UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA DE SUPORTE AO CÁLCULO DOS *USE CASE POINTS*

SANDRA CARLA GIELOW

BLUMENAU
2003

2003/2-37

SANDRA CARLA GIELOW

FERRA MENTA DE SUPORTE AO CÁLCULO DOS *USE CASE POINTS*

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Everaldo Artur Grahl - Orientador

BLUMENAU
2003

2003/2-37

FERRAMENTA DE SUPORTE AO CÁLCULO DOS

USE CASE POINTS

Por

SANDRA CARLA GIELOW

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Everaldo Artur Grahl, FURB
Membro:	Prof. Marcel Hugo, FURB
Membro:	Prof. Wilson Pedro Carli, FURB

Blumenau, 18 de fevereiro de 2004

A vocês que sempre compartilharam dos meus anseios e desalentos, vitórias e derrotas, alegrias e tristezas, incentivando-me a sonhar e lutar pelos meus sonhos, torcendo pelo meu sucesso, minha mais profunda admiração e gratidão. Obrigada minha mãe, meus irmãos e meu Pai onde quer que esteja.

AGRADECIMENTOS

À minha família, que sempre esteve presente mesmo nos momentos mais difíceis.

Agradeço Clodoaldo Tschöke, pela colaboração e a ajuda prestada nos momentos que precisei para a elaboração deste trabalho.

Ao meu orientador, Everaldo Artur Grahl, por ter acreditado na conclusão deste trabalho.

RESUMO

A modelagem de casos de uso é muito utilizada na análise orientada a objeto para capturar e descrever os requisitos do sistema. Estes por sua vez, podem servir como medição do tamanho e complexidade do sistema. Este trabalho tem como objetivo desenvolver uma ferramenta para cálculo dos *Use Case Points* a partir da leitura de diagramas gerados pela ferramenta CASE *Rational Rose*. Para implementação do trabalho foi utilizado o ambiente de desenvolvimento Visual C++ 6.0.

Palavras chaves: Métricas; *Use Case Points*; Casos de Uso.

ABSTRACT

In object oriented analysis the Use Cases modeling is very used to get and to describe system requirements and they are also good to verify the size and the complexity of the system. This work has as objective to develop a tool for calculation of the Use Case Points. I made software metrics studies observing Use Case Points mainly and to get data for calculation I use the Rational Rose tool case. For this work implementation was used Visual C++ 6.0 development set.

Key-Words: Metrics; Use Case Points; UseCase.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CASE – *Computer Aided Systems Engineering*

TCP/IP – *Transmission Control Protocol/Internet Protocol*

FTP – *File Transfer Protocol*

RUP – *Rational Unified Process*

API - *Application Programming Interface*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, com a dependência cada vez maior das organizações em relação a tecnologia da informação, a geração de produtos de software com qualidade e a um custo compensador, torna-se fator crítico de sucesso (FERNANDES, 1995). Conforme Pressman (2002), o rigor do dia-a-dia do trabalho do projeto de software deixa pouco tempo para raciocínio estratégico. Os gerentes de projeto de software estão preocupados em desenvolver estimativas significativas de projeto, produzir sistemas de alta qualidade e entregar produtos dentro do prazo. Com a crescente exigência dos consumidores pela qualidade, rapidez, comodidade, baixo custo de implantação e manutenção, é impossível não enxergar as métricas e estimativas de software como alavanca para um produto de melhor qualidade, com custos adequados. O ato de medir e estimar é a parte mais importante de um projeto de sistema bem-sucedido e alguns fatos, como a falta de maturidade, o desinteresse das empresas de desenvolvimento de sistemas e a baixa popularidade deste assunto entre os profissionais da área de informática são algumas das principais causas para o insucesso e o alto custo dos softwares.

Usando-se a medição para estabelecer um referencial do projeto, cada um desses aspectos torna-se mais gerenciável. Assim, tendo um referencial de métricas – uma base de dados que contém medições de produto e processo – engenheiros de software e gerentes de projeto podem ter melhor compreensão do trabalho e do software.

Atualmente, esta preocupação da medição é presente também no desenvolvimento de software orientado a objetos, mais particularmente nos diagramas de casos de uso. Segundo Freire (2003), a análise de sistemas orientados a objetos já utiliza, comumente, os diagramas de casos de uso para descrever as funcionalidades do sistema de acordo com a forma de utilização por parte dos usuários. A técnica de *Use Case Points* foi criada para permitir que seja possível estimar o tamanho do sistema ainda na fase de levantamento do projeto, utilizando-se dos próprios documentos gerados nesta fase de análise como subsídio para o cálculo. A forma de lançar uma estimativa é o principal diferencial da métrica por casos de uso. O método trata de estimar o tamanho de um sistema de acordo com o modo como os usuários o utilizarão, a complexidade de ações requeridas por cada tipo de usuário e uma análise em alto nível dos passos necessários para a realização de cada tarefa, em um nível muito mais abstrato que a técnica de pontos de função, muita usada pelas empresas em geral.

1.1 OBJETIVOS

O objetivo principal deste trabalho é desenvolver uma ferramenta de apoio ao cálculo do *Use Case Points.*

Os objetivos específicos do trabalho são:

a) capturar informações relacionadas aos casos de uso gerados pela ferramenta CASE *Rational Rose*;
b) disponibilizar relatório detalhando o cálculo do *Use Case Points.*

1.2 ESTRUTURA

Este trabalho é composto por seis capítulos.

O segundo capítulo apresenta conceitos e definições de métricas em software, além de descrever a Análise por Pontos de Função.

O terceiro capítulo apresenta a UML (*Unified Modeling Language*) e seus diagramas, detalhando mais o diagrama de casos de uso.

O quarto capítulo aborda informações específicas sobre os *Use Case Points*, demonstrando o seu cálculo.

O quinto capítulo descreve o desenvolvimento da ferramenta incluindo requisitos, especificação, implementação e o seu funcionamento.

O sexto capítulo apresenta as principais conclusões do trabalho.

2 MÉTRICAS EM SOFTWARE

Segundo Fernandes (1995), as métricas podem ser definidas como:

> Métodos para determinar, quantitativamente, a extensão em que o processo e o produto de software têm certos atributos. Isto inclui uma fórmula para determinar o valor da métrica como também sua forma de apresentação e as diretrizes de utilização e interpretação de resultados obtidos no contexto do ambiente de desenvolvimento.

Segundo Pressman (2002), as métricas em software, do ponto de vista da medição, podem ser divididas em duas categorias: medidas diretas e indiretas. Pode-se considerar como medidas diretas do processo de engenharia de software, o custo e o esforço aplicados no desenvolvimento e manutenção do software e do produto, a quantidade de linhas de código produzidas, velocidade de execução e o total de defeitos registrados durante um determinado período de tempo. As medidas indiretas do produto incluem funcionalidade, qualidade, complexidade, eficiência, confiabilidade e muitos outros aspectos do produto. As medidas indiretas são mais difíceis de serem medidas e avaliadas.

Outra divisão do domínio das métricas vista em Pressman (2002) são:

a) métricas orientadas ao tamanho: são medidas diretas do software e do processo por meio do qual ele é desenvolvido. Estas medidas são derivadas a partir de atributos de tamanho de software como linhas de código, esforço, custo e quantidade de documentação;
b) métricas orientadas a função: são medidas diretas do software e do processo através do qual o software é desenvolvido. Estas métricas concentram-se na funcionalidade ou qualidade do software. Um exemplo destas métricas é a técnica de Análise por pontos de função;
c) métricas orientadas às pessoas: estas métricas utilizam informações sobre a maneira pela qual as pessoas desenvolvem software e percepções humanas sobre a efetividade das ferramentas e métodos.

2.1 ORIGEM DAS MÉTRICAS

Segundo Möller (1995), a aplicação de métricas em software surgiu da necessidade de implementar qualidade e produtividade no desenvolvimento de software. A origem da aplicação de métodos quantitativos para o desenvolvimento de software foi estabelecida na década de 70. Houve quatro tendências na época, que foram evoluindo até chegarem nas práticas de métricas utilizadas atualmente:

a) medida da complexidade de código: em meados de 1970, os códigos métricos foram fáceis de se obter desde que fossem calculados pelo próprio código automatizado;
b) estimativa de custo de um projeto de software: esta técnica foi desenvolvida em meados de 1970, estimando o trabalho e o tempo gasto para se desenvolver um software, baseando-se além de outros fatores, no número de linhas de código necessário para implementação;
c) garantia da qualidade do software: estas técnicas foram melhoradas significativamente entre os anos de 1970 e 1980. Neste caso, se dá ênfase à identificação de informações faltantes, durante as várias fases do ciclo de vida do software;
d) processo de desenvolvimento do software: o projeto de software tornou-se grande e mais complexo sendo que a necessidade de se controlar este processo foi emergencial. O processo inclui a definição do ciclo de vida do software pela seqüência das fases, e mais ênfase na gerência de projetos de software com melhor controle deste recurso.

Na década de 80, estas quatro tendências impulsionaram a gerência de projetos de software por métodos quantitativos e os desenvolvedores de sistemas, a partir do conhecimento destas, começaram então a usar as métricas com o propósito de melhorar o processo de desenvolvimento de software.

2.2 IMPORTÂNCIA DA MEDIÇÃO

Segundo Fernandes (1995), o objetivo da aplicação de medição de software é fornecer aos gerentes e engenheiros de software um conjunto de informações tangíveis para planejar o

projeto, realizar estimativas, gerenciar e controlar os projetos com maior precisão. As necessidades de medição devem-se principalmente devido à realidade apontada nos itens descritos a seguir:

a) as estimativas de prazos, recursos, esforço e custo são realizadas com base no julgamento pessoal do gerente de projeto;
b) a estimativa do tamanho de software não é realizada;
c) a produtividade da equipe de desenvolvimento não é mensurada;
d) a qualidade dos produtos intermediários dos processos não é medida;
e) a qualidade do produto final não é medida;
f) o aperfeiçoamento da qualidade do produto ao longo de sua vida útil não é medido;
g) os fatores que impactam a produtividade e a qualidade não são determinados;
h) a qualidade do planejamento dos projetos não é medida;
i) os custos de não conformidade ou má qualidade não são medidos;
j) a capacidade de detecção de defeitos introduzidos durante o processo não é medida;
k) não há opções sistematizadas no sentido de aperfeiçoar o processo de desenvolvimento e gestão de software;
l) não há avaliação da satisfação dos usuários.

2.3 ANÁLISE POR PONTOS DE FUNÇÃO

Segundo Braga (1996), no final dos anos 70, Allan J. Albrecht começou analisar sistemas por solicitação do GUIDE (Grupo de Usuários IBM), buscando identificar os fatores críticos que determinassem o tamanho de um sistema e conseqüentemente uma estimativa do esforço gasto para desenvolvê-lo. Depois de analisar centenas de programas, desenvolveu os fundamentos da técnica de *Function Point Analysis* (FPA). A técnica de FPA mede uma aplicação através das funções desempenhadas para e por solicitação do usuário final, sendo baseado na visão do usuário. O FPA é independente da tecnologia e pode ser utilizada para estimativas. Esta técnica mede o que é o sistema e não como será, ou foi desenvolvido.

Até 1984, a técnica era usada para identificar o tamanho de um sistema e auxiliar na estimativa do esforço de desenvolvimento. Agora a utilização tem se dado através da implantação de programas de métricas para melhorar estimativas, gerenciar a qualidade

monitorar a produtividade, além de controlar indicadores financeiros, ou seja, melhorar todo o processo de desenvolvimento e manutenção de sistemas.

O objetivo principal da técnica de FPA é medir a funcionalidade de um software ou aplicativo, baseando-se primeiramente no desenho lógico e de acordo com a perspectiva do usuário. Esta técnica será detalhada a seguir, visto que foi a base para a criação dos *Use Case Points*, foco deste trabalho.

Segundo Fernandes (1995), o procedimento para a contagem de pontos de função inclui as seguintes etapas:

a) determinação do tipo de contagem;
b) identificação das fronteiras da contagem;
c) determinação dos pontos de função não ajustados;
d) determinação do fator de ajuste;
e) cálculo dos pontos de função ajustados.

2.3.1 DETERMINAÇÃO DO TIPO DE CONTAGEM

Segundo Braga (1996), o ponto de função pode estar associado com três tipos de contagem:

a) contagem de pontos de função em projetos de desenvolvimento: prevê a contabilização das funções identificadas no modelo lógico do sistema permitindo uma estimativa dos recursos de tempo e pessoal necessários ao desenvolvimento do sistema. São feitas várias medições durante o desenvolvimento do projeto, inclusive na implantação;
b) contagem de pontos de função em projetos de manutenção: consiste na medição das modificações que envolvem a inclusão, alteração e exclusão de funções;
c) contagem de pontos de função de uma aplicação: trata-se da técnica de pontos de função em um sistema já totalmente desenvolvido, também referenciado como pontos de função já instalados.

2.3.2 IDENTIFICAÇÃO DAS FRONTEIRAS DE CONTAGEM

Uma fronteira indica os limites entre a aplicação ou projeto que está sendo medido e as aplicações externas do domínio do usuário. A fronteira determina as funções que deverão ser medidas.

2.3.3 DETERMINAÇÃO DOS PONTOS DE FUNÇÃO NÃO AJUSTADOS

Segundo Braga (1996), a técnica divide as funções em dois tipos: dados e transações.

As funções de dados representam a funcionalidade proporcionada ao usuário para atender seus requisitos de dados internos e externos. Podem ser:

a) Arquivos Lógicos Internos (ALI): cada arquivo mestre lógico (grupo de dados lógico, que pode ser parte de uma base de dados maior ou um arquivo separados) é contado. Podem ser exemplificados por:
   - arquivos-mestres da aplicação;
   - tabelas criadas para atender a aplicação;
   - dados de segurança da aplicação;
   - dados de auditagem;
   - mensagens de auxílio (*help*);
   - mensagens de erro.

b) Arquivos de Interface Externa (AIE): todas as interfaces que são usadas para transmitir informação a outro sistema. Alguns exemplos:
   - arquivos-mestres de outras aplicações;
   - tabelas criadas para atender a outras aplicações;
   - mensagens de auxílio ou de erro criadas para atender a outras aplicações.

Funções transacionais representam a funcionalidade proporcionada aos usuários para processar os dados da aplicação. Podem ser definidos como:

a) Entrada Externa (EE): cada entrada do usuário, que fornece dados distintos orientados à aplicação do software, é contada. Entradas devem ser distinguidas de consultas, que são contadas separadamente. Pode ser exemplificada por:
   - entrada de dados *on-line*;
   - entrada de dados *batch*.

b) Saídas Externas (SE): cada saída do usuário, que fornece dados orientada à aplicação para o usuário, é contada. Nesse contexto, saída refere-se a relatório, telas, mensagens de erro, etc. Itens de dados individuais dentro de um relatório não são contados separadamente. Alguns exemplos:
   - dados transferidos para outra aplicação;
   - relatórios;
   - relatórios *on-line*;
   - gráficos gerados em forma de texto.

c) Consulta Externa (CE): uma consulta é definida como um a entrada *on-line*, que resulta na geração de alguma resposta imediata do software sob a forma de uma saída on-line. Cada consulta distinta é contada. Podem ser exemplificados por:
   - recuperação de dados;
   - consulta;
   - consultas de mensagens *help*.

Para o cálculo dos pontos não ajustados, deve-se identificar e enumerar as funções de aplicação, ou seja, determinar o número de ALI, AIE, EE, SE e CE. Posteriormente, classifica-se cada uma das funções conforme o nível de complexidade: simples, médio e complexo.

A avaliação da complexidade das funções de dados é determinada pelo número de Registros Lógicos Referenciados (RLR) e na quantidade de Dados Elementares Referenciados (DER). Um DER, é um campo único, reconhecido pelo usuário, que esteja presente em um ALI ou AIE. Um RLR é um subgrupo de elementos de dados, reconhecido pelo usuário, dentro de um ALI ou AIE.

