coordenadores de projetos, juntamente com as equipes de desenvolvimento, julgaram que o novo valor está muito próximo da realidade dos projetos já desenvolvidos por eles.

Contudo, não foi possível a continuidade do levantamento de dados nesses projetos, pois eles foram interrompidos com data para retomada posterior à publicação desse artigo.

6) ***TUCCA: Técnicas de Leitura para a Construção de Modelos de Casos de Uso e Análise do Documento de Requisitos*** **[Belgamo 2004].**

Embora possa parecer o contrário, a construção do Modelo de Casos de Uso para um determinado documento de requisitos não é uma tarefa trivial. Principalmente por que a

notação UML se restringe a simplesmente oferecer formalismo para a sintaxe do diagrama, não definindo diretrizes para a sua elaboração e tampouco para a sua especificação, deixando a qualidade desse trabalho ser totalmente influenciável pela subjetividade e experiência do projetista.

Dessa forma, em seu trabalho de mestrado, Belgamo definiu a técnica TUCCA. Esta técnica consiste em um conjunto de diretrizes que tem por objetivo principal a elaboração do Modelo de Casos de Uso a partir de um Documento de Requisitos.

Paralelamente a elaboração do Modelo de Casos de Uso, as diretrizes definidas pela TUCCA provocam uma inspeção no Documento de Requisitos dando suporte à detecção de vários defeitos no mesmo.

A técnica TUCCA é composta por duas leituras: AGRT - Actor Goal Reading Technique e UCRT – Use Case Reading Technique. Essas duas leituras são melhores explicadas na seqüência.

A AGRT, que tem por objetivo identificar os possíveis atores do sistema juntamente com os seus principais objetivos de uso. Essa técnica utiliza como artefato de entrada o Documento de Requisitos. A identificação dos atores e objetivos é feita através de uma técnica de marcação de verbos e substantivos. O artefato de saída é o Formulário Ator Objetivo (FAO), definido por Cockburn (2001), que registra, além dos atores e seus objetivos, os requisitos funcionais em que esses foram identificados no Documento de Requisitos.

Durante o processo de criação do FAO, as diretrizes definidas na técnica AGRT possibilitam encontrar-se defeitos no Documento de Requisitos. Desta forma, ao final da aplicação da AGRT, além do FAO também é gerado um Relatório de Defeitos encontrados.

A UCRT, com base no FAO, tem como objetivo elaborar o Modelo de Casos de Uso propriamente dito através do entendimento dos propósitos de cada ator com o intuito de criar os Casos de Uso, encontrar seus relacionamentos e especificá-los. Os Casos de Uso identificados, bem como seus relacionamentos e as referências aos requisitos funcionais geradores deste Caso de Uso são armazenados em um documento chamado Formulário de Casos de Uso Preliminares (FCUP) cujo *template* pode ser visto na Figura 7. Já a especificação dos Casos de Uso identificados são armazenadas em um documento chamado Formulário de Especificação de Casos de Uso (FEsp), o qual é mostrado na Figura 8.

Formulário de Casos de Uso Preliminares						
Nº	**Ator**	**Caso de Uso**	**Referência**	**Relacionamento**	***Include***	**Especificado**

Figura 7 – Formulário de Casos de Uso Preliminares [Belgamo, 2004].

Também, a exemplo da AGRT, as diretrizes definidas na técnica UCRT possibilitam encontrar defeitos no Documento de Requisitos. Desta forma, ao final da aplicação da UCRT, o Relatório de Defeitos gerados na AGRT é acrescido dos defeitos encontrados durante o processo de aplicação da UCRT.

	Especificação do Caso de Uso	Número:	Número do Caso de Uso
Nome do Caso de Uso	Nome do caso de uso criado		
Descrição ou Resumo	Pequena descrição do caso de uso		
Ator Participante	Proprietário da Informação		
Ator Operador	Manuseia o computador		
Ator Genérico	Ator genérico criado pela análise dos atores participantes.		
Pré-Condição	Condições que devem ser verdadeiras para que o caso de uso possa ser realizado		
Curso Normal	Corresponde a um fluxo de eventos		
Curso Alternativo	Corresponde a fluxos de eventos, porém mostrando os caminhos menos comuns de acontecer		
Evento Disparador	Descreve o critério de entrada para o caso de uso sendo especificado		
Include	Número dos casos de uso relacionados pelo estereótipo <<include>>		
Extend	Número dos casos de uso relacionados pelo estereótipo <<extend>>		
Requisitos Funcionais	Os números dos requisitos funcionais associados ao caso de uso criado		
Requisitos Não-Funcionais	Os números dos requisitos não funcionais associados ao caso de uso criado		
Autor	A pessoa que criou a especificação		
Data	A data da criação da especificação		
Versão	Código associado à versão da especificação		

Figura 8 - Formulário de Especificação de Casos de Uso

7) ***Um Estudo sobre a Influência da Sistematização da Construção de Modelos de Casos de Uso na Contagem dos Pontos de Caso de Uso*** **[Belgamo & Fabbri 2004c].** Esse artigo mostrou que os Pontos de Casos de Uso gerados a partir de modelos construídos com uma maior sistemática e padronização, apresentam uma maior qualidade do que os decorrentes dos modelos gerados por uma abordagem Ad-Hoc. Em outras palavras, mostrou que a forma como a especificação dos casos de uso é elaborada interfere nos PCU, os quais são bastante diferentes quando não se têm modelos gerados de uma maneira mais padronizada e que essa variação é pequena quando se têm modelos mais padronizada.

Para construir o Modelo de Casos de Uso a partir de um documento de requisitos foi utilizada a técnica de leitura TUCCA, a qual foi alvo de implementação no ambiente que está sendo apresentado neste trabalho no Capítulo 6.

O experimento mostrado nesse artigo foi realizado com 18 estudantes, divididos em seis grupos de 3, da disciplina de Engenharia de Software dos cursos de Bacharelado em Ciências da Computação e de Engenharia de Computação da Universidade Federal de São Carlos, sendo que nenhum dos participantes possuíam conhecimentos prévios de modelagem de requisitos através de casos de uso.

Cada grupo desenvolveu um documento de requisitos baseado no padrão IEEE de documentação de requisitos.

Usando como entrada os documentos de requisitos, dois Modelos de Casos de Uso foram desenvolvidos para cada documento, sendo que o primeiro modelo foi construído de maneira ad-hoc e o segundo modelo foi construído utilizando-se a técnica TUCCA.

Cada grupo desenvolveu um documento de requisitos, depois elaborou o modelo de casos de uso de maneira ad-hoc em um documento de um outro grupo e, posteriormente, aplicou a TUCCA em um documento de um terceiro grupo. Isso para que nenhum grupo utilizasse o documento criado por ele próprio ou ainda aplicasse as técnicas ad-hoc ou TUCCA em um mesmo documento.

Para efeito de controle, uma pessoa com domínio na técnica TUCCA modelou cada documento de requisitos gerando um Modelo Oráculo com o qual os Modelos desenvolvidos pelos grupos com as técnicas ad-hoc e TUCCA puderam ser comparados.

Como os modelos de casos de uso gerados com a TUCCA foram mais padronizados, isto é, foram bastante similares ao modelo oráculo, isso mostrou que os PCU apresentaram menor variabilidade do que os PCU gerados a partir dos modelos elaborados de forma ad-hoc, mostrando que a técnica TUCCA permite estimativas de PCU próximas entre si, mesmo quando executadas por pessoas diferentes.

Quando comparada com o PCU do Modelo Oráculo, pôde-se perceber que a média dos PCU gerados a partir dos modelos elaborados com a TUCCA ficaram 8% abaixo, enquanto que a média dos PCU gerados com a abordagem Ad-Hoc ficaram 105% acima.

Desta forma, a padronização do Modelo de Casos de Uso é fundamental para a maior precisão das métricas de Pontos de Caso de Uso e, consequentemente, para a estimativa de esforço, visto que a métrica em questão usa como medida fundamental para concluir o tamanho do software a quantidade de casos de uso, atores e transações e a TUCCA contribui para a padronização desses itens.

8) *Use Case Estimation – The Devil is in the Detail,* [Vinsen et al., 2004].

Nesse trabalho o desenvolvimento de um sistema de software grande e complexo é descrito pelos autores para mostrar alguns dos problemas nas estimativas de software baseadas em Casos de Uso e uma possível solução é proposta para melhorar este tipo de estimativa.

A introdução dos Casos de Uso no desenvolvimento de software pode ter melhorado bastante a eficiência da programação, mas não melhorou a estimativa de custo de sistemas complexos, e essa conclusão, segundo os autores, pode ser apoiada pelo fato de que ainda há muitas técnicas de estimativa de custo baseadas em função sendo usadas atualmente, mostrando que estimativas baseadas em Casos de Uso ainda não estão completamente maduras.

Os Casos de Uso são escritos de forma que torne fácil a leitura das funcionalidades do software pelo cliente. Os profissionais de estimativa são capazes de, rapidamente, ter uma visão geral sobre todos os requisitos, o que para sistemas pequenos é mais do que suficiente para uma estimativa de esforço razoavelmente precisa.

As funcionalidades detalhadas dos sistemas não são descritas no próprio Caso de Uso e sim em documentos auxiliares que podem chegar a milhares de páginas que incluem regras, condições, exclusões e comentários.

Essa característica dos Casos de Uso não permite que os profissionais em estimativa de software sejam capazes de incluir, com facilidade, os custos associados a componentes lógicos e infra-estrutura à estimativa de esforço de implementação do sistema. Dessa forma, a linguagem dos Casos de Uso contribui para o problema de identificação de elementos de infra-estrutura. com o propósito de mapear a contribuição desses elementos para estimativa do esforço de sistemas grandes e complexos.

Outro ponto que os Casos de Uso não levam em questão é a complexidade das interfaces gráficas, isso por que os Casos de Uso não descrevem as telas que deverão ser

disponibilizadas para os usuários, pois a atividade de projeto de interfaces costuma ser realizada apenas na fase de Projeto do ciclo de vida de desenvolvimento do software, a qual ainda não está pronta quando as estimativas de esforço são realizadas.

Para mostrar melhor todos os problemas relacionados à estimativa de esforço de desenvolvimento baseado em Casos de Uso, foi especificado um sistema contendo 134 Casos de Uso, 900000 (novecentas mil) linhas de código, e que levou cerca de 5 anos para ser desenvolvido.

Os Casos de Uso foram especificados por pessoas que tinham o domínio do escopo da aplicação e foram aprovados por gerentes que também tinham o domínio deste escopo.

A estimativa de esforço foi calculada baseada no Modelo de Casos de Uso, que segundo o autor, apresentava limitações, já que as pessoas que desenvolveram os Casos de Uso possuíam apenas domínio do processo diário das regras de negócio e não dos requisitos detalhados, os quais eram dominados apenas por alguns indivíduos. Desta forma, em pouco tempo, ficou aparente que o modelo estava escondendo várias complexidades do sistema.

Os Casos de Uso também mostraram problemas ao representar tarefas e processos que deveriam ser executados diariamente. O problema maior foi a falta de uma especificação nos Casos de Uso que definisse a interface da qual as tarefas deveriam retirar e depositar os dados trabalhados por elas. Essa informação só ficou disponível na fase de projeto, quando os Casos de Uso puderam ser totalmente especificados e a verdadeira complexidade da interface pôde ser definida.

Esse trabalho encontrou poucas evidências de que a comunidade acadêmica tenha investigado cientificamente a ligação que deve existir entre os custos relacionados aos Casos de Uso e os custos relacionados ao *WBS - Work Breakdown Structure* (árvore de composição que relaciona todo o software, hardware, serviços, dados e facilidades necessários para a implementação de um determinado produto).

Foi sugerido, através desses e outros trabalhos analisados pelo autor, que projetos grandes e complexos não têm um mapeamento completo entre o Modelo de Casos de Uso e o WBS, principalmente quando o Modelo de Caso de Uso apresenta um grande número de *Includes* e *Extends*.

Desta forma, o autor afirma que o que está sendo esquecido pelas técnicas de estimativa em questão é uma camada que possibilite o mapeamento entre o Modelo de Casos de Uso e o WBS, para prover a base para as estimativas.

Conseguir um detalhamento completo dos Casos de Uso nas fases iniciais do projeto é algo inviável; assim, esse mapeamento deve ser tal que possibilite ao máximo esclarecer os requisitos escondidos e exigências funcionais ainda não consideradas. Contudo, nesse artigo o autor não deixa claro o que seria "esclarecer ao máximo" esses requisitos escondidos. Dessa forma, para o autor desse artigo, mapear os Casos de Uso, com uma descrição funcional de alto nível pertencente ao WBS na fase de estimativa, irá permitir que a verdadeira complexidade dos Casos de Uso seja levantada. Mais uma vez, é mostrada a importância da descrição dos Casos de Uso para uma estimativa de maior qualidade.

Assim, nesse trabalho foi ressaltado o ponto de que Casos de Uso em sistemas grandes e complexos não são suficientes para mostrar o tamanho do software, pois estes escondem alguns fatores relacionados às interfaces com outros sistemas, complexidade de algumas funcionalidades e complexidade de infra-estrutura.

9) *A multiple-Case Study of Software Effort Estimation based on Use Case Points,* [Anda *et al.* 2005].

Esse trabalho, que segundo os autores é inédito, se concentra em mostrar que a estimativa de esforço de desenvolvimento de software*,* baseada na técnica de Pontos de Casos de Uso, ainda não está preparada para refletir o esforço relacionado:

1. ao processo de desenvolvimento de software*;*
2. à qualidade de código,

adotados na implementação do sistema o qual quer-se estimar.

Para tanto, Anda *et. al.* realizaram um estudo de caso em 4 empresas de desenvolvimento de software que possuem 2 características que as difere: processos de desenvolvimento variados e diferente ênfase na qualidade do código; dessa forma, encontraram nessas empresas desde um processo leve com pouca ênfase na qualidade do código desenvolvido, até processos mais bem elaborados, com ênfase em análise, projeto e qualidade do código desenvolvido, além de foco na qualidade do processo e do produto.

Para cada uma das quatro empresas foi proposto o desenvolvimento de um software para *web* com as mesmas funcionalidades e utilizando linguagem JAVA. Ao final do desenvolvimento, a métrica de esforço baseada em Pontos de Caso de Uso, foi comparada com o real esforço de desenvolvimento dos software.

Para que fosse possível a realização desse estudo de caso, as quatro empresas envolvidas registraram diariamente o esforço utilizado para a implementação de cada um dos Casos de Uso contidos no Modelo de Casos de Uso do software.

Utilizando a técnica de Pontos de Caso de Uso foi estimado o esforço de desenvolvimento do software que as 4 empresas teriam que implementar, chegando a um valor de 413 horas. Para a realização desta estimativa de esforço, foi utilizado o Modelo de Casos de Uso, que era comum para as 4 empresas, descrevendo os requisitos funcionais do sistema, partindo do princípio que os requisitos não funcionais eram triviais, segundo os autores.

O número de transações de um Caso de Uso é muito utilizado como uma medida do tamanho deste Caso de Uso, sendo que a técnica de PCU utiliza esse número como base para a determinação da métrica de PCU. Porém, segundo os autores deste artigo, há poucas experiências que mostram o correlacionamento existente entre o número de transações de um determinado Caso de Uso e o esforço despendido para a implementação deste Caso de Uso.

Com os dados coletados no experimento e mostrado na Tabela 8, os autores puderam mostrar o forte correlacionamento existente entre o esforço de desenvolvimento de um Caso de Uso e o número de transações deste mesmo Caso de Uso. Desta forma fica claro que o número de transações de um Caso de Uso pode ser usado como uma medida do tamanho deste Caso de Uso.

Foi observado também que o esforço real de desenvolvimento ficou entre 431-943 horas, sendo que a empresa com o processo de desenvolvimento mais leve foi a que gastou menos esforço para o desenvolvimento. Os esforços gastos por cada empresa para a implementação do sistema, bem como a estimativa feita no inicio do desenvolvimento podem ser observadas na Tabela 9.

Tabela 8 - Esforço de desenvolvimento por Caso de Uso

Caso de Uso	Transações	Esforço em horas			
		A	B	C	D
1	1	19	37	9	18
2	1	22	52	5	27
3	1	23	29	5	14
4	1	31	48	3	32
5	2	21	10	22	22
6	2	37	36	22	35
7	3	57	143	16	116
8	3	67	136	40	82
9	4	157	148	99	147
Total		434	639	221	493
Esforço não atribuído a nenhum caso de uso		153	304	210	336

Tabela 9 - Horas Estimadas X Horas Gastas dos projetos das quatro empresas

Empresa	Horas Estimadas	Horas Gasta	
A	413	587	Processo leve com foco no design visual
B	413	943	Processo elaborado com foco em análise e projeto.
C	413	431	Processo muito leve.
D	413	829	Processo muito bem elaborado com foco em análise, projeto e gerencia de projeto.

Dessa forma, segundo os autores, pôde-se constatar que apesar das quatro empresas terem desenvolvido um software com as mesmas funcionalidades, usar a mesma linguagem de desenvolvimento, houve uma grande diferença entre o estimado e o realizado.

Anda *et al*. atribuem essa grande diferença pelo fato das quatro empresas possuírem um processo de desenvolvimento de software diferente umas das outras.

Assim, os autores concluem que o modelo de Pontos de Caso de Uso ainda não reflete, em sua estimativa de esforço, o montante relativo ao processo de desenvolvimento de software utilizado para implementar o sistema.

10) *Software Estimation in the Maintenance Context,* [Diev, 2006].

Apesar do método definido por Karner (1993) ser muito utilizado e se mostrar útil para a estimativa de tamanho de software em fases iniciais do desenvolvimento do software, este método não foi projetado para dar suporte a algumas situações específicas, como por exemplo, o caso de um novo produto começar a ser desenvolvido tendo como base um produto pré-existente, ou seja, o desenvolvido de um novo produto gerado a partir da manutenção de um outro já existe.

Pensando em software construídos no contexto acima descrito, Diev (2006), seguindo a mesma linha de Mohagheghi et al. (2005), sugere um novo modelo de estimativa de software baseado nos Pontos de Casos de Uso chamado UCPm, o qual tem como objetivo refletir na estimativa de software, as especificidades da fase de manutenção do ciclo de desenvolvimento.

Para explicar melhor o UCPm, Diev define alguns conceitos, a saber:

a) Produto – Pode ser um novo sistema ou alterações de um sistema já existente;
b) Estimativa – Uma indicação sobre tamanho ou esforço;
c) Estimar – É o processo de se obter uma estimativa de algum produto;
d) Sistema Base – É o sistema a partir do qual um outro sistema será construído;
e) Largura do Software – Número de casos de uso que este software implementa;
f) Largura de um sistema base – É o número total de casos de uso do sistema base;
g) Largura das mudanças – É o número de casos de uso do sistema base que serão alterados em virtude da construção de um novo sistema a partir desse sistema base;
h) Larguras das Novas Funcionalidades – É o número dos novos casos de uso que serão implementados em um novo sistema que parte de um sistema previamente existente;

i) Profundidade das alterações – É a extensão de alterações que um Caso de Uso do sistema base irá sofrer em virtude da construção de um novo sistema a partir desse sistema base. Esse valor é definido em termos de transações de Casos de Uso. Sua forma de cálculo é mostrada na Tabela 10;
j) Tamanho de um software ou componente – número de pontos de caso de uso calculados usando o UCPm;

Tabela 10 – Regras para o cálculo da Profundidade das Alterações de um Caso de Uso

#	Tipo da Mudança	Profundidade (Ch_{UC})	Explicação
1	Há alterações na funcionalidade do Caso de Uso	N_2/N_1	N_1 é o número de transações do caso de uso antes da alteração. N_2 é o número de transações adicionadas ou alteradas ou ainda excluídas do Caso de Uso alterado
2	As mudanças são não funcionais	0,2	Exemplo: Alterações de layout do software.

O UCPm é baseado no método de Pontos de Caso de Uso e pode ser definido em 8 passos simples, a saber:

Passo1 – Calcular o peso dos atores não ajustados (PANA)

Essa parte é semelhante ao descrito por Karner (1996), ou seja, aqui cada ator é classificado como simples (peso 1), médio (peso 2) ou complexo (peso 3) para então realizar-se a soma desses pesos chegando ao valor do PANA.

Passo 2 – Calcular o peso dos casos de uso não ajustados (PCUNA)

Aqui cada novo Caso de Uso deve ser:

1) classificado em simples (contém de 1-3 transações) e atribuir peso 5; ou médio (contém de 4-7 transações) e atribui peso 10; ou complexo (contém mais do que 7 transações) e atribui o peso 15.

2) classificado com uma nota de 0 a 5 para cada um dos 4 Fatores de Complexidade Técnica abaixo descritos:

a) Sistema distribuído;
b) Processamentos Complexos;
c) Concorrência;
d) Segurança;

Se o Caso de Uso que está sendo classificado possuir nota maior do que 3 em qualquer um dos itens acima esse Caso de Uso deverá ser classificado com um grau de complexidade imediatamente superior ao que ele é atualmente. Se mais de dois itens influenciarem com grau maior do que 3 o Caso de Uso deverá ser classificado como complexo.

A justificativa de Diev para trazer os fatores de complexidade técnica para serem analisados para cada Caso de Uso ao invés de realizar esta análise para o produto inteiro, como definido na técnica de Pontos de Casos de Uso original, é que a técnica original foi pensada apenas para software uniformes nos quais cada Caso de Uso costuma ser influenciado por todos os fatores técnicos. Contudo Diev leva em consideração o fato de poder existir Casos de Uso que não sofrem influência de todos os fatores e, por isso, devem ser classificados separadamente quanto à sua complexidade técnica.

Para finalizar este passo basta somarmos todos os pesos dos Casos de Uso e obter assim o PCUNA.

Passo 3 – Calcular o Tamanho Não Ajustado das Novas Funcionalidades – Size(NF)

Size(NF) = PANA + PCUNA

Passo 4 – Calcular o Tamanho Não Ajustado das alterações – Size(Ch)

Para a realização desse passo é necessário identificar cada Caso de Uso que será alterado e calcular a sua largura e a sua profundidade

$$Size(\ Ch\ ) = \sum_{j=1...Largura(BS)}(\ Peso(BUCj\ ) * Profundidade(\ Chj\ )\ ),$$

Onde:

BUCj = é o j-nário caso de uso do sistema base;

Chj = são as mudanças do j-nário caso de uso do sistema base;

Passo 5 – Calcular o Fator de Complexidade Técnica do Projeto - FCT

Dos 13 fatores técnicos originalmente estabelecidos no método de Pontos de Caso de Uso, quatro já foram utilizados. Então, nesse passo devemos proceder como no método original, mas utilizando apenas os 9 fatores técnicos que ainda não foram utilizados.

É importante salientar que a classificação de cada fator técnico deve ser feita levando em consideração apenas o produto atual e não o produto base.

A fórmula para o cálculo do fator de complexidade técnica segue abaixo:

$$Fator = \sum_{i=fatores\ técnicos\ restantes} Nota_i * Peso_i$$
$$FCT = 0.6 + 0.01 * Fator$$

Passo 6 – Calcular o Tamanho do Produto – Size(Pr)

$$Size(Pr) = (\ Size(NF) + Size(Ch)\ ) * FCT$$

Passo 7 – Calcular o fator de complexidade ambiental - FCE

Nesse cálculo basta proceder da mesma forma descrita no método original de Pontos de Caso de Uso.

Passo 8 – Calcular o esforço

O esforço é calculado em pessoas/horas de acordo com a seguinte fórmula:

$$Esforço = Size(\ Pr\ ) * FCE * C(\ BS\ ) * R + Esforço(\ Suppl\ )$$

onde:

C(BS) = complexidade do sistema base;
R = Quantidade pessoas/horas para se implementar um ponto de caso de uso;

Esforço(Suppl) = Esforço necessário para produzir artefatos suplementares como testes de regressão e artefatos de infra-estrutura;

O UCPm aqui proposto não faz nenhuma alteração drástica nos princípios da técnica de Pontos de Caso de Uso e sim faz uma extensão deste, adicionando algumas especificidades para tratar projetos que estão em fase de manutenção.

Ainda Diev, a exemplo de Mohagheghi et al. (2005), ressalta a importância de se possuir uma padronização na escrita dos casos de uso, objetivando maior precisão das estimativas realizadas. Para tanto, recomenda o uso de repositórios de Casos de Uso que provêm reuso de requisitos e de estimativas.

11) Pontos de Caso de Uso e Pontos por Função na gestão de estimativa de tamanho de projetos de software orientados a objeto, [Andrade, 2004].

Nessa dissertação de mestrado Andrade define um processo de gestão de métrica de tamanho que propõe: i) utilizar a métrica Pontos de Caso de Uso (PCU) e a Análise de Pontos por Função (APF) quando são melhores aplicadas, tentando explorar a relação existente entre elas; ii) estimar o tamanho do software continuamente ao longo do processo de desenvolvimento do projeto; iii) sugerir ações gerenciais a serem tomadas durante o planejamento e controle do projeto.

Nesse trabalho, Andrade (2004) faz a relação entre Pontos de Caso de Uso (PCU) e Análise de Pontos por Função (APF). Na investigação dessa relação, os procedimentos de contagens foram aplicados em 19 projetos desenvolvidos com o paradigma de orientação a objetos na academia e na indústria.

Com base na análise estatística desse estudo de contagem, a autora definiu uma equação que representa a relação entre APF e PCU de projetos de software realizados pela academia e outra equação que representa a relação entre APF e PCU em projetos de software desenvolvidos pela indústria. Essas equações permitem a projeção de Pontos por Função a partir de valores de Pontos de Caso de Uso que são definidos logo no início do projeto, podendo-se então fazer uso de bases históricas de Pontos por Função disponibilizadas inclusive por algumas ferramentas de estimativa existentes no mercado, para realizar estimativas de cronograma, esforço entre outras.

12) ***Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department,*** **[Arnold & Pedross, 1998].**

Nesse artigo, Arnold & Pedross resumem as experiências realizadas com métricas de tamanho de software e produtividade no maior instituto bancário da Suíça. Para tanto, foram analisados os documentos de requisitos e os Pontos de Caso de Uso foram medidos com a intenção de testar e calibrar a técnica Pontos de Casos de Uso.

A análise realizada englobou 23 sistemas de grande escala, 64 questionários respondidos por gerentes de projeto, 11 entrevistas com gerentes de projetos selecionados.

Com base na análise foi possível mostrar que a métrica Pontos de Casos de Uso pode ser usada para medir o tamanho das funcionalidades a serem entregues para o usuário, pois se constatou que o tempo gasto para a realização das medidas de um sistema é de uma a duas horas; os líderes de projeto aceitaram muito bem a metodologia da técnica Pontos de Casos de Uso; segundo o autor pode-se realizar uma comparação direta entre PCU e PF, desta forma os PCU medidos variaram em um espaço de 90% a 110% (20% de variação) do valor dos PF calculados, o que, segundo os autores, é satisfatório, tendo em vista que a técnica Pontos por Função tem uma variação de até 30% quando um sistema é medido por líderes de projetos diferentes.

Contudo, algumas dificuldades foram encontradas no decorrer da análise em relação aos Casos de Uso, a saber: os Casos de Uso nunca são atualizados, os projetos eram muito diferentes em relação à integralidade dos Casos de Uso e a modelagem dos cenários não estava fácil de ser entendida. Ainda, o grau de detalhamento dos Casos de Uso analisados variava muito. Essa variação influencia a métrica de tamanho, já que esta varia de acordo com o grau de detalhamento da especificação dos Casos de Uso.

Portanto, Arnold & Pedross (1998) enfatizam que a descrição textual livre usada na especificação dos Casos de Uso pode gerar ambigüidades e, por isso, podem ser insuficientes para o cálculo de métricas de tamanho de software.

Sendo assim, os autores concluem que a automatização do processo de medida de tamanho de software utilizando a métrica Pontos de Caso de Uso ainda é inviável dada falta de formalidade na descrição dos Casos de Uso.

Fica clara, através deste artigo, a necessidade de, no mínimo, o estabelecimento de *frameworks* ou *tamplates* que possam vir a padronizar a escrita da especificação dos Casos de Uso.

13) *Estimating Software Project using ObjectMetrix,* [Cockburn, 2000].

Nesse *white paper,* Cockburn apresenta a técnica *ObjectMetrix* como um exemplo de técnica para se calcular estimativas de tempo, recursos e custo para o desenvolvimento de software orientados a objeto e baseados em componentes.

O desenvolvimento dessa técnica de estimativa foi baseado em projetos reais da indústria de software. A base teórica foi formalizada em um programa de pesquisa que durou dois anos.

Esta técnica de estimativa leva em consideração não somente o escopo e a complexidade do sistema, como também vai além, podendo levar em consideração a característica incremental do desenvolvimento e a reusabilidade oferecida por projetos baseados em componentes. Sendo assim, esta é uma técnica que se baseia em componentes e objetos ao invés de se basear em funções ou dados.

ObjectMetrix ainda é capaz de gerar medidas do software logo nas fases inicias do ciclo de vida de desenvolvimento, e, principalmente, pode ser continuamente refinada provendo uma precisão cada vez maior da medida de tamanho e custo.

A base da técnica *ObjectMetrix* é a medida do tamanho do sistema através da contagem e classificação dos Elementos de Escopo que, segundo a UML (2003), são características fundamentais da grande maioria dos sistemas orientados a objeto e baseados em componentes, a saber: Subsistemas, Classes, Casos de Uso, Componentes, e Interfaces.

