UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

SISTEMA DE VISUALIZAÇÃO GRÁFICA PARA APOIAR A TOMADA DE DECISÃO DURANTE A FASE DE TESTE

CARLOS BERNARDES DE ABREU

ORIENTADOR: PROF.DR. PLÍNIO R. S. VILELA

PIRACICABA, SP

2009

UNIVERSIDADE METODISTA DE PIRACICABA
FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

SISTEMA DE VISUALIZAÇÃO GRÁFICA PARA APOIAR A TOMADA DE DECISÃO DURANTE A FASE DE TESTE

CARLOS BERNARDES DE ABREU

ORIENTADOR: PROF.DR. PLÍNIO ROBERTO SOUZA VILELA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, da Faculdade de Ciências Exatas e da Natureza, da Universidade Metodista de Piracicaba – UNIMEP, como requisito para obtenção do Título de Mestre em Ciência da Computação.

PIRACICABA, SP

2009

SISTEMA DE VISUALIZAÇÃO GRÁFICA PARA APOIAR A TOMADA DE DECISÃO DURANTE A FASE DE TESTE

AUTOR: CARLOS BERNARDES DE ABREU

ORIENTADOR: PROF.DR. PLÍNIO ROBERTO SOUZA VILELA

Dissertação de Mestrado defendida e aprovada em 27 de maio de 2009, pela Banca Examinadora constituída dos Professores:

Prof. Dr. Plínio Roberto Souza Vilela (Orientador)
Profa. Dra. Simone do Rocio Senger de Souza / USP – São Carlos
Profa. Dra. Cecília Arias Sosa Peixoto / Unimep

AGRADECIMENTOS

Ao amigo e professor Dr. Plínio Roberto Souza Vilela pela orientação, compreensão e pelas várias revisões desta dissertação. Você acreditou desde o início. Muito obrigado pelo apoio nos momentos em que eu precisei.

Aos demais professores e colegas de mestrado pela convivência e trocas de conhecimentos.

A minha grande amiga Fabiane Pifer Furlan pelo companheirismo incentivo, pelos dias de estudos e por compartilhar tantas lembranças sendo sempre honesta e gentil. Muito obrigado por você estar sempre ao meu lado.

A meus pais Sebastião Felix de Abreu e Batista Bernardes que me incentivam por toda minha vida.

RESUMO

A busca pela melhoria da qualidade na fase de teste de software tem levado as empresas a um investimento considerável com o objetivo de reduzir falhas que possam gerar perdas financeiras, perdas de tempo e perdas de outras naturezas.

As informações necessárias para a tomada de decisão de um gerente de teste de software podem estar em formulários *checklist e* planilhas, que demandam tempo para análise e interação. Uma das soluções atuais é o uso de software para gerenciar o teste de software, mesmo assim essa abordagem não resolve totalmente o problema, apenas melhora a organização.

Este trabalho procura demonstrar que um gerente de teste de software pode utilizar a visualização gráfica como ferramenta de apoio para a tomada de decisão. Foi desenvolvimento de uma ferramenta interativa para gerar visualização gráfica, no entanto a ferramenta superou a expectativa, pois além do apoio na tomada de decisão, a ferramenta demonstrou que pode gerar uma base de conhecimento e conseqüentemente uma melhoria na qualidade do teste de software.

PALAVRAS CHAVES: Teste de Software; Visualização gráfica; Tomada de decisão

GRAPHIC VISUALIZATION SYSTEM TO SUPPORT THE DECISION-MAKING DURING THE SOFTWARE TEST

ABSTRACT

The guest to improved quality during software testing has led companies to a considerable investment with the objective of reducing failures that may generate financial losses, loss of time and loss of other kinds.

The information available to a software testing manager during his/her decision making process may be in checklist and spreadsheets, which require time for analysis and interaction. One current solution is the use of software to manage the software test, this approach still does not solve completely the problem, only improves the organization.

This paper aims to show that a software test manager can use the graphical visualization as a tool to support decision-making. A tool that interactively display graphic information was developed, however the tool went over the expectation, as well as support in decision making, showed that the tool can generate a knowledge base and consequently an improvement in the quality of the software test.