A avaliação da complexidade das funções transacionais é determinada a partir da quantidade de Arquivos Lógicos Referenciados (ALR) e o número de DERs. Um ALR é um ALI lido ou mantido por um tipo de função e/ou um AIE lido por uma função. Um DER, neste caso, é um campo único reconhecido pelo usuário e tratado pelas funções transacionais.

As tabelas 1, 2, 3, 4 e 5 apresentam a determinação da complexidade das respectivas funções.

Tabela 1 – Complexidade de cada ALI

	1 a 19 DER	**20 a 30 DER**	**51 ou mais DER**
1 RLR	Simples	Simples	Médio
2 a 5 RLR	Simples	Médio	Complexo
6 ou mais RLR	Médio	Complexo	Complexo

Fonte: adaptado de Braga (1996).

Tabela 2 – Complexidade de cada AIE

	1 a 19 DER	**20 a 30 DER**	**51 ou mais DER**
1 RLR	Simples	Simples	Médio
2 a 5 RLR	Simples	Médio	Complexo
6 ou mais RLR	Médio	Complexo	Complexo

Fonte: adaptado de Braga (1996).

Tabela 3 – Complexidade de cada EE

	1 a 4 DER	**5 a 15 DER**	**16 ou mais DER**
0 ou 1 ALR	Simples	Simples	Médio
2 ALR	Simples	Médio	Complexo
3 ou mais ALR	Médio	Complexo	Complexo

Fonte: adaptado de Braga (1996).

Tabela 4 – Complexidade de cada SE

	1 a 5 DER	**6 a 19 DER**	**20 ou mais DER**
0 ou 1 ALR	Simples	Simples	Médio
2 a 3 ALR	Simples	Médio	Complexo
4 ou mais ALR	Médio	Complexo	Complexo

Fonte: adaptado de Braga (1996).

Tabela 5 – Complexidade de cada CE

	1 a 4 DER	**5 a 15 DER**	**16 ou mais DER**
0 ou 1 ALR	Simples	Simples	Média
2 ALR	Simples	Média	Complexo
3 ou mais ALR	Médio	Complexo	Complexo

Fonte: adaptado de Braga (1996).

A tabela 6 mostra um exemplo da determinação dos pontos de função não ajustados em uma aplicação, onde o peso é o fator padrão de multiplicação do número de ocorrências de determinada complexidade (simples, médio ou complexo) e o peso padrão da métrica.

Tabela 6 – Exemplo de determinação dos pontos de função não ajustados

			Fator de peso			
Função	**Contagem**		**Simples**	**Médio**	**Complexo**	**Resultado**
EE		**x**	3	4	6	
SE		**x**	4	5	7	
ALI		**x**	7	10	15	
AIE		**x**	5	7	10	
CE		**x**	3	4	6	
Contagem total						

2.3.4 DETERMINAÇÃO DO FATOR DE AJUSTE

Segundo Braga (1996), o cálculo de fator de ajuste é baseado em quatorze características gerais do sistema. Cada característica está associada a descrições que auxiliam na determinação do nível de influência de cada uma.

O cálculo do fator de ajuste é realizado a partir dos seguintes passos:

a) avaliar o impacto de cada uma das quatorze características gerais do sistema no aplicativo que está sendo contado, atribuindo peso de 0 a 5 para cada característica segundo o grau de influência;
b) calcular o nível de influência através da soma dos pesos de cada uma das quatorze características;
c) calcular o fator de ajuste através da equação conforme quadro 1, onde NI é o total do nível de influência.

Fator de ajuste = (NI * 0,01) + 0,65

Fonte: Braga (1996).

QUADRO 1 – Cálculo do fator de ajuste

Segundo Braga (1996), estas são as quatorze características gerais do sistema:

a) comunicação de dados;
b) funções distribuídas;
c) performance;
d) configuração do equipamento;
e) volume de transações;
f) entrada de dados *on-line*;
g) interface com o usuário;

h) atualização *on-line*;
i) processamento complexo;
j) reusabilidade;
k) facilidade de implantação;
l) facilidade operacional;
m) múltiplos locais;
n) facilidade de mudanças.

Cada característica geral do sistema deve ser avaliada em termos de sua influência, usando-se uma escala de zero (0) a cinco (5), conforme tabela 7.

Tabela 7 – Grau de influência para cada característica geral do sistema

Grau	**Descrições**
0	Nenhuma influência
1	Influência mínima
2	Influência moderada
3	Influência média
4	Influência insignificante
5	Influência forte

Fonte: Braga (1996).

2.3.5 CÁLCULO DOS PONTOS DE FUNÇÃO AJUSTADOS

O total de pontos de função da aplicação será obtido através da multiplicação do número de pontos de função não-ajustados pelo fator de ajuste. Mais informações sobre o cálculo do FPA podem ser obtidas em Braga (1996), Valcanaia (1998) e BFPUG (1998).

3 UML (*UNIFIED MODELING LANGUAGE*)

Segundo Tonsig (2003), para conseguir desenvolver um software capaz de satisfazer as necessidades de seus usuários, com qualidade, por intermédio de uma arquitetura sólida que aceite modificações, de forma rápida, eficiente, com o mínimo de desperdício e retrabalho, faz-se necessário o emprego de modelagem. Segundo Booch (2000) os modelos são construídos para que o sistema que será desenvolvido seja melhor compreendido. Construir o modelo de um sistema não é uma atividade fácil, são várias as abordagens a serem consideradas tais como a organização da empresa, os processos existentes ou requeridos, as informações existentes (ou requeridas) e os recursos envolvidos. Com a modelagem, pode-se alcançar quatro objetivos:

a) os modelos ajudam a visualizar o sistema como ele é ou como se deseja;
b) os modelos permitem especificar a estrutura ou o comportamento de um sistema;
c) os modelos proporcionam um guia para a construção do sistema;
d) os modelos documentam as decisões tomadas.

Existem limites para a capacidade humana de compreender complexidades, mais especificamente, reter todos os detalhes que envolvem uma realidade complexa, os relacionamentos existentes, as possíveis situações que possam ocorrer dependendo da combinação de cada aspecto envolvido. Com a ajuda da modelagem, é delimitado o problema, restringindo o foco num único aspecto por vez. Quanto mais complexo for o sistema, maior será a probabilidade de ocorrência de erros ou de construção de itens errados. Caso não se utilize nenhuma modelagem, pode-se esquecer de detalhes que irão comprometer o produto quando estiver sendo utilizado (TONSIG, 2003).

3.1 BREVE HISTÓRICO

Segundo Bezerra (2003), a construção da UML teve muitos contribuintes, mas os principais autores do processo foram Grady Booch, James Rumbaugh e Ivar Jacobson. No processo de definição da UML, procurou-se o melhor das características das notações preexistentes, principalmente das técnicas *Booch Method*, OMT (*Object Modeling Technique*) e OOSE (*Object-Oriented Software Engineering*), proposta pelos "três amigos", citados anteriormente. A notação definida para a UML é uma união de diversas notações

preexistentes, com alguns elementos removidos e outros elementos adicionados com o objetivo de torná-la mais expressiva. Finalmente, em 1997, a UML foi aprovada pelo OMG (*Object Management Group*). Desde então, a UML tem tido grande aceitação pela comunidade de desenvolvedores de sistemas.

3.2 CONCEITOS

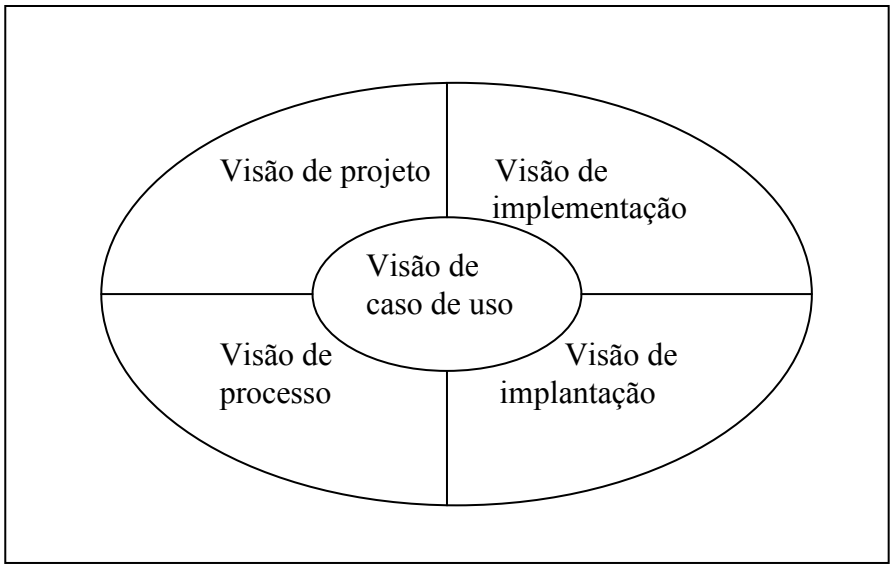
Segundo Booch (2000), a UML é uma linguagem-padrão para elaboração da estrutura de projetos de software. A UML poderá ser empregada para visualização, especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software. A UML é apenas uma linguagem e portanto, é somente uma parte de um método para desenvolvimento de software.

Segundo Tonsig (2003), a UML tem como objetivo prover as necessidades de desenvolvedores de software com uma linguagem de modelagem visual completa, buscando atingir os seguintes aspectos:

a) disponibilização de mecanismos de especificações que possam expressar os níveis conceituais;
b) independência de processos de desenvolvimento e linguagens de programação;
c) incentivo do crescimento das aplicações desenvolvidas no conceito da orientação a objetos;
d) permissão de suporte a conceitos de desenvolvimento de alto nível, tais como *frameworks*, padrões e componentes.

Segundo Booch (2000), visualizar, especificar, construir e documentar sistemas complexos de software são tarefas que requerem a visualização desses sistemas sob várias perspectivas. Vários participantes – usuários finais, analistas, desenvolvedores, integradores de sistemas, o pessoal que testa os sistemas, escritores técnicos e gerentes de projetos – cada um traz contribuições próprias ao projeto e observa o sistema de maneira distinta em momentos diferentes ao longo do desenvolvimento do projeto. A arquitetura do sistema talvez seja o artefato mais importante a ser utilizado com o objetivo de gerenciar esses diferentes pontos de vistas e assim, tornar possível um controle do desenvolvimento iterativo e incremental de um sistema durante seu ciclo de vida.

Conforme mostra a figura 1, a arquitetura de sistema complexo de software pode ser descrita mais adequadamente por cinco visões interligadas. Cada visão constitui uma projeção na organização e estrutura do sistema, cujo foco está voltado para determinado aspecto desse sistema.



Fonte: Bezerra (2003).

FIGURA 1 – Visões de um sistema de software

A visão do caso de uso abrange os casos de uso que descrevem o comportamento do sistema conforme é visto pelos seus usuários finais, analistas e pessoal do teste. Essa visão não especifica realmente a organização do sistema de um software. Porém, ela existe para especificar as forças que determinam a forma da arquitetura do sistema. Com a UML, os aspectos estáticos dessa visão são capturados em diagramas de caso de uso, enquanto os aspectos dinâmicos são capturados em diagramas de interação, diagramas de gráfico de estados e diagramas de atividades.

A visão de projeto de um sistema abrange as classes, interfaces e colaborações que formam o vocabulário do problema e de sua solução. Essa perspectiva proporciona principalmente um suporte para os requisitos funcionais do sistema, ou seja, os serviços que o sistema deverá fornecer a seus usuários finais. Com a UML, os aspectos estáticos dessa visão são capturados em diagramas de classes e de objetos; os aspectos dinâmicos são capturados em diagramas de interações, diagramas de estados e diagramas de atividades.

A visão do processo abrange as *threads* e os processos que formam os mecanismos de concorrência e de sincronização do sistema. Essa visão cuida principalmente de questões referentes ao desempenho, à escabilidade e ao *through-put* do sistema. Com a UML, os aspectos estáticos e dinâmicos dessa visão são capturados nos mesmos tipos de diagramas da visão de projeto, mas com o foco voltado para as classes ativas que representam *threads* e processos.

A visão de implementação de um sistema abrange os componentes e os arquivos utilizados para a montagem e fornecimento do sistema físico. Essa visão envolve principalmente o gerenciamento da configuração das versões do sistema, composta por componentes e arquivos de alguma maneira independentes, que podem ser reunidos de diferentes formas para a produção de um sistema executável. Com a UML, os aspectos estáticos dessa visão são capturados em diagramas de interações, de estados e de atividades.

A visão de implantação de um sistema abrange os nós que formam a topologia de hardware em que o sistema é executado. Essa visão direciona principalmente a distribuição, o fornecimento e a instalação das partes que constituem o sistema físico. Com a UML, os aspectos estáticos dessa visão são capturados em diagramas de implantação; os aspectos dinâmicos são capturados em diagrama de interações, de estados e diagramas de atividades.

Cada uma dessas cinco visões pode ser considerada isoladamente, permitindo que diferentes participantes dirijam seu foco para os aspectos da arquitetura do sistema que mais interessam. Essas cinco visões também interagem entre si – os nós da visão da implantação contêm componentes da visão de implementação que, por sua vez, representa a realização física de classes, interfaces, colaborações e classes ativas provenientes das visões de projeto e de processo.

3.3 DIAGRAMAS DA UML

Segundo Furlan (1998), o modo para descrever os vários aspectos de modelagem pela UML é através da notação definida pelos seus vários tipos de diagramas. Um diagrama é uma apresentação gráfica de uma coleção de elementos de modelo, freqüentemente mostrado como um gráfico conectado de arcos (relacionamentos) e vértices (outros elementos do modelo).

Cada visão é descrita em um número de diagramas que contém informação enfatizando um aspecto particular do sistema. Analisando-se o sistema através de visões diferentes, é possível se concentrar em um aspecto de cada vez .

Os diagramas propostos pela UML são:

a) diagrama de classes: exibe conjunto de classes, interfaces e colaborações, bem como seus relacionamentos. Esses diagramas são encontrados com maior freqüência em sistemas de modelagem orientados a objeto e abrangem uma visão estática da estrutura do sistema. Os diagramas de classes incluem classes ativas direcionam a perspectiva do processo estático do sistema;
b) diagrama de objetos: exibe um conjunto de objetos e seus relacionamentos. Representa retratos estáticos de instâncias de itens encontrados em diagramas de classes. São diagramas que abrangem a visão estática da estrutura ou processo de um sistema, como ocorre nos diagramas de classes, mas sob perspectiva de casos reais ou de protótipos;
c) diagrama de casos de uso: exibe um conjunto de caso de uso e atores (um tipo especial de classe) e seus relacionamentos. Diagramas de caso de uso abrangem a visão estática de casos de uso do sistema. Esses diagramas são importantes principalmente para a organização e a modelagem de comportamentos do sistema;
d) diagrama de interação: exibe uma interação, consistindo de um conjunto de objetos e seus relacionamentos, incluindo as mensagens que podem ser trocadas entre eles. Diagramas de interação abrangem a visão dinâmica do sistema. Composto por diagrama de seqüências e diagrama de colaborações descritos a seguir;
e) diagrama de seqüências: é um diagrama de interação cuja ênfase está na organização estrutural dos objetos que enviam e recebem mensagens;
f) diagrama de colaborações: é um diagrama de interação cuja ênfase está na organização estrutural dos objetos que enviam e recebem mensagens;
g) diagrama de estados: exibem uma máquina de estados, formada por estados, transições, eventos e atividades. Os diagramas de estados abrangem a visão dinâmica de um sistema. São importantes principalmente para a modelagem de comportamentos de uma interface, classe ou colaboração e para dar ênfase a comportamentos de um objeto ordenados por eventos;

h) diagrama de atividade: é um tipo essencial de diagrama de gráficos de estado, exibindo o fluxo de uma atividade para outra no sistema. Abrange a visão dinâmica do sistema e é importante principalmente para a modelagem da função de um sistema e dá ênfase ao fluxo de controle entre objetos;
i) diagrama de componente: exibe as organizações e as dependências existentes em um conjunto de componentes. Abrange a visão estática da implementação de um sistema. Está relacionado aos diagramas de classes, pois tipicamente os componentes são mapeados para uma ou mais classes, interfaces ou colaborações;
j) diagrama de implantação: mostra a configuração dos nós de processamento em tempo de execução e os componentes neles existentes. Abrange a visão estática do funcionamento de uma arquitetura diagramas de implantação. Está relacionado aos diagramas de componentes, pois tipicamente um nó inclui um ou mais componentes.