Sendo assim, considera-se o projeto de software dividido em duas partes: a primeira, composta de Subsistemas constituído de Classes que implementam os Casos de Uso; a segunda, composta por Classes que implementam um conjunto de Interfaces que constituem Componentes em um desenvolvimento baseado em componentes.

Portanto, a técnica considera o conjunto de Casos de Uso como apenas um tipo de elemento de escopo, sendo que outros tipos de elementos de escopo podem ser considerados como, por exemplo, o projeto de classes, componentes e páginas *webs*. Por isso, em um primeiro momento, logo no início do ciclo de desenvolvimento do software, a técnica já consegue

oferecer estimativas de produtividade, custo e esforço importantes para a tomada de decisão e, em momentos posteriores do desenvolvimento, essa estimativa pode ser apurada conforme outras informações estejam disponíveis, pois quanto mais itens de escopo forem analisados, mais precisas ficam as métricas de tamanho e complexidade dos software.

14) *Estimating Effort by Use Case Points: Method, Tool and Case Study*, [Kusumoto *et al.*, 2004].

Kusumoto et al. mostram, nesse trabalho, o desenvolvimento de uma ferramenta para aferir a medida de Pontos de Caso de Uso que automaticamente classifica a complexidade dos atores e Casos de Uso de um Modelo de Caso de Uso. Para validar a ferramenta ela foi aplicada a Modelos de Caso de Uso existentes e o resultado foi comparado com as estimativas realizadas por especialistas em estimativa de esforço.

Várias ferramentas que dão suporte à contagem de Pontos de Caso de Uso estão disponíveis no mercado. Essas ferramentas extraem os Casos de Uso e Atores do Modelo de Caso de Uso e o usuário insere a complexidade de cada um desses itens manualmente e, em seguida, faz a mesma coisa para os valores dos fatores técnicos e ambientais do projeto.

Dessa forma, a ferramenta basicamente aplica a fórmula nos valores inseridos pelo usuário. Ou seja, o julgamento da complexidade dos Atores e Casos de Uso, segundo os autores o passo mais importante do processo dessa técnica, é determinado pelo usuário e não são calculados automaticamente.

Neste trabalho Kusumoto et al. sugerem um método de classificação automática de Casos de Uso e Atores.

Para a classificação automática da complexidade dos Atores são propostas três regras:

1) Classificação baseada no nome do ator: Um Ator pode representar uma pessoa ou um outro sistema. Segundo os autores, pelo nome do Ator é possível inferir o que ele está representando, ou seja, nomes com as palavras “sistema”, “servidor”, “aplicação”, “ferramenta”, entre outros, leva à classificação do Ator como SISTEMA, caso contrário como PESSOA;

2) Classificação baseada em palavras chaves incluídas nos Casos de Uso: Quatro listas de palavras chaves foram definidas pelos autores, a saber:
    a. Palavras chave para Atores Simples: requisito, envia e informa;
    b. Palavras chave para Atores Médio Sistema: mensagem, *mail* e envia;
    c. Palavras chave para Atores Médio Pessoa: comando, texto, inserir e *GUI*;
    d. Palavras chave para Atores Complexos: *Enter,* botão, apertar, arrastar, selecionar, mostrar, *GUI,* janela;

    As transações dos Casos de Uso que se relacionam com um determinado Ator são comparadas com as listas de palavras chaves e aquela lista que melhor representar os Casos de Uso é a que vai classificar o Ator. Dessa forma, se as transações presentes nos Casos de Uso com os quais um determinado ator se relaciona, contiverem as palavras chaves (ou forem melhores representadas) da lista "c" então o Ator poderá ser classificado como de complexidade média;

3) Classificação baseada em dados de base histórica: Quando os passos 1 e 2 são insuficientes para determinar a classificação, uma base de dados histórica deve ser consultada. Se for encontrado um Ator com o mesmo nome do Ator em questão, então a classificação de complexidade deverá ser a mesma;

Segundo o autor, para a classificação automática dos Casos de Uso, poder-se-ia simplesmente realizar a contagem do número de transações que cada Caso de Uso apresenta, contudo o autor decidiu realizar uma análise prévia da especificação para a determinação do que é uma transação, já que a UML não define nenhuma formalidade sobre como especificar um Caso de Uso.

Essa análise prévia dos Casos de Uso se resume a uma análise morfológica e sintática do fluxo de eventos de especificação dos Casos de Uso em busca de padronizar o conceito de transação para só então realizar a contagem.

Utilizando os conceitos acima descritos a ferramenta U-EST foi desenvolvida com a ajuda do sistema de análise gramatical CaboCha. Na Figura 9 é mostrada a arquitetura da ferramenta U-EST.

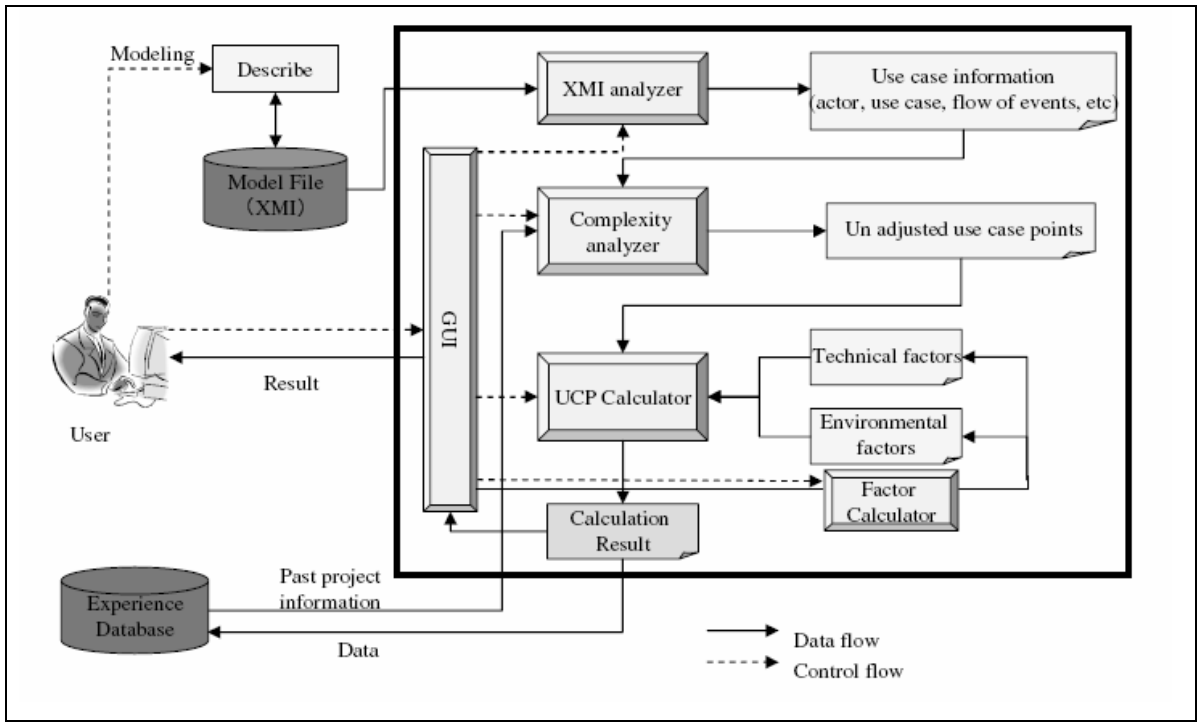

Figura 9 - Arquitetura da ferramenta U-EST

O processo para o uso da ferramenta é mostrado nos seguintes:

1) O projetista escreve o Modelo de Casos de Uso e o salva em formato XMI;

2) O analisador de XMI extrai os Atores e os Casos de Uso automaticamente do XMI;

3) O analisador de complexidade julga automaticamente as complexidades dos Atores e dos Casos de Uso e calcula os Pontos de Caso de Uso Não Ajustados (PCUNA);

4) U-EST mostra os Casos de Uso e Atores com suas respectivas classificações e permite que o usuário as altere, caso seja necessário. Se a alteração for feita então o PCUNA é recalculado;

5) Usuário julga os fatores técnicos e ambientais e a ferramenta calcula o PCU;

6) O esforço de desenvolvimento então é calculado com um fator de 20 homens-horas por unidade de Pontos de Caso de Uso;

Para mostrar a potencialidade dessa ferramenta ela foi aplicada nos Modelos de Caso de Uso de cinco projetos de tamanho médio da empresa Hitachi Systems & Services.

Os resultados obtidos pela medição realizada com o uso da ferramenta U-EST foram comparados com os resultados obtidos por especialistas em contagem de Pontos de Caso de Uso.

Ao todo, os resultados das classificações de complexidade de Atores e Casos de Uso feitas pela ferramenta foram muito similares aos realizadas pelo especialista, contendo algumas diferenças apenas onde havia falta de informações no Modelo de Casos de Uso; nesse momento, a experiência do especialista foi o diferencial na classificação.

Neste trabalho fica claro que a maior dificuldade na aplicação dos Pontos de Caso de Uso é a definição do que é transação dentro do fluxo de eventos que descrevem o Caso de Uso. Isso por que não há formalismo nenhum sobre a escrita do fluxo de eventos, sendo este apenas balizado por melhores práticas e algumas regras existentes na literatura sobre esse assunto.

Então, uma forma encontrada por Kusumoto foi a análise morfológica e sintática do fluxo de eventos para a identificação das transações, já que a entrada para a ferramenta era o Modelo de Caso de Uso já desenvolvido.

14) Diretrizes para elaboração de Documento de Requisitos com ênfase nos Requisitos Funcionais [Kawai, 2005].

Segundo Kawai, embora haja vários padrões para a elaboração de um documento de requisitos, nenhum deles explica o que realmente deve ser escrito na seção que especifica um requisito funcional.

Ainda, muitas vezes, cada requisito é exposto de uma forma diferente num mesmo DR, alguns abordando mais detalhes que outros ou com as informações agrupadas em um único parágrafo. Desta forma, partes do requisito funcional ficam subentendidas, tendo o desenvolvedor que fazer suposições sobre o que está sendo requisitado.

Além de afetar a fase de implementação do software, um Documento de Requisitos que traz um requisitos escrito de várias maneiras diferentes e com graus de detalhamento diferentes acaba dificultando a aplicação da técnica TUCCA [Belgamo, 2004].

Dessa forma, nesse trabalho de mestrado, Kawai explorou a criação de diretrizes para a construção de Documentos de Requisitos com enfoque na especificação dos requisitos funcionais, objetivando facilitar a aplicação da técnica TUCCA [Belgamo, 2004], abordando:

- formato para especificação de requisitos funcionais no Documento de Requisitos;
- recomendações de escrita do Documento de Requisitos;
- *checklist* pré-inspeção, no intuito de definir se o Documento de Requisitos deve ou não ser submetido a uma inspeção.

Como resultado do trabalho foi apresentado um conjunto de elementos que devem ser levantados e documentados para cada um dos Requisitos Funcionais, conforme mostra a Figura 10.

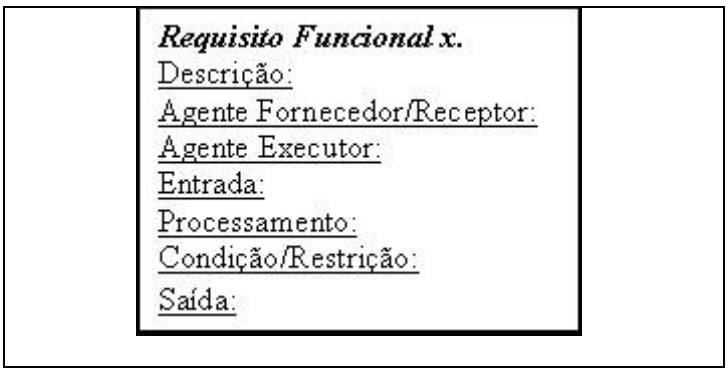

Figura 10 – Formato de um Requisitos Funcional [Kawai, 2005].

Abaixo segue a explicação de cada item da Figura 10.

- *Identificação do Requisito*: este cabeçalho permite que cada requisito funcional seja identificado unicamente através de um número denotado por x.
- *Descrição:* este item serve para explicar, de forma sucinta, a idéia principal da funcionalidade de um requisito, descrevendo "o que" fazer e não "como" fazer.
- *Agente Fornecedor/Receptor:* representar qualquer entidade que interage com o sistema de forma indireta, ou seja, que fornece as informações ou dados de entrada para o sistema e conseqüentemente recebe a resposta.
- *Agente Executor:* representar qualquer entidade que interage com o sistema de forma direta, ou seja, informa ao sistema as informações obtidas do "Agente Fornecedor/Receptor". É um agente que opera, que executa o sistema.
- *Entrada*: neste item são descritas as informações de entrada, ou seja, o(s) dado(s) que o sistema recebe e processa para a execução de uma determinada funcionalidade.
- *Processamento*: ao contrário do item "Descrição", que é bem geral, neste item é descrito com mais detalhes o fluxo de atividades ou ações que o sistema deve realizar para alcançar o que foi escrito no item "Descrição".

- *Condição/Restrição*: como o próprio nome diz, este item indica a condição ou restrição necessária para que seja realizada a funcionalidade requerida do sistema.
- *Saída*: este item indica a resposta do sistema dada uma determinada funcionalidade. Descreve a saída gerada pelo sistema de acordo com o Requisito Funcional especificado.

Kawai, com o intuito de melhorar a forma de registrar os requisitos no Documento de Requisitos, apresenta recomendações para a escrita desses requisitos. Essas recomendações abrangem regras gramaticais para a escrita de um requisito, como usar acrônimos, termos e símbolos que devem ser evitados na escrita de um requisito entre outros.

Neste trabalho foi realizado um estudo de caso por meio do qual foi possível constatar que documentos de requisitos construídos seguindo o padrão estabelecido por Kawai e que foram submetidos à TUCCA para o levantamento de defeitos no Documento de Requisitos apresentaram uma menor quantidade de defeitos que os documentos que não seguiram as diretrizes de Kawai.

Sintetizando os trabalhos analisados nesta seção, pode-se concluir que a métrica de Pontos de Casos de Uso tem sido bastante explorada e tem se mostrada, através dos dados apresentado pelos autores, ser uma medida bastante significativa para se realizar várias estimativas sobre um produto de software a ser desenvolvido, embora ainda necessite de investigação para torná-la de aplicação mais geral.

Nesse sentido, Andrade (2004) fez um relacionamento linear da medida de Pontos de Caso de Uso (PCU) com a medida de Pontos por Função, medida esta consagrada na indústria de desenvolvimento de software e na academia, sugerindo o uso da técnica de PCU para a estimativa de esforço utilizando bases históricas já consolidas de medidas de Pontos por Função.

Ainda salientando a eficiência da métrica PCU, Anda (2001), Anda (2002) e Arnold & Pedross (1998) mostraram a aplicação satisfatória dessa métrica em vários desenvolvimentos de software nas mais variadas áreas de aplicação utilizando várias tecnologias diferentes.

Contudo, vários autores, entre eles Anda (2001), Mohagheghi et al. (2005), Kusumoto et al (2004), Vinsen et al. (2004), Diev (2006) deixaram claro que, apesar do desempenho satisfatório dos PCU na estimativa de tamanho de software, essa técnica apresenta uma deficiência causada pela forma com a qual o Modelo de Casos de Uso é construído e os Casos de Uso são especificados, pois essas atividades não são totalmente formalizadas pela linguagem UML. Isso deixa o indivíduo que está modelando os requisitos do sistema desenvolver o Modelo de Casos de Uso sem nenhuma sistematização e especificar os Casos de Uso com o grau de detalhamento desejado e sem nenhum tipo de padronização. Essa grande variedade de formas de especificação acaba influenciando diretamente a contagem dos Pontos de Caso de Uso, já que as transações da especificação dos Casos de Uso são usadas para a classificação da complexidade desses Casos de Uso.

Belgamo (2004) desenvolveu uma técnica para sistematizar a construção do Modelo de Casos de Uso e mostraram que o uso dessa técnica, em conjunto com a técnica de PCU, gera estimativas com maior qualidade em relação a estimativas geradas pela aplicação da técnica de PCU em Modelos de Casos de Uso desenvolvidos sem nenhuma sistematização (ad-hoc).

Já Kusumoto et al (2004), adotaram uma técnica diferenciada, baseada em análise gramatical (sintática e morfológica), para a determinação do que é uma transação que deve ser contada na especificação dos Casos de Uso para efeito de classificação da complexidade desses Casos de Uso na técnica de Pontos de Casos de Uso. Contudo, essa técnica ainda apresenta algumas falhas quando o Modelo de Caso de Uso utilizado como artefato de entrada para a ferramenta U-EST apresenta alguma deficiência de informação.

Um outro problema apresentado pelos autores está relacionado ao avanço das fases de desenvolvimento do software, pois conforme o desenvolvimento do software evolui, novas informações, como Diagrama de Classes e Interfaces, são adicionadas ao projeto podendo deixar a métrica de tamanho de software mais precisa [Cockburn, 2000]. Porém, a métrica original de Pontos de Caso de Uso, proposta por Karner (1993), não leva em consideração esse tipo de informação, perdendo a oportunidade de ficar mais precisa à medida que a construção do software avança.

Também, a métrica original de Pontos de Caso de Uso proposta foi idealizada e testada apenas em projetos pequenos e nunca utilizando um ciclo de vida incremental. Por isso, Mohagheghi et al., (2005) propõem uma adaptação da técnica de Pontos de Caso de Uso para

que ela também possa ser utilizada em software concebidos através desse tipo de ciclo de vida.

Observados esses trabalhos e analisadas as formas com que o PCU é calculado, fica clara a necessidade de uma ferramenta que seja capaz de deixar a métrica de tamanho de software sempre atualizada, de acordo com o documento de requisitos e que busque a padronização do Modelo de Casos de Uso e da especificação dos Casos de Uso de forma que a métrica não cause muitas distorções e que seja o mais precisa possível, independentemente do indivíduo que a utiliza. Ressalta-se que no ambiente que foi desenvolvido como parte deste trabalho, implementou-se a técnica PCU na sua forma original, o que já traz uma contribuição bastante significativa para o engenheiro de software, para que este possa, ao menos, ter uma previsão de valores como esforço, custo e tempo de desenvolvimento, utilizando o PCU em técnicas de estimativa.

5.3 *Ferramentas para Automatização de Métricas de Software*

Pontos por Função e Pontos de Casos de Uso têm sido propostos para estimar desenvolvimentos de software em várias organizações. Com isso, várias ferramentas que dão suporte a eles foram desenvolvidas para suprir essa demanda [Kusumoto et al., 2004].

Esta seção comenta algumas ferramentas disponíveis no mercado para a medida de tamanho de software usando a técnica Pontos de Caso de Uso e Pontos por Função. Procurou-se identificar as funcionalidades oferecidas por essas ferramentas e até que ponto elas automatizam o processo de medida de tamanho de software.

Contudo, dentre as ferramentas pesquisadas, o que mais se encontrou foram ferramentas que usam a métrica de tamanho de software, que foi calculada de forma semi-automática, para calcular as estimativas de esforço e tempo de desenvolvimento.

Diz-se semi-automática a forma do cálculo da métrica, pois esta não é baseada em algum modelo formal de descrição de requisitos do software através do qual a ferramenta possa fazer a leitura em busca de dados como complexidade de Casos de Uso ou número de arquivos internos (ALIs) e, automaticamente, determinar a medida. Em outras palavras,

sempre que alguma dessas ferramentas calcula alguma métrica de tamanho é porque o usuário teve antes que analisar o software e informar seus dados de complexidade, número de arquivos, etc., para a ferramenta aplicar uma fórmula simples e chegar ao valor da métrica.

1. Construx Estimate

Construx Estimate é uma ferramenta desenvolvida pela *Construx* Software *Builders* que ajuda a melhorar a qualidade da métrica do tamanho do software [Construx, 2005]. Ela utiliza as técnicas de estimativa COCOMOII. A Tabela 11 mostra as funcionalidades providas por essa ferramenta.

Tabela 11 – Principais funcionalidades da ferramenta Construx Estimate [Construx, 2005].

Funcionalidade	Descrição
Calibração através de dados históricos	Através dessa funcionalidade o usuário pode calibrar a ferramenta com dados da sua própria base histórica.
Dados da indústria	Caso o usuário ainda não tenha uma base histórica de dados, a ferramenta oferece uma proveniente de diferentes tipos de projeto da indústria de software.
Estimativas através da interface gráfica	A ferramenta possibilita a estimativa de software, ainda nas fases iniciais do projeto, baseando-se na contagem das caixas de diálogos, relatórios, saídas gráficas e etc.
Pontos por Função e linhas de código	Para o cálculo do esforço e elaboração de cronogramas, esta ferramenta permite a entrada direta da contagem de Pontos por Função e/ou linhas de código.

Como se pode perceber pela Tabela 11, a *Construx Estimate* não faz a análise de um modelo para a geração da estimativa. A estimativa é baseada na interface gráfica, o que faz necessária a construção de um protótipo, mesmo que em papel (*mock-up*), para a geração das estimativas. Uma vantagem dessa ferramenta é que ela traz uma base de dados histórica, já pronta, para usuários novos na tarefa de estimativa de software, que ainda não possuem o próprio histórico.

2. Cost Xpert 3.3 – Software *Cost Estimation and Project Planning*

Essa ferramenta foi desenvolvida pela empresa Cost Xpert *Group Incorporated* e estima tamanho de software utilizando vários tipos de modelos diferentes para calcular essa métrica

e ainda disponibiliza o modelo COCOMO com a opção de vários tipos de ciclo de vida de desenvolvimento para o cálculo de esforço e custos [Costxpert, 2005]. A Tabela 12 traz algumas das funcionalidades desse software.

Tabela 12 – Principais funcionalidades da ferramenta Cost Xpert [Costxpert, 2005].

Funcionalidade	Descrição
Cálculo da estimativa de custo e esforço	Esta ferramenta faz o cálculo da estimativa de custo e esforço através do modelo COCOMO que tem a opção de trabalhar com os mais variados tipos de ciclo de desenvolvimento de software.
Medida do tamanho do software	O Cost Xpert é capaz de calcular a métrica do tamanho do software através de várias técnicas como SLOC, *Feature Points, Internet Points*, *Domino Points,* Pontos de Caso de Uso, Pontos por Função entre outros.
Distribuição dos custos do sistema	Após o cálculo do esforço a ferramenta é capaz de distribuir os custos de desenvolvimento entre as diferentes fases do projeto como Análise, Projeto, Implementação entre outros.

3. COSMOS – *The* Software *Cost Modeling System*

O COSMOS é uma ferramenta de cálculo de tamanho, esforço e cronograma de software desenvolvido pela *East Tennessee State University.* Essa é a única ferramenta que combina três técnicas: Pontos por Função, COCOMO e o modelo *Putman* proposto por Lawrence Putnam [Cosmos, 2005]. A Tabela 13 mostra as principais características desse software.

Tabela 13 – Principais funcionalidades da ferramenta COSMOS [Cosmos, 2005].

Funcionalidade	Descrição
Cálculo da estimativa de custo	Esta ferramenta faz o cálculo da estimativa de custo e esforço através do modelo COCOMO, *Rayleigh* e Pontos por Função.
Medida do tamanho do software	O COSMOS é capaz de realizar o cálculo da métrica do tamanho do software através do uso de Pontos por Função.
Cálculo da complexidade do sistema	A complexidade do sistema é calculada através do fator de ajuste técnico do Pontos por Função.
Cálculo da métrica SLOC – Linhas de código fonte	Faz o cálculo de SLOC através da métrica Pontos por Função e da linguagem que será usada para a implementação do software.

4. EEUC – *Estimate Easy Use Case*

O EEUC é uma ferramenta de medida de tamanho e estimativa de custo de software, desenvolvida pela empresa Duvessa Software, que realiza as medidas nas fases iniciais do projeto através da técnica Pontos de Caso de Uso [EEUC, 2005]. A Tabela 14 mostra algumas das funcionalidades apresentadas pela ferramenta EEUC.

Tabela 14 – Principais funcionalidades da ferramenta EEUC [EEUC, 2005].

Funcionalidade	Descrição
Cálculo da estimativa de custo	Este cálculo é realizado com base no COCOMO II.
Medida do tamanho do software	O tamanho do software é medido através dos Pontos de Caso de Uso.
Cálculo da complexidade do sistema	A complexidade do sistema é calculada através do fator de complexidade técnica e do fator ambiental provenientes dos Pontos de Caso de Uso
Elaboração de cenários comparativos	É possível a elaboração de cenários através dos quais o usuário pode alterar atores e/ou Casos de Uso e verificar o impacto disto nas estimativas de software.
Importação de Casos de Uso e atores de outras ferramentas	Pode-se realizar a importação das informações de Casos de Uso e atores de outros software como Rational Rose, Rational XDE e outros.

A Tabela 15 resume as funcionalidades das ferramentas apresentadas nesse trabalho.

Tabela 15 – Resumo das funcionalidades das ferramentas analisadas

Funcionalidade	**Ferramentas**
Calibração da ferramenta por meio de dados históricos	1
Base de dados histórica com dados da indústria já cadastrados na ferramenta.	1
Estimativa utilizando a interface gráfica do software	1
PF ou SLOC	1,2,3
Cálculo de estimativa de custo e esforço	1,2,3,4
Outras métricas além de PF, PCU e SLOC	2

Distribuição dos custos do projeto entre as fases do ciclo de vida do desenvolvimento	2
PCU	2,4
Cálculo da complexidade do software	3,4
Elaboração de cenários comparativos	4
Coleta automática das informações	0

Outras ferramentas, menos expressivas para este trabalho, foram analisadas, algumas delas estão abaixo relacionadas:

- ACEIT – *Automated Cost Estimating Integrated Tools*: *Framework* de ferramentas desenvolvida através de um projeto patrocinado pelo Governo norte americano que visa a reportar estimativas de desenvolvimento de projeto de software [Aceit, 2005].
- *PRICE Cost Models* – Esta ferramenta desenvolvida pela *Price System LLC* traz o histórico de centenas de projetos de software com os seus respectivos custos de desenvolvimento, além de estimar projetos de desenvolvimento de *hardware* através de metodologias desenvolvidas pelo departamento de defesa dos Estados Unidos [Price, 2005].
- *CoStar Software Estimation Tool* – Esta é uma ferramenta desenvolvida pela *Softstar* com o intuito de estimar duração, esforço e custo de projetos de software através do uso do COCOMO II [Costar, 2005].

Conforme observado nesta seção, o cálculo da métrica de tamanho de software feita pelos software acima descritos não é totalmente automático, ou seja, o usuário sempre tem que dar informações como as complexidades de atores e Casos de Uso para o cálculo dos Pontos de Caso de Uso, ou então prover informações sobre a complexidade das Entradas Externas, Saídas Externas, Consultas Externas, Arquivos Lógicos Internos e Interface Lógica Externa para o cálculo dos Pontos por Função.

Das ferramentas analisadas, algumas se propõem a fazer algum tipo de medida de tamanho de software, já outras simplesmente realizam a estimativa do custo ou cronograma baseada em informações de métricas de tamanho de software que o usuário insere, ou seja, não há

nenhuma automatização da métrica de tamanho de software, há simplesmente um cálculo da estimativa de esforço baseado em métricas que o usuário já deve possuir inicialmente.

Assim sendo, dentro do conjunto de ferramentas pesquisadas, não existe nenhuma que aceite como entrada um modelo de requisitos bem formatado, como um Modelo de Casos de Uso ou algum outro tipo de diagrama e, através dele gere, automaticamente, algum tipo de métrica de tamanho de software como SLOC, Pontos de Caso de Uso ou Pontos por Função.

O ambiente COCAR desenvolvido neste trabalho foi especificado com o objetivo de cobrir as restrições apresentadas pelas ferramentas analisadas no que tange a disponibilizar uma forma semi-automática e padronizada de inserir requisitos/casos de uso no sistema para o cálculo dos Pontos de Caso de Uso.

5.4 *Considerações Finais*

Apresentaram-se neste capítulo alguns trabalhos teóricos relacionados com estimativa de tamanho de software principalmente com a utilização da técnica PCU, isso, pois esta técnica é a base para a implementação da parte de estimativa de tamanho de software do ambiente COCAR que foi o objetivo principal desta dissertação de mestrado.

Além dos trabalhos teóricos também foram apresentadas as principais ferramentas acadêmicas ou do mercado de desenvolvimento de software que, de certa forma, automatizam o cálculo da métrica de tamanho de software, sendo importante o entendimento das principais funcionalidades dessas ferramentas buscando sempre o *benchmarking* (processo contínuo de comparação de produto, serviço e práticas) com objetivo de identificar os requisitos que este tipo de ferramenta deve conter.

Conhecendo o que o mercado acadêmico/profissional estuda e espera de uma ferramenta de estimativa de tamanho de software, no próximo capítulo será apresentado o Ambiente COCAR, mostrando detalhes de funcionalidade e implementação da ferramenta.

Capítulo 6
O Ambiente COCAR

6.1 *Considerações Iniciais*

Como visto nos capítulos anteriores, diversos autores falam da necessidade e da importância de ferramentas que dêem apoio às atividades do desenvolvimento de software [Munson & Nguyen, 2005, Marcus et al, 2005, Egyed & Grübacher, 2002, Sommerville, 2003, Alexander, 2003]. Tais ferramentas são denominadas ferramentas CASE (*Computer Aided Software Engineering*) e o sucesso do uso está diretamente ligado à capacidade que têm de abrangerem o processo de desenvolvimento de software de forma integrada. De acordo com Munson e Nguyen (2005), quanto maior o número de etapas do processo de desenvolvimento de software que a mesma ferramenta abranger melhor o resultado que ela poderá proporcionar, evitando erros de integração e favorecendo a rastreabilidade dos artefatos gerados, aumentando a qualidade no processo.