KEYWORDS: Software Test, Graphic Visualization, Decision-making

SUMÁRIO

Lista de Figuras

LISTA DE ABREVIATURAS E SIGLAS

LOC	Lines of Code
GQM	Goal Question Metrics
ISO	International Standards Organization
IEEE	Institute of Electrical and Electronics Engineering (EUA).
FPA	Function Points Analysis
RUP	Rational Unified Process
UFC	Unadjasted Function-point Count
UML	Unified Modeling Language

LISTA DE TABELAS

1 INTRODUÇÃO

A visualização gráfica é uma forma de comunicação que facilita o entendimento e interpretação de uma informação ou mensagem. Desde o inicio da humanidade foram feitos desenhos em pedras e cavernas, os quais são formas de visualização gráfica de uma mensagem, mesmo antes do surgimento da escrita, é possível afirmar que temos facilidades na interpretação de imagens gráficas. Já no século XVI, com a expansão marítima, surgiram os primeiros mapas, também uma maneira encontrada para representar graficamente um conjunto de informações.
Leonardo da Vinci (1452 - 1519) transpôs seu conhecimento e conseguiu detalhar máquinas e movimentos em ilustrações gráficas.

A Engenharia de Software tem se desenvolvido significativamente nas últimas décadas, novas ferramentas de alto nível para desenvolvimento de software, novas ferramentas de teste, novos tipos de teste tem sido desenvolvidos, mesmo assim os defeitos permanecem no software. Para a revelação dos defeitos de software são utilizados diversos tipos de teste, e a qualidade do software está relacionada diretamente com o teste de software. O acompanhamento das métricas é feito através de formulários ou tabelas contendo os dados de teste.

No processo de teste de software existe uma série de abordagens buscando padronizar métodos, técnicas e padrões de teste de software através de medidas que são valores numéricos que podem representar a qualidade e a confiabilidade do software.

Conforme citado por Sommerville (2003) a criticidade do software está relacionada ao nível de confiança do software, que pode ser baixa, média, alta, muito-alta e ultra-alta. A confiança do software é a métrica que define o padrão de qualidade exigido para o software antes de sua entrada em produção, e está relacionado ao prazo necessário para o processo de produção, custo do software, prazos de entrega e risco do software falhar em produção. A confiança interfere diretamente na qualidade no processo de teste de software.

Para definir um conjunto de métricas será utilizado o paradigma Goal-Question-Metrics (GQM) [BASILI, CALDIERA, ROMBACH, 1994], onde as métricas serão selecionadas de acordo com a definição de necessidades claras para cada objetivo ou meta.

Este trabalho tem como proposta apoiar o processo de tomada de decisão relacionada ao teste de software, economizando tempo, custos desnecessários e estabelecer técnicas visuais como apoio a tomada de decisão. Para realizar essa avaliação foi desenvolvida uma ferramenta de visualização gráfica, interativa, utilizando os atributos de visualização [CARD, 1999] para gerar visualização gráfica. Através desses gráficos foi possível comparar o tempo para de análise de tabelas de dados brutos com as análises feitas através dos gráficos gerados pela ferramenta. Foi possível notar que a percepção das informações através de análise de gráficos ocorre em menor tempo, e conseqüentemente foi possível concluir que a visualização gráfica pode apoiar no processo de tomada de decisão.

1.1 MOTIVAÇÃO

A motivação principal deste trabalho é a dificuldade que gerentes de projeto têm encontrado na tomada de decisão desde a fase de levantamento de requisitos, fase de desenvolvimento e teste de software.

As diversas abordagens sobre visualização gráfica [CARD, 1999], têm demonstrado a eficiência em relação à clareza, facilidade de interpretação e compreensão de um grande volume de informações [FAYYAD, 1996]. A motivação vem da possibilidade de pesquisar a utilização de visualização gráfica em Engenharia de Software, na fase de teste de software, apoiando a tomada de decisão nesse processo, e visando principalmente a redução de prazos e custos, e melhoria da qualidade na fase de teste de software. Uma ferramenta de visualização gráfica apresenta formas e padrões adequados para apoio à tomada de decisão.

A visualização gráfica pode ser observada em todos os setores de análise, desde mapas, diagramas estatísticos, e gráficos [CARD, 1999]. Uma tabela de dados para ser analisada demanda tempo e atenção, com o uso do recurso de visualização gráfica essa atividade pode ser facilitada.

A possibilidade de estabelecer um padrão de análise de teste de software, através de uma base de conhecimento entre vários experimentos, poderá contribuir para a capacitação de novos gerentes de testes de software.