Cada tipo de diagrama captura uma perspectiva diferente e um determinado elemento poderá existir em múltiplos diagramas, embora haja apenas uma definição daquele elemento no modelo subjacente. A seguir será detalhado o diagrama de casos de uso, visto que o mesmo é a base para o *Use Case Points*.

3.3.1 DIAGRAMA DE CASOS DE USO

Segundo Booch (2000), os diagramas de casos de uso são importantes para visualizar, especificar e documentar o comportamento de um elemento. Esses diagramas fazem com que sistemas, subsistemas e classes fiquem acessíveis e compreensíveis, por apresentarem uma visão externa sobre como esses elementos podem ser utilizados no contexto geral.

É importante que na modelagem a ser construída sejam especificados os limites do sistema, definidos pela funcionalidade que é exigida do software. A funcionalidade de todo o sistema é representada por um conjunto de casos de uso. Cada caso de uso por sua vez deve ser extensivamente avaliado, para encontrar todas as possíveis situações de uso daquela funcionalidade que está sendo modelada (TONSIG, 2003).

Segundo Tonsig (2003), no momento da construção do diagrama, não deve se ter a preocupação quanto aos aspectos de implementação, visto que os principais objetivos da representação são:

a) captar a funcionalidade necessária para a resolução dos problemas existentes, sob o ponto de vista do cliente ou usuários;
b) mostrar uma visão funcional coesa sobre tudo o que o software deverá fazer, pois esse diagrama será a base para todo o processo de desenvolvimento;
c) deverá ser aplicado para testes de validação, ou seja, verificar se o software quando pronto, realmente possui funcionalidade inicialmente planejada;
d) propiciar facilidades para a transformação dos requisitos funcionais em classes e operações reais do software.

O modelo de casos de uso de um sistema é composto de casos de uso, de atores e de relacionamentos entre estes. Nas próximas seções, esses componentes serão descritos separadamente.

3.3.1.1 CASOS DE USO

Segundo Booch (2000), um caso de uso descreve um conjunto de seqüências, cada uma representando a interação de itens externos ao sistema (seus atores) com o próprio sistema (e suas principais abstrações). Um caso de uso, conforme figura 2, é a descrição de um conjunto de seqüências de ações, inclusive variantes, que um sistema executa para produzir um resultado de valor observável por um ator.

FIGURA 2 – Exemplo de caso de uso

3.3.1.2 ATORES

Na UML, qualquer elemento externo que interage com o sistema é denominado ator. O termo “externo” indica que atores não fazem parte do sistema. O termo “interage” significa

que um ator troca (envia e/ou recebe) informações com o sistema. As categorias de atores, junto com exemplos de cada uma, são listadas a seguir:

a) pessoas: empregado, cliente, gerente, almoxarife, vendedor, etc;
b) organizações: empresa fornecedora, agência de impostos, administradora de cartões;
c) outros sistemas: sistema de cobrança, sistema de estoque de produtos;
d) equipamentos: leitora de código de barras, sensor.

Um ator pode participar de muitos casos de uso. Do mesmo modo, um caso de uso pode envolver vários atores, o que resulta na classificação dos atores em primários ou secundários. Um ator primário é aquele que inicia uma seqüência de interações de caso de uso. São os agentes externos para os quais o caso de uso tem benefício direto. As funcionalidades principais do sistema são definidas tendo em mente os objetivos dos atores primários. Atores secundários supervisionam, operam, mantêm ou auxiliam na utilização do sistema. Um exemplo de ator é representado na figura 3.

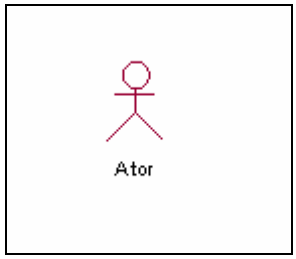


FIGURA 3 – Exemplo de ator

3.3.1.3 RELACIONAMENTOS

Segundo Bezerra (2003), os casos de uso e atores não existem sozinhos. Por exemplo, um ator deve estar relacionado a um ou mais casos de uso de um sistema. A UML define diversos tipos de relacionamentos no modelo de casos de uso:

a) relacionamento de comunicação: representa a informação de quais atores estão associados a que casos de uso, exemplificado na figura 4. O fato de um ator estar associado a um caso de uso significa que esse ator interage (troca informações) com o sistema. Um ator pode se relacionar com mais de um caso de uso do sistema.

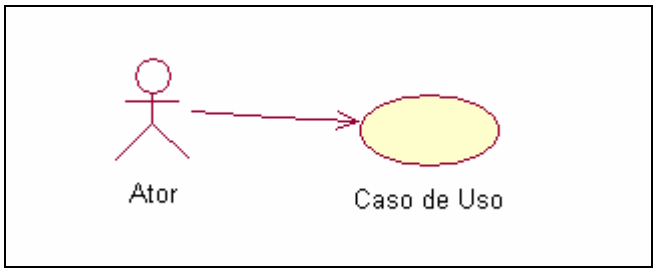


FIGURA 4 – Exemplo de relacionamento de comunicação

b) relacionamento de inclusão: existe somente entre casos de uso. Para entender o princípio deste tipo de relacionamento, é útil uma analogia com um conceito de linguagens de programação: a rotina. Quando dois ou mais casos de uso incluem uma seqüência comum de interações, essa seqüência comum deve ser descrita em um outro caso de uso. A partir daí, os casos de uso do sistema podem usar esse caso de uso comum. Isso evita a descrição de uma mesma seqüência de interações mais de uma vez e, tornando assim, a descrição dos casos de uso como um todo mais simples, conforme apresentado na figura 5. Neste tipo de relacionamento deve-se usar o estereótipo <<inclui>>.

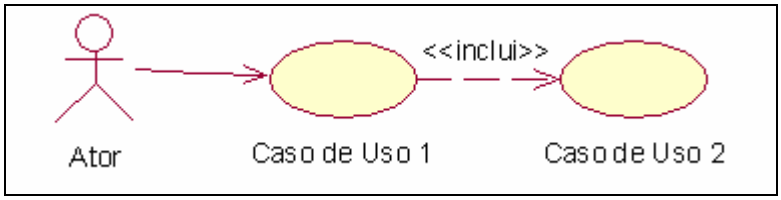


FIGURA 5 – Exemplo de relacionamento de inclusão

c) relacionamento de extensão: utilizado para modelar situações em que diferentes seqüências de interações podem ser inseridas em um caso de uso, chamado caso de uso estendido, exemplificado na figura 6. Cada uma dessas diferentes seqüências representa um comportamento opcional, ou seja, um comportamento que só ocorre sob certas condições, ou cuja realização depende da escolha de um ator. Quando um ator opta por executar a seqüência de interações definida no extensor, este é executado. Após sua execução, o fluxo de interações volta ao caso de uso estendido, recomeçando logo após o ponto em que o extensor foi inserido. Neste tipo de relacionamento deve-se usar o estereótipo <<estende>>.

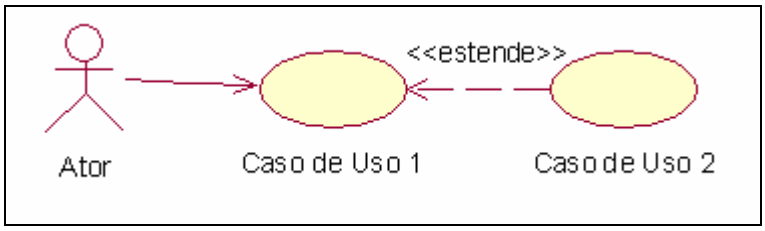


FIGURA 6– Exemplo de relacionamento de extensão

d) relacionamento de generalização: este relacionamento permite que um caso de uso (ou um ator) herde características de um caso de uso (ator) mais genérico. Este último normalmente chamado de caso de uso (ator) base. O caso de uso (ator) herdeiro pode especializar o comportamento do caso de uso (ator) base. O relacionamento de generalização pode existir entre dois casos de uso ou entre dois atores, conforme figura 7.

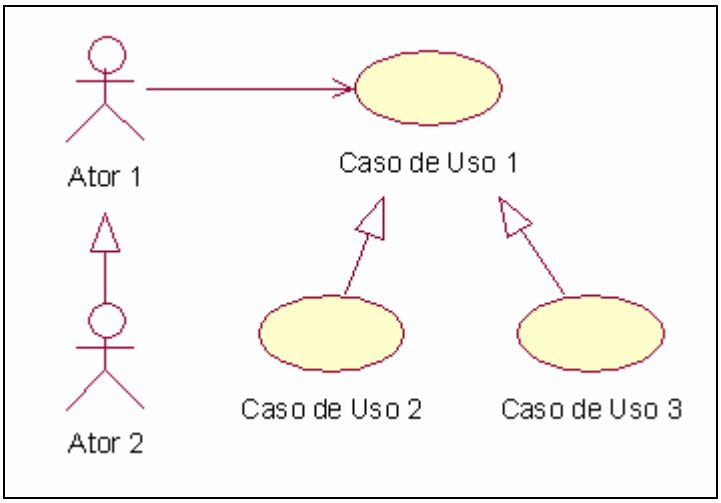


FIGURA 7 – Exemplo de relacionamento de generalização

4 USE CASE POINTS

Inúmeras vezes, a estimativa de prazos de entrega de projetos são adiadas. Muitos utilizam a prática pessoal para chegar aos valores das estimativas. Quanto mais desenvolvimento de software alguém fizer, mais exatas serão as estimativas. Independente do nível de habilidade, é necessária uma melhor abordagem para levar em conta as tendências pessoais bem como aqueles projetos que não têm um pessoal com grande experiência.

Aplicar a análise de pontos por função aos casos de uso é um relacionamento natural. Os pontos de função continuam sendo o método mais comumente usado para medição do software. Os casos de uso estão sendo utilizados com maior freqüência para descrever os requisitos de software. Os pontos de função são um método para medição dos requisitos do software e os casos de uso é uma linguagem para desenvolver esses requisitos (LONGSTREET, 2001).

Ambos os pontos de função e os casos de uso tentam ser independentes da tecnologia usada para executar a solução do software. Os casos de uso são usados para assegurar que todos os requisitos serão atendidos. Outro benefício dos casos de uso é que são definidos cedo no ciclo de vida do software. Se os casos forem definidos e medidos corretamente, as estimativas do projeto serão mais exatas. Desde que os pontos de função tenham bases históricas grandes, pode-se comparar a produtividade de utilizar métodos de caso de uso com outros métodos.

4.1 BREVE HISTÓRICO

Segundo Banerjee (2001), em 1993, a métrica por pontos de caso de uso foi desenvolvida por Gustav Karner da Objectory. A *Rational Software* adquiriu a *Objectory* de Ivar Jacobson em 1995. Juntamente, foi adquirida esta pesquisa, a estimativa homens/hora para projetos baseados em casos de uso. Embora esse trabalho tenha sido baseado em um trabalho anterior de Albrecht que envolvia o uso da análise de pontos por função, este trouxe novas descobertas usando artefatos que eram derivados diretamente dos casos de uso.

O método é uma extensão da análise por pontos de função e MK II, uma adaptação de FPA. O trabalho de Karner – métrica por pontos de caso de uso - foi escrito para uma tese na Universidade de Linkoping.

4.2 PROCESSO DE CONTAGEM DOS *USE CASE POINTS*

Segundo Freire (2003), após levantados os casos de usos principais do sistema, pode-se então estimar o tamanho do software como um todo baseando-se em um conjunto simples de métricas similares a técnica de FPA. É importante lembrar que o grau de detalhes dos casos de uso analisados influenciam diretamente na qualidade final da medição do software, devendo haver uma preocupação em medir somente casos de uso em um nível de sistema, onde seja possível diferenciar transações e interações com o usuário.

O processo envolve quatro processos:

a) avaliação dos atores;
b) avaliação dos casos de uso;
c) cálculo dos fatores de ajuste;
d) cálculo final dos *Use Case Points*.

4.2.1 AVALIAÇÃO DOS ATORES

O primeiro passo é determinar o número dos atores no sistema para se calcular o *Unajusted Actor Weight* (UAW). Segundo Freire (2003), Banerjee (2001) e Anda (2001), classificam-se os atores envolvidos em cada caso de uso, como simples, médio ou complexo conforme mostra a tabela 8.

Um ator simples representa outro sistema acessado através de uma API de programação. Um ator médio é quando outro sistema interage através de um protocolo de comunicação (tais como TCP/IP, FTP). Um ator complexo, significa que um usuário está interagindo através de uma interface gráfica ou uma página *Web*.

Tabela 8 – Peso dos atores

Tipo de ator	Peso	Descrição
Ator simples	1	Outro sistema acessado através de uma API de programação
Ator médio	2	Outro sistema interagindo através de um protocolo de comunicação, como TCP/IP
Ator complexo	3	Um usuário interagindo através de uma interface gráfica

Fonte: Freire (2003).

Determinado o número de atores em cada tipo, multiplica-se o total de número de atores pelo peso correspondente. Um ator simples tem peso 1, um ator médio possui peso 2, e um ator complexo possui peso 3. Logo, o UAW total é calculado pela soma dos produtos do cálculo de cada tipo de ator.

4.2.2 AVALIAÇÃO DOS CASOS DE USO

Uma vez calculado o peso dos atores do sistema, parte-se para o cálculo inicial do *Unajusted Use Case Weight* (UUCW). Para fins de cálculo classificam-se os casos de uso em três níveis de complexidade, simples, médio e complexo, de acordo com o número de transações envolvidas em seu processamento conforme a tabela 9. Por transação, entende-se como uma série de processos que devem, garantidamente, ser realizados em conjunto – ou cancelados em sua totalidade, caso não seja possível concluir o processamento (FREIRE, 2003).

Segundo Banerjee (2001 apud KARNER, 1993), propõe que os casos de uso de relacionamento de inclusão e extensão não devam ser considerados, mas não comenta claramente o motivo.

Tabela 9 – Peso dos casos de uso por número de transações

Tipo de caso de uso	Número de transações	Peso
Simples	Até 3	1
Médio	4 a 7	2
Complexo	Mais de 7	3

Fonte: Freire (2003).

O cálculo do UUCW é realizado como no cálculo de peso dos atores, soma-se os produtos da quantidade de casos de uso classificados em cada tipo pelo peso correspondente.

Segundo Freire (2003), uma outra maneira de se calcular o peso dos casos de uso do sistema é levar em consideração o número de classes envolvidas no processo, no lugar das transações.

O cálculo e os pesos para cada tipo de caso de uso permanecem da mesma forma que na abordagem anterior. Um caso de uso simples possui 5 ou menos classes, caso de uso médio possui de 5 a 10 classes, e um caso de uso complexo possui mais de 10 classes, conforme mostrado na tabela 10. Cada tipo de caso de uso é multiplicado pelo seu respectivo peso e o somatório destes produtos resultará no UUCW. Existem outros meios para o cálculo do UUCP, pode ser consultado em Banerjee (2001).

Tabela 10 – Peso dos casos de uso por número de classes

Tipo de caso de uso	**Número de classes**	**Peso**
Simples	Menos de 5	1
Médio	5 a 10	2
Complexo	Mais de 10	3

Fonte: Freire (2003).

Após determinado o UAW e UUCW, pode-se então calcular o *Unajusted Use Case Points* (UUCP), conforme fórmula do quadro 2.

UUCP = UAW + UUCW

Fonte: Freire (2003).

QUADRO 2 – Cálculo do UUCP

4.2.3 CÁLCULO DOS FATORES DE AJUSTE

O método de ajuste é similar ao adotado pela técnica de pontos de função, constituído de duas partes, um cálculo de fatores técnicos, cobrindo uma série de requisitos funcionais do sistema; e um cálculo de fatores de ambiente, requisitos não-funcionais associados ao processo de desenvolvimento – tais como experiência da equipe, motivação e estabilidade do projeto (FREIRE, 2003).