Além disso, como também foi mencionado anteriormente, os Modelos de Casos de Uso são, atualmente, uma técnica bastante usada para modelar os requisitos do software, a qual pode ser adotada em diversos paradigmas de desenvolvimento e, com base na qual, são extraídas várias outras informações que podem auxiliar em outras etapas do processo de desenvolvimento.

Assim, o ambiente COCAR foi idealizado com objetivo de dar suporte a algumas atividades do desenvolvimento de software, considerando como base fundamental os Modelos de Casos de Uso. Visando à construção desse ambiente, alguns trabalhos de mestrado estão sendo conduzidos com objetivo de avaliar o estado da arte e outras propostas existentes na literatura de forma a estabelecer estratégias e soluções que possam auxiliar e melhorar a qualidade no processo de desenvolvimento de software.

Sendo um ambiente baseado em Modelos de Casos de Uso o suporte à elaboração dos mesmos foi a primeira funcionalidade a ser implementada, para que outras funcionalidades

pudessem ser providas pelo ambiente, inclusive a automação da técnica PCU, que foi o principal objetivo deste trabalho. A construção desses modelos foi totalmente baseada na técnica TUCCA [Belgamo, 2004] e a definição dos requisitos funcionais com base nos quais a TUCCA é aplicada para gerar os modelos, foi baseada nas diretrizes propostas por Kawai (2005). Esses dois trabalhos foram comentados no Capítulo 5 e a implementação dos mesmos no ambiente COCAR foi realizada no âmbito deste trabalho, em conjunto com o mestrado de Di Thommazo (2007), uma vez que essas funcionalidades eram necessárias para ambos os trabalhos. Dessa forma, salienta-se que as Seções 6.2 e 6.3 deste capítulo, por se tratarem de partes do ambiente que foram desenvolvidas em conjunto, também foram escritas em conjunto e, portanto, constam dos dois trabalhos.

Assim, o objetivo deste capítulo é apresentar os aspectos funcionais e de implementação do ambiente COCAR. Na Seção 6.2 são apresentadas as principais funcionalidades. Os aspectos de implementação são abordados na Seção 6.3. Em seguida, na Seção 6.4, apresentam-se os requisitos funcionais do ambiente COCAR no que diz respeito aos aspectos da automatização da técnica de Pontos de Caso de Uso e por fim, a Seção 6.5 contém as considerações finais.

6.2 *Funcionalidades do Ambiente* COCAR

O ambiente COCAR apresentado neste trabalho corresponde à sua primeira versão, que contém as seguintes funcionalidades:

i) Inserção dos Requisitos de um sistema.

ii) Elaboração de Modelos de Casos de Uso.

iii) Geração Automática de Pontos de Casos de Uso.

iv) Suporte ao Gerenciamento de Requisitos.

v) Geração de Casos de Teste com base em Casos de Uso.

Cada uma dessas cinco funcionalidades, comentadas brevemente a seguir, corresponde a um trabalho de mestrado.

i) Inserção dos Requisitos de um sistema:

Esse módulo do COCAR corresponde ao trabalho de mestrado de Kawai (2005), intitulado "Diretrizes para elaboração de Documento de Requisitos com ênfase nos Requisitos Funcionais" que, como apresentado no Capítulo 5, propôs diretrizes para a construção do Documento de Requisitos, facilitando a identificação das entidades envolvidas e seus papéis, informações de entrada e saída para o processamento associado à funcionalidade, descrição do processamento da funcionalidade em si, além de restrições e condições de execução de cada funcionalidade. No caso desse trabalho, o ambiente COCAR dá suporte à inserção dos requisitos, de acordo com as diretrizes propostas. Essa funcionalidade permite que as informações de um documento de requisitos sejam inseridas no ambiente para que, a partir delas, o Modelo de Casos de Uso possa ser gerado. Em vista das necessidades específicas da rastreabilidade de requisitos, dois outros campos de informação são tratados no ambiente, além daqueles necessários para contemplar a proposta de Kawai (2005), a saber:

- Solicitante do Requisito: refere-se ao *stakeholder* que levantou esse requisito como necessidade para o sistema. Caso alguma alteração seja proposta nesse requisito ao longo do desenvolvimento, essa pessoa deve ser questionada sobre a mudança. Como já visto neste trabalho, o acompanhamento do requisito desde sua origem, é denominado pré-rastreamento ([Zisman e Spanoudakis, 2004] e [Letelier, 2002]).
- Gerente do Requisito: refere-se à pessoa responsável por acompanhar o ciclo de vida de um requisito, ao longo do processo de desenvolvimento de software. Dessa forma, é possível a realização de pesquisa para verificar quais são os requisitos sob responsabilidade de determinado gerente.

O trabalho de Salem (2005) também salienta a importância de armazenar os requisitos em um banco de dados.

ii) Elaboração de Modelos de Casos de Uso:

Esse módulo do COCAR corresponde ao trabalho de mestrado de Belgamo (2004) comentado no Capítulo 5. O trabalho de Belgamo propôs uma técnica de leitura cujo objetivo principal é auxiliar na elaboração de Modelos de Casos de Uso fazendo com que essa atividade torne-se mais sistematizada e independente da experiência do desenvolvedor, como também menos suscetível a maus entendimentos que gerem modelos não coerentes com o Documento de Requisitos. Ao mesmo tempo em que auxilia na atividade de construção, os passos da

TUCCA propiciam a identificação de defeitos no Documento de Requisitos. Cada Caso de Uso contém a descrição de seus passos, informações de entrada e saída, condições para execução do curso normal e dos cursos alternativos, restrições de tempo e desempenho, relações de dependência com outros Casos de Uso, além dos relacionamentos de inclusão e extensão em que o Caso de Uso está envolvido. Ao final da aplicação da TUCCA, têm-se o Diagrama de Casos de Uso e as Especificações dos Casos de Uso, sendo que as informações para compor o diagrama estão contidas nas próprias especificações. Ou seja, o ambiente não gera o diagrama, graficamente, mas as especificações possuem toda informação para que se consiga construí-lo. Ressalta-se que, como a implementação do trabalho de Belgamo (2004) aconteceu no âmbito deste trabalho, por questões de tempo e de focar no que era essencial para implementação dos PCU, a parte relacionada ao relato de discrepâncias não foi implementa nesta primeira versão do ambiente COCAR.

iii) Geração de Pontos de Casos de Uso:

Esse módulo do COCAR corresponde a este trabalho de mestrado. Essa funcionalidade faz a geração dos Pontos de Casos de Uso com base nas especificações dos Casos de Uso elaboradas com a aplicação da TUCCA. Seguindo o *template* proposto em Belgamo (2004), o qual é usado para descrever as especificações, as informações necessárias para a geração dessa métrica são organizadas de maneira a facilitar a sua geração. O valor dos Pontos de Casos de Uso também pode ser convertido em Pontos por Função. Esses dados auxiliam na fase de planejamento do software, uma vez que podem ser usados para estimar tempo, custo e esforço de desenvolvimento. No futuro, pretende-se que tais estimativas sejam também geradas pela ferramenta.

iv) Suporte à Rastreabilidade entre os Requisitos e o Modelo de Casos de Uso:

Esse módulo do COCAR corresponde ao trabalho de mestrado de Di Thommazo (2007) intitulado "Gerenciamento de Requisitos no Ambiente COCAR" e possibilita o gerenciamento dos requisitos. Ele oferece suporte para a rastreabilidade tanto no âmbito do próprio Documento de Requisitos como entre esse documento e o Modelo de Casos de Uso. Considerando a natureza dinâmica dos requisitos, essa funcionalidade possibilita o acompanhamento das alterações, informação esta que poderá ser usada também para auxiliar a geração de Casos de Teste de regressão, o qual deve ser aplicado sempre que o sistema sofrer alterações.

Nesse módulo é gerada a Matriz de Rastreabilidade de Requisitos, a qual indica, de forma automática, qual a relação existente entre os requisitos funcionais do software. Essa matriz pode ser utilizada para a previsão do impacto gerado nos demais requisitos do software, quando ocorrem modificações em um requisito específico. Por meio desse relacionamento, o usuário pode averiguar a necessidade ou não de alterar os demais requisitos relacionados com aquele em que a mudança foi planejada ou executada.

Esse módulo do ambiente COCAR também calcula o Indicador de Estabilidade, proposto por Hazan e Leite (2003), o qual fornece o percentual de alterações, inclusões ou exclusões de requisitos no sistema, em qualquer período do ciclo de desenvolvimento do software. Além disso, também são oferecidas facilidades para consultas sobre os requisitos cadastrados.

v) Geração de Casos de Teste com base em Casos de Uso:

Esse módulo do COCAR corresponde à proposta de trabalho de mestrado de Gregolin (2006), que esta em andamento, intitulada "Uma Proposta de Ferramenta para a Geração de Casos de Teste a partir de Casos de Uso", que consiste em gerar casos de teste a partir dos Casos de Uso. Como esse trabalho ainda está em andamento, não se tem maiores detalhes sobre essa funcionalidade. Em linhas gerais, pode-se dizer que, a partir dos Casos de Usos definidos com a aplicação da TUCCA, serão gerados casos de teste que explorem cenários de uso do sistema, definidos a partir da elaboração de um grafo construído com base nas especificações dos casos de uso. Além disso, pretende-se que, no futuro, a ferramenta dê suporte à geração de casos de teste que possam ser aplicados para testar os Casos de Uso individualmente. Como o ambiente COCAR prevê o controle das alterações que ocorrem nos requisitos, este módulo deverá dar suporte à geração de Casos de Teste de regressão. Dessa forma, será atendida a característica extensível do ambiente COCAR, ampliando a rastreabilidade dos artefatos envolvidos no processo de desenvolvimento de software.

Em resumo, o propósito deste mestrado foi integrar os trabalhos de [Kawai, 2005] (funcionalidade i) e [Belgamo, 2004] (funcionalidade ii) com ênfase na Geração dos Pontos de Caso de Uso (funcionalidade iii) permitindo gerar a métrica de Pontos de Caso de Uso de forma mais padronizada, automática e cada vez mais independente do indivíduo que projeta o Modelo de Casos de Uso.

Ainda, a funcionalidade de transformação da métrica de Pontos de Caso de Uso para a métrica de Pontos por Função proposta por Andrade [2004] foi implementada, já que

atualmente a métrica de Pontos por Função é largamente utilizada e há várias bases históricas de tempo de desenvolvimento de software baseadas nessa métrica.

6.3 *Aspectos de Implementação do Ambiente COCAR*

Nesta seção serão comentados os aspectos mais relevantes relacionados ao desenvolvimento do ambiente COCAR. Primeiramente fala-se sobre o processo de desenvolvimento escolhido para o desenvolvimento do ambiente; em seguida, comenta-se sobre a linguagem de programação bem como os padrões de projeto presentes na literatura e utilizados na implementação do ambiente COCAR; e, finalmente, fala-se da implementação, abordando-se o modelo utilizado para a codificação do ambiente.

6.3.1 Processo de Desenvolvimento

De acordo com Pressman (2006) um processo de desenvolvimento de software deve ser escolhido baseado em três fatores:

- na natureza do projeto e da aplicação;
- nos métodos e ferramentas a serem utilizados;
- nos controles e produtos que precisam ser entregues;

Quanto à natureza do projeto e da aplicação, o ambiente COCAR pode ser considerado um software relativamente complexo, uma vez que seu objetivo é dar suporte a várias atividades do desenvolvimento de software que, embora completamente diferentes, são baseadas no mesmo alvo, isto é, os Modelos de Casos de Uso. Consequentemente, a implementação do ambiente deveria ser iniciada com as funcionalidades relativas à automatização da TUCCA [Belgamo, 2004] e ao tratamento da entrada dos requisitos funcionais, de acordo com as diretrizes propostas por Kawai (2005).

No que diz respeito à TUCCA, embora a técnica estivesse totalmente bem especificada e com seus passos bem detalhados e definidos, transformá-la em uma ferramenta implica na definição de uma série de outros detalhes que não são necessários para sua aplicação manual, como por exemplo, detalhes de interface, entradas e saídas de cada passo, etc. Dessa forma,

havia o domínio completo da aplicação da técnica manual e possuíam-se os requisitos gerais para a sua automatização, mas vários detalhes precisavam ser definidos.

Quanto aos métodos, o ambiente COCAR foi desenvolvido utilizando-se a própria abordagem que essa ferramenta implementa, ou seja, a especificação de requisitos foi feita por meio da estrutura definida por Kawai (2005), para a especificação de requisitos, a geração do Modelo de Casos de Uso foi realizada aplicando-se, manualmente, a TUCCA [Belgamo, 2004] e o cálculo da estimativa de tamanho foi realizado aplicando-se, manualmente, Pontos de Caso de Uso [Karner, 1993]. Uma vez gerado o Modelo de Casos de Uso do ambiente COCAR por meio da TUCCA, para se obter a representação gráfica do diagrama de casos de uso utilizou-se a ferramenta Rational Rose [Rose, 2006].

Quanto aos controles e produtos que precisavam ser entregues, considerou-se como imprescindíveis: o documento de especificação de requisitos do ambiente definido de acordo com as diretrizes de Kawai (2005); o Modelo de Casos de Uso gerado pela técnica TUCCA, manualmente; e, versões parciais do ambiente COCAR, implementando as funcionalidades especificadas por meio de ciclos de desenvolvimento.

Levando em consideração os critérios sugeridos por Pressman (2006) para a escolha do processo de desenvolvimento de software decidiu-se adotar para o desenvolvimento do ambiente COCAR o ciclo de vida de prototipação descartável seguida de prototipação evolutiva, pois:

- Como dito anteriormente, o ambiente COCAR, no inicio de seu desenvolvimento, se caracterizava como sendo um projeto no qual se conhecia os requisitos gerais do sistema, mas não todos os seus detalhes. Para um projeto dessa natureza o modelo de prototipagem funciona como um mecanismo para identificar esses requisitos, principalmente quando se tem a definição de um conjunto de objetivos gerais para o software, mas os requisitos detalhados ainda não foram identificados ou o desenvolvedor está inseguro em relação à forma com que a interação homem-máquina deve acontecer [Pressman, 2006];
- relativo aos métodos utilizados, segundo Sommerville (2003), o modelo de prototipagem descartável exige ao final da fase de Engenharia de Requisitos um protótipo descartável juntamente com um documento de especificação, contudo não define nenhum *template* desse documento e também esse modelo não faz restrições a

forma de modelagem e especificação dos requisitos do software, sendo permitido portanto o uso do documento de requisitos de acordo com as diretrizes propostas por Kawai (2005) e a aplicação da técnica TUCCA para a geração do Modelo de Casos de Uso;

- relativo às ferramentas utilizadas, a prototipação evolucionária não exige o uso de nenhuma ferramenta de apoio específica, enquanto que a prototipação descartável sugere o uso de alguma ferramenta que seja capaz de gerar interfaces gráficas rapidamente. No caso deste trabalho, para a geração do protótipo descartável, foi escolhida a linguagem HTML que, embora não seja uma ferramenta propriamente dita, é uma linguagem pela qual se desenvolve interfaces gráficas rapidamente e foi utilizada para gerar as telas do protótipo descartável e também, posteriormente, para a implementação da interface gráfica do protótipo evolucionário;
- relativo aos produtos e controles a serem entregues, a prototipação evolucionária, como já mencionado, exige a entrega de um documento de requisitos o qual foi realizado na especificação do ambiente COCAR por meio das diretrizes especificadas por Kawai (2005). Outro controle exigido pela prototipação descartável seguida pela prototipação evolucionária são os releases de protótipos e posteriormente os releases de versões executáveis, exigências essas que atendem perfeitamente as necessidades de desenvolvimento do ambiente COCAR.

O Processo de prototipação descartável se deu conforme enunciado por Sommerville (2003):

1) Desenvolvimento de um documento de requisitos, segundo as diretrizes propostas por Kawai (2005), contendo as principais funcionalidades a serem atendidas pelo ambiente COCAR, dentre elas a técnica TUCCA [Belgamo, 2004], a técnica Pontos de Casos de Uso e o Gerenciamento de Requisitos. Além disso, também foram consideradas características de ferramentas relacionadas ao contexto do ambiente COCAR e discutidas no capítulo anterior.
2) Projeto e construção de um protótipo, a partir do documento de requisitos.
3) Submissão do protótipo à avaliação do grupo envolvido no projeto (professores e alunos) que continha pessoas experientes principalmente no uso da técnica TUCCA.
4) Se o protótipo fosse avaliado como completo, seguia-se para o passo 7; caso contrário, continuava-se no passo 5.
5) Refinamento do documento de requisitos com base nos dados da avaliação do passo 3.

6) Retorno ao passo 2.
7) Entrega do protótipo final com a especificação.

Após algumas iterações do processo de prototipação e com uma versão final do documento de requisitos do ambiente COCAR, disponível no Apêndice I, aplicou-se, manualmente, a técnica TUCCA gerando-se o Modelo de Casos de Uso apresentado no Apêndice II.

6.3.2 Projeto

Com objetivo de facilitar o acesso ao ambiente COCAR foi decidida a sua implementação em ambiente *Web* Cliente Servidor. Com essa abordagem abstraem-se as dificuldades referentes à instalação da ferramenta, versões de software terceiros como banco de dados e não preocupação com requisitos mínimos para a execução do sistema. Dessa forma, foi estabelecido que a ferramenta fosse desenvolvida como um servidor que pudesse ser acessada através de um *browser* em uma máquina remota via *Internet*, permitindo assim um fácil acesso de todos os envolvidos no projeto com a ferramenta.

Para a implementação da ferramenta em um ambiente *Web* Cliente Servidor foi escolhida a linguagem Java por vários motivos, a saber:

1. Vasta documentação sobre desenvolvimento *Web* disponível tanto no site do fabricante da linguagem bem como na Internet de maneira geral;
2. Compiladores e ferramentas de desenvolvimento disponibilizados sem ônus nenhum na Internet. Por exemplo, o ambiente de desenvolvimento Eclipse [Eclipse, 2006];
3. Suporte completo ao paradigma de orientação a objeto;
4. Amplamente utilizada no mercado e na academia por 5 milhões de desenvolvedores se configurando como a maior comunidade de desenvolvedores de software; [SUN, 2006a].
5. O Java *Enterprise Editon* 5 ganhou o prêmio de melhor plataforma de *Web Services* em janeiro de 2006 do "2005 SOA *Web Services Journal Readers' Choice Awards*" [SUN, 2006b].

Como o ambiente COCAR é uma ferramenta cujo crescimento já esta previsto com a adição de outros recursos e funcionalidades, como por exemplo, a geração automática de Casos de Teste prevista na proposta de trabalho de Gregolin (2006), buscou-se utilizar padrões de projetos existentes na literatura que permitam a fácil manutenção do software buscando conferir ao produto final alta manutenibilidade.

Dessa forma o projeto do ambiente COCAR foi realizado com padrões MVC (*Model-View-Controller*), de maneira a separar as camadas de regras de negócio, de apresentação e de controle, promovendo um menor acoplamento do software, permitindo que manutenções de uma determinada parte da ferramenta possam ser feitas sem interferir nas demais.

Para garantir a independência da camada *VIEW* das demais foi utilizado o framework Struts [APACHE 2006] por já estar consagrado no mercado de desenvolvimento de software *Web*, pela facilidade em usá-lo e pela vasta documentação disponível na Internet.

Com o propósito de implementar a camada *MODEL* com baixo acoplamento em relação às demais, foi utilizando o padrão de acesso a banco de dados DAO (*Data Access Object*) e o *Transfer Object* como exemplificado nas Figuras: Figura 11, Figura 12, Figura 13 e Figura 14 respectivamente. [Java, 2006]

A Figura 15 traz o diagrama de classe que implementa a funcionalidade "Cadastrar Sistema" do sistema COCAR. Lembrando que todas as outras funcionalidades foram implementadas seguindo a mesmas estrutura de classes.

Como opção para o Gerenciador de Banco de Dados foi adotado o *MySQL*, que atende às necessidades da ferramenta, além de ser gratuito e amplamente utilizado, contando com ampla documentação. Caso haja necessidade, a migração para outro gerenciador de banco de dados é facilitada devido ao uso do padrão DAO.

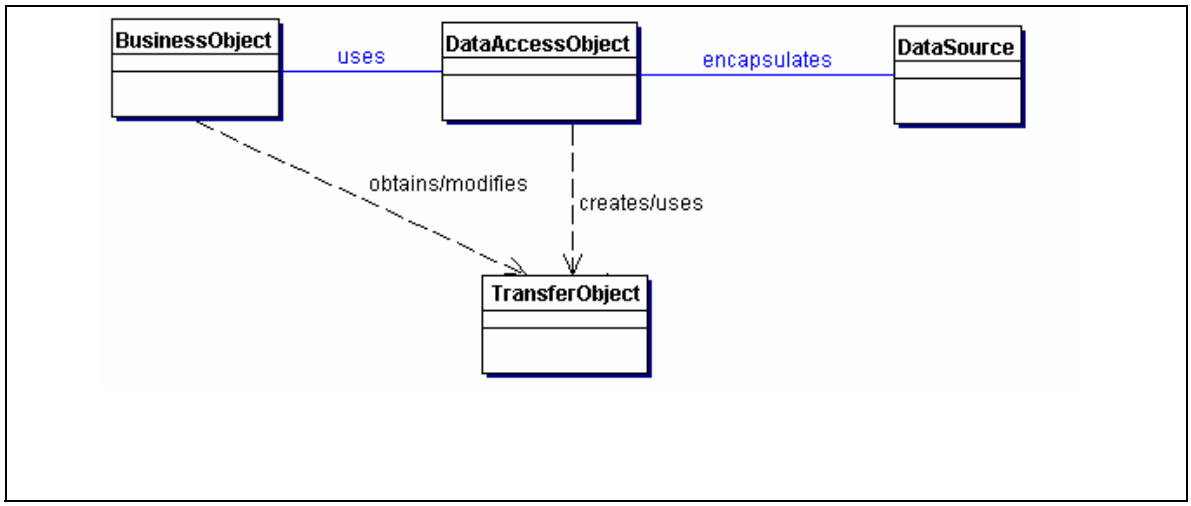

Figura 11 – Diagrama de Classe UML do Padrão DAO[Java, 2006]

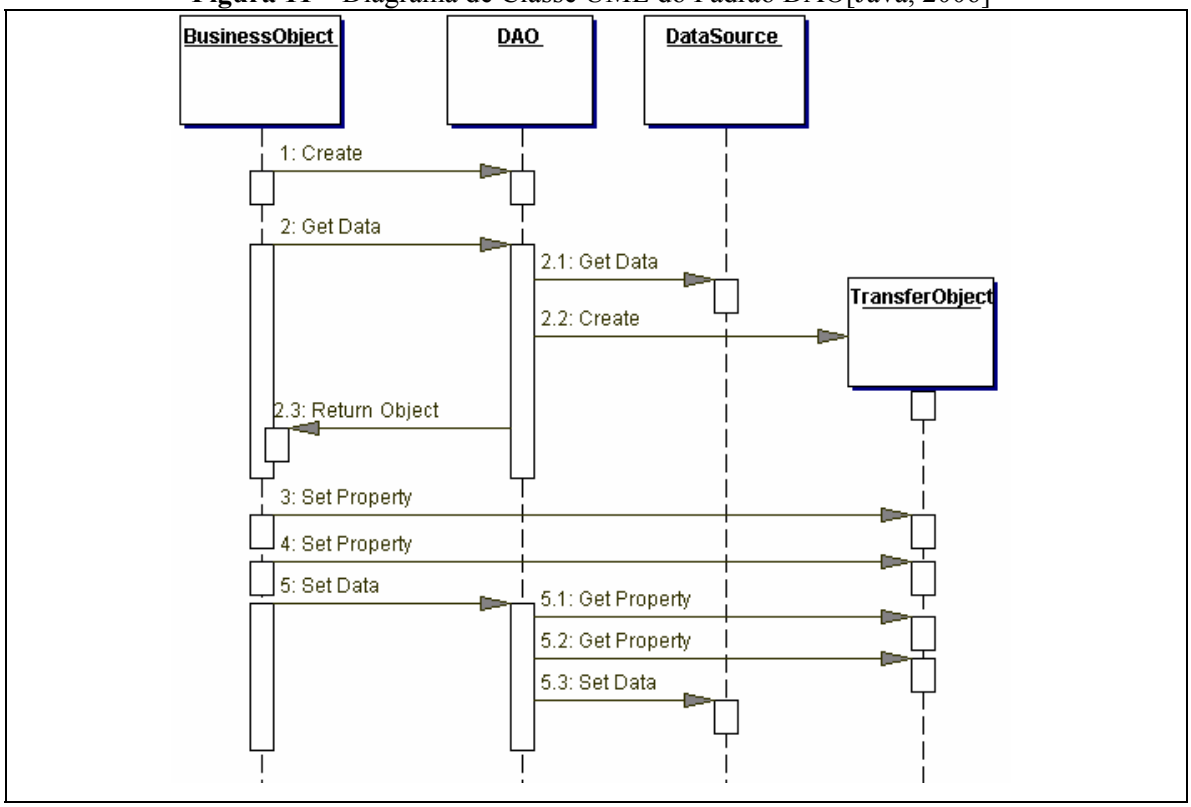

Figura 12 - Diagrama de Classe UML do Padrão DAO[Java, 2006]

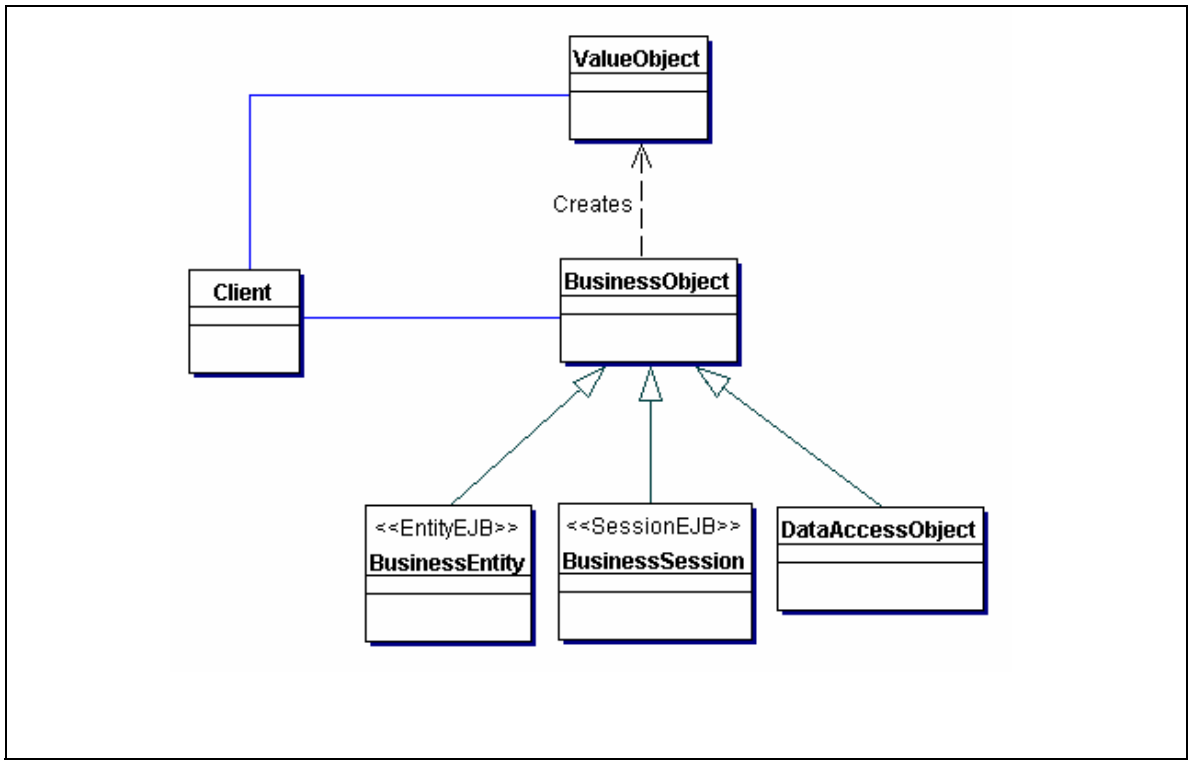

Figura 13 - Diagrama de Classes UML do Padrão Transfer Object [Java, 2006].