4.2.3.1 FATORES TÉCNICOS

Para o cálculo do *Technical Complexity Factor* (TCF), utiliza-se a tabela 11. Deve-se atribuir um valor a cada fator técnico (T1 a T13) em uma escala de 0 a 5, conforme tabela 7 vista anteriormente, indicando a influência do requisito no sistema.

Tabela 11 – Peso dos fatores técnicos

Fator	**Requisito**	**Peso**
T1	Sistema distribuído	2
T2	Tempo de resposta	2
T3	Eficiência	1
T4	Processamento complexo	1
T5	Código reusável	1
T6	Facilidade de instalação	0.5
T7	Facilidade de uso	0.5
T8	Portabilidade	2
T9	Facilidade de mudança	1
T10	Concorrência	1
T11	Recursos de segurança	1
T12	Acessível por terceiros	1
T13	Requer treinamento	1

Fonte: Freire (2003).

Este fator é calculado conforme a seguinte fórmula descrita no quadro 3. O TFactor, é o somatório dos produtos do valor atribuído a cada fator técnico pelo respectivo peso.

TCF = 0.6 + (0.01 x TFactor)

Fonte: Freire (2003).

QUADRO 3 – Cálculo dos fatores técnicos

4.2.3.2 FATORES AMBIENTAIS

A tabela 12 mostra os fatores ambientais previstos pela metodologia dos pontos de caso de uso e seus pesos associados.

Tabela 12 – Peso dos fatores ambientais

Fator	Descrição	Peso
E1	Familiaridade com RUP ou processo formal	1.5
E2	Experiência com a aplicação em desenvolvimento	0.5
E3	Experiência em orientação a objetos	1
E4	Presença de analista experiente	0.5
E5	Motivação	1
E6	Requisitos estáveis	2
E7	Desenvolvedores em meio-expediente	-1
E8	Linguagem de programação difícil	2

Fonte: Freire (2003).

No caso dos fatores ambientais, o nível de influência indica o nível de disponibilidade de cada recurso no decorrer do projeto. Desta forma, determinar que um dado fator tem nível de influência zero (nenhuma influência) a um fator que indica que o mesmo não está presente no processo de desenvolvimento. Considere o seguinte contexto:

a) para os quatro primeiros fatores (E1 a E4), 0 significa nenhuma experiência no assunto, 3 significa experiência média e 5 significa especialista no assunto;
b) para o quinto fator (E5), 0 significa nenhuma motivação para o projeto, 3 significa motivação média e 5 significa motivação alta;
c) para o sexto fator (E6), 0 significa requisitos extremamente instáveis, 3 significa requisitos médios e 5 significa requisitos inalterados;
d) para o sétimo fator (E7), 0 significa nenhum pessoal técnico em tempo parcial, 3 significa médio e 5 significa todo o pessoal técnico em tempo parcial;
e) para o oitavo fator (E8), 0 significa linguagem de programação fácil de usar, 3 significa média e 5 significa linguagem de programação difícil de usar.

A título de ilustração pode-se dizer que, um grau de influência mínimo (0) atribuído ao fator E3 indica uma equipe com total desconhecimento de orientação a objetos – enquanto o grau máximo (5) indica a disponibilidade de uma equipe experiente neste paradigma de desenvolvimento.

O *Environmental Factor* (EF) é calculado pela fórmula conforme descrita no quadro 4.

EF = 1.4 + (-0.03 x EFactor)

Fonte: Freire (2003).

QUADRO 4 – Cálculo dos fatores ambientais

Onde o valor de EFactor é dado pela soma dos produtos entre o peso de cada fator (E1 a E8) e seu grau de influência atribuído.

Na maioria dos casos, os fatores ambientais tendem a diminuir o valor em pontos de caso de uso do sistema: isto reflete o ganho de velocidade proporcionado pelos diversos fatores ambientais descritos na tabela, quando os mesmos encontram-se disponíveis.

Tanto o cálculo dos fatores técnicos e fatores ambientais são considerados mais subjetivos assim como ocorre no cálculo das características gerais do sistema no cálculo do FPA.

4.2.4 CÁLCULO FINAL DOS *USE CASE POINTS*

Finalmente, pode-se calcular o valor total do sistema em *Use Case Points* (UCP) utilizando-se a seguinte fórmula apresentada no quadro 5:

UCP = UUCP x TCF x EF

Fonte: Freire (2003)

QUADRO 5 – Cálculo dos *Use Case Points*

Conforme Freire (2003 apud KARNER, 1993), em sua pesquisa, estima-se que o tempo necessário para o desenvolvimento de um projeto seja em média de 20 horas de trabalho por ponto de caso de uso (UCP), sendo que experiências demonstram uma variação entre 15 e 30 horas por ponto de caso de uso.

Outros autores sugerem um refinamento na técnica de Karner. A técnica sugere que a presença de certos atributos influencia diretamente na média de horas por ponto calculado. Sugere-se que conte a quantidade de fatores técnicos entre T1 a T6 que receberam nível de influência maior que 3; e deve-se somar o valor obtido a quantidade de fatores ambientais entre E7 e E8 que receberam valor de influência menor que 3.

Neste caso, os autores da técnica sugerem que o projeto seja revisto: talvez haja a necessidade de treinamento pessoal, adequação de tecnologia ou revisão de requisitos, para garantir um melhor aproveitamento de recursos e redução no cronograma previsto. O somatório indica a quantidade sugerida de horas por ponto de caso de uso a ser adotada no projeto, sendo a média sugerida de:

a) 20 horas por ponto, para um resultado de 2 ou menor;
b) 28 horas por ponto, caso o somatório resulte em 3 ou 4;
c) 36 horas por ponto, para valores maiores que 4.

4.3 EXEMPLO DE CÁLCULO

Inicialmente, faz-se avaliação dos atores. A classificação se baseia no fato do ator ser considerado simples, médio ou complexo. Para tornar o cálculo será usado um estudo de caso de um controle de pesca conforme quadro 6. O diagrama de caso de uso, contendo somente os casos de uso primários, pode ser observado na figura 8. Os diagramas de seqüência deste estudo de caso, estão descritos na figura 9, 10 e 11. O diagrama de classes, utilizado na elaboração dos diagramas de seqüência, é mostrado na figura 12.

Uma entidade ambientalista decidiu criar um banco de dados sobre as pescas realizadas na sua região de atuação, a fim de disponibilizar dados de interesse dos pescadores, entidades de pescadores e a comunidade em geral. Foi realizado um censo onde foram coletadas as seguintes informações: Dados sobre embarcações : proprietário, nome da embarcação, comprimento, inscrição na capitania dos portos, ano construção. Sobre o proprietário são cadastrados o nome, endereço, CPF, apelido e município; As embarcações podem ser para pesca artesanal ou industrial. Quando for barco para pescaria industrial, devem ser armazenados ainda dados como a capacidade de estocagem e se a embarcação possui tanque de isca. No caso da pesca artesanal devem ser informados ainda o tipo de material do casco e o tipo de propulsão do barco (motor, vela, remo, etc.); Para cada embarcação serão armazenados os diversos tipos de petrechos de pesca utilizados (rede, caniço, etc.) e também o tipo de conservação do pescado (refrigerado, sem refrigeração , etc.) Cabe aos pesquisadores o cadastramento das espécies de animais encontrados na área marítima considerada pelo sistema. Sobre cada espécie é anotado código, nome científico e nome popular. Os fiscais vão informar os dados coletados sobre as pescas, que foram anotados nos pontos de desembarque, registrando a data, hora, embarcação e para cada espécie capturada, será registrada a quantidade em quilos obtida e o petrecho utilizado para sua captura.Sobre as pescas realizadas (desembarques), o sistema disponibilizará aos usuários em geral dois relatórios mensais: quantidade total pescada por espécie e embarcações com maior quantidade de pesca (quilos) utilizando rede.

QUADRO 6 – Estudo de caso “Controle de Pesca”

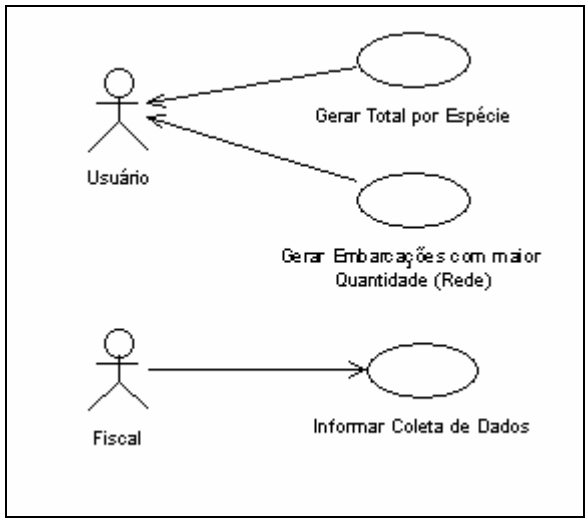


FIGURA 8 – Exemplo do diagrama de caso de uso do estudo de caso

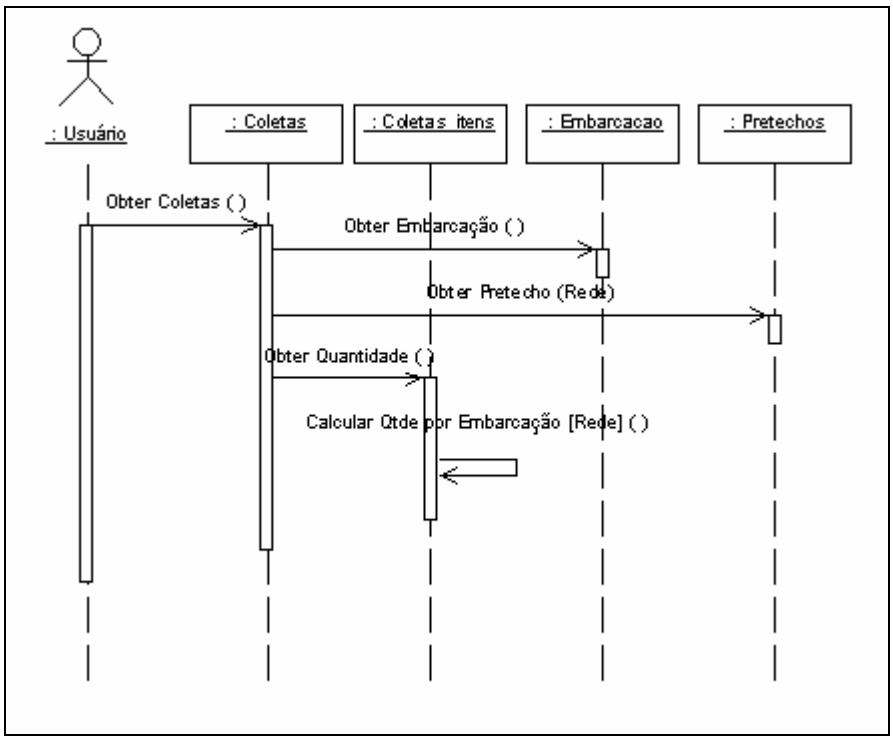


FIGURA 9 – Exemplo do diagrama de gerar embarcações

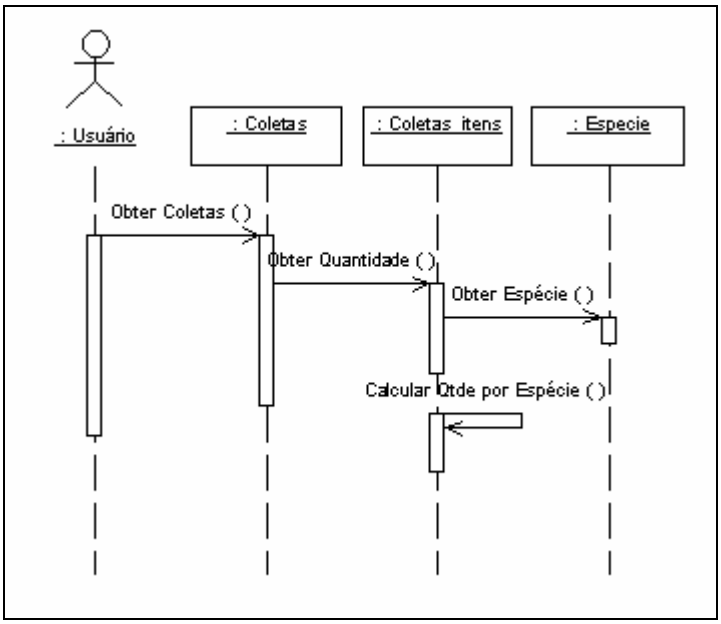


FIGURA 10 – Exemplo do diagrama de seqüência total por espécie

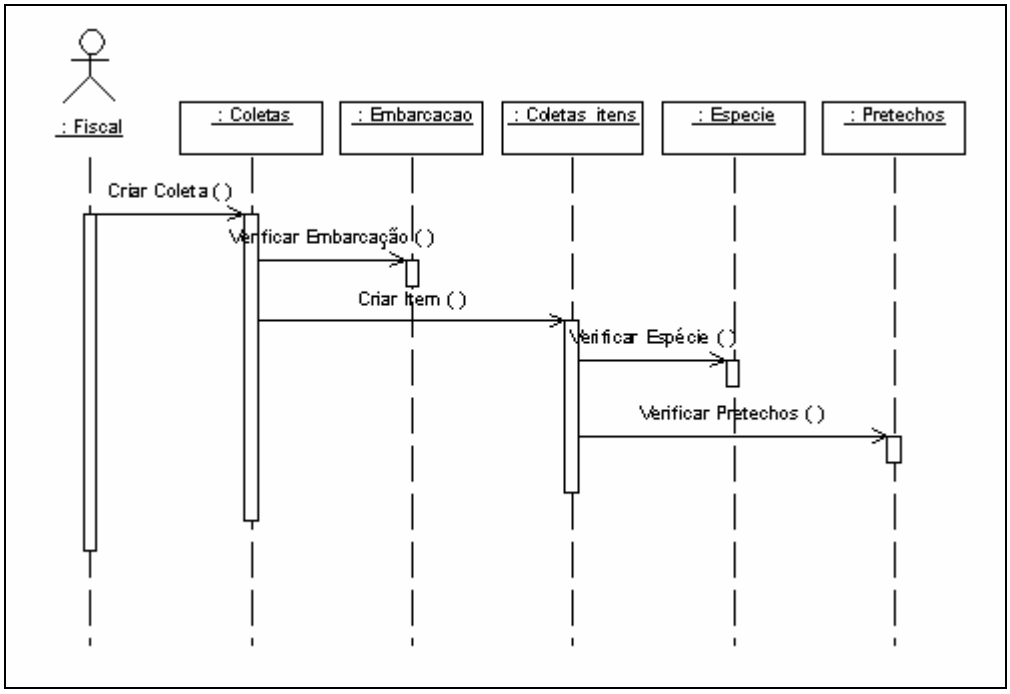


FIGURA 11 – Exemplo do diagrama de seqüência coleta de dados

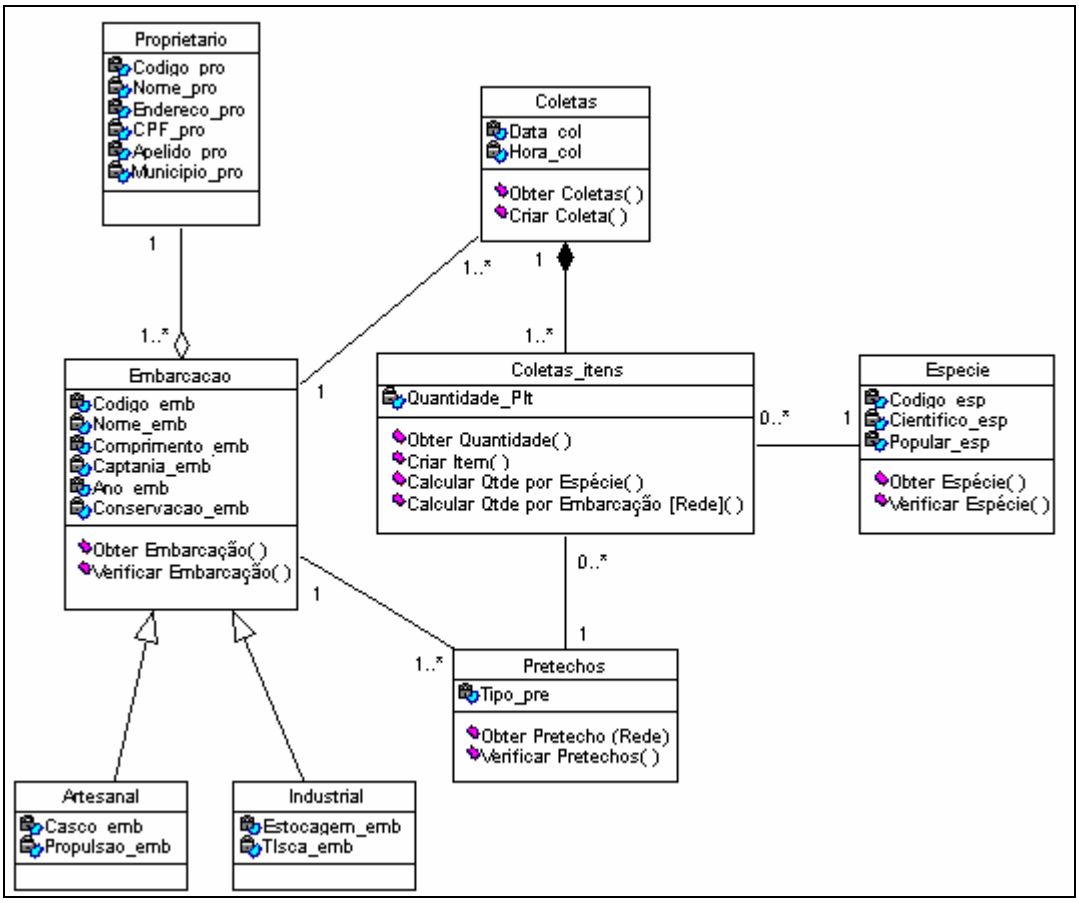


FIGURA 12 – Exemplo do diagrama de classes do estudo de caso

Neste estudo de caso, tem-se dois atores (fiscal e usuário) que são classificados como complexos, como mostrado na tabela 13, onde tem-se um exemplo do cálculo do UAW.

Tabela 13 – Exemplo do cálculo da avaliação dos atores

Ator	**Tipo de ator**	**Peso**
Fiscal	Complexo	3
Usuário	Complexo	3
	Total UAW:	**6**

O próximo passo é a avaliação dos casos de uso. Classificam-se os casos de uso conforme a tabela 10. Um exemplo do UUCW é descrito na tabela 14.

Tabela 14 – Exemplo do cálculo da avaliação dos casos de uso

Caso de uso	Tipo de caso de uso	Número de classes	Peso
Gerar total por espécie	Simples	3	1
Gerar embarcações com maior quantidade	Simples	4	1
Informar coleta de dados	Médio	5	1
		Total UUCW:	**3**

Logo tem-se o resultado dos pontos não ajustados, UUCP = 6 (UAW)+3 (UUCW)=9.

Posteriormente é necessário o cálculo dos fatores de ajuste: fatores técnicos e fatores ambientais. No exemplo conforme tabela 15, atribui-se o grau de influência a cada fator técnico. Conforme já citado anteriormente, aqui está uma parte mais subjetiva do cálculo, visto que depende do bom conhecimento e experiência do engenheiro de software que está realizando o cálculo.

Tabela 15 – Exemplo do cálculo dos fatores técnicos

Fator	Descrição	Nível de influência		Peso	Total Peso
T1	Sistema distribuído	2	**x**	2	4
T2	Tempo de resposta	2	**x**	2	4
T3	Eficiência	3	**x**	1	3
T4	Processamento complexo	1	**x**	1	1
T5	Código reusável	3	**x**	1	3
T6	Facilidade de instalação	1	**x**	0.5	0.5
T7	Facilidade de uso	1	**x**	0.5	0.5
T8	Portabilidade	1	**x**	2	2
T9	Facilidade de mudança	2	**x**	1	2
T10	Concorrência	1	**x**	1	1
T11	Recursos de segurança	1	**x**	1	1
T12	Acessível por terceiros	1	**x**	1	1
T13	Requer treinamento especial	0	**x**	1	0
				Total dos pesos:	**23**

Calculado o total dos pesos dos fatores técnicos, aplica-se a fórmula descrita no quadro 3. Tem-se então, TCF = 0.6 + (0.01 x 23) = 0.83.

Conforme tabela 16, calculam-se agora os fatores ambientais, onde atribui-se o nível de influência a cada fator.

Tabela 16 – Exemplo do cálculo dos fatores ambientais

Fator	Descrição	Nível de influência		Peso	Total Peso
E1	Familiaridade com RUP ou outro processo formal	1	**x**	1.5	1.5
E2	Experiência com a aplicação em desenvolvimento	1	**x**	0.5	0.5
E3	Experiência em orientação a objeto	3	**x**	1	3
E4	Presença de analista experiente	2	**x**	0.5	1
E5	Motivação	1	**x**	1	1
E6	Requisitos estáveis	1	**x**	2	2
E7	Desenvolvedores em meio-expediente	0	**x**	-1	0
E8	Linguagem de programação difícil	2	**x**	2	4
				Total dos pesos:	**13**

Calculado o total dos pesos dos fatores ambientais, aplica-se a fórmula descrita no quadro 4. Tem-se então, EF = 1.4 + (-0.03 x 13) = 1.01.

Depois de efetuados os cálculos do UUCP, EF e TCF, calcula-se então o valor total do sistema conforme fórmula vista no quadro 5. Tem-se como resultado de UCP = 9 x 0.83 x 1.01 = 7.54.

Logo, multiplica-se o valor de UCP pelo número de horas homem, UCP = 7.54 x 20 = 150.89. Portanto, para este estudo de caso a previsão é de 150.89 horas/homem.

5 DESENVOLVIMENTO DA FERRAMENTA

Neste capítulo são apresentadas as especificações da ferramenta desenvolvida, juntamente com a implementação do mesma.

5.1 REQUISITOS PRINCIPAIS

A ferramenta deve permitir o cálculo dos *Use Case Points* a partir dos diagramas de casos de uso e diagramas de seqüência criados pela ferramenta CASE *Rational Rose*. Esta ferramenta CASE foi adotada para o trabalho em virtude da mesma ser utilizada no mercado e estar disponível para uso na Universidade (FURB).

Dos diagramas de casos de uso serão obtidos os atores e os casos de uso. Dos diagramas de seqüência serão obtidas as classes utilizadas em cada caso de uso.

Resumindo, os principais requisitos funcionais identificados neste trabalho foram:

a) a ferramenta deve ser capaz de ler o arquivo gerado pela ferramenta CASE *Rational Rose* e a partir deste manipular os dados necessários para o cálculo dos *Use Case Points*;
b) a ferramenta deve gerar um relatório com os dados dos cálculos.

Como requisitos não funcionais pode-se destacar a necessidade de facilidade de uso da ferramenta e a devida precisão dos resultados visto que tratam-se de cálculos matemáticos.

5.2 ESPECIFICAÇÃO

A especificação do sistema foi feita utilizando-se de alguns diagramas da UML. A ferramenta CASE utilizada para esta especificação foi o *Rational Rose.* A seguir serão apresentados estes diagramas.

5.2.1 DIAGRAMA DE CASOS DE USO

Na ferramenta desenvolvida foram observados os seguintes diagramas de casos de uso primários (Figura 13):

a) ler arquivo UML: o analista abre um arquivo do *Rational Rose* e a ferramenta gera na tela os dados referentes ao diagrama de caso de uso e o diagrama de seqüência necessárias ao cálculo. A partir do diagrama de caso de uso serão listados na tela, os atores e os casos de uso com os dados necessários ao cálculo. A partir do diagrama de seqüência, utiliza-se número de classes envolvidas em cada caso de uso para o cálculo dos pesos dos casos de uso;

b) efetuar cálculo: o analista deverá informar o nível de influência dos fatores técnicos e ambientais, bem como classificar os atores e informar o número de horas por ponto dos *Use Case Points* e então efetuar o cálculo, mostrando na tela os resultados obtidos ;

c) gerar relatório: gera um relatório com os dados completos do cálculo.

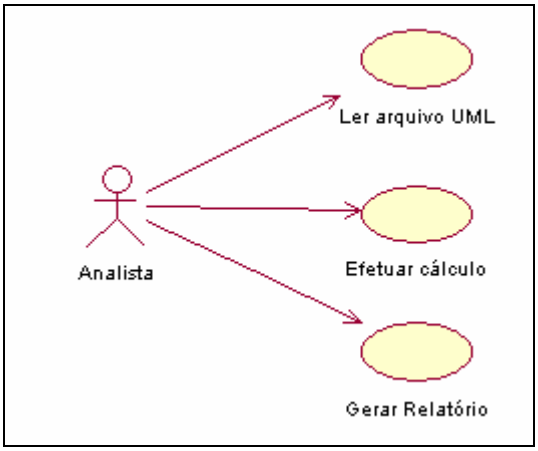


FIGURA 13 – Diagrama de casos de uso

5.2.2 DIAGRAMA DE CLASSES

As classes identificadas para esta ferramenta são apresentadas na figura 14.

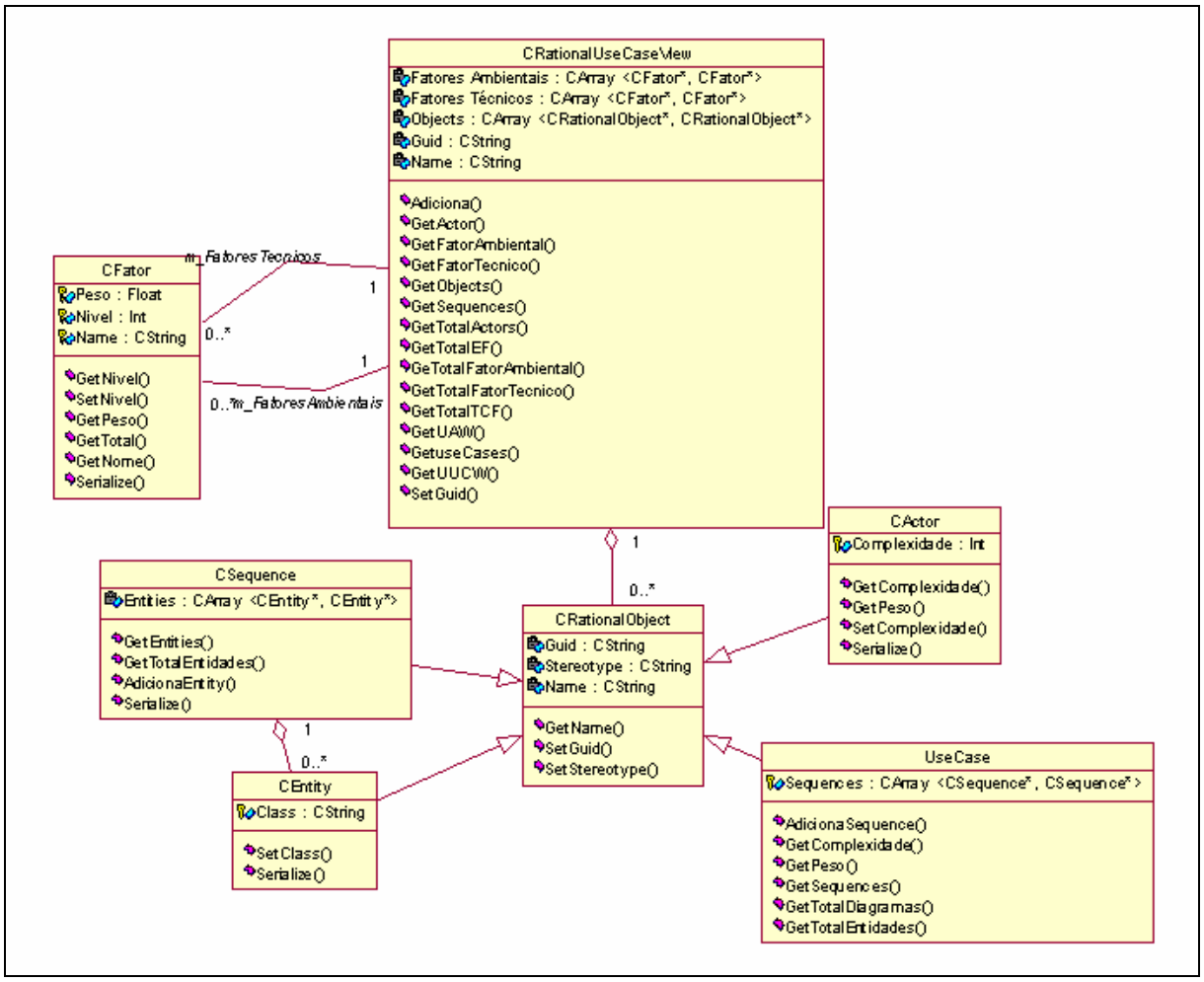


FIGURA 14 – Diagrama de classes

As classes possuem as seguintes finalidades:

a) CRationalUseCaseView: é responsável pela manipulação dos dados obtidos do arquivo UML;

b) CRationalObject: contém informações comuns para as entidades, casos de uso, atores e diagrama de sequências;

c) CEntity: representa as entidades extraídas do arquivo UML;

d) CSequences: representa o diagrama de sequência do arquivo UML;

e) CActor: representa os atores do arquivo UML;

f) Use Case: representa os casos de uso do arquivo UML;

g) CFator: representa os fatores técnicos e ambientais.

5.2.3 DIAGRAMA DE ATIVIDADES

Os diagramas de atividades descrevem o fluxo de atividades de um processo e podem ser usados para facilitar o entendimento dos casos de uso. Serão descritos a seguir os três casos de uso primários:

a) ler arquivo UML;

b) efetuar cálculo;

c) gerar relatório.

Na figura 15 é apresentado o diagrama de atividades “ler arquivo UML”.

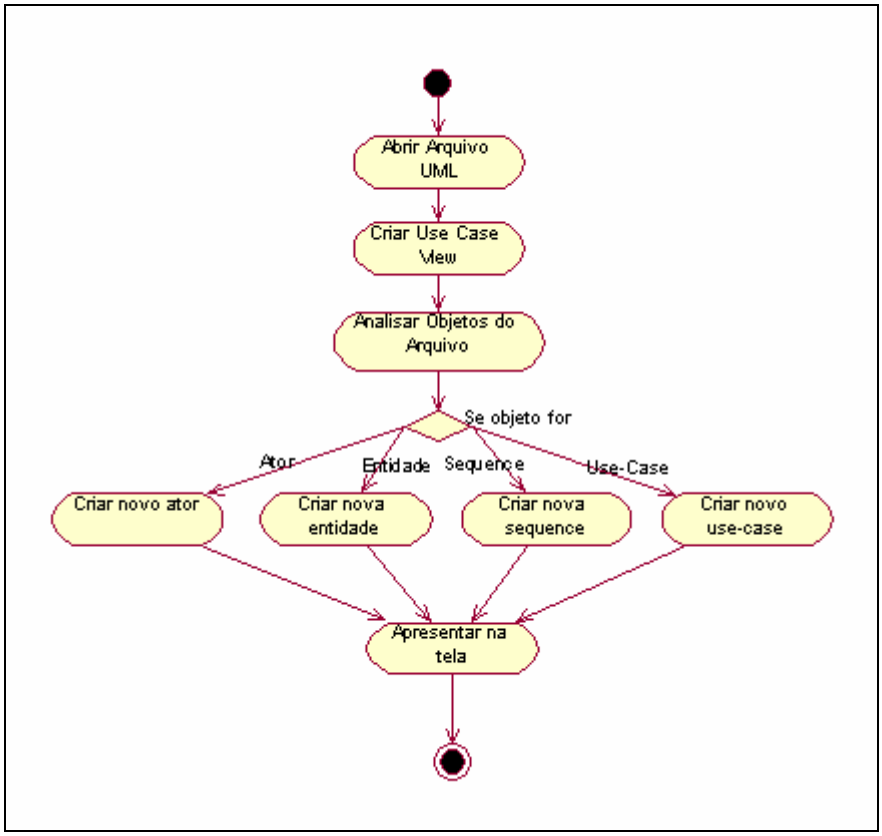


FIGURA 15 – Diagrama de atividades do ler arquivo UML

Este diagrama inicia com a solicitação de leitura de um arquivo gerado pela ferramenta *Rational Rose*, feita pelo analista, sendo posteriormente interpretado. Este arquivo deverá conter um diagrama de caso de uso e um diagrama de seqüência. O diagrama de seqüência deve ser criado a partir de um caso de uso. Após isto, começa o processo de criação dos objetos.