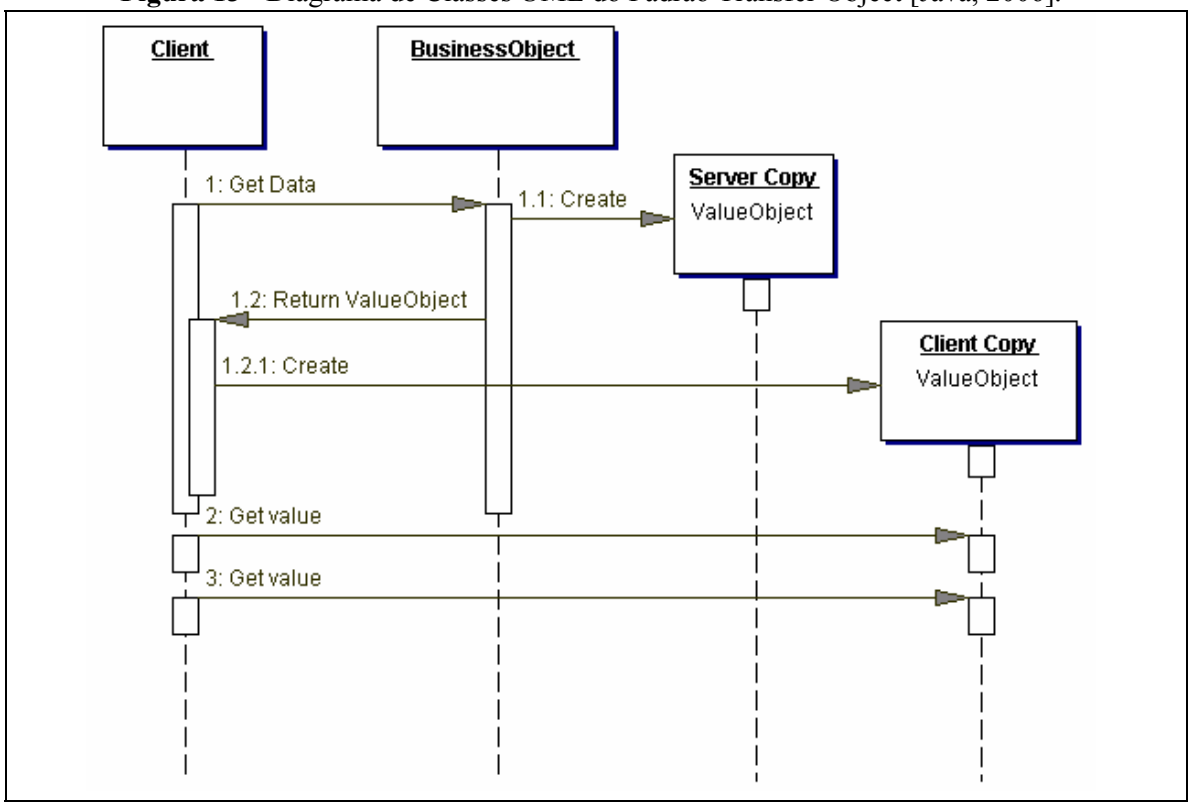

Figura 14 - Diagrama de Classes e Seqüência UML do Padrão Transfer Object [Java, 2006].

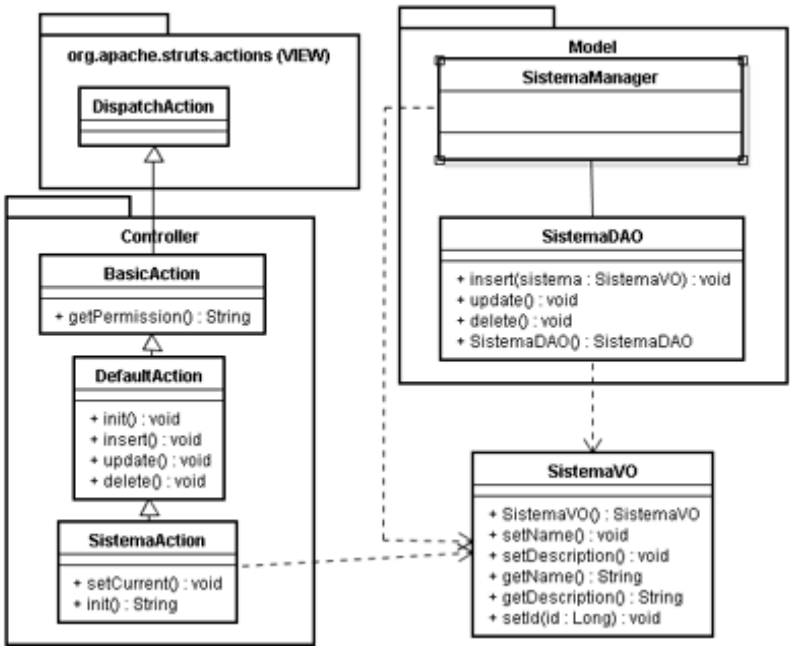

Figura 15 - Diagrama de Classes UML da funcionalidade Cadastrar Sistema.

6.3.3 Implementação

Conforme discutido na seção 6.2.1 e 6.2.2, para a implementação do Ambiente COCAR foi usado a Prototipação Evolucionária, os Padrões de Projeto DAO e *Transfer Object* e a linguagem Java, sendo que o software foi implementado conforme o processo definido em Sommerville (2003) e mostrado na Figura 16 abaixo:

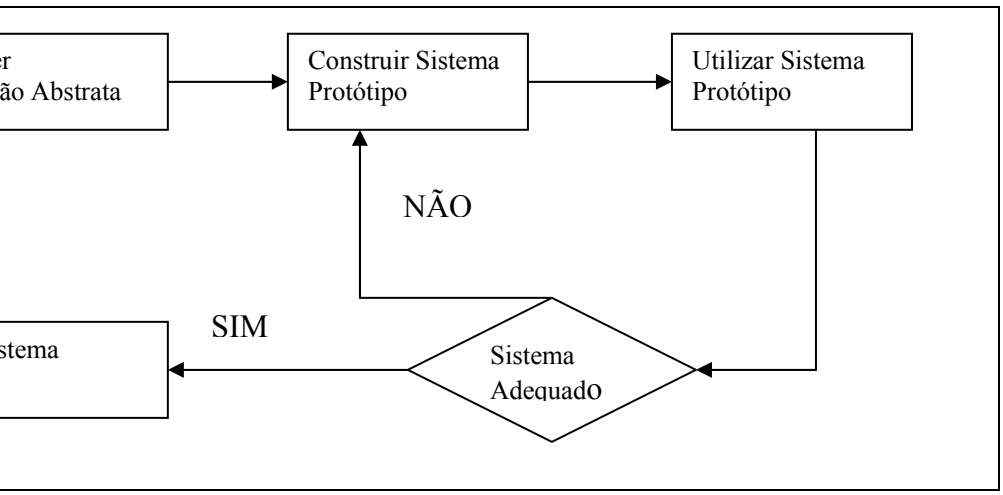

Figura 16 - Prototipação Evolucionária [Sommerville, 2003]

Contudo, houve uma pequena adaptação ao processo de Prototipação Evolucionária definido em Sommerville (2003) no que diz respeito à sua primeira fase, "Desenvolver Especificação Abstrata", que foi substituída pelo uso da Prototipação Descartável. Dessa forma, a Prototipação Descartável gerou um documento de requisitos segundo as diretrizes especificadas por Kawai (2005) que serviu como artefato de entrada para o processo de Prototipação Evolucionária.

6.4 *Requisitos Funcionais do Ambiente COCAR – Pontos de Caso de Uso*

Como visto no Capítulo 4, Anda *et. al,* (2001), Mohagheghi et al (2005), Vinsen et al. (2004), Diev (2006) e Arnold & Pedross (1998) mostram a necessidade de padronizar a especificação dos Modelos de Casos de Uso para a geração mais precisa da medida de Pontos de Caso de Uso.

Neste cenário, no qual há a falta de padronização na especificação do Modelo de Casos de Uso, Kusumoto et al. (2004) enfatizam a necessidade da criação de ferramentas capazes de automatizar o cálculo da métrica de PCU por meio da classificação automática da complexidade dos Casos de Uso e dos Atores do Modelo de Casos de Uso, alertando também sobre a importância de uma forma de padronizar o modelo de Casos de Uso antes de se iniciar a contagem de PCU.

Belgamo & Fabbri (2004c) mostram que o uso da técnica TUCCA é capaz de padronizar a identificação dos Casos de Uso no Modelo de Caso de Uso, além de influenciar na contagem de PCU gerando métricas mais padronizadas e com maior qualidade.

Nesse sentido, o ambiente COCAR foi projetado no escopo da estimativa de Ponto de Caso de Uso com os requisitos funcionais apresentados a seguir:

- Classificação automática da complexidade dos Casos de Uso;
- Cálculo da medida dos Pontos de Caso de Uso;
- Transformação da métrica de PCU em PF.

Na seqüência, cada um desses requisitos será detalhado.

6.4.1 Classificação Automática da Complexidade dos Casos de Uso

Segundo a técnica de PCU proposta por Karner (1993) um dos passos do processo de contagem é a classificação da complexidade de cada Caso de Uso. Essa classificação é baseada no número de transações da especificação que cada Caso de Uso contém. Porém, os Casos de Uso podem ser especificados com diversos graus de detalhamento [Cockburn, 2001; Boock et al., 2000] influenciando diretamente na classificação da complexidade dos Casos de Uso, bem como na contagem dos PCU.

Nesse sentido, o trabalho de Mohagheghi et al (2005) propõe uma regra de simplificação de Casos de Uso simplesmente para o efeito de contagem dos PCU, buscando padronizar o grau de detalhamento da especificação dos Casos de Uso modelados. Já Kusumoto et al. (2004) realizam uma análise morfológica e sintática do fluxo de eventos da especificação dos Casos de Uso visando a padronizar o conceito de transação para só então realizar a classificação dos Casos de Uso para a contagem de PCU.

O ambiente COCAR gera o Modelo de Caso de Uso por meio da aplicação da técnica TUCCA. Para aplicar esta técnica, o ambiente COCAR usa como entrada um documento de requisitos, que fica cadastrado no próprio ambiente, seguindo as diretrizes especificadas por Kawai (2005). Desta forma é possível a geração de Modelos de Casos de Uso mais padronizados [Belgamo & Fabbri, 2004c] e preparados para a aplicação direta da técnica de contagem de PCU.

Assim, o ambiente COCAR é capaz de realizar a classificação automática da complexidade dos Casos de Uso por meio da contagem direta das transações especificadas em cada Caso de Uso, nos termos estabelecidos na técnica de PCU proposta por Karner (1993).

6.4.2 Cálculo da Medida dos Pontos de Caso de Uso

No cálculo da medida de Pontos de Caso de Uso Karner (1993) propõe, resumidamente, os seguintes passos:

1. Classificação da complexidade dos Atores;
2. Classificação da complexidade dos Casos de Uso;
3. Cálculo do Fator de Complexidade Ambiental;
4. Cálculo do Fator de Complexidade Técnica;
5. Cálculo da medida de PCU;

Para a classificação da complexidade dos Atores do Modelo de Casos de Uso, Kusumoto et al. (2004) propõem uma forma automática de proceder essa classificação baseando-se em 3 regras: nome do Ator; palavras chaves que constam nos Casos de Uso relacionadas com os Atores; dados de base histórica. No entanto, não foi possível a implementação direta desta técnica no ambiente COCAR antes de um prévio estudo de sua adaptação para o idioma Português, pois suas as regras baseiam-se quase totalmente no significado semântico das palavras e o estudo foi realizado em sistemas especificados no idioma japonês. Assim, sugere-se para trabalhos futuros o estudo da adaptação da técnica de Kusumoto et al (2004) para o idioma Português.

Portanto, o ambiente COCAR prove suporte para a classificação manual da complexidade dos Atores por meio de uma interface. Nesta interface são mostrados todos os Atores modelados no Modelo de Casos de Uso e as definições dos níveis de complexidade de Atores proposto por Karner (1993), viabilizando ao usuário um melhor julgamento sobre a complexidade do Ator.

A classificação dos Casos de Uso acontece de forma automática conforme discutido no item 6.4.1.

Tanto para o cálculo do Fator de Complexidade Ambiental quanto para o cálculo do Fator de Complexidade Técnica, o ambiente COCAR disponibiliza interfaces que trazem os itens relativos a cada fator para serem classificados pelo usuário em uma escala de 0 a 5, conforme proposto pela técnica de PCU. Após a classificação dos itens, o ambiente COCAR realiza todos os cálculos necessários para aferir o valor do FCA e do FCT.

Tendo todos os Atores e os Casos de Uso classificados de acordo com as suas complexidades e os fatores FCA e FCT calculados, o ambiente COCAR calcula automaticamente o valor da métrica de PCU através da aplicação direta da fórmula descrita na técnica de PCU.

6.4.3 Transformação da métrica de PCU em PF

Não obstante a técnica de PF não ter foco no desenvolvimento de software orientado a objeto e não possuir nenhuma ferramenta que faça o seu cálculo automaticamente [Chen et al, 2004], ela é uma das principais técnicas de estimativa de software existente no mercado [Caldiera et al., 1998] contando com um instituto internacional, o IFPUG, que padroniza o seu processo de contagem e certifica profissionais a realizá-lo [IFPUG, 1999].

Nesse sentido, Andrade (2004) desenvolveu uma equação que representa a relação existente entre a APF e PCU, possibilitando a projeção do valor de Pontos por Função de um determinado projeto de software a partir do valor da métrica de Pontos de Caso de Uso.

Dada a importância já comentada da métrica de Pontos por Função no mercado de desenvolvimento de software, o ambiente COCAR automatizou a equação de Andrade (2004) implementando a funcionalidade de transformação da métrica de PCU em PF, permitindo que estimativas de esforço possam ser calculadas usando informações de bases históricas de Pontos por Função, disponibilizadas, inclusive, por algumas ferramentas de estimativa existentes no mercado.

6.5 *Considerações Finais*

Neste capítulo foram apresentadas as características do ambiente COCAR, cujo início de implementação foi fruto deste trabalho de mestrado. Lembra-se que, embora o objeto principal de pesquisa deste trabalho tenha sido o aspecto do cálculo automático da métrica de Pontos de Caso de Uso, para que essa questão pudesse ser implementada, fez-se necessário dar início à implementação do ambiente como um todo, automatizando-se a aplicação da TUCCA [Belgamo, 2004], e da especificação de requisitos [Kawai, 2005], frutos de outros trabalhos de mestrado.

Apresentaram-se também as principais funcionalidades do ambiente e seus aspectos de implementação, descrevendo o processo pelo qual o ambiente foi construído, as tecnologias utilizadas e as justificativas para as decisões tomadas. Foram ainda exploradas as funcionalidades implementadas no ambiente COCAR referentes ao cálculo automático da

métrica de Pontos de Caso de Uso, justificando-se cada uma delas com o apoio da literatura consultada e das propostas de outros autores, como foi discutido com detalhes no Capítulo 5.

A seguir, no Capítulo 7, será apresentado um estudo de caso, no qual todas as características das funcionalidades aqui descritas poderão ser observadas e entendidas com maior clareza.

Capítulo 7
Exemplo de Uso do Ambiente COCAR

7.1 *Considerações Iniciais*

No capítulo anterior foi apresentado o ambiente COCAR, descrevendo-se suas principais funcionalidades, sobretudo aquelas relacionadas à automatização da técnica de Pontos de Caso de Uso.

Neste capítulo será apresentado um exemplo no qual uma especificação de software retirada da indústria de desenvolvimento de software foi utilizado para mostrar o uso da ferramenta, descrevendo os passos que devem ser seguidos para:

- a criação de um sistema e cadastramento de seus requisitos, seguindo o modelo proposto por Kawai (2005);
- a criação do Modelo de Casos de Uso, por meio da aplicação da técnica TUCCA [Belgamo,2004];
- a automação da técnica de PCU e a geração da medida de PF;

Posteriormente, os resultados obtidos por meio das funcionalidades disponíveis no ambiente COCAR serão comparados com os resultados obtidos na indústria de software, tendo como objetivo avaliar a viabilidade de aplicação do ambiente e os resultados gerados por ele.

O exemplo em questão, utilizado tanto para mostrar as funcionalidades do ambiente COCAR quanto para efeito de comparação dos resultados obtidos na indústria por meio de um estudo de caso, é referente a um sistema denominado UserPrePaid.

O UserPrePaid é um sistema Web desenvolvido pela empresa VoxAge Teleinformática, que atua no seguimento de CTI (*Computer Telephone Integration*), com sede em São Paulo. Trata-se de um sistema simples e com poucas funcionalidades, projetado para dar suporte aos usuários de telefonia pré-paga para a realização da compra de créditos e verificação do histórico de atividades, exigindo para isso a autenticação deste usuário.

Apesar de simples, o sistema UserPrePaid é capaz de ilustrar o funcionamento do ambiente COCAR, abrangendo todas as funcionalidades referidas anteriormente. Ainda, devido à simplicidade do sistema e ao seu uso prático para a VoxAge, este sistema já foi implementado e pode-se realizar um estudo de caso no qual foi possível comparar:

- Modelo de Casos de Uso gerado pela empresa com o Modelo de Casos de Uso gerado pelo ambiente COCAR;
- PCU calculados manualmente pela empresa com o PCU gerado automaticamente pelo ambiente COCAR;
- Esforço real de implementação do software com a estimativa de esforço calculada baseada na métrica de PCU da empresa e na métrica de PCU do ambiente COCAR

Este capítulo está organizado da seguinte forma: na Seção 7.2 é realizada uma breve descrição do ambiente COCAR. Na Seção 7.3 descreve-se a criação de um sistema no ambiente COCAR. O cadastramento dos requisitos relacionados ao sistema criado é abordado na Seção 7.4. A criação do Modelo de Casos de Uso com base na TUCCA é detalhada na Seção 7.5 e os aspectos de automatização dos PCU e transformação para PF, na Seção 7.6. Na seção 7.6 apresenta-se uma avaliação do ambiente COCAR e, por fim, na Seção 7.7 apresentam-se as considerações finais.

Ainda vale lembrar, conforme já salientado no Capítulo 6, que como duas das principais funcionalidades do ambiente COCAR (inserção de requisitos de software baseado nas diretrizes definidas por Kawai (2005), e a geração do Modelo de Casos de Uso baseado na técnica TUCCA de Belgamo (2004)) são partes comum deste trabalho e do mestrado de Di Thommazo (2007), as Seções 7.2, 7.3, 7.4 e 7.5 foram escritas em conjunto com esse outro mestrando.

7.2 *Apresentação do Ambiente COCAR*

O projeto de interfaces do ambiente COCAR foi implementado pensando na melhor forma de interação entre o usuário e o sistema, valendo-se dos benefícios gerados pelos protótipos criados, conforme comentados na Seção 6.3.1. Para auxiliar a navegação, as funcionalidades ficam disponíveis ao usuário no momento em que podem ser aplicadas.

Para aumentar o entendimento do usuário em relação à forma correta de uso do ambiente COCAR, a primeira tela mostrada diz respeito a um texto de ajuda mostrando ao usuário as principais funcionalidades, com *hyperlinks* para detalhes de cada uma das funcionalidades. Essa tela é mostrada na Figura 17

Após a apresentação das instruções para o usuário, a primeira etapa a ser processada para a elaboração do Modelo de Casos de Uso é a escolha de um sistema já existente ou a criação de um novo sistema. Essa interface se fez necessária, pois os menus referentes à técnica TUCCA e à geração da métrica de PCU só ficam disponíveis quando se tem um sistema cadastrado e ativo no ambiente COCAR, como é mostrado na Figura 18 e Figura 19.

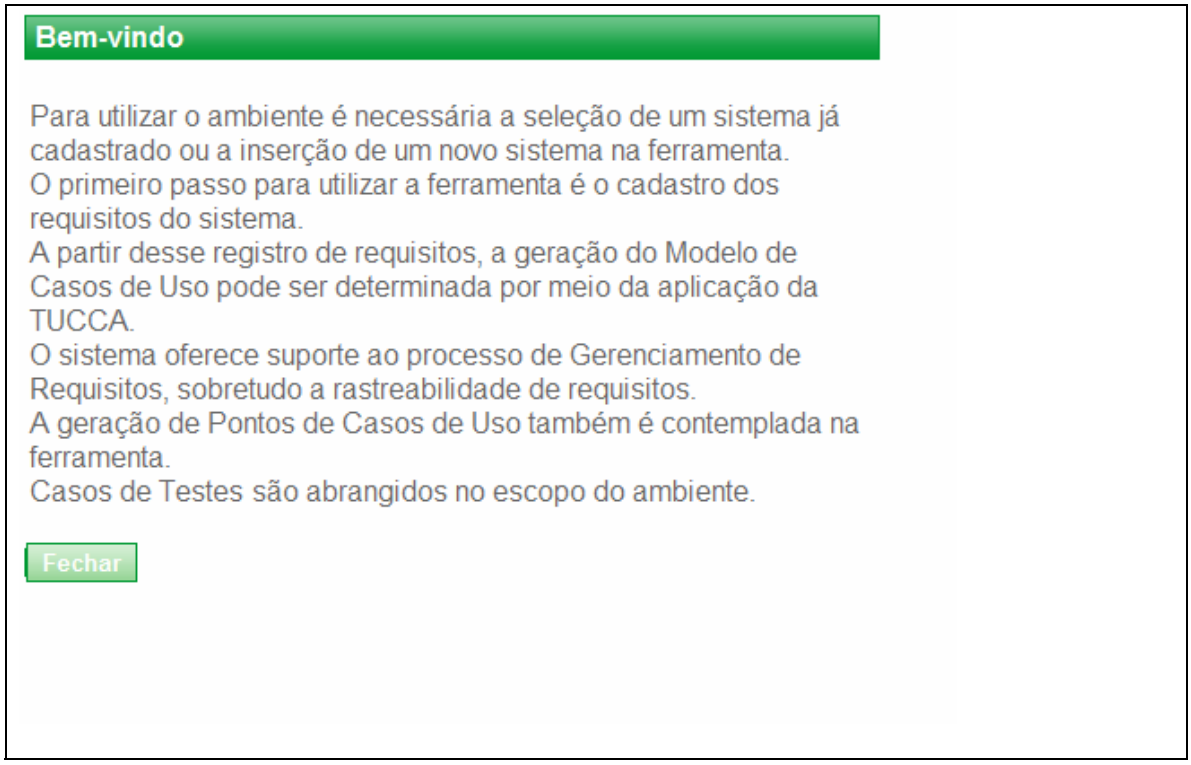

Figura 17 – Interface de *help* do sistema COCAR

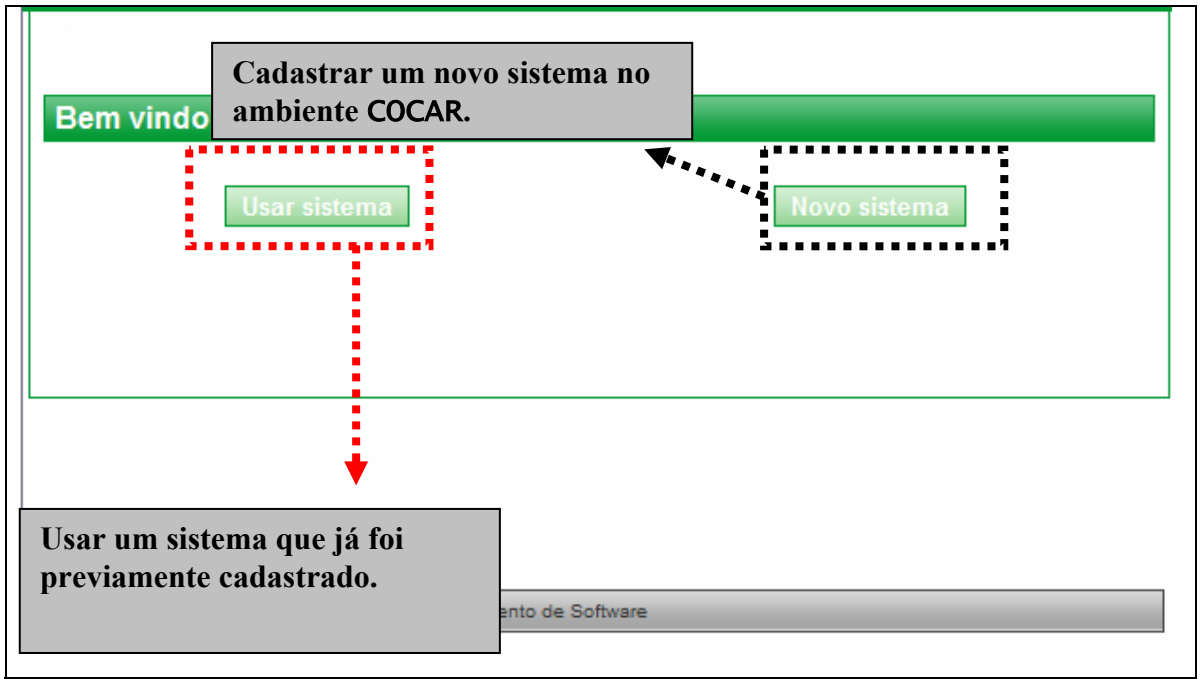

Figura 18 – Escolha de um sistema ou a criação de um novo sistema

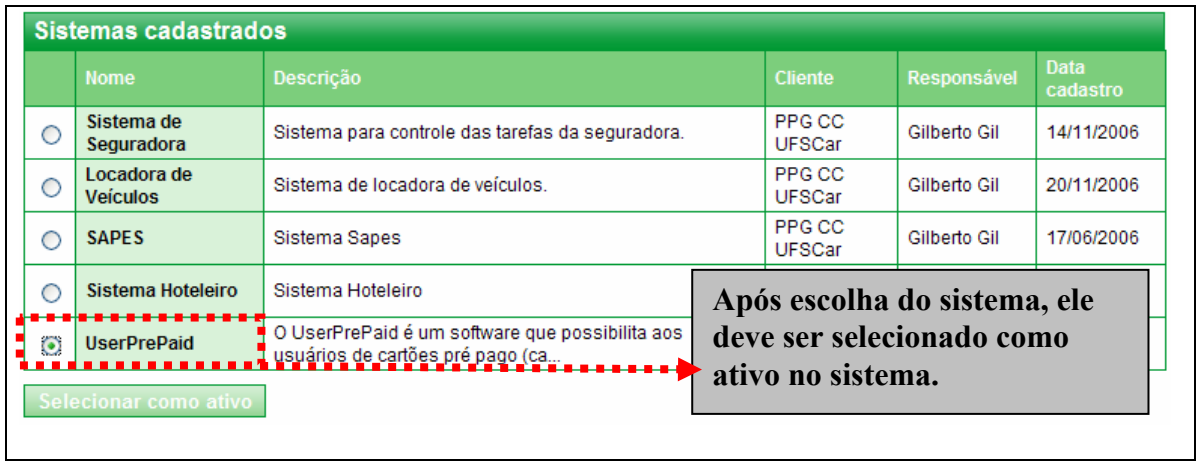

Figura 19 – Selecionar um sistema já cadastrado como ativo

A criação do sistema e os passos subseqüentes para a geração do Modelo de Casos de Uso são mostrados nas próximas seções.

7.3 *Criação de Sistema*

O ambiente COCAR organiza os dados por meio de Sistemas. Neles são inseridos os requisitos do software. Quando um Sistema é criado, deve-se fornecer seu nome, descrição, o cliente que o solicitou, o gerente responsável e as funções do produto. A Figura 20 ilustra o cadastramento de um sistema.

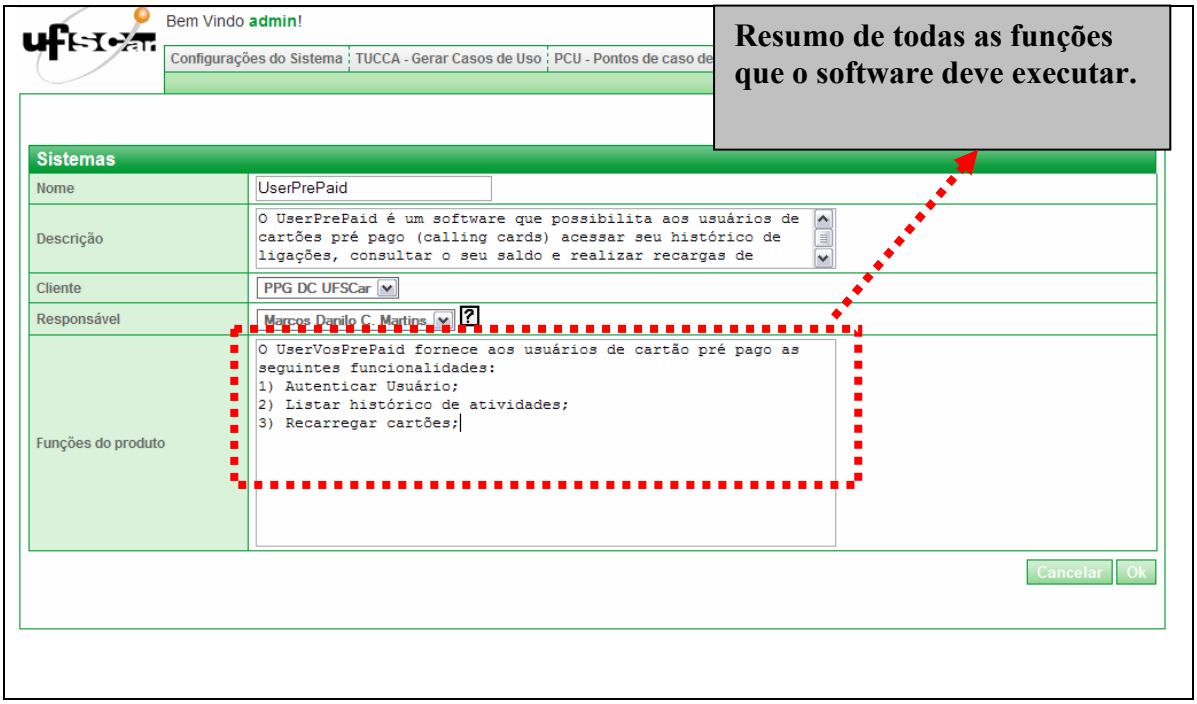

Figura 20 – Cadastramento de um sistema no ambiente COCAR

7.4 *Cadastramento de Requisitos*

Conforme discutido no Capítulo 6, o repositório de requisitos do ambiente COCAR segue o modelo proposto por Kawai (2005), detalhado no Capítulo 5. Além dos atributos propostos nesse modelo, foram ainda inseridos dois atributos baseados na proposta de Wiegers (1999b), sendo eles: o solicitante e o gerente do requisito. Apesar de boa parte desses atributos já terem sido apresentados e discutidos no trabalho de Kawai (2005), eles serão detalhados na seqüência, enfocando-se suas particularidades no ambiente COCAR.

- Identificação do Requisito: a identificação dos requisitos é feita internamente, mantendo-se a consistência no banco de dados. Esse aspecto de implementação é transparente para o usuário.
- Descrição: explicação breve do requisito. O objetivo deste item é registrar "o quê" a funcionalidade deve fazer e não o "como". Uma vez cadastrados diversos requisitos no sistema, uma listagem deste item "descrição" permite uma boa idéia de tudo o que o sistema deve realizar.
- Processamento: composto pelo fluxo de atividades ou ações que o sistema deve realizar para alcançar o que está registrado no item "Descrição". O campo do processamento permite inserção de textos grandes.

- Entradas: nesse campo são indicados todos os dados que estão envolvidos na funcionalidade. É composto por uma lista de dados que são cadastráveis no ambiente, indicando-se o nome, a descrição e o tipo (inteiro, flat, texto, data ou lógico) do dado de entrada.
- Restrição: nesse campo são registradas restrições ou condições necessárias para que o requisito seja realizado. O nome original desse campo era Condição/Restrição e no contexto do ambiente foi renomeado simplesmente para Restrição.
- Saída: a resposta do sistema ao requisito deve ser indicada nesse campo.
- Ator Principal: sua nomenclatura original era Agente Fornecedor/Receptor. No contexto do ambiente foi renomeado para Ator Principal. É representado por uma lista de atores que podem ser cadastrados no sistema, indicando-se o nome e a descrição. Mais de um agente pode ser selecionado para um requisito. Representa o ator que fornece ou utiliza os dados no sistema.
- Ator Executor: originalmente era chamado Agente Executor. Representado por uma lista de atores que podem ser cadastrados no sistema, indicando-se o nome e a descrição. Mais de um agente pode ser selecionado para um requisito. Representa o ator que utiliza o sistema, inserindo os dados fornecidos pelo Ator Principal.
- Solicitante do Requisito: esse atributo não foi baseado nas diretrizes apresentadas por Kawai (2005). Ele é fruto da sugestão de Wiegers (1999b) e foi inserido no ambiente COCAR com o propósito de dar suporte à pré-rastreabilidade, conforme definição de Zisman e Spanoudakis (2004) e Letelier (2002). Nesse caso, o usuário do sistema deve escolher os solicitantes do requisito dentre os disponíveis em uma lista. Os solicitantes são cadastrados no ambiente COCAR, fornecendo-se nome e descrição.
- Gerente do Requisito: esse atributo também não foi baseado nas diretrizes apresentadas por Kawai (2005), sendo proposta de Wiggers (1999b). Esse atributo também tem foco na rastreabilidade. Nesse caso, o usuário do sistema deve escolher somente um gerente dentre os disponíveis em uma lista. Os gerentes são cadastrados fornecendo-se nome e descrição.

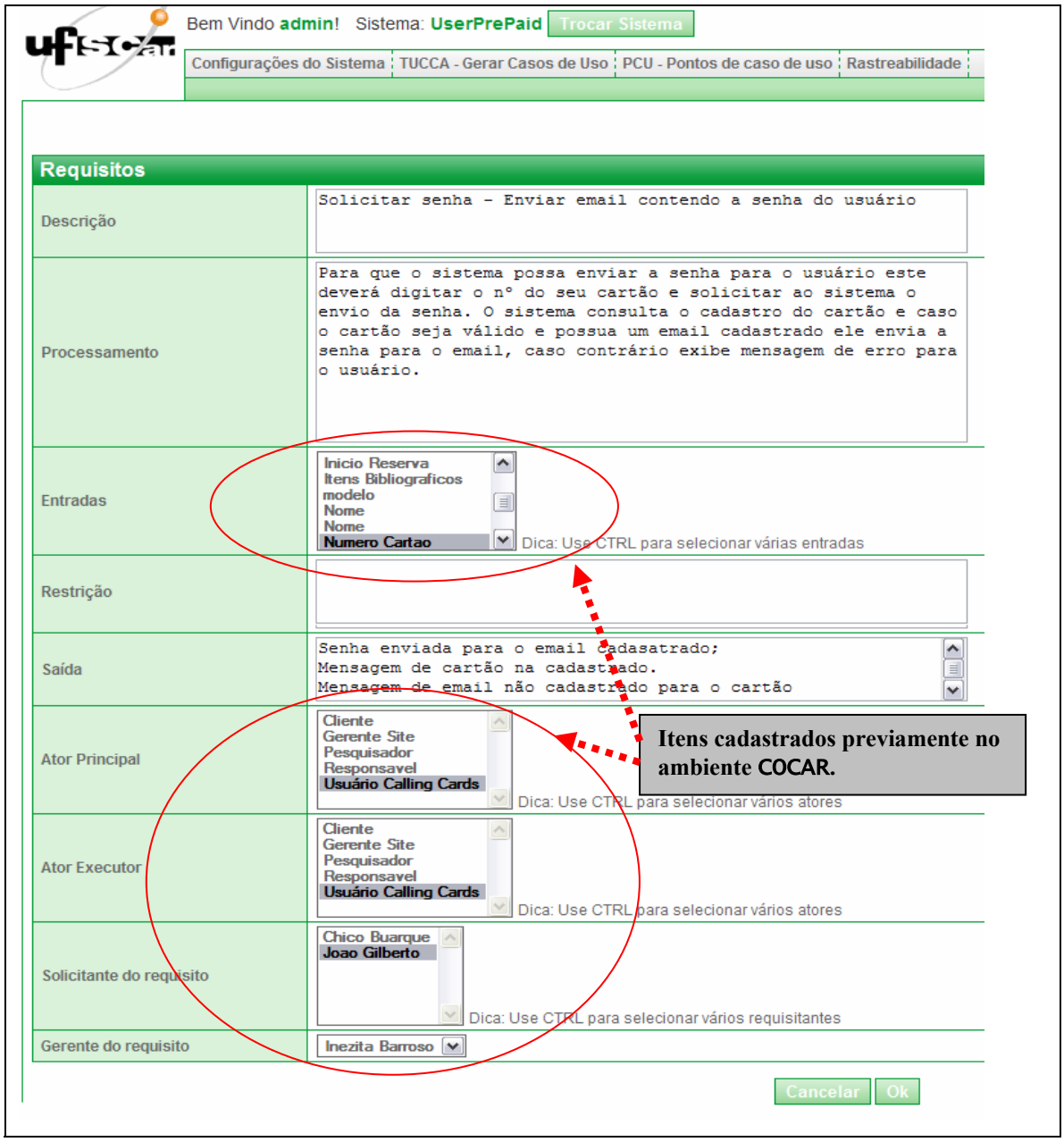

Figura 21 – Cadastramento de Requisitos no ambiente COCAR.

A Figura 21 mostra a tela do ambiente COCAR, na opção de cadastro de requisito. Percebe-se que nela, conforme especificado por Kawai (2005), os campos Entrada, Ator Principal, Ator Executor, Solicitante do Requisito e Gerente do Requisito são representados por um componente *list box* (componente HTML que permite a seleção de vários itens previamente cadastrados) e composto por uma lista de dados que são cadastráveis no ambiente. O cadastro desses dados deve ser feito antes da inserção dos requisitos por meio das telas representadas pela Figura 22, Figura 23, Figura 24 e Figura 25.

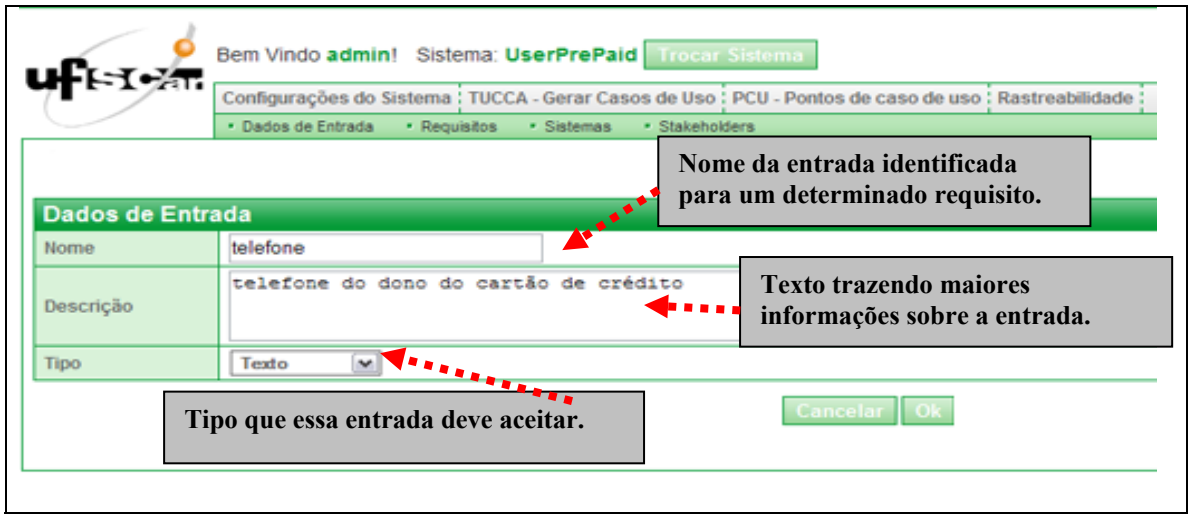

Figura 22 – Cadastramento de Entradas no ambiente COCAR

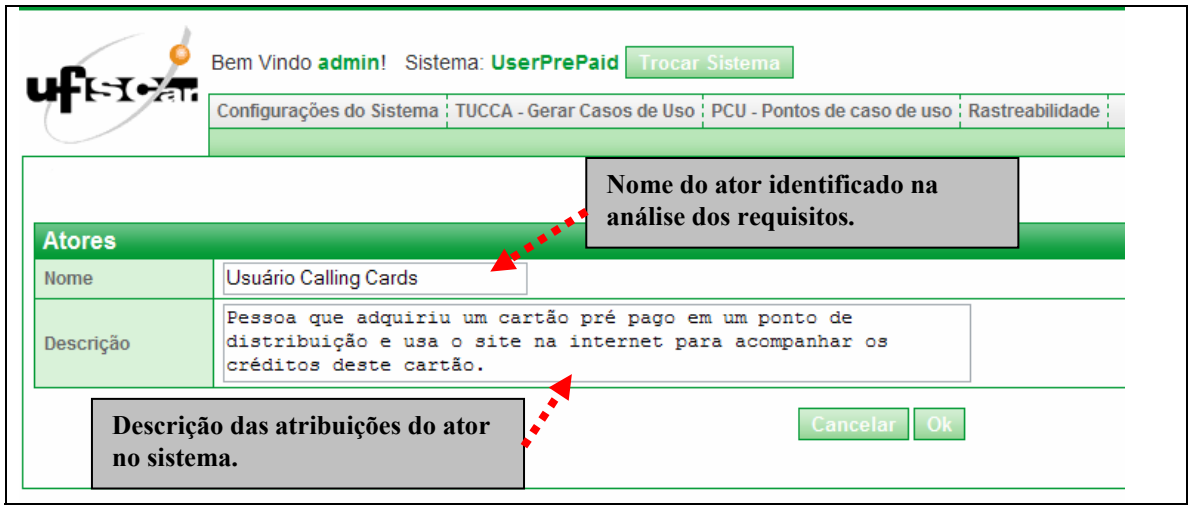

Figura 23 – Cadastramento de Atores (primário e secundário)

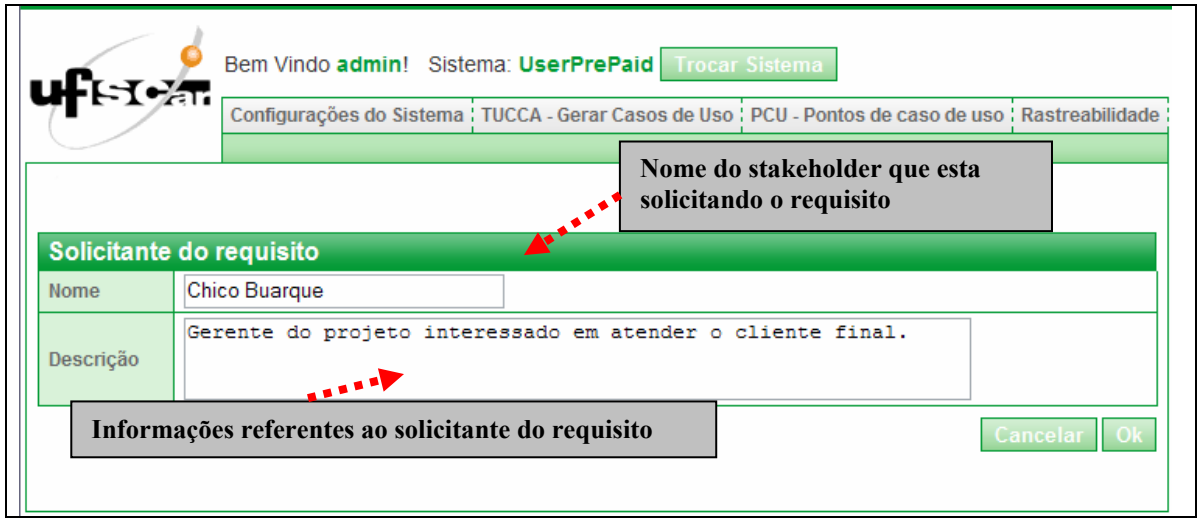

Figura 24 – Cadastramento de Solicitante de Requisitos

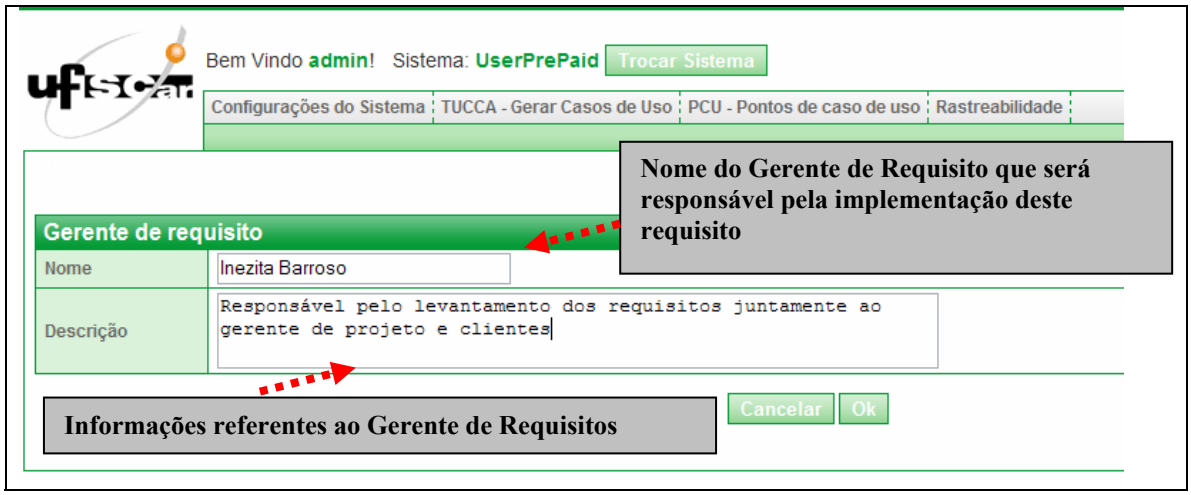

Figura 25 – Cadastramento de Gerente de Requisito

7.5 *Criação do Modelo de Casos de Uso*

No ambiente COCAR o Modelo de Casos de Uso é gerado seguindo os passos que compõem a TUCCA, tendo como ponto de partida o documento de requisitos definido de acordo com as diretrizes estabelecidas por Kawai (2005), implementadas da forma apresentada na seção anterior. Como já foi apresentada no Capítulo 5, a TUCCA é composta de duas técnicas de leituras: a *Actor-Goal Reading Technique* (AGRT), que recebe como entrada o documento de requisitos e tem o objetivo de identificar os possíveis atores e os objetivos de uso do sistema; e a *Use Case Reading Technique* (UCRT), responsável pela identificação dos Casos de Uso e da elaboração de suas especificações.

As interfaces foram implementadas por meio de uma abordagem *wizard*, ou seja, o ambiente COCAR oferece um passo a passo para a execução das atividades que fazem parte do processo de geração do Modelo de Casos de Uso, respeitando a precedência entre os passos a serem seguidos. Dessa forma, procura-se facilitar a aplicação da TUCCA pelo usuário, à medida que a seqüência dos passos conduz naturalmente a criação do Modelo de Casos de Uso.

7.5.1 - Actor-Goal Reading Technique

Após a inserção dos requisitos do sistema para o qual se deseja definir o modelo de casos de uso, o usuário pode dar início à aplicação da AGRT. Ela é composta de quatro passos principais, nem todas executadas seqüencialmente:

i) Marcação dos atores, funções e restrições.
ii) Determinação dos atores e objetivos.
iii) Resolução de problemas de redundância.
iv) Criação do Formulário Ator X Objetivo (FAO).

Em seguida, será abordado cada um desses passos, mostrando as telas do ambiente COCAR para uma melhor visualização da solução.

i) Marcação dos atores, funções e restrições

Nesse passo, são exibidos para o usuário cada um dos requisitos funcionais cadastrados, além das funções do produto, que foram inseridos no ambiente previamente, conforme descrito nos itens 7.2 e 7.3. O usuário deve ler cada um dos itens apresentados e marcar os candidatos a ator (amarelo), as funções (objetivos) (rosa) e as restrições (azul) existentes no texto. Esse processo é apresentado na Figura 26.

ii) Determinação dos atores e objetivos

A partir da marcação, toma-se como referência cada uma das funções levantadas e solicita-se ao usuário a determinação do objetivo e de qual dos atores está vinculado com esse objetivo. Esse processo deve ser realizado para cada uma das funções que foram marcadas no passo anterior. Salienta-se que, com a execução desse passo, o passo de criação do FAO também está sendo realizado, simultaneamente. A Figura 27 ilustra esse processo.

iii) Resolução de problemas de redundância

Uma vez determinados os atores e objetivos do sistema, é necessário retirar as possíveis redundâncias que possam existir, tanto ao que diz respeito aos objetivos identificados como também aos atores que interagem com o sistema. Deve-se atentar para dois tipos de redundância.

A primeira, chamada redundância intra-atores, refere-se ao caso de dois objetivos, de um mesmo ator, serem exatamente os mesmos, apesar de terem denominação diferente no documento de requisitos, ou seja, apesar de sintaticamente diferentes, semanticamente eles são iguais. Como exemplo, tem-se “Cadastrar Cliente” e “Salvar Cliente” que, em uma

determinada situação, para um determinado documento de requisitos, podem significar a mesma funcionalidade e, portanto, devem ser fundidos em um único objetivo.

O segundo tipo de redundância é chamada de redundância inter-atores. Nesse caso, temos objetivos sintaticamente diferentes relacionados a atores distintos mas, contendo o mesmo teor semântico. Como exemplo, podemos ter o objetivo "Cadastrar Pedido" sendo realizado pelo ator "Cliente" e o objetivo "Armazenar Pedido" sendo realizado pelo ator "Usuário", os dois objetivos, apesar de estarem escritos de forma diferente, significam a mesma coisa, então devem ser agrupados em apenas um objetivo.

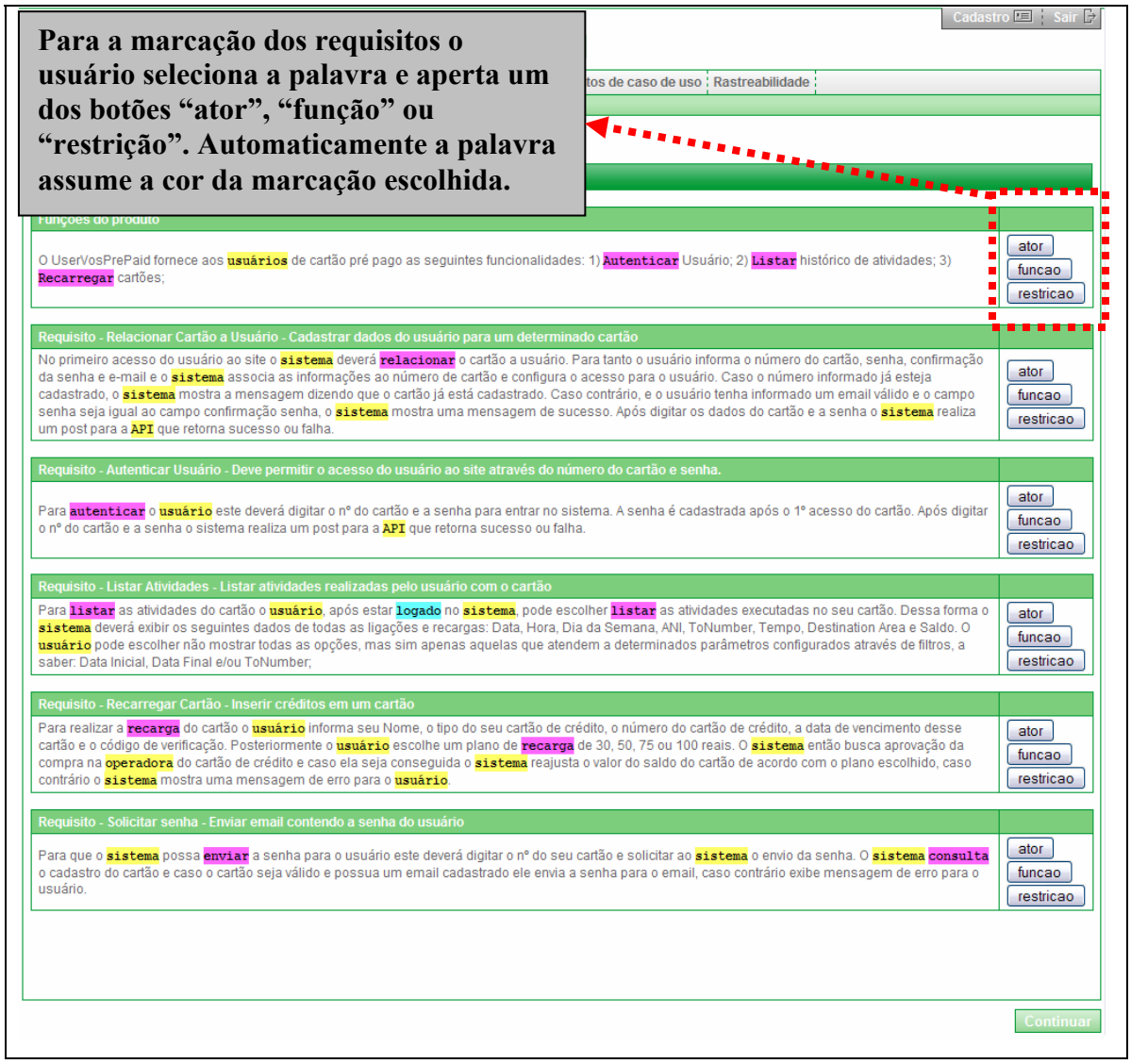

Figura 26 - Processo de marcação de atores, funções e restrições.

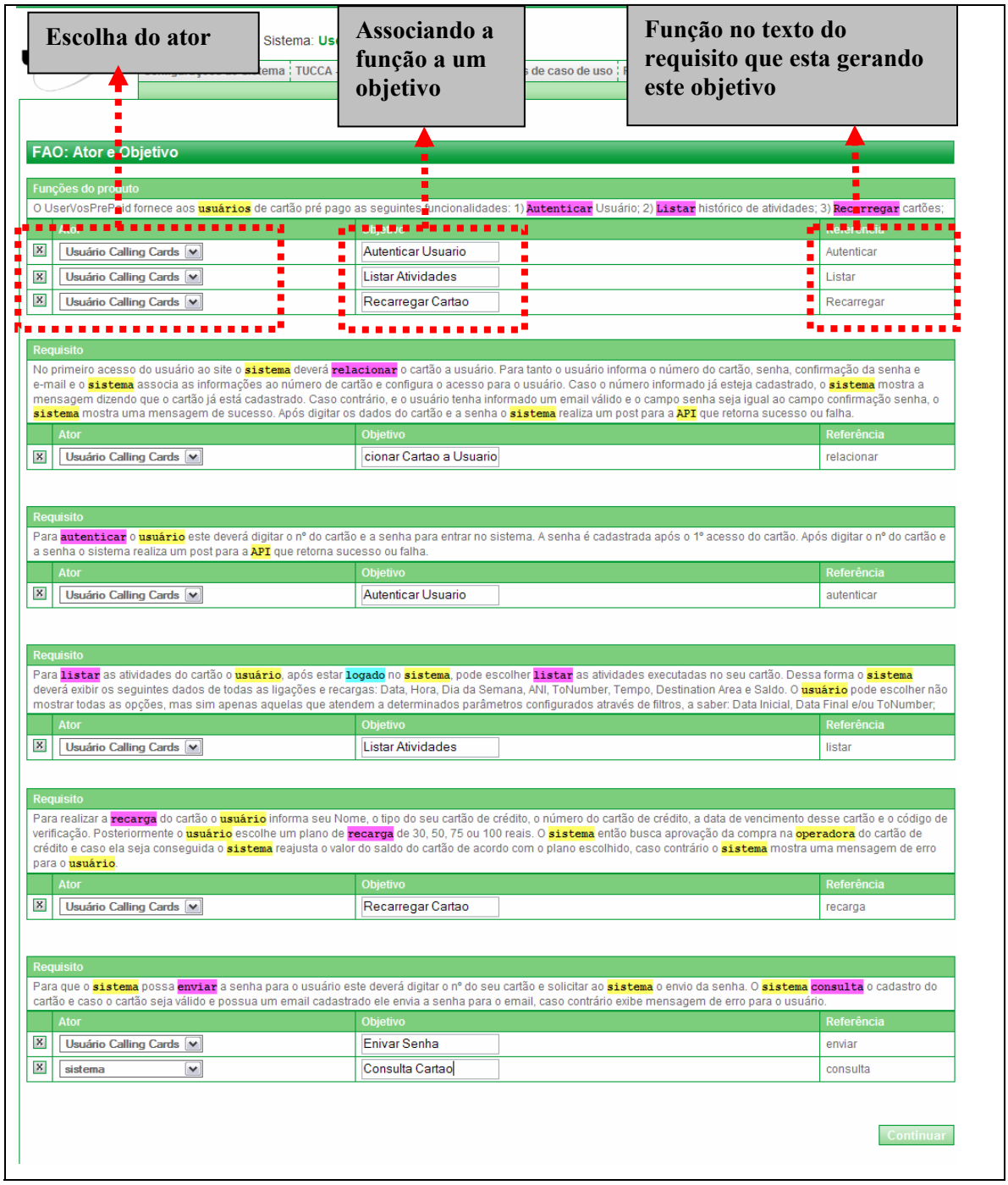

Figura 27 – Determinação dos atores e objetivos.

Na Figura 28, é mostrado a interface disponibilizada pelo ambiente COCAR para resolver as redundâncias intra-atores e inter-atores. No exemplo em questão não há nenhuma redundância desse tipo a ser resolvida.

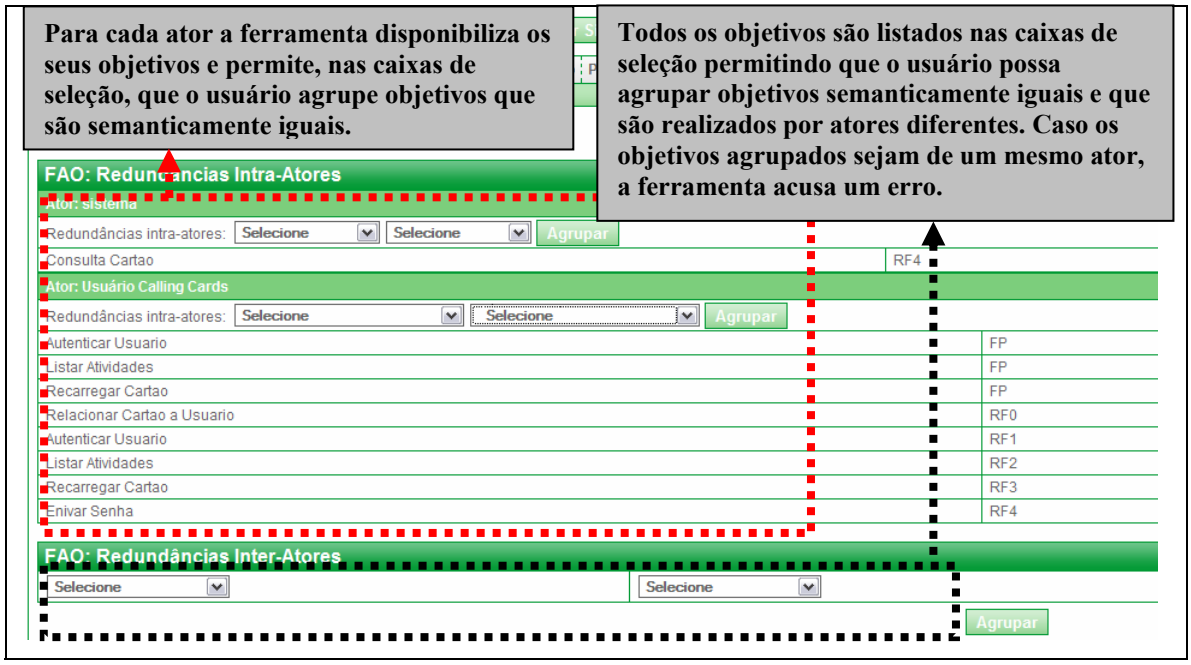

Figura 28 - Exemplo da interface de resolução de redundância intra-atores e inter-atores.

iv) Criação do Formulário Ator X Objetivo (FAO)

Depois de executados os passos anteriores, tem-se o FAO concluído, uma vez que ele vai sendo elaborado à medida que os outros passos são realizados. Esse formulário apresenta os atores e todos os seus objetivos no sistema. Um exemplo do FAO é mostrado na Figura 29.