A figura 16 mostra o diagrama de atividades “Efetuar cálculo”.

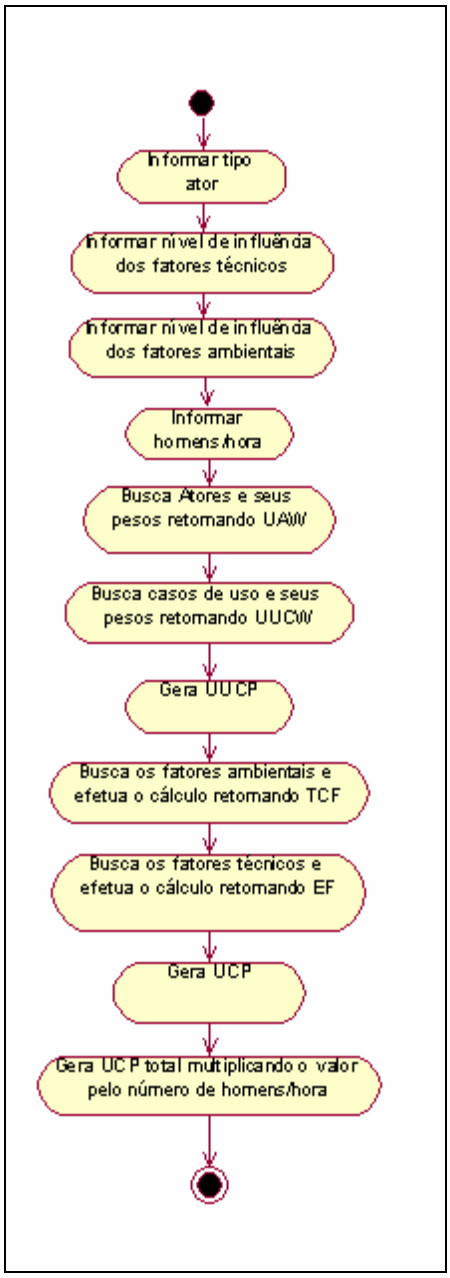


FIGURA 16 – Diagrama de atividades do efetuar cálculo

Primeiramente o analista deve definir a classificação dos atores listados, como simples, médio ou complexo. Deve-se então definir, o nível de influência de 0 a 5 dos fatores técnicos e dos fatores ambientais, e informar o valor dos pontos por caso de uso (homens/hora).

Após definidas estas informações, a ferramenta busca a quantidade de atores e os respectivos pesos definidos pelo analista, somando os pesos e retornando o valor de UAW.

Seguindo, busca-se a quantidade dos casos de usos e seus respectivos pesos. Nesta etapa, calcula-se o peso dos casos de uso, através da quantidade de entidades envolvidas no

processo trazidas do diagrama de seqüência conforme visto na tabela 10. Somando-se os pesos, tem-se então o valor de UUCW. Logo tem-se, o resultado do UUCP, conforme visto na fórmula do quadro 2.

Nesta etapa, busca-se o nível de influência dos fatores técnicos, informados anteriormente, logo calcula-se o TCF conforme visto na fórmula do quadro 3. Após, busca-se o nível de influência dos fatores ambientais, informados anteriormente, logo calcula-se o EF conforme visto na fórmula do quadro 4.

Depois de efetuadas as etapas, pode-se então calcular os UCP (*Use Case Points)* conforme visto na fórmula do quadro 5. Multiplica-se o número de pontos por casos de uso (homens/hora), informado pelo analista anteriormente, pelo resultado do cálculo do UCP.

Na figura 17 é apresentado o diagrama de atividades “gerar relatório”.

Primeiramente, cria-se um arquivo com extensão TXT, sendo que, a cada etapa efetuada, de busca de dados ou resultados, grava-se no arquivo criado. Busca-se a quantidade de casos de uso e seus dados, como peso, complexidade, número de entidades envolvidas em cada caso de uso. Seguindo, busca-se os atores e seus respectivos pesos e complexidades. Nesta etapa, busca-se os fatores técnicos e seus dados, bem como os fatores ambientais e seus dados.

Depois de efetuadas as etapas, busca-se todos os resultados do cálculo, seguindo na ordem, UAW, UUCW, UUCP, TCF, EF, UCP e UCP final, que é multiplicado pelo número de pontos por caso de uso. Seguindo, chama-se o executável notepad onde tem-se o relatório conforme apêndice A.

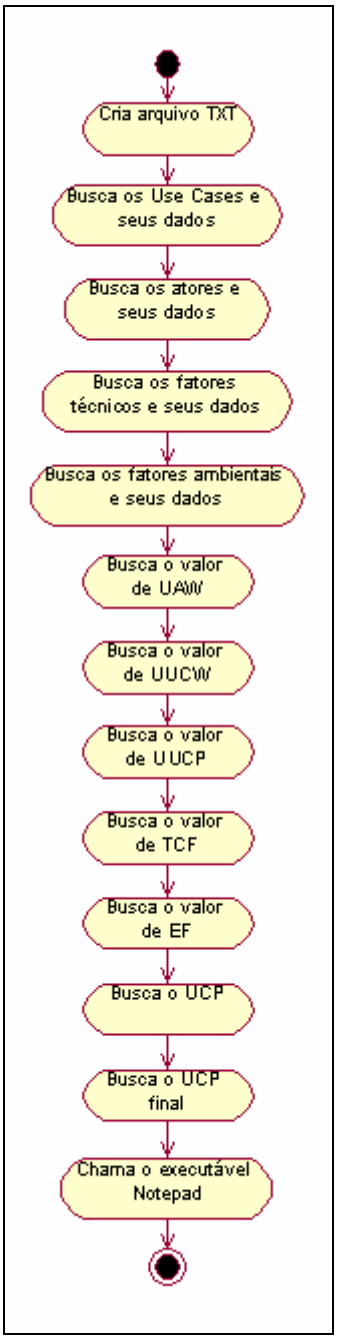


FIGURA 17 – Diagrama de atividades do gerar relatório

5.3 IMPLEMENTAÇÃO

Neste capítulo serão apresentadas as ferramentas e técnicas utilizadas na construção do protótipo da ferramenta.

5.3.1 PROGRAMAÇÃO VISUAL (AMBIENTE VISUAL C++)

Segundo Holzner (1999), o Visual C++ é um ambiente de desenvolvimento poderoso, contido em um pacote formado por inúmeras ferramentas para auxiliar na programação Windows. Também conhecido por Microsoft Visual Studio, integra ferramentas para criação, edição e compilação e teste de software em uma única interface.

Para o desenvolvimento dos programas utilizou-se C++, originalmente criado para desenvolver programas grandes, tornando o processo mais gerenciável, ou seja, facilitando o tratamento de programação Windows. Além disso, com a migração da linguagem C para a linguagem C++, a metodologia de programação procedural foi substituída pela orientação a objetos.

O ambiente dispõe da biblioteca *Microsoft Foundation Class* (MFC), um pacote de código pré-escrito e pronto para uso. Esta biblioteca fornece uma coleção de classes escritas em C++ que realizam a maioria do trabalho penoso no desenvolvimento de aplicações Windows.

O Visual C++ também oferece muitas ferramentas de programação tais como editor de menu para a criação de menus, o editor de diálogo para a criação de caixas de diálogo e os Assistentes do Visual C++, que escrevem boa parte do programa apenas com base em algumas informações fornecidas na etapa de execução da ferramenta (HOLZNER, 1999).

Os projetos em desenvolvimento são colocados em espaços de trabalho (*workspaces)* sendo que um espaço de trabalho pode conter vários projetos. A grande quantidade de arquivos que um programa Windows utiliza nos projetos organizados automaticamente pelo ambiente. O trabalho foi desenvolvido utilizando a versão 6.0 do ambiente Microsoft Visual Studio.

5.3.2 FERRAMENTA CASE RATIONAL ROSE

Segundo Silva (2001), ferramentas CASE são definidas como um conjunto integrado de ferramentas que oferecem impulso, atuando em todas as fases do ciclo de desenvolvimento do projeto de software.

O *Rational Rose*, utilizado também para especificar a ferramenta, visualizar, documentar e construir artefatos de um sistema utilizando diagramas que fazem parte da UML (RATIONAL, 2000). A figura 18 mostra o *Rational Rose* sendo utilizado durante a especificação da ferramenta desenvolvida neste trabalho.

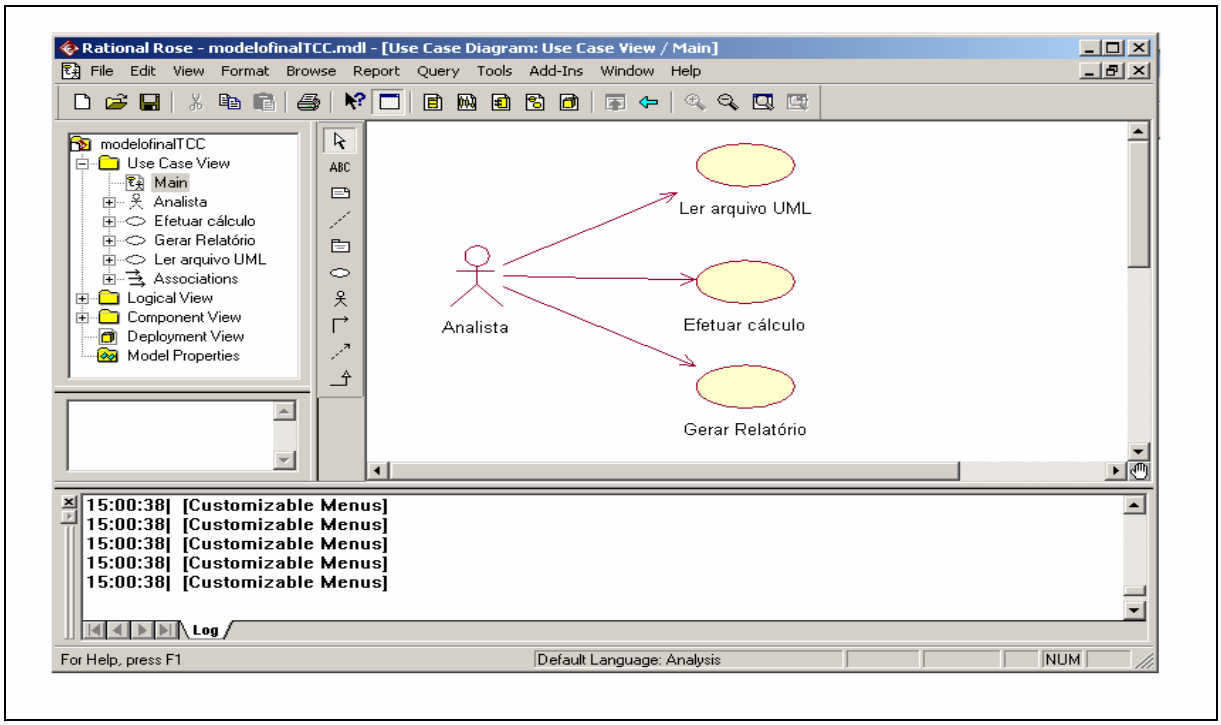


FIGURA 18 – Ferramenta CASE *Rational Rose*

5.3.3 OPERACIONALIDADE

Para demonstrar a operacionalidade, será utilizado o mesmo estudo de caso descrito no quadro 6. Os diagrama de caso de uso pode ser observado na figura 8. Os diagramas de seqüência deste estudo de caso estão descritos na figura 9, 10 e 11. O diagrama de classes na figura 12. Estes, foram utilizados anteriormente no capítulo 3.4 como exemplo de cálculo.

Para utilizar a ferramenta deve-se inicialmente criar o diagrama de caso de uso, o diagrama de classe e os diagramas de seqüências, citados anteriormente, na ferramenta CASE *Rational Rose*. Tendo o diagrama salvo, pode-se iniciar o cálculo.

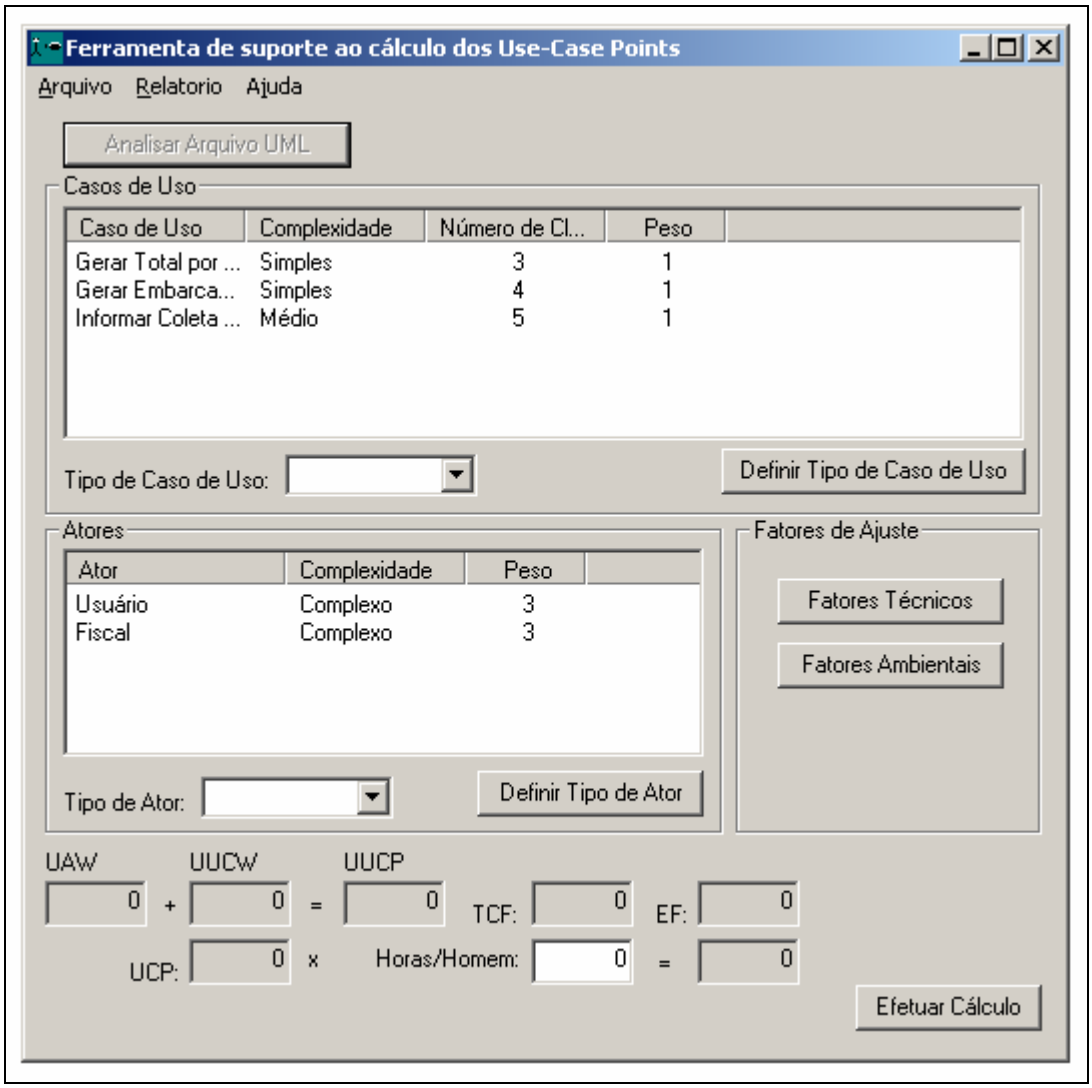


FIGURA 19 – Tela principal

As principais funções são descritas a seguir, conforme figura 19:

a) Arquivo:
   - abrir: abre uma definição do diagrama de casos de uso e o diagrama de caso de seqüência;
   - salvar: salva a definição do diagrama de casos de uso e o diagrama de seqüência;
   - abrir arquivo UML: abre um arquivo do *Rational Rose* ;
   - sair: sair da tela;

b) Relatório:
   - emitir relatório: imprime relatório referente ao cálculo efetuado;

c) Ajuda:
   - Sobre: mostra informações sobre a versão da ferramenta desenvolvida.