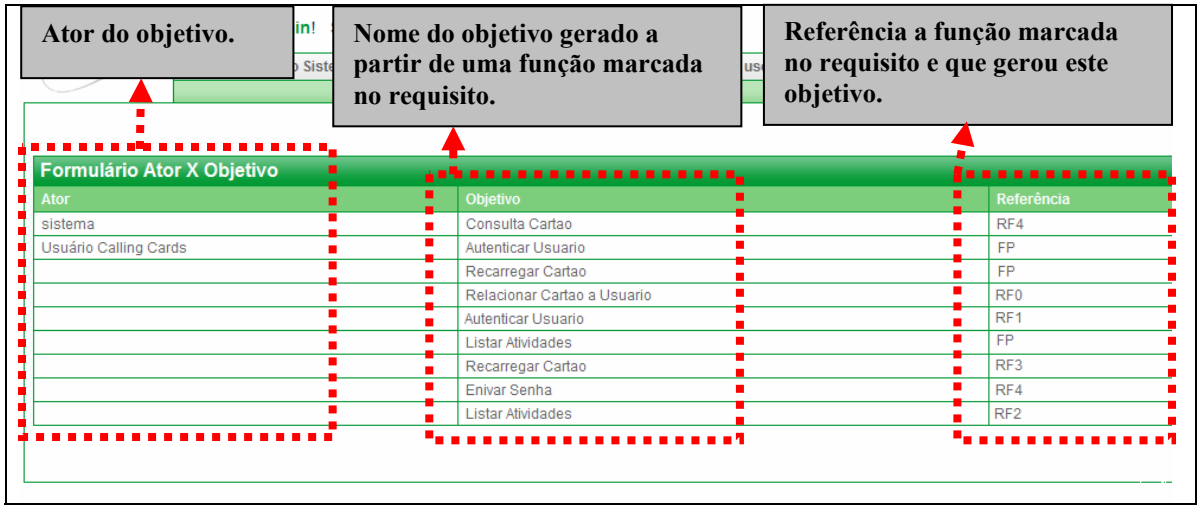

Figura 29 - Exemplo do FAO gerado pela aplicação da AGRT no ambiente COCAR.

7.5.2 - Use Case Reading Technique (UCRT)

A UCRT utiliza como entrada o FAO gerado pela AGRT e os requisitos do sistema. Como saída são gerados o Diagrama de Casos de Uso e a Especificação de Casos de Uso.

A UCRT é composta por duas etapas: Criação dos Casos de Uso Preliminares e Criação das Especificações e Possíveis Relacionamentos entre os Casos de Uso. A seguir é mostrado como cada uma dessas etapas foi implementada no ambiente COCAR.

i) Criação de Casos de Uso Preliminares (Etapa I)

O intuito principal desta etapa é identificar os Casos de Uso preliminares usando para isso os objetivos existentes no FAO. Além disso, também se busca nessa fase a identificação dos possíveis relacionamentos e associações existentes entre esses casos de uso preliminares.

Para tanto, o primeiro passo dessa etapa consiste em tornar cada objetivo do FAO, que tem como referência a seção "Funções do Produto", em um caso de uso preliminar, pois como essa seção do documento de requisitos apresenta as principais funcionalidades do sistema, elas são fortes candidatas a se tornarem casos de uso. Após isso, devem-se percorrer os objetivos do FAO em busca daqueles que tiverem o mesmo nome dos objetivos já transformados em Casos de Uso Preliminares, com o intuito de agrupar as referências associadas a eles, uma vez que esses objetivos foram detalhados em outros requisitos do documento de requisitos, além de serem citados na seção "Funções do Produto". Com isso, facilita-se a especificação do caso de uso, uma vez que todos os pontos em que esse objetivo foi citado estão agora agrupados. O ambiente COCAR faz essa operação automaticamente como pode ser percebido na Figura 30.

No segundo passo dessa etapa analisa-se a possibilidade de agrupamento ou não de todos os objetivos que possuem o mesmo conjunto de referências (daqueles ainda não transformados em Casos de Uso Preliminares). No exemplo da Figura 30 o requisito 4 tem a mesma referência "RF2" e os dois requisitos aparecem no campo "Fundir" dessa tela. No exemplo em questão, "consultar cartão" e "enviar senha" são objetivos que, apesar de terem origem no mesmo requisito, são semanticamente diferentes um do outro e, por isso, não serão fundidos.

O terceiro passo dessa etapa corresponde a transformar em casos de uso independentes todos os objetivos restantes no FAO. O ambiente COCAR realiza este passo automaticamente.

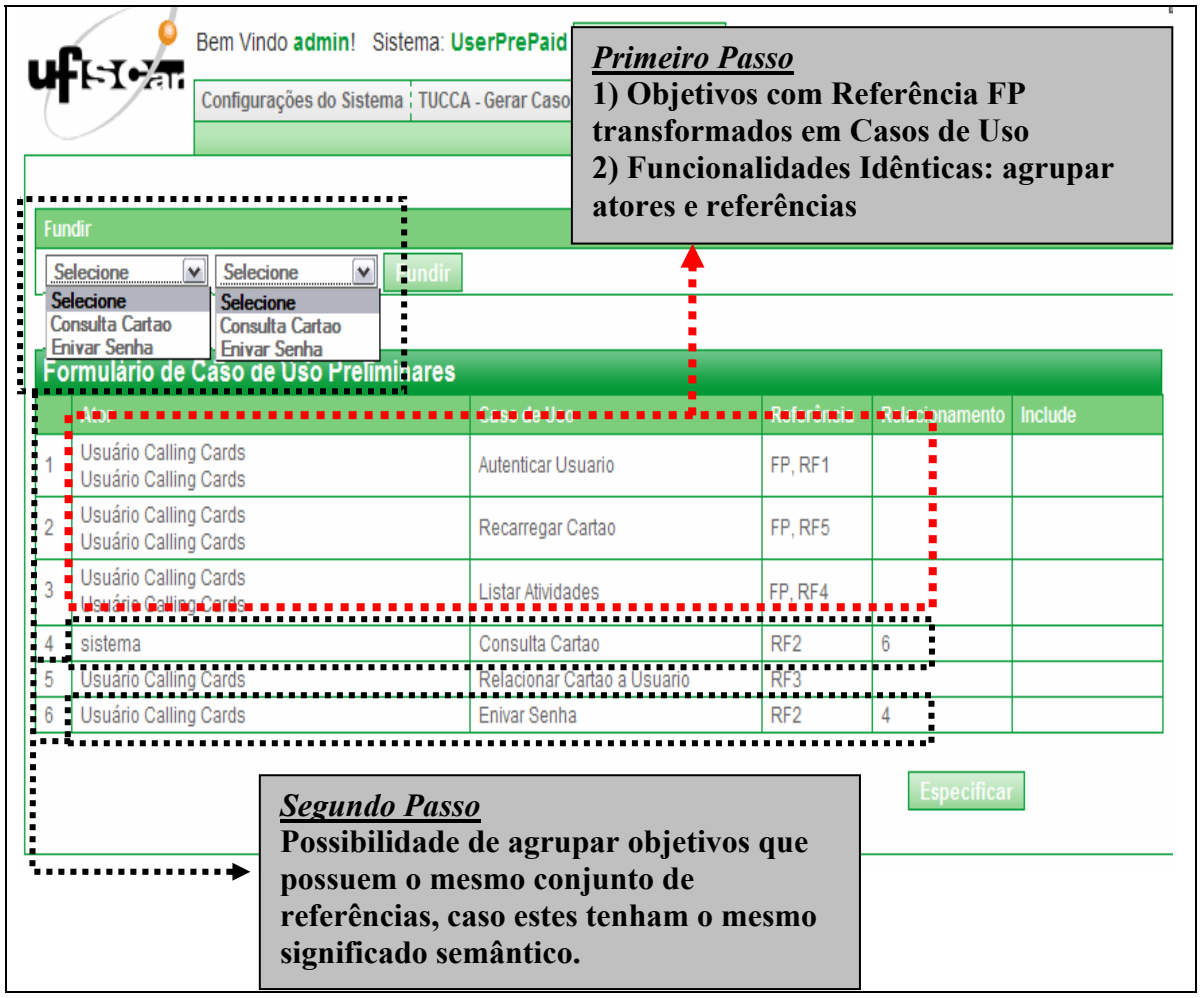

Figura 30 – Primeira Etapa: Formulário de Casos de Uso Preliminares

ii) Criação das Especificações e possíveis relacionamentos entre os Casos de Uso (Etapa II)

Nessa etapa, o objetivo principal é especificar os Casos de Uso definidos no Formulário de Casos de Uso Preliminares gerado na etapa anterior.

Para tanto, o ambiente COCAR oferece uma interface que disponibiliza os Casos de Uso Preliminares para a especificação, sendo que os Casos de Uso que contêm apenas referência para a seção de "Funções do Produto" são disponibilizados primeiro, seguidos dos Casos de Uso que tiverem menos referências, conforme o definido na técnica UCRT [Belgamo, 2004]. O objetivo dessa ordem é especificar primeiro os casos de uso que têm probabilidade de serem mais simples do que aqueles que estão associados com vários requisitos do documento de requisitos. Esses, que estão associados com vários requisitos, além de, normalmente, serem mais complexos, ainda têm a probabilidade de envolver casos de uso que já tenham

sido especificados e que podem ter uma associação do tipo <<include>> ou <<extend>> com o caso de uso mais complexo.

Para facilitar a elaboração da especificação do caso de uso, o ambiente COCAR disponibiliza em sua interface a facilidade de mostrar os requisitos que estão relacionados com o caso de uso que esta sendo especificado, possibilitando ao usuário ler e aproveitar o texto do requisito para poder compor os cursos normal e alternativo do Caso de Uso.

Além disso, é possível incluir cursos alternativos por meio do botão "Incluir" da linha de curso alternativo, ou ainda, definir associações entre casos de uso com estereotipo <<includes>> ou <<extends>> por meio da lista de Casos de Uso relacionados.

A Figura 31 mostra a interface oferecida pelo ambiente COCAR para especificação dos Casos de Uso, de acordo com os detalhes discutidos.

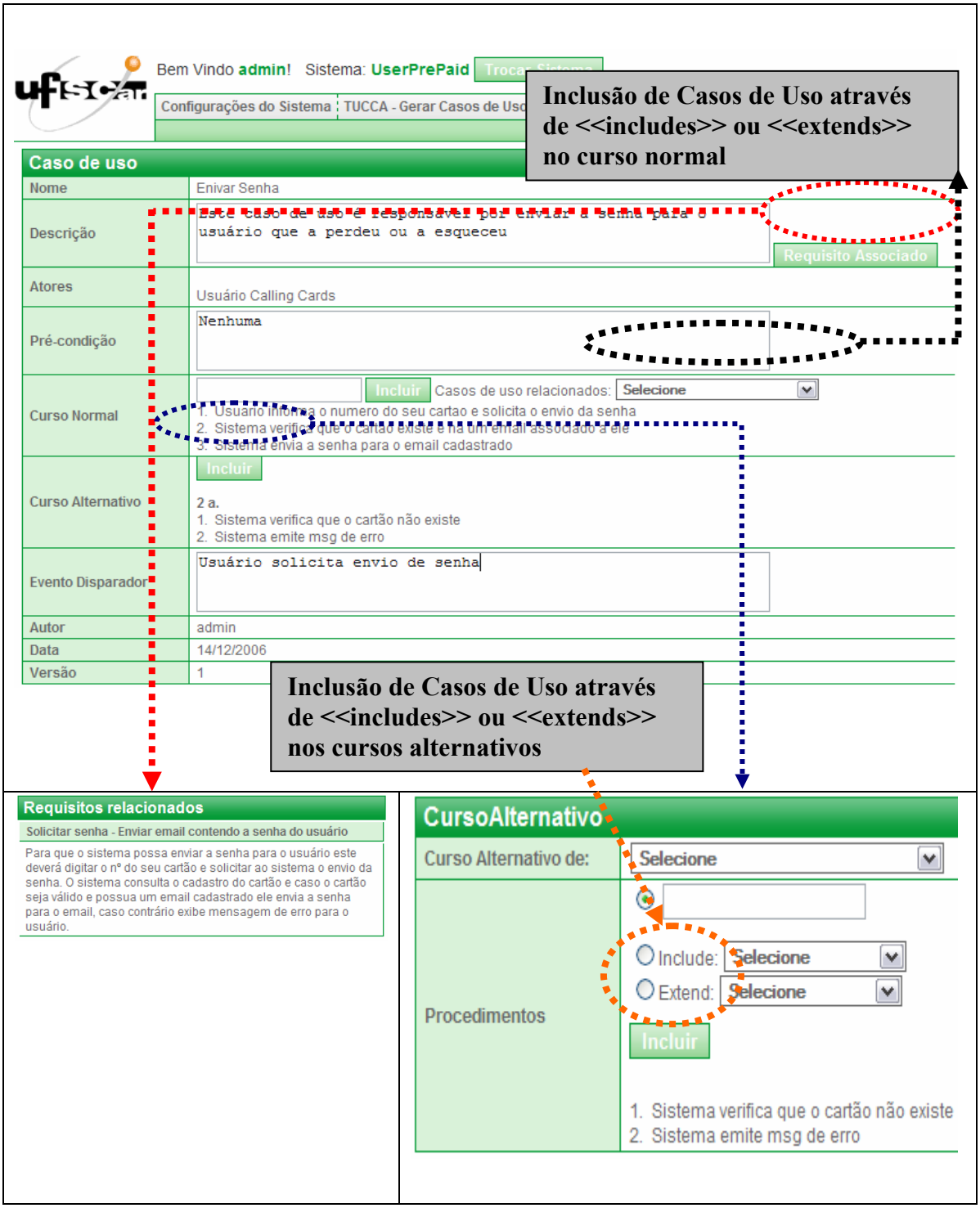

Figura 31 – Interface para a especificação dos Casos de Uso e Janela com a janela dos requisitos relacionados e janela de especificação de curso alternativo

7.6 *Automatização da Técnica PCU e Geração da métrica de PF*

Para contemplar a automatização da métrica de PCU o ambiente COCAR realiza as seguintes etapas:

- Classificação da complexidade dos Atores;
- Classificação automática da complexidade dos Casos de Uso gerando a medida de PCU não ajustado;
- Julgamento dos fatores ambientais e de complexidade técnica;
- Geração da medida de PCU ajustada e PF.

Conforme discutido anteriormente, o ambiente COCAR não prove a classificação automática da complexidade dos Atores, ao invés disso, oferece suporte para que o usuário possa fazê-lo de maneira amigável como mostra a Figura 32.

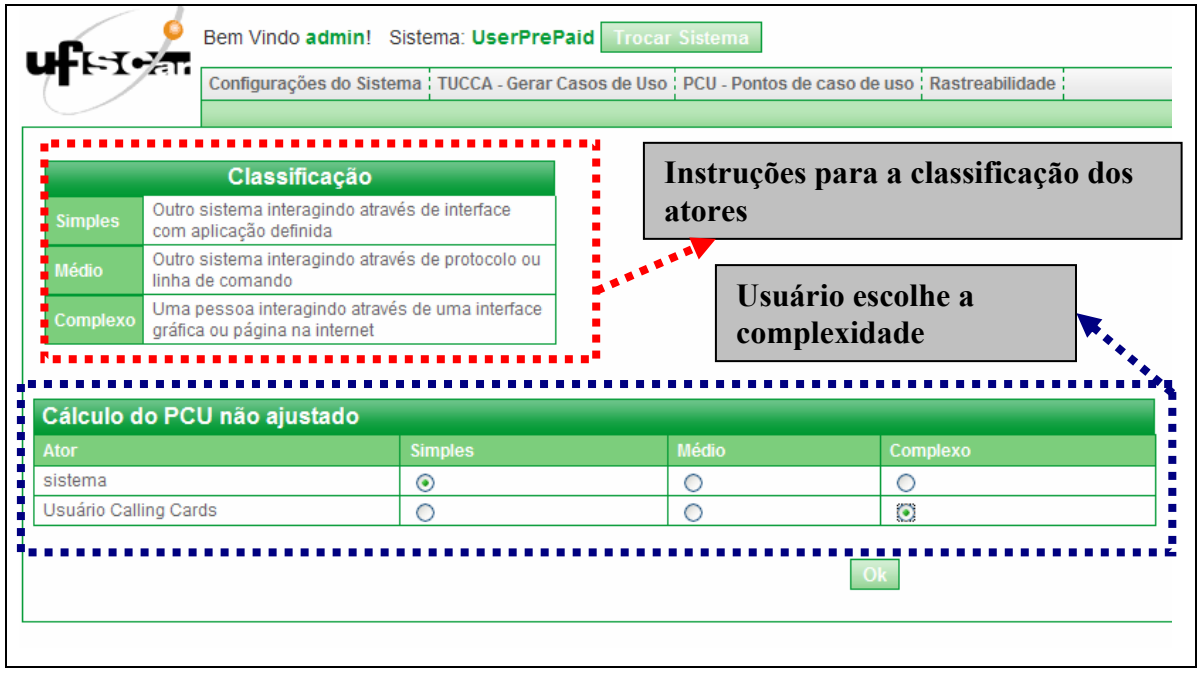

Figura 32 – Classificação de complexidade dos atores

Após proceder a classificação de complexidade dos atores, o ambiente COCAR realiza a classificação automática dos Casos de Uso baseando-se no Modelo de Caso de Uso gerado pela TUCCA e nas regras estabelecidas por Karner (1993), sendo que após a classificação automática o ambiente COCAR já realiza os cálculos necessários para a geração da métrica de

PCU não ajustados disponibilizando para o usuário o valor dessa métrica, conforme mostrado na Figura 33.

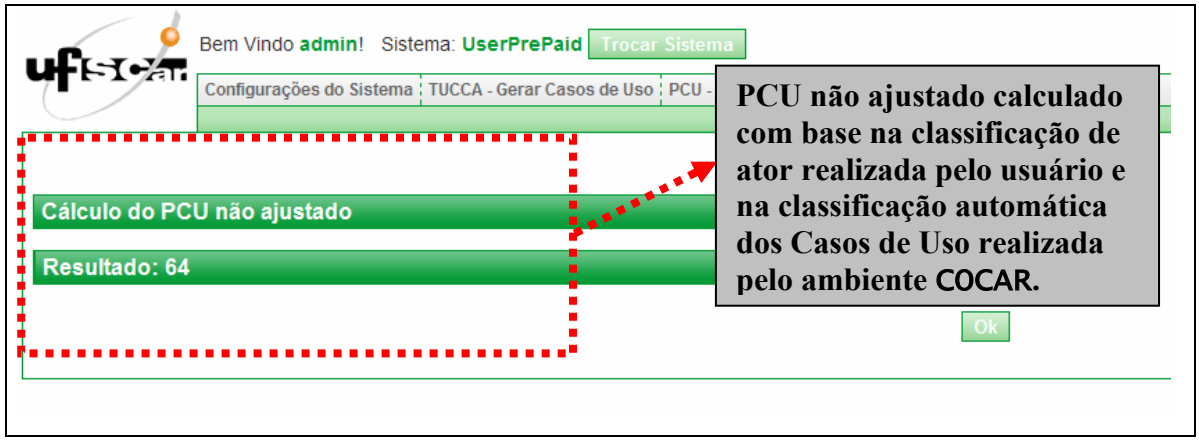

Figura 33 – Resultado do Cálculo do PCU não ajustado

Para concluir o processo de cálculo da métrica PCU ainda é necessário calcular o fator ambiental e o fator de complexidade técnica. Para realizar esse cálculo o ambiente COCAR disponibiliza uma interface que traz todos os itens que devem ser classificados pelo usuário. Essa interface é mostrada na Figura 34. Com a classificação desses itens, o ambiente COCAR processa os cálculos necessários para a geração do fator de complexidade técnica, do fator de complexidade ambiental e do valor da métrica de Pontos de Caso de Uso Ajustados e o valor da métrica Pontos por Função, utilizando a equação estabelecida em Andrade (2004). Todas essas informações podem ser observadas no relatório da Figura 35.

Cálculo do PCU ajustado

Fatores que contribuem para a complexidade do sistema	Irrelevante	Muito pouco relevante	Pouco relevante	Relevante	Muito relevante	Essencial
Distribuição do sistema	○	◉	○	○	○	○
Resposta aos objetivos de desempenho	○	◉	○	○	○	○
Eficiência do usuário final	○	○	○	○	○	◉
Complexidade do processamento interno	◉	○	○	○	○	○
Código deve ser reutilizado	○	○	○	○	○	◉
Facilidade de instalação	○	○	○	○	○	◉
Facilidade de uso	○	○	○	○	○	◉
Portabilidade	○	○	○	○	○	◉
Fácil de alterar	○	○	○	○	○	◉
Concorrência	○	○	◉	○	○	○
Feature de segurança	○	○	◉	○	○	○
Acesso direto a dispositivos de parceiros	○	○	○	○	○	◉
Treinamento especial aos usuários	◉	○	○	○	○	○
Fatores que contribuem para a eficiência	**Irrelevante**	**Muito pouco relevante**	**Pouco relevante**	**Relevante**	**Muito relevante**	**Essencial**
Familiaridade com o Processo de Interativo Unificado	◉	○	○	○	○	○
Experiência na Aplicação	◉	○	○	○	○	○
Experiência com orientação a objeto	○	◉	○	○	○	○
Capacidade de Liderança de Análise	○	◉	○	○	○	○
Motivação	○	○	○	○	○	◉
Estabilidade de Requisitos	○	○	○	○	○	◉
Consultores Part-Time	◉	○	○	○	○	○
Dificuldade de Programação na Linguagem	○	○	◉	○	○	○

Figura 34 – Itens de complexidade ambiental e complexidade técnica julgados pelo usuário

PCU - Dados consolidados

Ator	Complexidade
sistema	1
Usuário Calling Cards	3
TOTAL	**4**

Caso de Uso	Complexidade
Enviar Senha	10
Relacionar Cartão a Usuario	15
Consulta Cartao	5
Autenticar Usuário	10
Listar Atividades	10
Recarregar Cartao	10
TOTAL	**60**

Métrica	Valor
Pontos de caso de uso não ajustado	**64**
Fator de Complexidade Técnica	1.02
Fator Ambiental	0.74
Pontos de caso de uso ajustado	**48.31**
Pontos por função	**139.64**

Figura 35 – Relatório de Pontos de Caso de Uso Ajustado

7.7 *Avaliação do Ambiente COCAR*

O ambiente COCAR foi avaliado por meio de um estudo de caso que levou em consideração as funcionalidades implementadas neste trabalho.

Este estudo de caso teve por objetivo comparar o Modelo de Casos de Uso e a métrica de PCU gerados pelos profissionais de uma empresa da indústria de desenvolvimento de software com o Modelo de Casos de Uso e a métrica de PCU gerados pelo ambiente COCAR. Além disso, tomando por base a métrica de PCU gerada pela empresa e essa mesma métrica gerada pelo ambiente COCAR, foi estimado o esforço de desenvolvimento com base nesses dois valores o qual foi comparado com o esforço real de desenvolvimento do software.

A empresa em questão é a VoxAge Teleinformática LTDA. a qual atua no mercado de software para CTI (*Computer Telephone Integration*) contando hoje com 54 colaboradores entre estagiários e funcionários. Localizada em São Paulo, capital, a empresa desenvolve vários produtos de software para *Call Center* e interação por voz.

O projeto escolhido dentro da VoxAge para este Estudo de Caso diz respeito a um site para usuários de cartão de telefonia pré-paga.

7.7.1 Planejamento do Estudo de Caso

O exemplo selecionado para o Estudo de Caso foi um projeto vendido pela VoxAge para um cliente. Desta forma, os profissionais que trabalharam no desenvolvimento desse projeto o trataram como um projeto normal dentro da empresa, submetendo-o aos processos usuais de gerenciamento e desenvolvimento de software.

Os pontos de comparação de interesse nesse Estudo de Caso podem ser sintetizados nas seguintes questões:

Q1) Há diferença de tempo entre a geração da métrica PCU utilizando a abordagem Ad-Hoc e utilizando a abordagem do ambiente COCAR (deve-se levar em consideração a construção do Modelo de Casos de Uso)?

Q2) Há diferenças no número de Casos de Uso entre o Modelo de Caso de Uso gerado pela abordagem Ad-Hoc e o gerado pela abordagem do ambiente COCAR?

Q3) Há diferenças entre o valor de PCU encontrado pelo ambiente COCAR e pela abordagem Ad-Hoc?

Q4) Qual estimativa de esforço ficou mais próximo do real, aquela baseada no PCU gerado pelo ambiente COCAR ou aquela baseada no PCU gerado de maneira Ad-Hoc?

Para responder as questões acima definidas, este estudo de caso foi projetado da seguinte forma:

1) um projeto simples e real da empresa VoxAge foi escolhido e os requisitos desse projeto foram especificados em um documento de requisitos baseado nas diretrizes sugeridas por Kawai (2005) e nas necessidades da empresa;
2) um colaborador da empresa gerou o Modelo de Casos de Uso e calculou a métrica PCU deste projeto baseando-se no documento de requisitos elaborado em 1;
3) um pesquisador acadêmico submeteu o documento de requisitos ao ambiente COCAR gerando-se o Modelo de Caso de Uso e a métrica de PCU automaticamente;
4) As estimativas de esforço foram calculadas para as métricas de PCU conseguidas em 2 e 3;
5) O projeto foi implementado e o esforço real de implementação foi observado;

7.7.2 –Execução do Estudo de Caso

Os seguintes itens caracterizam a execução deste Estudo de Caso.

1) A equipe de colaboradores que desenvolveu o site já estava acostumada a desenvolver software dessa natureza e era composta por 3 desenvolvedores e 1 gerente de projetos, sendo que apenas o gerente de projetos sabia que o projeto em questão seria utilizado para este Estudo de Caso. Desta forma evitou-se qualquer desvio de comportamento da equipe que poderia ocorrer em virtude do conhecimento de que o projeto em questão seria usado também para fins acadêmicos.

2) O gerente de projetos foi o responsável por gerenciar o registro do tempo de desenvolvimento gasto no projeto.

3) A aplicação deste Estudo de Caso se deu no mês de outubro e novembro de 2006, quando o documento de requisitos do software foi entregue para um dos desenvolvedores da empresa que elaborou o Modelo de Casos de Uso usando para isso uma ferramenta CASE.

4) Em seguida, o gerente de projetos calculou da métrica de PCU e distribui as tarefas para a equipe de desenvolvimento por meio de um software de controle de atividades que a VoxAge possui, no qual todas as horas de desenvolvimento são registradas.

5) Ao final da implementação, o documento de requisitos utilizado foi disponibilizado para este trabalho, e só então ele foi submetido ao ambiente COCAR por meio do qual foi gerado o Modelo de Casos de Uso e calculada a métrica PCU automaticamente.

6) Foram calculadas duas estimativas de esforço, a primeira baseada na métrica de PCU gerada pelo desenvolvedor da empresa VoxAge utilizando uma abordagem Ad-Hoc, e a segunda, baseada na métrica de PCU, gerada por este trabalho, utilizando o ambiente COCAR.

7) Por fim, o gerente de projetos fez o levantamento do total de horas utilizadas no desenvolvimento do projeto, disponibilizando o resultado para este trabalho. Só então o cálculo da estimativa de esforço foi realizado usando para isso o índice de esforço definido por Karner (1993), ou seja, multiplicando-se o valor da métrica pelo esforço de 20 horas/homens.

7.7.3 –Coleta e Análise de Dados

Após a execução dos passos descritos anteriormente os dados coletados tinham por objetivo responder às questões estabelecidas. Esses dados estão apresentados na

Tabela 16.

Tabela 16– Dados coletados do Estudo de Caso

	Empresa	COCAR		
Tempo de Projeto do Modelo de Caso de Uso e do cálculo da métrica de PCU	5 horas	11 horas		
Número de Casos de Uso	6 casos de uso	6 casos de uso		
Número de Casos de Uso Semelhantes	5	5		
Numero de Casos de Uso Estendidos ou Incluídos	1	0		
Métrica de PCU	45,594 pcu	48,634 pcu		Real
Estimativa de Esforço	911,880 horas	972,672 horas		265,5 horas

i) Há uma diferença de tempo entre a geração da métrica de PCU utilizando a abordagem Ad-Hoc e utilizando o ambiente COCAR (deve-se levar em consideração a construção do Modelo de Casos de Uso)?

Pelo que pode ser verificado nos dados da

Tabela 16, há uma diferença de tempos entre a geração do Modelo de Casos de Uso e o cálculo da métrica de PCU gerados por meio da abordagem Ad-Hoc e por meio do ambiente COCAR. Em seguida seguem alguns fatores que podem ter contribuído para esta diferença de tempo:

1) há a necessidade de se cadastrar todo o documento de requisitos na ferramenta, sendo que o tempo gasto nesta atividade foi de aproximadamente 4 horas;
2) há um processo sistemático a ser seguido na automação da técnica TUCCA.

ii) Há diferenças no número de Casos de Uso entre o Modelo de Caso de Uso gerado pela abordagem Ad-Hoc e o gerado utilizando o ambiente COCAR?

Segundo a

Tabela 16, não há diferença no número de Casos de Uso. Porém, a abordagem Ad-Hoc gerou um Modelo de Casos de Uso com 6 Casos de Uso dos quais um deles era estendido, enquanto

o ambiente COCAR gerou um Modelo de Casos de Uso com 6 Casos de Uso sendo que nenhum deles era estendido.

Além disso, a abordagem ad-hoc gerou 5 Casos de Uso semelhantes à abordagem do ambiente COCAR e apenas um Caso de Uso não semelhante.

iii) Há diferenças entre o valor de PCU encontrado pelo ambiente COCAR e pela abordagem Ad-Hoc?

A diferença encontrada entre a abordagem Ad-Hoc e a abordagem do ambiente COCAR foi muito pequena consistindo em cerca de 3 PCU de diferença.

iv) Qual estimativa de esforço ficou mais próximo do real, aquela baseada no PCU gerado pelo ambiente COCAR ou aquela baseada no PCU gerado de maneira Ad-Hoc?

Como a diferença na métrica de PCU foi muito pequena, as estimativas ficaram muito próximas uma da outra, sendo que a estimativa realizada pela métrica gerada utilizando a abordagem pelo ambiente COCAR apresentou um valor 9% superior ao da abordagem Ad-Hoc. Porém, as duas métricas ficaram muito distantes do esforço real de desenvolvimento do software registrado pela empresa, mostrando que o índice de estimativa de esforço de Karner (1993) usando 20 homens/horas por PCU não retratou o verdadeiro esforço de desenvolvimento para o contexto deste Estudo de Caso em particular.

Por meio deste Estudo de Caso, é possível perceber que para o contexto em questão a aplicação do ambiente COCAR se assemelha muito às práticas de especialistas de mercado, sugerindo até a sua aplicabilidade em empresas desenvolvedoras de software para dar suporte à atividade de medida de tamanho de software.

O índice de esforço de 20 homens/horas por PCU não foi adequado para o contexto deste Estudo de Caso. Porém, outros autores como Scheneider and Winters (2001) e Mohagheghi et al. (2005) já haviam identificado alguns problemas com esse índice e em seus trabalhos sugerem outras formas de calcular a estimativa de esforço. De qualquer forma, ressalta-se que vários estudos devem ser conduzidos para que se consiga identificar o motivo e reduzir a diferença do valor estimado para o valor real, verificando também uma maneira para que o próprio ambiente possa dar suporte em relação a isso.

7.8 *Considerações Finais*

Nesse capítulo foi mostrado o uso do ambiente COCAR, apresentando-se o passo a passo do uso da ferramenta, por meio de um exemplo de documento de requisitos retirado da indústria de desenvolvimento de software, ressaltando como:

1) cadastrar os requisitos na ferramenta utilizando as diretrizes definidas por Kawai (2005);
2) gerar o Modelo de Casos de Uso utilizando a automatização da técnica TUCCA de Belgamo (2004);
3) gerar a medida de PCU e como transformá-la em PF utilizando a automatização da equação sugerida por Andrade (2004);

Por fim, foi descrito um pequeno Estudo de Caso, realizado em parceria com a indústria de desenvolvimento de software, no qual foi mostrado que os resultados gerados pelo ambiente COCAR são muito semelhantes àqueles gerados pela empresa participante.

Capítulo 8
Conclusão

Este trabalho apresentou uma ferramenta criada para dar suporte ao processo de desenvolvimento de software: o ambiente COCAR, que em sua primeira versão, implementou os seguintes requisitos funcionais:

- o cadastramento dos requisitos de um sistema, segundo diretrizes definidas no trabalho de Kawai (2005);
- a geração automática do Modelo de Casos de Uso, a partir dos requisitos cadastrados, por meio da aplicação da técnica TUCCA definida em Belgamo (2004);
- a automatização da técnica PCU para a geração automática da métrica de PCU, segundo processo definido em Karner (1993);
- a geração automática da métrica PF por meio da automação da equação definida em Andrade (2004);
- o suporte ao gerenciamento de requisitos [Di Thommazo, 2007].

O ambiente COCAR foi projetado com o objetivo de ser extensível e flexível a ponto de poder incorporar no futuro outras atividades do processo de desenvolvimento de software, que possam estar apoiadas no Modelo de Casos de Uso, como é o caso da geração de casos de teste com base nesse modelo e que está atualmente sendo desenvolvida por outro mestrando desse grupo de pesquisa. Busca-se, dessa forma, a criação de um ambiente capaz de agregar outros trabalhos acadêmicos e que seja passível de uma utilização prática, tendo como restrição o uso da técnica de modelagem por Casos de Uso o que se espera não seja efetivamente uma restrição já que ela é amplamente utilizada.

Como o objetivo maior do ambiente COCAR é dar suporte a atividades do processo de desenvolvimento de software apoiando-se no Modelo de Casos de Uso, a primeira funcionalidade a ser implementada por este trabalho foi o cadastramento de requisitos seguindo as diretrizes estabelecidas por Kawai (2005) e a técnica TUCCA para a geração do Modelo de Casos de Uso definida por Belgamo (2004). A implementação dessas duas

funcionalidades foi compartilhada com outro mestrando que desenvolveu a parte do ambiente COCAR que provê suporte ao gerenciamento de requisitos Di Thommazo (2007). Dessa forma, algumas conclusões aqui apresentadas, aquelas relacionadas à parte da implementação que foi desenvolvida em conjunto com esse outro mestrando, são coincidentes nos dois trabalhos.

No escopo específico deste trabalho, foram implementados aspectos de automatização da técnica PCU descrita no Capítulo 4 e cujas funcionalidades no contexto do ambiente foram apresentadas no Capítulo 6. No Capítulo 7 essas funcionalidades foram exemplificadas por meio de interfaces que compõem o próprio ambiente COCAR, além de ter sido mostrado um exemplo prático do uso da ferramenta por meio de um exemplo relativo a um sistema real desenvolvido na indústria de software.

A primeira funcionalidade tratada na automatização da técnica PCU foi a Classificação Automática da Complexidade dos Casos de Uso. Segundo vários autores da literatura disponível sobre o assunto, a maior dificuldade em se produzir métricas PCU com qualidade é a falta de padronização existente na especificação do Modelo de Casos de Uso. Dessa forma, este trabalho tentou resolver esse problema gerando o Modelo de Casos de Uso através da automatização da técnica TUCCA, a qual pode contribuir para melhorar a qualidade dessa métrica [Belgamo, 2004] [Belgamo & Fabbri, 2004c].

A segunda funcionalidade específica tratada pela automatização da técnica PCU foi o cálculo da métrica PCU, tendo como base a classificação de complexidade dos Casos de Uso mencionada anteriormente.

A terceira funcionalidade foi a transformação da métrica PCU para a métrica PF, proposta por Andrade (2004). Segundo Caldiera et al. (1998), a técnica APF é uma das principais técnicas de estimativa de software existentes no mercado e conta com várias bases de informações históricas consistentes que ajudam no processo de estimativa de esforço para o desenvolvimento de software.

Para dar subsídio teórico a este trabalho e mapear o conhecimento científico existente na área de métricas de tamanho de software, mais especificamente de Pontos de Caso de Uso, foi apresentada, no Capítulo 5, uma revisão bibliográfica, realizada, em princípio, de maneira *ad hoc* e, posteriormente, por meio de uma Revisão Sistemática [Kitchenham, 2004] [Biolchini

et al., 2005]. Com a Revisão Sistemática é possível conduzir um processo de pesquisa bibliográfica abrangente, passível de repetição, confiável e não dependente dos revisores [Mafra & Travassos 2006]. No entanto, da revisão sistemática apenas os artigos com uma forte ligação com este trabalho foram comentados.

8.1 *Contribuições e Limitações deste Trabalho*

A seguir relacionam-se, resumidamente, algumas contribuições deste trabalho:

- Criação de uma primeira versão do ambiente COCAR, para suporte a algumas atividades do desenvolvimento de software, com base em Casos de Uso, como por exemplo, a inserção de requisitos, criação do Modelo de Casos de Uso automatização da métrica PCU, suporte ao gerenciamento de requisitos e geração de casos de teste baseada em casos de uso.
- Implementação das diretrizes para elaboração de documento de requisitos com ênfase nos requisitos funcionais propostas por Kawai (2005). A implementação da proposta favorece seu uso e avaliação.
- Implementação da TUCCA [Belgamo, 2004] para a geração do Modelo de Casos de Uso. Por ser composta de uma série de passos que compõe um algoritmo, ela pôde ser implementada, tendo alguns passos automatizados. A aplicação da técnica com suporte de uma ferramenta oferece ao usuário um caminho único, a ser seguido passo a passo, minimizando a possibilidade de erros e facilitando sua aplicação.
- Criação de um ambiente extensível para agregar futuras funcionalidades baseadas em Casos de Uso, que possam contribuir para melhorar a qualidade do desenvolvimento de software.
- Implementação da métrica PCU, de acordo com a definição de Karner (1993) e geração do PF [Albrecht, 1979]] a partir do PCU de acordo com Andrade (2004). Essa transformação possibilita que estimativas de esforço sejam realizadas pelo engenheiro de software, baseando-se em métricas PCU, porém utilizando bases históricas baseadas em PF.

A seguir relacionam-se, resumidamente, algumas limitações deste trabalho:

- Apesar da proposta do ambiente COCAR ser de um ambiente que abranja algumas fases do processo de desenvolvimento de software, a ferramenta deve ser capaz de interagir com ferramentas existentes no mercado, por meio de importação/exportação de arquivos XML ou ainda por meio de algum padrão de mercado como o XMI, criado pela OMG com o objetivo de facilitar a troca de "metadados" de modelos da UML entre as ferramentas de modelagem de software baseadas na linguagem UML [OMG, 2006].
- Como já discutido anteriormente, uma das primeiras funcionalidades implementada pelo ambiente COCAR foi a automatização da técnica TUCCA [Belgamo, 2004]. Apesar dessa técnica possuir dois objetivos, a geração do Modelo de Casos de Uso e a revisão do documento de requisitos gerando um relatório de discrepância, somente a geração do modelo foi implementada como parte deste trabalho de mestrado. Desta forma, o ambiente COCAR ainda não está preparado para relatar as discrepâncias encontradas no documento de requisitos por meio da aplicação da técnica TUCCA.
- O ambiente não provê, atualmente, a possibilidade de geração de relatórios impressos ou exportáveis para arquivos texto.
- Atualmente, para o cálculo da métrica PCU, por uma questão de tempo, o ambiente COCAR focou-se em realizar automaticamente apenas a classificação dos Casos de Uso. A classificação de complexidade dos atores, bem como a determinação dos valores dos itens de complexidade técnica e ambiental não foram implementados no âmbito deste trabalho, pois nas pesquisas bibliográficas realizadas encontrou-se apenas um trabalho referente a isso. Esse trabalho foi realizado por Kusumoto et al (2004), que desenvolveu uma técnica para a classificação automática da complexidade de atores baseada em um banco de palavras chaves japonesas, por meio do qual a classificação automática se torna possível. Dessa forma, para a implementação dessa funcionalidade no ambiente COCAR, um trabalho de adaptação da técnica de Kusumoto et al (2004) para o Português deve ser realizado, o que estava fora do âmbito deste trabalho.
- Apesar da TUCCA fornecer Modelos de Caso de Uso padronizados e consistentes, essa técnica não entra no detalhe de como descrever as transações dos Casos de Uso, o que é fundamental para o cálculo dos PCU. Assim, neste trabalho, adotou-se a alternativa de dar suporte à especificação dos Casos de Uso por meio de uma interface amigável, que apresenta sempre os requisitos relacionados ao Caso de Uso que esta sendo especificado. Mas, fica a cargo da pessoa que está especificando, saber o conceito do que vem a ser uma transação para a técnica PCU. Também neste caso, Kusumoto et al (2004) propõem uma forma automática de decidir-se o que é e o que não é uma transação, por meio de recursos de análise sintática e gramatical baseada nos cursos normal e alternativo da

especificação do caso de uso. Além de implementar algum suporte na linha do trabalho de Kusumoto, o ambiente poderia disponibilizar conceitos, via uma função de ajuda, para esclarecer o conceito de transação.

- Foi realizado neste trabalho um pequeno Estudo de Caso para mostrar o uso prático do ambiente COCAR. Contudo, esse exemplo não é suficiente para validar a ferramenta ou mostrar a sua efetividade e eficiência na geração automática da métrica de PCU, necessitando, para isso, de um experimento prático maior, envolvendo um contexto mais abrangente.

8.2 *Lições Aprendidas*

Durante o desenvolvimento deste trabalho de mestrado diversas lições foram aprendidas, algumas das quais são relatada a seguir.

Como duas das funcionalidades do ambiente COCAR – inserção de requisitos de software e geração do Modelo de Casos de Uso – eram parte comum deste trabalho e do mestrado de Di Thommazo (2007), tanto a documentação, projeto e implementação dessas funcionalidades, bem como a descrição das mesmas nos trabalhos, foi realizada em conjunto. Esse trabalho em equipe exigiu organização e sinergia dos envolvidos, sendo necessária à discussão das decisões de projeto e dos aspectos de implementação. Dessa forma, o compartilhamento das experiências dos envolvidos favoreceu para a qualidade do trabalho e para a garantia do caráter extensível da ferramenta. Entretanto, a necessidade da implementação de partes em comum ocasionou erros e atrasos no desenvolvimento. Caso a ferramenta fosse atender apenas as funcionalidades específicas deste trabalho, aceitando como entrada um Modelo de Casos de Uso já projetado, sua especificação, implementação e teste teriam sido simplificados.

Vale ressaltar que a especificação do ambiente COCAR, disponíveis no Anexo I, foi criada utilizando-se os próprios modelos implementados neste trabalho, ou seja, as diretrizes propostas por Kawai (2005). Outro artefato criado para a implementação foi o modelo de Casos de Uso, gerado a partir da especificação citada anteriormente com a aplicação da TUCCA, e disponível no Anexo II. Dessa forma, procurou-se avaliar os benefícios dos

trabalhos científicos que seriam incorporados no ambiente COCAR na própria criação do ambiente, além de documentar a criação do mesmo.

Ainda sobre o ambiente COCAR, pode-se concluir que a prototipação descartável utilizada para a validação dos requisitos, bem como a prototipação evolucionária utilizada para a construção da ferramenta foram essenciais para o sucesso da implementação do ambiente, pois, desta forma, foi possível projetar uma interface mais amigável e ainda realizar aos poucos a integração entre os desenvolvimentos realizados por este trabalho e pelo trabalhado de Di Thommazo(2007).

Outro aspecto a ser considerado como lição aprendida é o da revisão bibliográfica que, apesar de ser iniciada de forma ad-hoc, já no final do trabalho tomou-se conhecimento da abordagem de Revisão Sistemática [Kitchenham, 2004], fazendo-se uma aplicação da mesma para atualizar o conjunto de artigos lidos. Embora esse processo não tenha sido conduzido com o rigor exigido, conclui-se que esse tipo de revisão permite organização e sistematização do trabalho, agregando qualidade nesta etapa da dissertação.

8.3 *Trabalhos Futuros*

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

- Realização de experimentos para avaliar a aplicação da TUCCA de forma manual e de forma assistida pelo ambiente.
- Realização de experimentos para a validação da praticidade do ambiente COCAR no que diz respeito à automatização da técnica de PCU, bem como para a validação da eficiência e efetividade dessa abordagem.
- Avaliar a utilização do ambiente COCAR na indústria para que sejam utilizados em sistemas complexos, de diferentes portes do exemplo utilizado neste trabalho.
- Criar formas de classificação automática da complexidade dos atores, bem como da avaliação dos itens de complexidade técnica e ambiental.
- Implementar no ambiente COCAR técnicas de estimativa de esforço de desenvolvimento de software baseadas em PF e/ou PCU.

- Implementar no ambiente COCAR a parte de identificação de defeitos do Documento de Requisitos por meio do relatório de discrepância descrito na técnica TUCCA [Belgamo, 2004].

Referências Bibliográficas

[Aceit, 2005] ACEIT - Automated Cost Estimating Integrated Tools. Desenvolvida por Tecolote Research Inc. Disponível em: < http://www.aceit.com/>. Acessado em: 30/03/2005.

[Albrecht, 1979] ALBRECHT, A.J. Measuring application development productivity. In: IBM Applic. Dev. Joint SHARE/GUIDE Symposium, Monterey, CA, p. 83-92, 1979.

[Alexander, 2003] ALEXANDER, I. SemiAutomatic Tracing of Requirement Versions to Use Cases Experiences & Challenges. In: Second International Workshop on Traceability, Montreal, Outubro 2003. Acessado em: 02/01/2007. Disponível em http://www.soi.city.ac.uk/~gespan/paper6.pdf. Acesso em 10/07/2006.

[Anda et al., 2001] ANDA, B., DREIEM, D., SJOBERG, D.I.K., JORGESEN, M.. Estimating Software Development Effort Based on Use Cases – Experiences from industry. In: **UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools, 4th International Conference (LNCS 2185)**, Toronto, CA, p. 487-502, 2001.

[Anda, 2002] ANDA, B., Comparing Effort Estimates Based on Use Case Points with Expert Estimates. **Empirical Assessment in Software Engineering (EASE 2002),** Keele, UK, April 8-10, 2002, 13 p..

[Anda & Sjoberg, 2002] ANDA B., SJOBERG, D.I.K.. "Towards an Inspection Technique for Use Case Models", In: **Fourteenth IEEE Conference on Software Engineering and Knowledge Engineering (SEKE)**, 2002.

[Anda et al., 2005] ANDA, B. BENESTAD, H.C. HOVE, S.E.. "A multiple-case study of software effort estimation based on use case points", In: **Empirical Software Engineering, 2005. 2005 International Symposium on**, 10 pp.-, 2005. Disponível em: < http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1541849>

[Andrade, 2004] ANDRADE, E. L. P. **Pontos de Caso de Uso e Ponto de Função na gestão de estimativa de tamanho de projetos de software orientado a objeto.** 2004. 143 f. Tese (Mestrado em Gestão do Conhecimento e Tecnologia da Computação) - Universidade Católica de Brasília. Brasília, 2004.

[APACHE, 2006] Apache Software Fundation. Disponivel em : http://www.apache.org/. Acesso em: 13/10/2006

[Arnold & Pedross, 1998] ARNOLD, P. and PEDROSS, P. Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department. **IEEE Comput. Soc**, Los Alamitos, CA, USA, p. 490-493. 1998.

[Belgamo, 2004] BELGAMO, A. **GUCCRA: Técnicas de Leitura para Construção de Modelos de Casos de Uso e Análise do Documento de Requisitos**. 2004. 157 f. Tese (Mestrado em Ciências da Computação) - Universidade Federal de São Carlos, São Carlos, 2004.

[Belgamo & Fabbri, 2004a] BELGAMO, A., FABBRI, S.C.P.F.. Constructing Use Case Model by Using a Systematic Approach: Description of a Study. **In: WER – Workshop em Engenharia de Requisitos**, Tandil, Argentina, p. 251 – 262, 2004.

[Belgamo & Fabbri, 2004b] BELGAMO, A., FABBRI, S.C.P.F.. GUCCRA: Contribuindo para a Identificação de Defeitos em Documentos de Requisitos Durante a Construção de Modelos de Casos de Uso. **In: WER – Workshop em Engenharia de Requisitos**, Tandil, Argentina, p. 100 – 111, 2004.

[Belgamo & Fabbri, 2004c] BELGAMO, A., FABBRI, S.C.P.F.. GUCCRA: Um Estudo sobre a Influência da Sistematização da Construção de Modelos de Casos de Uso na Contagem dos Pontos de Casos de Uso. In**: III Simpósio Brasileiro de Qualidade de Software**. Brasília 2004.

[Belgamo & Fabbri, 2005] BELGAMO, A., FABBRI, S.C.P.F.. GUCCRA: Técnica de Leitura para apoiar a Construção de Modelos de Caso de Uso e a Análise de Documentos de Requisitos. **In: SBES – 19º Simpósio Brasileiro de Engenharia de Software**, Uberlândia, MG, 3-7 out., 2005.

[Belgamo et al., 2005] BELGAMO, A., FABBRI, S.C.P.F., MALDONADO, J.C.. Avaliando a Qualidade da Técnica GUCCRA com Técnica de Inspeção. **In: WER – Workshop em Engenharia de Requisitos**, Porto, Portugal, 13-14 jun., 2005.

[Biolchini et al., 2005] BIOLCHINI, J. ; MIAN, P. ; NATALI, A. ; TRAVASSOS, G. . Systematic Review in Software Engineering, Relatório Técnico RT - ES 679 - 05, 2005.

[Boehm et al., 1995] BOEHM, B., CLARK, B., HOROWITZ, E., WESTLAND, C., MADACHY, R., SELBY, R. Cost models for future software life cycle processes: COCOMO 2.0. In: USC center for software engineering, 1995. Disponível em: URL: http://sunset.usc.edu/publications/TECHRPTS/1995/index.html. Acesso em: 03/03/2005.

[Boock et al., 2000] BOOCH, G; JACOBSON, I; RUMBAUGH, J. **UML Guia do Usuário.** Tradução de Fábio Freitas da Silva. Rio de Janeiro. Editora Campus, 2000. 472 p.

[Caldieira et al., 1998] CALDIERA, G, G. ANTONIOL, R. FIUTEM, C. LOKAN, "Definition and Experimental Evaluation of Function Points for Object-Oriented Systems," In: International Symposium on Software Metrics , 5., Bethesda, Maryland. p 167 – 179.

[Carbone & Santucci, 2002] CARBONE, M., SANTUCCI, G. Fast&&Serious: a UML based metric for effort estimation. **6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'02).** 12p. Spain., 2002.

[Carrol, 2005] CARROLL, E. R.. Practitioner reports: Estimating software based on use case points. In: **Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications OOPSLA '05**, San Diego, CA, USA, p. 257-265, 2005.

[Chen et. al 2004] CHEN, Y., BOEHM, B.W, MADACHY, R., VALERDI, R.. An empirical study of eServices product UML sizing metrics. **ACM-IEEE Int'l Symposium on Empirical Software Engineering (ISESE 2004)**. Redondo Beach CA, USA, Ago. 2004.

[Construx, 2005] CONSTRUX – Construx Estimate. Desenvolvida pela Construx Software Builders. Disponível em: < http://www.construx.com/resources/estimate/> . Acesso em: 20/03/2005.

[Cosmos, 2005] COSMOS - The Software Cost Modeling System. Desenvolvida pela East Tennessee State University. Disponível em: < http://www-cs.etsu.edu/cosmos/>. Acesso em: 25/03/2005.

[Costar, 2005] COSTAR - *CoStar* Software *Estimation Tool*. Desenvolvida por Softstar Acesso em 10/03/2005. Disponível em: < >.

[Costxpert, 2005] COSTXPERT - Cost Xpert Group Incorporated. Quick Start Guide. Acesso em 15/03/2005. Disponível em: <http://www.costxpert.com/products/downloads.html>.

[Cockburn, 2000] COCKBURN, C.. The Object Factory. Estimating Software Projects using ObjectMetrix. White paper. April 2000.

[Cockburn, 2001] Cockburn, A. **Escrevendo Casos de Uso Eficazes**. Tradução de Roberto Vedoato.Porto Alegre. Bookman, 2001. 254 p.

[Costagliola et al., 2006] COSTAGLIOLA, G., DI MARTINO, S., FERRUCCI, F., GRAVINO, C., TORTORA, G., VITIELLO, G.. Effort estimation modeling techniques: a case study for web applications. In: **Proceedings of the 6th international conference on Web engineering ICWE '06**. Palo Alto, California, USA. p 9-16. 2006.

[Damodaran, 2003] DAMODARAN, M; WASHINGTON, A. Estimation using use case points. **Computer Science Program**. Texas – Victoria: University of Houston. 2003. Disponível em: <http://bfpug.com.br/Artigos/UCP/Damodaran-Estimation_Using_Use_Case_Points.pdf>. Acesso em 20/05/2005.

[Diev, 2006] Diev, S.. Software estimation in the maintenance context. In: **SIGSOFT Softw. Eng. Notes 2006**, New York, NY, USA , v. 31, n. 2 p 1-8, 2006. Acesso em 02/01/2007. Disponível em: < http://portal.acm.org/citation.cfm?id=1118540&jmp=cit&coll=Portal&dl=GUIDE&CFID=1868278&CFTOKEN=50669673#>

[Di Thommazo, 2007] DI THOMMAZO, A. Gerenciamento De Requisitos No Ambiente COCAR. 2007. 159 f. Tese (Mestrado em Ciências da Computação) - Universidade Federal de São Carlos, São Carlos, 2007.

[Eclipse, 2006] ECLIPSE. Disponivel em : http://www.eclipse.org/. Acesso em: 13/08/2006.

[EEUC,2005] EEUC *Estimate Easy Use Case*. Disponível em: http://www.duvessa.com. Acesso em: 12/12/2006

[Egyed & Grübacher, 2002] EGYED, A.; GRÜNBACHER, P. Automating Requirements Traceability: Beyond the Record & Replay Paradigm; In: **Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on.** p. 163 – 171. 2002. Acessado em: 02/02/2007. Disponível em: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1115010

[Fetcke et al., 1998] FETCKE. T., ABRAN, A. and NGUYEN, T.-H. Mapping the OO-Jacobson Approach into Function Point Analysis. **International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-23). IEEE Comput. Soc**, Los Alamitos, CA, USA, p. 192-202. 1998.

[Fowler, 2005] FOWLER, M., KOBRYN, C., BOOCH, G., **UML Essencial**. 3a. edição. Bookman, 2005, 160 p.

[Garmus & Herron, 2000] GARMUS, D., HERRON, D. Function Point Analysis: Measure Practices for successful software projects. Addison Wesley: EUA. 2000. 363 p.

[Gregolin, 2006] GREGOLIN, B.T. Uma Proposta de Ferramenta para a Geração de Cados de Teste a partir de Casos de Uso. 83 f. (Exame de Qualificação de Mestrado em Ciências da Computação) - Universidade Federal de São Carlos, São Carlos, 2006.

[Hazan, 1999] HAZAN, C. **Análise de Pontos por Função**: uma abordagem gerencial. Rio de Janeiro,SEPRO. 1999. 1v

[Hazan & Leite, 2003] Hazan, C.; Leite, J.C.S.P. Indicadores para a Gerência de Requisitos. In WER; 2003.

[Heimberg & Grahl, 2005] HEIMBERG, V., EVERALDO, E. A., Estudo de Caso da Aplicação da Métrica de Pontos de Caso de Uso numa Empresa de Software. In: **XIV Seminário de Computação**, Blumenau, SC, Brasil, 2005. Disponível em: < http://www.inf.furb.br/seminco/2005/artigos/130-vf.pdf >. Acesso em: 25/12/2005.

[IFPUG, 1999] IFPUG International Function Point Users Group. **Function Point Counting Pratices Manual.** Release 4.1. Ohio: IFPUG. 1999. 1 v.

[Jacobson et al., JACOBSON, I. **Object-Oriented Software Engineering –**

1992] **A Use Case Driven Approach.** USA. Addison-Wesley, 1992, 528 p.

[Java, 2006] JAVA - Core J2EE Pattern Catalog. Disponível em: http://java.sun.com/blueprints/corej2eepatterns. Acessado em: 30/10/2006.

[Karner, 1993] KARNER, G. **Use Case Points:** resource estimation for Objectory projects. Objective Systems SF AB (copyright owned by Rational/IBM), 1993.

[Kawai, 2005] KAWAI, K. **Diretrizes para elaboração de Documento de Requisitos com ênfase nos Requisitos Funcionais**. 2005. 170 f. Tese (Mestrado em Ciências da Computação) - Universidade Federal de São Carlos, São Carlos, 2005.

[Kitchenham, 2004] KITCHENHAM, B Procedures for Performing Systematic Reviews, Joint Technical Rreport, Keele University TR/SE-0401 e NICTA 0400011T.1, 2004

[Kulak & Guiney, 2000] KULAK, D.; GUINEY, E. **Use Cases: Requirements in Context.** USA**.** Addison-Wesley, 2000. 329 p.

[Kusumoto et. al., 2004] KUSUMOTO, S.; MATUKAWA, F.; INOUE, Y.,HANABUSA, S.; MAEGAWA, Y. Effort Estimation Tool Based on Use Case Points Method. **Computer, Software Metrics, 10th International Symposium on Metrics (METRICS'04)** Chicago, Illinois, set. 2004, p. 9 – 11.