O botão “Analisar arquivo UML” deverá ser utilizado para a ferramenta analisar o arquivo aberto anteriormente.

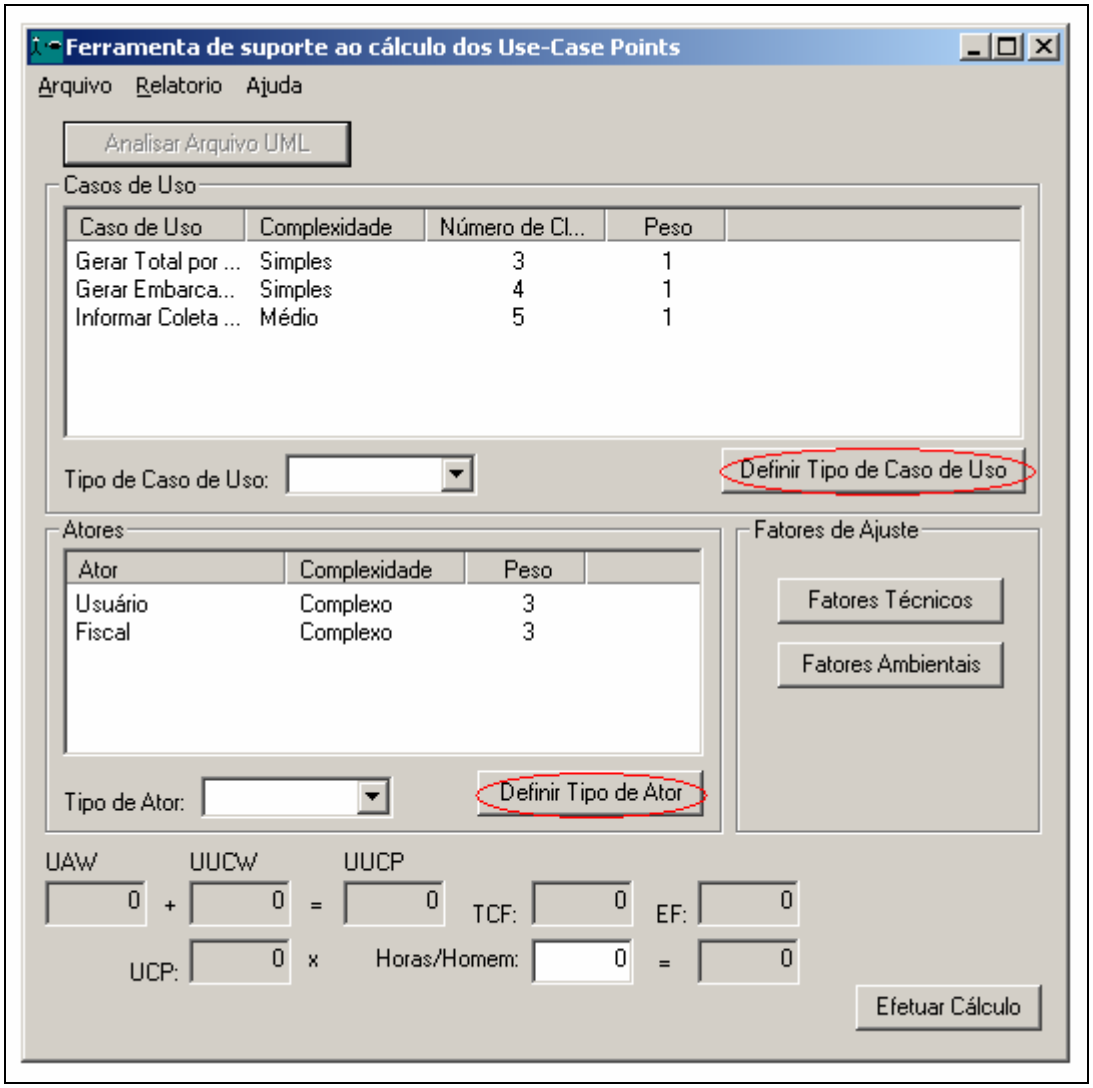


FIGURA 20 – Tela principal – Definir Tipo de Ator e Caso de Uso

A partir da opção “Definir Tipo de Ator”, classificam-se os atores de acordo com a complexidade do ator, conforme figura 20. Na opção “Definir Tipo de Caso de Uso”, classificam-se de acordo com a complexidade do caso de uso, mostrado na figura 20.

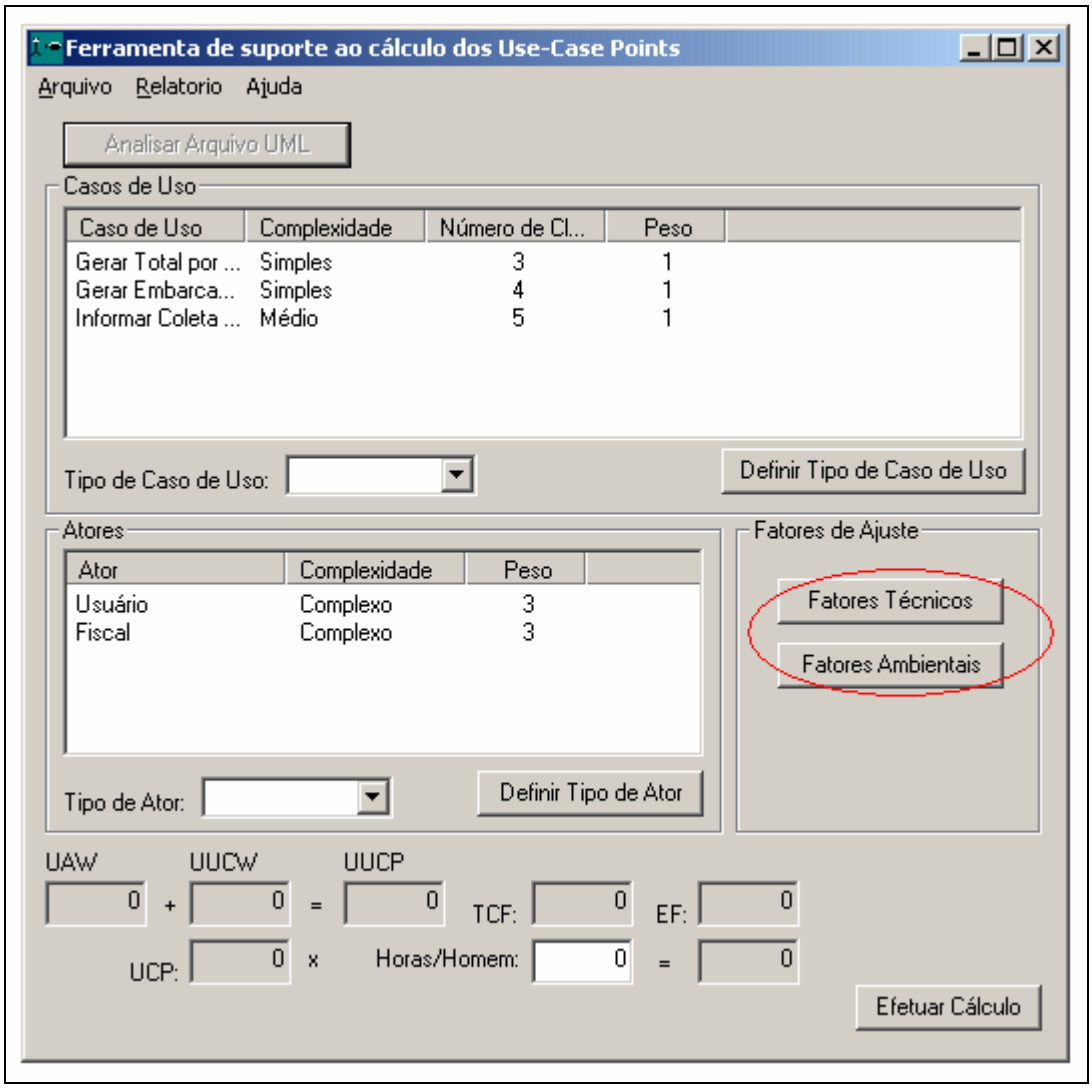


FIGURA 21 – Tela principal – Fatores de ajuste

Para o cálculo dos fatores de ajustes, tem-se duas telas que podem ser visualizadas, pressionando a opção “Fatores Técnicos” e a opção “Fatores Ambientais”, conforme exemplificado na figura 21.

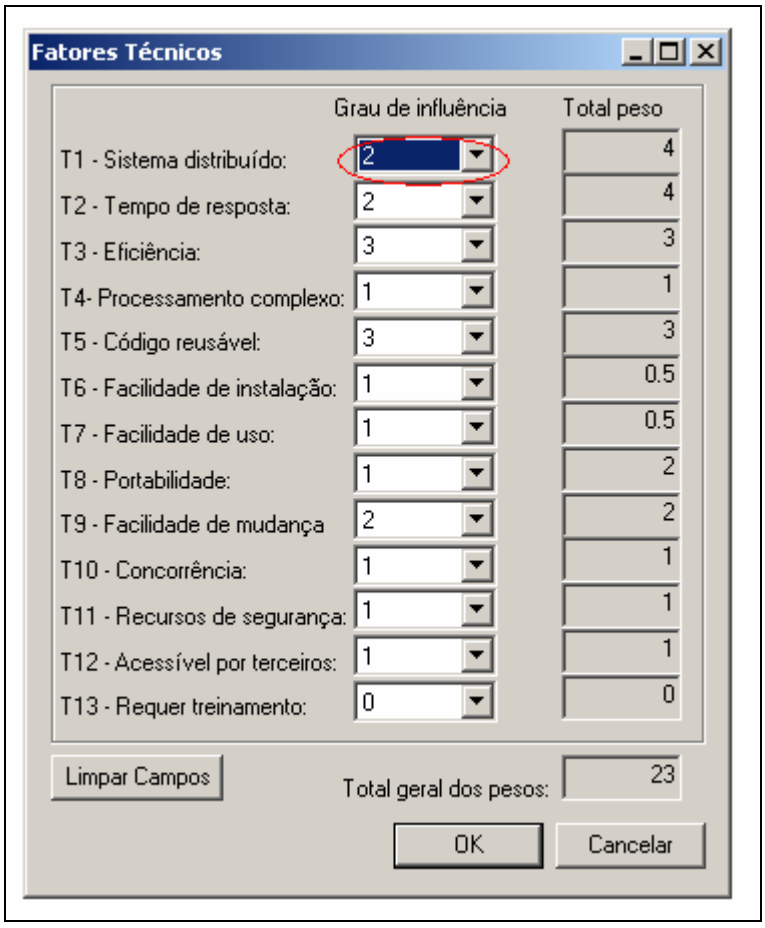


FIGURA 22 – Tela fatores técnicos

Define-se o então grau de influência para cada fator técnico, conforme a tela da figura 22. Para que os dados informados nesta tela sejam utilizados no cálculo, é necessário pressionar o botão OK. Ao lado de cada grau de influência, é apresentado o total do peso para cada fator. Tem-se ainda, o total geral dos pesos dos fatores técnicos.

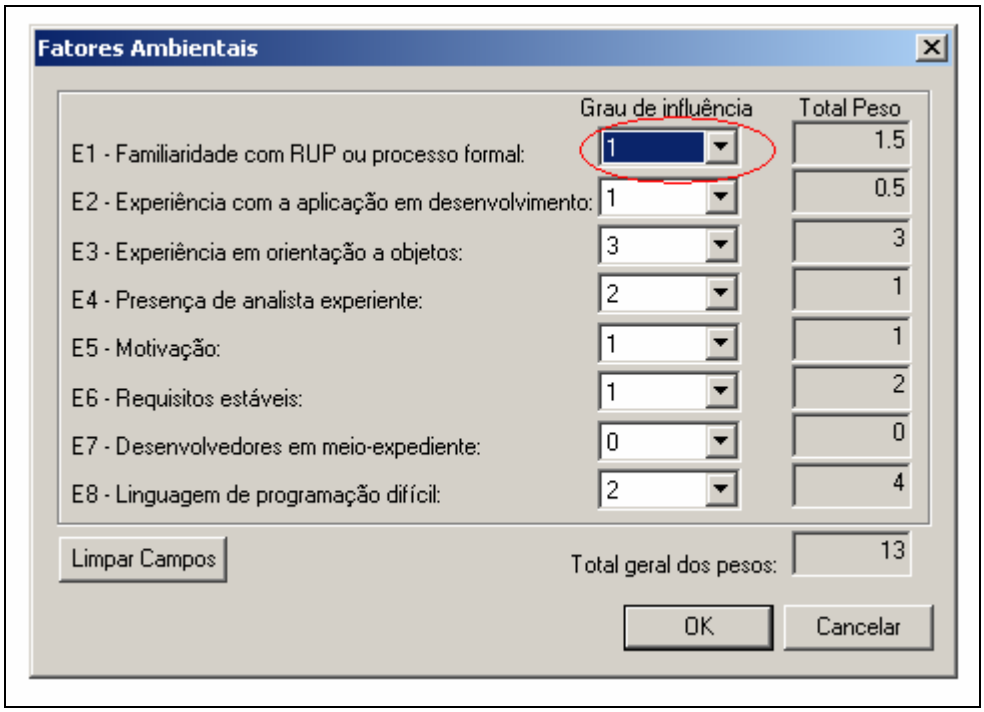


FIGURA 23 – Tela fatores ambientais

Na tela mostrada na figura 23, define-se o grau de influência dos fatores ambientais. Ao lado de cada grau de influência, é exemplificado o total do peso para cada fator. Tem-se ainda, o total geral dos pesos dos fatores ambientais.

Nesta etapa, deverá ser informado o valor de homens/hora. Depois de informados todos os dados necessários ao cálculo utiliza-se o botão “Efetuar Cálculo” para finalizar o cálculo, conforme tela apresentada na figura 24.

Para a geração do relatório, deve-se selecionar a opção Relatório/Emitir relatório, imprimindo assim o relatório com os resultados do cálculo. No apêndice A, tem-se o relatório emitido com os resultados do cálculo do estudo de caso utilizado anteriormente.