[Letelier, 2002] LETELIER, P. A Framework for Requirements Traceability in UML-based Projects. In: **1st International Workshop on Traceability in Emerging Forms of Software Engineering**, Edinburgh, U.K, p32-41, 2002

[Longstreet, 2002] LONGSTREET, D., **Fundamentals of Function Points Analysis.** Blue Springs: Longstreet Consulting Inc., 2002. Disponível em: http://www.ifpug.com/Articles/default.htm. Acesso em 20/12/2006.

[Longstreet, 2004] LONGSTREET, D., **Function Points Analysis Training**

Course. October 2004. 116 p.. Disponível em: www.softwaremetrics.com. Acesso em: 21/03/2004.

[Marcus et al. , 2005] MARCUS A.; XIE, X.; POSHYVANYK, D. When and how to visualize traceability links? In: **Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering- Automated Software Engineering**, Long Beach, USA, 2005

[McPhee, 1999] McPHEE, C. Seng 621: Software process management:_software size estimation. University of Calgary. 1999. 11p. Disponível em: http://sern.ucalgary.ca/~cmcphee/SENG621/Software_Size_Estimation.html. Acesso em 19/09/03

[Medeiros, 2004] MEDEIROS, E. S., **Desenvolvendo Software com UML 2.0: Definitivo**. 1ª edição . São Paulo. Makron Books, 2004, 264 p.

[MCT, 2002] MCT. MINISTÉRIO DE CIÊNCIAS E TECNOLOGIA. **Qualidade e produtividade no setor de software brasileiro**. Brasília: Ministério de Ciências e Tecnologia. Secretaria de Política de Informática. 2002. 258p. (n. 4).

[MCT, 2003] MCT. MINISTÉRIO DE CIÊNCIAS E TECNOLOGIA. Notícias de TI: uma vitória do software brasileiro. Notícia publicada no Estadão em 29/09/03. 2003b. Acesso em 06/01/2004. Disponível em: <http://www.mct.gov.br/Temas/info/Imprensa/Noticias_Anteriores/Noticias_2003/Software_2003.htm >.

[MCT, 2005] MCT. MINISTÉRIO DE CIÊNCIAS E TECNOLOGIA. Tecnologia da Informação – Qualificação. Desenvolvida pelo Ministério de Ciências e Tecnologia. Disponível em: <http://www.mct.gov.br/Temas/info/Dsi/qualidad/CMM.htm>. Acesso em: 08/05/2005.

[Mohagheghi et al., 2005] MOHAGHEGHI, P.; ANDA, B.; CONRADI, R., Effort Estimation of Use Cases for Incremental Large-Scale Software Development. **International Conference of Software Engineering, ICSE'05**, St Louis, Missouri, USA, May 15-21, 2005

[Munson & Nguyen, 2005] MUNSON, E. V.; NGUYEN, T. N. Concordance, conformance, versions, and traceability; In: **Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering**, Long Beach, California, 2005.

[OMG, 2005] OMG. **Object Management Group.** Disponível em: http://www.omg.org/. Acesso em 01/05/2005.

[OMG, 2006] OMG. **Object Management Group.** Disponível em: http://www.omg.org/technology/documents/formal/xmi.htm. Acesso em 01/05/2006.

[Pressman, 2006] PRESSMAN, R, S. **Engenharia de Software**. 6a. edição. USA. McGrawHill, 2006, 843 p.

[Price, 2005] PRICE - Price Cost Models. Desenvolvida por Price System LLC. Disponível em: < http://www.pricesystems.com/productservice/pricetp.html#costmodels>. Acessado em: 01/04/2005.

[ROSE, 2006] RATIONAL ROSE – Rational Rose Enterprise. Desenvolvida pela Rational/IBM. Acesso em: 20/12/2006. Disponível em: < http://www-306.ibm.com/software/awdtools/developer/rose/index.html> .

[Ryser & Glinz, 1999] RYSER, J.; GLINZ, M. A Method Employing Scenarios to Systematically Derive Test Cases for System Test. University of Zurich, Insitut für Informatik, Zürich, 1999a. Disponível em: http://www.ifi.unizh.ch/groups/req/ftp/SCENT/SCENT.pdf Acesso em: 24/07/2005

[Ryser & Glinz, RYSER, J.; GLINZ, M. Using Dependence Charts to

2000] Improve Scenario-Based Testing. In: **17th International Conference on Testing Computer Software TCS'2000**. Washintong, D.C. 2000

[Salem,2006] SALEM, A.M. Improving Software Quality through Requirements Traceability Models; In: The 4th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2006), Dubai, Sharjah, UAE, 2006.

[Schneider & Winters, 2001] SCHNEIDER, G.; WINTERS, J. P. **Applying Use Cases, A Pratical Guide.** 2a Edition, USA, Addison-Wesley, 2001. 245 p.

[Smith, 1999] SMITH, J. The Estimation of Effort Based on Use Cases. Rational Software, White paper.1999.

[SUN, 2006a] SUN. Acessado em: 15/11/2006. Disponível em: http://www.sun.com/java/everywhere/#facts.

[SUN, 2006b] SUN. Acessado em: 15/11/2006. Disponível em: http://www.sun.com/java/everywhere/#awards.

[Sommerville, 2003] SOMMERVILLE, I. **Engenharia de Software**. 6a. edição. Traduzido por: Mônica Maria G. Ttravieso – Rio de Janeiro. Addison Wesley, 2003, 580 p.

[SWEBOK, 2004] IEEE Computer Society Professional Practices Committee. **Guide to the Software Engineering Body of Knowledge SWEBOK.** Los Alamitos, California: BOURQUE, P., DUPUIS, R., 2004. 202 p.

[Travassos & Mafra, 2006] MAFRA, S.; TRAVASSOS, G**. Estudos Primários e Secundários apoiando a busca por Evidência em Engenharia de Software**. . Relatório Técnico 687/06 PESC – Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ,. Rio de Janeiro, RJ, 2006.

[Thayer & Dorfman, 1997] THAYER, R. H.; DORFMAN, M. **Introduction to Tutorial Software Requirements Engineering**. **Software Requirements Engineering.** IEEE-CS Press, 2a Edition, 1997, p.p. 1-2.

[UML, 2003] UML. **OMG Unified Modeling Language Specification.** Versão 1.5, março, 2003, 736 p.. Disponível em: http://www.omg.org/technology/documents/formal/uml.htm . Acesso em: 01/02/2005.

[Vinsen et al, 2004] VINSEN, K. JAMIESON, D. CALLENDER, G. Use case estimation - the devil is in the detail. In: **Requirements Engineering Conference,** 2004. Proceedings. 12th IEEE International, p. 10-15, 2004.

[Wiegers, 1999a] WIEGERS, K. Software Requirements, Microsoft Press, 1999. 1a edição.

[Wiegers, 1999b] WIEGERS, K. Automating Requirements Management. Acesso em: 21/04/2005. Disponível em: http://www.processimpact.com/articles/rm_tools.html.

[Zisman & Spanoudakis, 2004] ZISMAN, A.; SPANOUDAKIS, G. Software Traceability: Past, Present, and Future. In : **The Newsletter of the Requirements Engineering Specialist Group of the British Computer Society** – Disponível em : http://www.resg.org.uk/archive/rq33.pdf p.13. Acesso em: 05/03/2005

Apêndice I
Exemplos de Requisitos Funcionais do Ambiente COCAR

Na seqüência são disponibilizados alguns requisitos funcionais da especificação do ambiente Ícaro.

Exemplo de requisito de Inserção de um Sistema

Requisito: 001

Descrição: Cadastrar Sistema

Agente Fornecedor/Receptor: Responsável

Agente Executor: Responsável

Entrada:

nro	campo	formatação	restrição	descrição
1	Nome do sistema	String	64 caracteres	Nome do sistema
2	Descrição do sistema	String	512 caracteres	Descrição do sistema
3	Cliente	String	Deve estar num conjunto de clientes cadastrados	Cliente que solicitou o sistema
4	Data de cadastro	Data	Formato dd/mm/aaaa	Data que foi realizado o cadastro
5	Responsável pelo sistema	String	Deve estar num conjunto de responsáveis cadastrados	Responsável pelo sistema
6	Funções do Produto	String	4096 caracteres	Funções que o sistema deve apresentar

Processamento: O Responsável deve solicitar à ferramenta a inclusão de um novo Sistema. O Responsável deve preencher todos os campos de entrada exibidos na interface e mostrados no quadro de entradas (Nome do sistema, Descrição do sistema, Cliente, Data de cadastro, Responsável pelo sistema, Funções do Produto). O Responsável salva os dados. A ferramenta deve atribuir, automaticamente, um número de identificação auto-incremental ao sistema. A qualquer momento o Responsável pode escolher a opção de Cancelar o cadastro de um Sistema. Quando isso ocorrer, deve ser exibida a interface inicial da ferramenta.

Condição/Restrição: O campo "nome do sistema" deve ser único.

Saída: Dados do Sistema salvos no banco de dados da ferramenta.

Solicitante do Requisito: André

Gerente do Requisito: André

Data: 28/11/2005

Exemplo de requisito do módulo de Inserção dos Requisitos de um Sistema

Requisito: 005

Descrição: Cadastrar Requisito

Agente Fornecedor/Receptor: Responsável

Agente Executor: Responsável

Entrada:

nro	campo	formatação	restrição	descrição
1	Descrição	String	256 caracteres	Descrição do requisito
2	Agente Fornecedor/Receptor	String	Devem estar num conjunto de agentes cadastrados	Agentes fornecedor/receptor envolvidos no requisito
3	Agente Executor	String	Devem estar num conjunto de agentes cadastrados	Agentes executores envolvidos no requisito
4	Entrada	String	Devem estar num conjunto de Tipos de Entrada cadastrados	Conjunto de dados de entrada do requisito que está sendo especificado
5	Processamento	String	4096 caracteres	Explicação do requisito como uma seqüência de passos
6	Condição/Restrição	String	4096 caracteres	Restrições do requisito
7	Saída	String	4096 caracteres + conjunto de dados cadastrados no conjunto de dados cadastrados	Qualquer dado de saída, incluindo mensagens, textos ou dados produzidos e armazenados no

				sistema.
8	Stakeholder	String	Devem estar num conjunto de stakeholders cadastrados	Quem propôs o requisito
9	Data	Data	Formato dd/mm/aaaa	Data de entrada do requisito na ferramenta
10	Responsável	String	Deve estar num conjunto de responsáveis cadastrados	Responsável pelo cadastramento do requisito

Processamento: A ferramenta lista os sistemas cadastrados conforme o requisito 002. O Responsável solicita a inclusão de um novo Requisito para o sistema escolhido. O Requisito representa uma funcionalidade que o Sistema deve oferecer. O Responsável deve preencher todos os dados apresentados no quadro de entradas. Para salvar os dados do requisito o Responsável deve escolher a opção Salvar. A ferramenta deve atribuir, automaticamente, dois números ao requisito: um número de identificação auto-incremental e um número de versão. A qualquer momento o Responsável pode escolher a opção de Cancelar o cadastro de um Requisito no sistema. Quando isso ocorrer, deve ser exibida a interface inicial da ferramenta.

Condição/Restrição: Requisito 002 concluído com sucesso. O campo "descrição" deve ser único.

Saída: Dados do requisito salvos no banco de dados da ferramenta.

Solicitante do Requisito: André

Gerente do Requisito: André

Data: 28/11/2005

Exemplo de requisito do módulo de Elaboração do Modelo de Casos de Uso

Requisito: 029

Descrição: Marcar candidatos a atores, funcionalidades e restrições

Agente Fornecedor/Receptor: Responsável

Agente Executor: Responsável

Entrada:

nro	campo	formatação	restrição	descrição
1	Ator	String	4096 caracteres	Conjunto de atores do requisito ou das funções do produto
2	Função	String	4096 caracteres	Conjunto de Funções do

				requisito ou das funções do produto
3	Restrição	String	4096 caracteres	Conjunto de Restrições do requisito ou das funções do produto

Processamento: A ferramenta deve exibir os sistemas cadastrados para que o Responsável selecione um deles. O Responsável solicita a aplicação da técnica AGRT para o sistema escolhido. A ferramenta faz sugestões de atores, funcionalidades e restrições, de acordo com os campos Agente Fornecedor/Receptor, Descrição e Condição/Restrição de cada requisito cadastrado no sistema, respectivamente. O Responsável ou aceita cada sugestão ou a edita de acordo com a necessidade. Também é exibido a seção "Funções do produto" do sistema para que o Responsável identifique candidatos a atores, funcionalidades e restrições nesta seção.

Condição/Restrição: ao menos um sistema esteja cadastrado.

Saída: requisitos e seção "Funções do produto" com candidatos a atores, funcionalidades e restrições marcados.

Solicitante do Requisito: Marcos

Gerente do Requisito: Marcos

Data: 24/01/2006

Exemplo de requisito do módulo de Geração de Pontos de Casos de Uso

Requisito: 047

Descrição: Gerar medida de Pontos de Casos de Uso não ajustados para um sistema.

Agente Fornecedor/Receptor: Responsável

Agente Executor: Responsável

Entrada:

nro	campo	formatação	restrição	descrição
1	Tipo de complexidade de cada ator do sistema	Inteiro	1 caracter para cada ator	Classificar os atores como simples (1) médio (2) ou complexo (3)

Processamento: A ferramenta lista os sistemas cadastrados conforme o requisito 002, sendo que a opção Tem diagrama de casos de uso associado é marcada verdadeiro e não pode ser alterada. O Responsável escolhe um sistema e confirma a geração da medida de pontos de

casos de uso. O Responsável atribui um tipo de complexidade para cada ator do sistema, ao qual é atribuído um fator de peso, de acordo com a tabela.

Tipo de ator	Descrição	Fator de peso
Simples	Outro sistema interagindo através interface com aplicação definida	1
Médio	Outro sistema interagindo através de protocolo ou linha de comando	2
Complexo	Uma pessoa interagindo através de uma interface gráfica ou página na internet	3

Para cada caso de uso, a ferramenta conta os passos dos cursos normal e alternativos, que representam as transações em cada caso de uso, e então o caso de uso é classificado de acordo com o número de transações e recebe um fator de peso, de acordo com a tabela.

Tipo de caso de uso	Descrição	Fator de peso
Simples	Até 3 transações (até 5 objetos lógicos)	5
Médio	4 a 7 transações (6 a 10 objetos lógicos)	10
Complexo	Mais que 7 transações (mais que 10 objetos lógicos)	15

A medida de pontos de casos de uso não ajustados é obtida pela fórmula $PCU(na) = \sum(PesosDosAtores) + \sum(PesosDosCasosDeUso)$.

Condição/Restrição: Requisito 002 concluído com sucesso.

Saída: Medida de pontos de casos de uso não ajustados.

Solicitante do Requisito: Marcos

Gerente do Requisito: Marcos

Data: 08/12/2005

Exemplo de requisito do módulo de Suporte ao Gerenciamento de Requisitos

Requisito: 051

Descrição: Visualizar Indicador de Estabilidade

Agente Fornecedor/Receptor: Responsável

Agente Executor: Responsável

Entrada:

nro	campo	formatação	restrição	descrição
1	Data inicial	Data	Formato dd/mm/aaaa	Data de início do período de busca para gerar o indicador

2	Data final	Data	Formato dd/mm/aaaa	Data de final do período de busca para gerar o indicador

Processamento: A ferramenta lista os sistemas cadastrados conforme o requisito 002. O Responsável escolhe um sistema e solicita a visualização do Indicador de Estabilidade. Além disso ele deve fornecer o período (data inicial e data final) para o qual deve ser gerado o indicador. O sistema deve exibir os valores do Indicador de Estabilidade, descrevendo:

- Qual o percentual de novos requisitos no período: para a determinação desse indicador deve-se buscar o número de requisitos que existiam na data inicial do período e o número de requisitos que existiam na data final do período. Subtraindo-se o número de requisitos que existiam na data final do período pelo número de requisitos que existiam na data inicial do período obtém-se o número de requisitos que foram inseridos no período. Para a determinação do percentual de novos requisitos, deve-se dividir esse valor obtido pelo número de requisitos da data inicial do período.
- Qual o percentual de requisitos alterados no período: para a determinação desse indicador deve-se buscar o número de requisitos que foram alterados entre o período compreendido entre a data inicial e data final fornecidas. Para a determinação do percentual de requisitos alterados, deve-se dividir esse valor obtido pelo número de requisitos da data final do período.
- Qual o percentual de requisitos excluídos no período: para a determinação desse indicador deve-se buscar o número de requisitos que foram excluídos entre o período compreendido entre a data inicial e a data final fornecidas. Para a determinação do percentual de requisitos excluídos, deve-se dividir esse valor obtido pelo número de requisitos da data final do período.

Condição/Restrição: Requisito 002 concluído com sucesso.

Saída:

nro	campo	formatação	restrição	descrição
1	Requisitos novos	Float	Valores percentuais com 2 casas decimais	Percentual de requisitos novos no período especificado.
2	Requisitos alterados	Float	Valores percentuais com 2 casas decimais	Percentual de requisitos alterados no período especificado.
3	Requisitos Excluidos	Float	Valores percentuais com 2 casas decimais	Percentual de requisitos excluídos no período especificado.

Solicitante do Requisito: André

Gerente do Requisito: André

Data: 17/12/2005

Apêndice II
Exemplos de Especificação de Caso de Uso do Ambiente COCAR

Na seqüência são disponibilizadas algumas especificações de Casos de Uso do ambiente Ícaro construídas a partir da aplicação da técnica TUCCA no documento de requisitos apresentado no Apêndice I.

Exemplo de Especificação de Caso de Uso de Inserção de um Sistema

Especificação do Caso de Uso	**Número:**	**01**
Nome do Caso de Uso	Cadastrar sistema	
Descrição ou Resumo	Cadastrar sistema na ferramenta	
Ator Participante	Responsável	
Ator Operador	Responsável	
Ator Genérico	-	
Pré-condição	-	
Curso Normal	1. Ferramenta exibe campos para preenchimento 2. Responsável preenche todos os campos e salva 3. Ferramenta confirma que o nome do sistema ainda não existe 4. Ferramenta atribui um número auto-incremental para o sistema e salva no banco de dados	
Curso Alternativo	2a. Responsável não preenche todos os campos 2a1. Ferramenta informa que é necessário preencher todos os campos 2a2. Voltar ao passo 2 2b. Responsável cancela cadastro 2b1. Ferramenta exibe tela inicial 3a. Ferramenta confirma que o nome do sistema já existe 3a1. Ferramenta informa que deve ser utilizado outro nome do sistema 3a2. Voltar ao passo 2	
Evento Disparador	O Responsável seleciona a opção	
Include		
Extend	-	
Requisitos Funcionais	001	
Requisitos Não-Funcionais	-	

Exemplo de Especificação de Caso de Uso de Inserção dos Requisitos de um Sistema

Especificação do Caso de Uso	**Número:**	**03**
Nome do Caso de Uso	Cadastrar requisito	
Descrição ou Resumo	Cadastrar requisito para um determinado sistema	
Ator Participante	Responsável	
Ator Operador	Responsável	
Ator Genérico	-	
Pré-condição	-	
Curso Normal	1. Include: Listar sistemas 2. Responsável seleciona um sistema no qual deseja cadastrar um requisito 3. Ferramenta exibe campos para preenchimento 4. Responsável preenche todos os campos e salva 5. Ferramenta confirma que a descrição do requisito ainda não existe 6. Ferramenta atribui um número auto-incremental para o sistema e o número de versão 1 e salva no banco de dados	
Curso Alternativo	2a. Responsável cancela alteração 2a1. Ferramenta exibe tela inicial 4a. Responsável cancela alteração 4a1. Ferramenta exibe tela inicial 5a. Ferramenta confirma que a nova descrição do requisito já existe 5a1. Ferramenta informa que deve ser utilizada outra descrição do requisito 5a2. Voltar ao passo 4	
Evento Disparador	O Responsável seleciona a opção	
Include		
Extend	-	
Requisitos Funcionais	005	
Requisitos Não-Funcionais	-	

Exemplo de Especificação de Caso de Uso de Aplicação da AGRT

Especificação do Caso de Uso	**Número:**	**09**
Nome do Caso de Uso	Aplicar AGRT	
Descrição ou Resumo	Aplicar AGRT num determinado sistema	
Ator Participante	Responsável	
Ator Operador	Responsável	
Ator Genérico	-	
Pré-condição	Deve haver ao menos um sistema cadastrado na ferramenta	
Curso Normal	1. Ferramenta exibe uma lista selecionável com todos os sistemas cadastrados 2. Responsável seleciona um sistema para aplicar a AGRT 3. Ferramenta exibe requisito e faz sugestões de ator, funcionalidade e restrição 4. Responsável confirma as sugestões feitas 5. Ferramenta inclui o ator, a funcionalidade e o número do requisito no FAO 6. Voltar ao passo 3 até que todos os requisitos tenham sido considerados 7. Ferramenta exibe seção 'Funções do Produto' do sistema 8. Responsável identifica funcionalidades na seção 'Funções do Produto' 9. Responsável associa estas funcionalidades à funcionalidades já identificadas 10. Ferramenta exibe lista de funcionalidades de um mesmo ator com possibilidade de seleção múltipla 11. Responsável seleciona funcionalidades com o mesmo significado e seleciona opção combinar 12. Responsável seleciona o termo mais apropriado para as funcionalidades 13. Ferramenta agrupa as referências das funcionalidades selecionadas numa só, utilizando o termo selecionado 14. Ferramenta substitui os termos descartados pelo termo selecionado nos requisitos referenciados pela funcionalidade 15. Responsável preenche campos da discrepância e confirma 16. Ferramenta salva discrepância 17. Voltar ao passo 11 até que não haja mais funcionalidades ambíguas 18. Voltar ao passo 10 até que todos os atores tenham sido considerados 19. Ferramenta exibe lista de funcionalidades dos atores, podendo associar uma letra ou um asterisco a cada funcionalidade 20. Responsável marca cada objetivo do ator 'Sistema'	

	ou <nome do sistema> com uma letra diferente, e marca as funcionalidades com o mesmo significado com a mesma letra 21. Responsável seleciona o termo mais apropriado para as funcionalidades marcadas com uma mesma letra 22. Ferramenta substitui os termos descartados pelo termo selecionado nos requisitos referenciados pelas funcionalidades 23. Ferramenta agrupa as referências das funcionalidades marcadas com a mesma letra numa só e atribui ao ator que não 'Sistema' ou <nome do sistema>, utilizando o termo selecionado 24. Responsável preenche campos da discrepância e confirma 25. Voltar ao passo 21 até que todas as letras utilizadas para marcar as funcionalidades sejam consideradas 26. Responsável associa funcionalidade marcada com asterisco a algum ator 27. Voltar ao passo 26 até que todas as funcionalidades marcadas com asterisco tenham sido consideradas 28. Ferramenta exibe lista de funcionalidades dos atores, podendo associar um número a cada funcionalidade 29. Responsável marca funcionalidades com o mesmo significado com um mesmo número 30. Ferramenta agrupa as referências das funcionalidades selecionadas numa só, utilizando o termo selecionado 31. Responsável seleciona o termo mais apropriado para as funcionalidades marcadas com um mesmo número 32. Ferramenta substitui os termos descartados pelo termo selecionado nos requisitos referenciados pelas funcionalidades 33. Responsável preenche campos da discrepância e confirma 34. Voltar ao passo 30 até que todos os números utilizados para marcar as funcionalidades sejam considerados 35. Ferramenta verifica que todo requisito foi referenciado em alguma funcionalidade
Curso Alternativo	2a. Responsável cancela aplicação da AGRT 2a1. Ferramenta exibe tela inicial 4a. Responsável não confirma sugestões feitas 4a1. Responsável altera sugestões de acordo com a necessidade e confirma 4a2. Ferramenta inclui o ator, a funcionalidade e o número do requisito no FAO 4a3. Voltar ao passo 3 4b. Responsável não confirma sugestões feitas

	4b1. Responsável associa requisito a uma funcionalidade já identificada 4b2. Ferramenta inclui número do requisito na funcionalidade identificada 4b3. Voltar ao passo 3 4c. Responsável não confirma sugestões feitas 4c1. Desconsiderar requisito e voltar ao passo 3 4d. Responsável não confirma sugestões feitas 4d1. Responsável seleciona opção nova funcionalidade 4d2. Responsável preenche campos para nova funcionalidade e confirma 4d3. Ferramenta inclui o ator, a funcionalidade e o número do requisito no FAO 4d4. Voltar ao passo 3 9a. Responsável não encontra funcionalidade para associar 9a1. Responsável seleciona opção nova funcionalidade 9a2. Responsável preenche campos para nova funcionalidade e confirma 9a3. Ferramenta inclui o ator, a funcionalidade e a seção 'Funções do Produto' no FAO 9a4. Responsável preenche campos da discrepância e confirma 9a5. Ferramenta salva discrepância 9a6. Voltar ao passo 9 11a. Não há funcionalidades com o mesmo significado 11a1. Ir para o passo 19 20a. Não há funcionalidades com o mesmo significado de uma funcionalidade do ator 'Sistema' ou <nome do sistema> 20a1. Responsável marca funcionalidade com um asterisco 20a2. Voltar ao passo 20 26a. Funcionalidade não pode ser associada a nenhum ator 26a1. Ferramenta move a funcionalidade para a lista de objetivos não-associados 26a2. Voltar ao passo 26 29a. Não há funcionalidades com o mesmo significado 29a1. Ir para o passo 35 35a. Ferramenta verifica que um requisito não foi referenciado em nenhuma funcionalidade 35a1. Responsável preenche campos da discrepância e confirma 35a2. Voltar ao passo 35 e checar demais requisitos
Evento Disparador	O Responsável seleciona a opção
Include	
Extend	-
Requisitos Funcionais	029, 030, 031, 032, 033, 034, 035
Requisitos Não-Funcionais	-

Exemplo de Especificação de Caso de Uso de Geração de PCU

Especificação do Caso de Uso	**Número:**	**10**
Nome do Caso de Uso	Gerar medida de PCU	
Descrição ou Resumo	Gerar medida de pontos de casos de uso para um determinado sistema	
Ator Participante	Responsável	
Ator Operador	Responsável	
Ator Genérico	-	
Pré-condição	-	
Curso Normal	1. Include: Listar sistemas (com a opção 'Tem diagrama de casos de uso associado' marcada verdadeiro) 2. Responsável seleciona um sistema 3. Ferramenta exibe a lista de atores do sistema selecionado 4. Responsável atribui peso a cada um dos atores 5. Ferramenta conta os passos dos cursos normal e alternativos de cada caso de uso do sistema e atribui peso a cada um deles 6. Ferramenta calcula pontos de casos de uso não ajustados para o sistema 7. Ferramenta exibe o questionário de Fator de Complexidade Técnica 8. Responsável atribui pontuação a cada um dos fatores do questionário 9. Ferramenta exibe o questionário de Fator Ambiental 10. Responsável atribui pontuação a cada um dos fatores do questionário 11. Ferramenta calcula os pontos de casos de uso ajustados para o sistema 12. Ferramenta transforma a medida de pontos de casos de uso em medida de pontos por função 13. Ferramenta exibe as medidas de pontos de casos de uso ajustados e a medida de pontos por função do sistema	
Curso Alternativo	2a. Responsável cancela geração de PCU 2a1. Ferramenta exibe tela inicial 4a. Responsável não atribui peso a todos os atores 4a1. Ferramenta informa que o responsável deve atribuir peso a todos os atores do sistema 4a2. Voltar ao passo 4 8a. Responsável não atribui pontuação a todos os fatores do questionário 8a1. Ferramenta informa que o responsável deve atribuir pontuação a todos os fatores do questionário 8a2. Voltar ao passo 8 10a. Responsável não atribui pontuação a todos os fatores do questionário 10a1. Ferramenta informa que o responsável deve	

	atribuir pontuação a todos os fatores do questionário 10a2. Voltar ao passo 10
Evento Disparador	O Responsável seleciona a opção
Include	
Extend	-
Requisitos Funcionais	047, 048, 049
Requisitos Não-Funcionais	-

Exemplo de Especificação de Caso de Uso de Indicador de Rastreabilidade

Especificação do Caso de Uso	**Número:**	**11**
Nome do Caso de Uso	Visualizar indicador de estabilidade	
Descrição ou Resumo	Visualizar indicador de estabilidade para um determinado sistema	
Ator Participante	Responsável	
Ator Operador	Responsável	
Ator Genérico	-	
Pré-condição	-	
Curso Normal	1. Include: Listar sistemas 2. Responsável seleciona sistema cujo indicador de estabilidade deseja visualizar 3. Ferramenta exibe campos para data inicial e data final 4. Responsável informa data inicial e data final do período a ser calculado o indicador de estabilidade 5. Ferramenta calcula o indicador de estabilidade do sistema selecionado para o período informado 6. Ferramenta exibe o indicador de estabilidade do sistema no período informado	
Curso Alternativo	2a. Responsável cancela visualização do indicador de estabilidade 2a1. Ferramenta exibe tela inicial 4a. Responsável não informa data inicial e/ou data final 4a1. Ferramenta informa que o responsável deve informar a data inicial e a data final do período 4a2. Voltar ao passo 4	
Evento Disparador	O Responsável seleciona a opção	
Include		
Extend	-	
Requisitos Funcionais	051	
Requisitos Não-Funcionais	-	