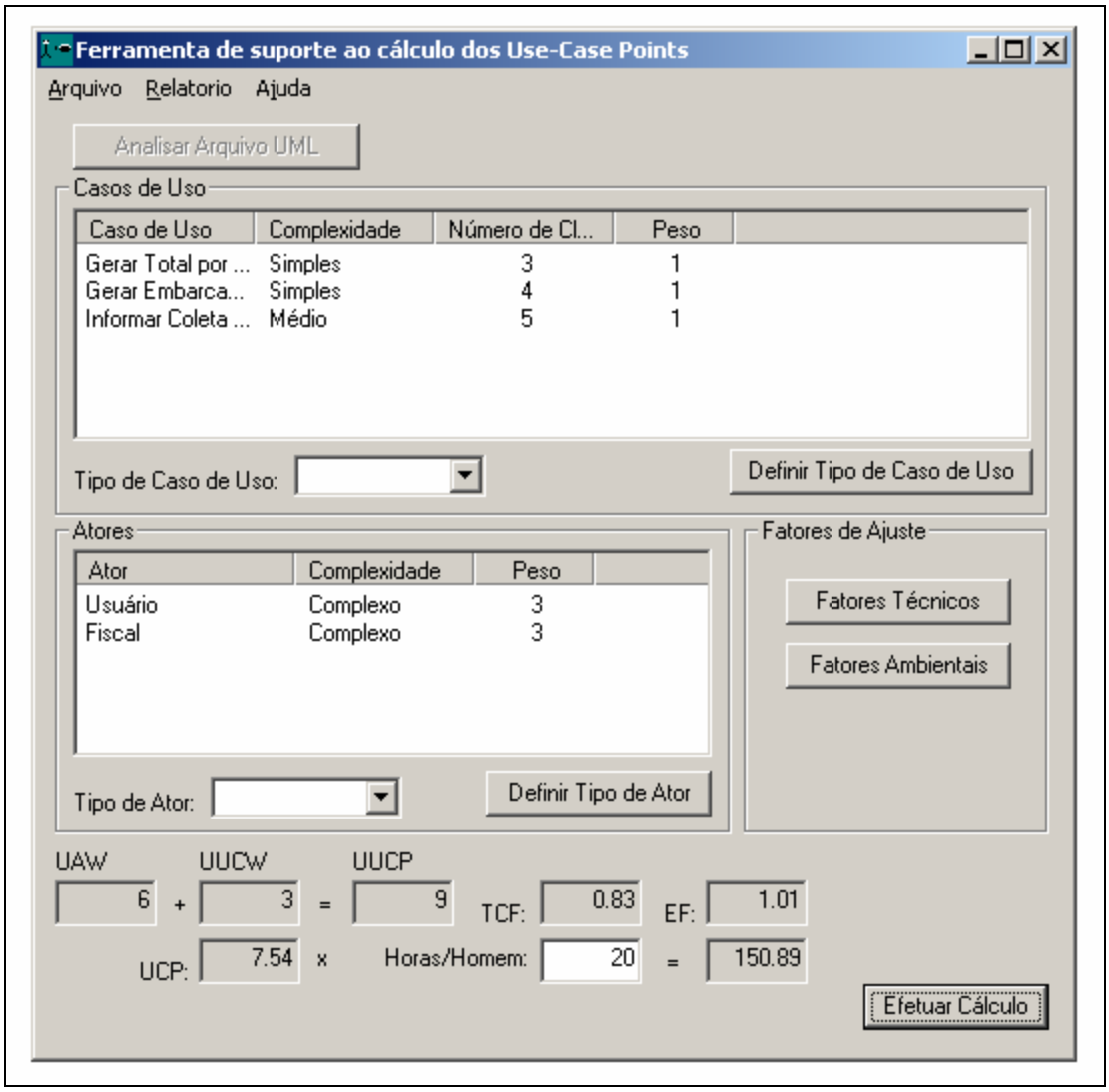


FIGURA 24 – Tela principal – Efetuar Cálculo

5.4 RESULTADOS E DISCUSSÃO

Como dificuldades deste trabalho cita-se a interpretação do arquivo gerado pela ferramenta *Rational Rose*. O arquivo MDL contém uma estrutura definida por parte da ferramenta CASE *Rational Rose*, contendo palavras reservadas no arquivo MDL, dando a possibilidade de identificação da informação desejada. Por exemplo, a palavra "object UseCase" identifica que logo após deverá cont er o nome do caso de uso especificada dentro do ambiente Use Case View do *Rational Rose*. Tendo identificadas todas as palavras reservadas do arquivo MDL, poderá identificar-se o ator, o caso de uso do diagrama de casos de uso, e as classes no diagrama de seqüência. No apêndice B, verifica-se parte do código fonte criado para a leitura do *Rational Rose*. Apesar do arquivo MDL possuir palavras

reservadas, este não possui fácil compreensibilidade, pois tem-se muitas informações, por exemplo, para apenas um ator.

A utilização do recurso de serialização, utilizado para abrir e gravar o arquivo oferecido pelo MFC, facilita a leitura e gravação dos estados de objetos por ter sido desenvolvido especialmente para esse fim, onde passa-se os valores dos objetos para um arquivo em uma seqüência simples e lógica. O MFC encarrega-se de armazená-los no arquivo. Para extrair, basta utilizar a mesma seqüência da gravação, criando novos objetos e passando os valores existentes no arquivo para os mesmos.

A automação de parte do cálculo dos *Use Case Points*, tornou-se necessária a utilização de três diagramas: diagrama de casos de uso, diagrama de seqüência e o diagrama de classes utilizado para elaborar o diagrama de seqüência. Deste modo, avança-se mais no processo de projeto do que inicialmente planejava-se.

Para a realização dos testes da ferramenta foram utilizados alguns diagramas gerados na disciplina de Requisitos de Software no curso de Ciências da Computação da FURB. Os resultados demonstraram alguns problemas no início, mas depois foram superados com os devidos ajustes na leitura do arquivo da ferramenta CASE *Rational Rose*.

6 CONCLUSÕES

Apesar de ainda não estarem sendo utilizadas em grande escala na nossa região, as métricas de software estão cada vez mais se tornando indispensáveis, pois com um mercado tão competitivo, é indispensável a existência de ferramentas que auxiliem no processo de gerenciamento de sistemas, para não extrapolar custos e prazos. A ferramenta construída pode contribuir na disseminação desta cultura a partir da utilização dos diagramas de casos de uso e da métrica correspondente *Use Case Points*. Além disso, pode-se utilizar a mesma nas aulas de engenharia de software como instrumento de aprendizado.

Os objetivos do trabalho foram alcançados visto que a ferramenta faz a leitura apropriada dos diagramas de casos de uso e diagramas de seqüência gerados pela ferramenta CASE *Rational Rose* e também disponibiliza um relatório com os dados do cálculo.

Um ponto importante a ressaltar na ferramenta construída é a automação de parte do cálculo dos *Use Case Points* a partir dos diagramas obtidos na ferramenta CASE *Rational Rose.* Inicialmente não se planejava obter os dados do diagrama de seqüência, mas com a evolução do trabalho percebeu-se como uma alternativa a ser adotada. Para a criação do diagrama de seqüência é necessário que tenha-se definido o diagrama de classes. Por um lado, isto sugere que o usuário tenha os três diagramas criados na ferramenta CASE.

Quanto ao ambiente de desenvolvimento utilizado na codificação, Visual C++ 6.0, o mesmo mostrou-se apropriado para a ferramenta desenvolvida.

Como extensões para este trabalho sugere-se:

a) a ferramenta poderia permitir o cálculo dos *Use Case Points* sem a leitura de arquivos de ferramenta CASE, apenas com a intervenção manual;
b) a ferramenta poderia fazer a leitura de outras ferramentas CASE e em especial algumas opções de software livre como o *Poseidon* – software que suporta nove diagramas de UML, mais informações podem ser obtidas em Gentleware (2001);
c) transformar a ferramenta desenvolvida numa DLL e incorporar na ferramenta CASE *Rational Rose*.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDA, Bente. **Comparring effort estimatives based on use case points with expert estimates.** [S.l], 2001?. Disponível em: <http://www.ifpug.com/articles/ Anda-Comparing_Effort_Estimates_Based_on_UCP.pdf>. Acesso em: 12 ago. 2003.

BANERJEE, Gautam. **Use Case Points:** an estimation approach, [S.l], 2001?. Disponível em: < http://www.bfpug.com.br/Artigos/UCP/Banerjee-UCP_An_Estimation_Approach.pdf>. Acesso em: 12 ago. 2003.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML.** Rio de Janeiro: Campus, 2003.

BFPUG, **Brazilian Function Point Users Group**, [S.l.], 1998. Disponível em: <http://www.bfpug.com.br>. Acesso em: 12 ago. 2003.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML:** guia do usuário: o mais avançado tutorial sobre Unified Modeling Language (UML). Tradução Fábio Freitas. Rio de Janeiro: Campus, 2000.

BRAGA, Antônio Braga. **Análise de pontos de função.** Rio de Janeiro: Infobook, 1996.

FERNANDES, Aguinaldo A. **Gerência de software através de métricas.** São Paulo: Atlas, 1995.

FREIRE, Herval. **Calculando estimativas:** o método de pontos de caso de uso, [S.l], 2003. Disponível em: <http://www.cnnt.com.br/files/usecasepoints.pdf> . Acesso em: 12 ago. 2003.

FURLAN, José Davi. **Modelagem de objetos através da UML.** São Paulo: Makron Books, 1998.

HOLZNER, Steven. **Programando Visual C++ 6 :** em tempo recorde. São Paulo: Makron Books, 1999.

KARNER, Gustav. **Metrics for objectory.** Tese – Linkoping University Departament of Computer and Information Science. Suécia: 1993.

LONGSTREET, David. **Use case and function points**, [S.l], 2001. Disponível em: <http://www.ifpug.com/articles/usecases.htm>. Acesso em: 12 ago. 2003.

MOLLER, K. H.; PAULISH D. J.. **Software metrics:**a practitioneris guide to improved product development. Los Alamitos: IEEE, 1995.

GENTLEWARE. **Poseidon for UML**, [S.l], 2001?. Disponível em: <www.gentleware.com>. Acesso em: 12 dez. 2003.

PRESSMAN, Roger S.. **Engenharia de software**. Tradução José Carlos Barbosa dos Santos. São Paulo: Makron Books, 2002.

RATIONAL, Software Corporation. **Unified modeling language v.1.**, [S.l], jan. 2001. Disponível em: <http://www.rational.com/uml/resources/documentation>. Acesso em: 12 ago. 2003.

SILVA, Patrícia Regina Ramos da. **Ferramenta para cálculo de métricas em softwares orientados a objetos codificados em Delphi.** 2001. 86 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

TONSIG, Sérgio Luiz. **Engenharia de software:** análise e projeto de sistemas. São Paulo: Futura, 2003.

VALCANAIA, Tibério César. **Protótipo de ferramenta ao cálculo de FPA sobre Microsoft Acess 97.** 1998. 126 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

APÊNDICE A – Relatório emitido pela ferramenta

```
-------------------------------------------------------------------------
CASOS DE USO:
-------------------------------------------------------------------------
Nome                                  Complexidade   Classes   Peso
Gerar Total por Espécie               Simples              3    1
Gerar Embarcações com maior Quant     Simples              4    1
Informar Coleta de Dados              Médio                5    1
-------------------------------------------------------------------------
ATORES:
-------------------------------------------------------------------------
Nome                                  Complexidade   Peso
Usuário                               Complexo         3
Fiscal                                Complexo         3
-------------------------------------------------------------------------
FATORES TÉCNICOS:
-------------------------------------------------------------------------
Nome                                                  Grau  Peso
T1  - Sistema distribuído                               2  2.00
T2  - Tempo de resposta                                 2  2.00
T3  - Eficiência                                        2  1.00
T4  - Processamento complexo                            2  1.00
T5  - Código reusável                                   2  1.00
T6  - Facilidade de instalação                          2  0.50
T7  - Facilidade de uso                                 2  0.50
T8  - Portabilidade                                     2  2.00
T9  - Facilidade de mudança                             2  1.00
T10 - Concorrência                                      1  1.00
T11 - Recursos de segurança                             0  1.00
T12 - Acessível por terceiros                           0  1.00
T13 - Requer treinamento                                0  1.00
-------------------------------------------------------------------------
FATORES AMBIENTAIS:
-------------------------------------------------------------------------
Nome                                                  Grau  Peso
E1 - Familiaridade RUP/Processo formal                  1  1.50
E2 - Experiência com a Aplicação em desenv.             2  0.50
E3 - Experiência em OO                                  5  1.00
E4 - Presença analista experiente                       3  0.50
E5 - Motivação                                          1  1.00
E6 - Requisitos estáveis                                1  2.00
E7 - Desenvolvedores em meio expediente                 1  -1.00
E8 - Ling. Programação difícil                          1  2.00
-------------------------------------------------------------------------
RESULTADOS:
-------------------------------------------------------------------------
UAW  : 6
UUCW : 3
UUCP : 9
TCF  : 0.83
EF   : 1.01
UCP  : 7.54
TOTAL: 150.89
```

APÊNDICE B – Leitura do arquivo gerado pela ferramenta *Rational Rose*

```
void CLeRationalDlg::OnAnalisar()
{
    const LPCSTR ROOT_STRING = "root_usecase_package";
    const LPCSTR PARENTESES = "()";

   UpdateData(TRUE);

   CWaitCursor wc;

   CStdioFile arquivo;
   CFileException e;

   if ( m_sArquivo != "" )
   {
          if (m_sArquivo != m_sArquivoAnterior)
             m_sArquivoAnterior = m_sArquivo;
         else
         {
            AfxMessageBox("Arquivo já analisado anteriormente");
            return;
         }

         if (m_pUseCaseView != NULL)
            delete[] m_pUseCaseView;

      if (arquivo.Open(m_sArquivo,  CFile::modeRead|CFile::typeText,&e) == TRUE)
        {
          CString linha = "";
          CString name = "";
          CString valor = "";

          int parenteses = 0;
          bool elementFound = false;
          bool fim = false;

          CRationalObject* object = NULL;
          CAssociation* lastAssociation = NULL;
          CSequence* lastSequence = NULL;
          CUseCase* lastUseCase = NULL;

          TipoObjeto currentObject = ObjetoNaoTratado;
```

```
while (arquivo.ReadString(linha))
{
      if (linha.Find(ROOT_STRING) >= 0)
      {
            parenteses = 0;
            elementFound = true;
      }


      if (elementFound && !fim)
      {
            int index = linha.FindOneOf(PARENTESES);
             while (index >= 0)
            {
               if (linha[index] == '(')
                 parenteses++;

               if (linha[index] == ')')
               {
                 parenteses--;
                 if (parenteses == 0)
                 {

                    if (object != NULL &&
                    currentObject != Role &&
                    currentObject != Object)
                    {
                      m_pUseCaseView-Adiciona(object);
                     }

                    fim = true;
                    break;
                 }
               }

               int temp = linha.Find("(", index+1);
               if (temp == -1)
               temp = linha.Find(")", index+1);

               index = temp;
      }

      TipoObjeto tipoObjeto = ParseObject(linha, name);

      if (tipoObjeto != ObjetoNaoTratado)
      {
            if (object != NULL &&
                currentObject != Mechanism &&
                currentObject != Object)
               {
                      m_pUseCaseView->Adiciona(object);
               }
```

```
                        switch (tipoObjeto)
                        {
                        case Class_Category:
                  m_pUseCaseView = new  CRationalUseCaseView(name);
                              break;

                        case Class:
                              object = new CActor(name);
                              break;

                       case UseCase:
                             object = new CUseCase(name);
                  lastUseCase = dynamic_cast<CUseCase*>(object);
                              break;

                        case Mechanism:
                             object = new CSequence(name);
                  lastSequence = dynamic_cast<CSequence*>(object);

                  if (lastUseCase != NULL)
lastUseCase->AdicionaSequence(dynamic_cast<CSequence*>(object));
                        break;

                      case  Object:
                         object = new CEntity(name);
                         if (lastSequence != NULL)
     lastSequence->AdicionaEntity(dynamic_cast<CEntity*>(object));

                         break;
               }

               currentObject = tipoObjeto;
          }
          else
          {
               TipoValor tipoValor = ParseValor(linha, valor);



               if (tipoValor != ValorNaoTratado)
               {
                    switch (tipoValor)
                    {
                    case Guid:
                    {
                         if (object != NULL)
                              object->SetGuid(valor);
                         else
                              m_pUseCaseView->SetGuid(valor);

                         break;
```

```
                    }

                case Stereotype:
                {
                    if (object != NULL)
                        object->SetStereotype(valor);

                    break;
                }

                case Entity:
                {
                   if (currentObject == Object)
                   {
                CEntity* temp = dynamic_cast<CEntity*>(object);
                temp->SetClass(valor);
                   }
                }

            }
          }
       }
    }

    if (fim)
    break;
    }
   }
   else
   {
       LPTSTR errorMessage = "\0";
       e.GetErrorMessage(errorMessage, 1);

       TRACE(errorMessage);
   }
  }
  else
  {
       AfxMessageBox("É necessário abrir um arquivo
                      com extensão .mdl");
  }

  if (m_pUseCaseView != NULL)
  {
            // ATORES
            if (m_pUseCaseView->GetTotalActors() > 0)
                CarregaListaAtores();

            // USES CASES
            if (m_pUseCaseView->GetTotalUseCases() > 0)
                CarregaListaCasos();
       }
```

```
        UpdateData(FALSE);
    }
```