A tomada de decisão no caso de teste de software tem se tornado um grande dilema, visto que não é possível executar todos os testes necessários para

garantir que o software esteja sem defeitos, então a grande dificuldade é determinar quando parar de testar. Para esta tomada de decisão deverá ser considerada a confiança e criticidade para definir o padrão de qualidade requerido para o software testado.

1.2 - HIPÓTESES

A primeira hipótese tem origem na afirmação de que a visualização gráfica tem demonstrado eficiência e clareza para interpretação de um volume de informações. Para demonstrar a veracidade da hipótese, os dados simulados de teste foram digitados em uma tabela de dados brutos e depois a tabela foi transformada para melhorar a interação, mesmo assim quando comparado com relação à clareza, a visualização gráfica mostrou-se mais eficiente.

A segunda hipótese refere-se à afirmação de que a visualização gráfica pode reduzir o tempo necessário para tomar conhecimento das informações contidas nas tabelas, onde uma grande quantidade de informações exige um esforço maior para análise. Para verificar esta hipótese, os dados foram analisados diretamente na tabela de dados simulados e depois comparados com a visualização gráfica desses dados. Foi constatado que há um ganho de tempo quando utilizado a visualização gráfica.

1.3 – METODOLOGIA

O trabalho foi desenvolvido baseado em três fases. Na primeira fase foi realizado o levantamento bibliográfico onde foram pesquisados livros, artigos e teses que possibilitaram reunir informações de técnicas, definições e conceitos nas áreas de teste de software, visualização gráfica e estudos das dificuldades no processo de tomada de decisão durante a fase de teste de software. Na segunda fase foi desenvolvida uma ferramenta com o objetivo de avaliar as hipóteses levantadas. Para desenvolver a ferramenta foi feito estudos da linguagem de programação C# que é uma linguagem de alto nível e totalmente orientada a objetos. Na terceira fase foi feito um plano de simulações. Nesta fase foram feitas pesquisas para encontrar dados e tabelas utilizadas para a tomada de decisão durante a fase de teste e não foi encontrada nenhuma pesquisa contendo essas tabelas, sendo assim, foram

utilizados dados simulados que podem representar dados reais. Foram feitas simulações de interpretação de tabela, como é feito a tomada de decisão de uma forma tradicional e simulações de interpretação da mesma tabela utilizando a ferramenta desenvolvida utilizando visualização gráfica.

Foi observada durante os experimentos da ferramenta que realmente é muito difícil a interação com uma tabela e que a mente humana tem maior facilidade em analisar e interagir com visualização gráfica, comprovando a veracidade das hipóteses levantadas.

1.4 – ORGANIZAÇÃO DO TRABALHO

O Capítulo 2 apresenta a revisão bibliográfica destacando fundamentos teóricos de teste de software, visualização gráfica, indispensáveis para a realização deste trabalho.

O Capítulo 3 apresenta o desenvolvimento do protótipo da ferramenta, destacando os recursos utilizados.

O Capitulo 4 apresenta um experimento simulado, demonstrando a veracidade das hipóteses.

O Capítulo 5 apresenta a conclusão do trabalho, destacando as contribuições, sugerindo possíveis experimentos da ferramenta desenvolvida em situações reais.

O Capítulo 6 apresenta a bibliografia.

2 – REVISÃO DA LITERATURA

A complexidade na produção de um software compreende desde a maturidade da equipe ao tamanho e complexidade dos algoritmos. Durante o processo de desenvolvimento do software são encontrados defeitos de especificação de requisitos [PFLEEGER,2004], os quais são corrigidos pela equipe de desenvolvimento durante o processo de produção do software.

O processo de teste de software envolve técnicas e estudos que serão abordados neste trabalho de pesquisa. A maioria destas técnicas foi desenvolvida para linguagens estruturadas e, atualmente, estão sendo moldadas para as técnicas de desenvolvimento orientado a objetos, procurando adaptar novos conceitos de teste. O importante é ter conhecimento de que a atividade de teste de software é uma atividade requerida para a revisão do software e melhoria da qualidade

Na seção a seguir serão abordados os conceitos, técnicas e métodos de diversos autores sobre teste de software e visualização gráfica.

2.1 – Definições de Defeito, Erro, e Falha

A IEEE tem realizado esforços para padronizar e definir a forma de comunicação utilizada no processo de Engenharia de Software, mas ainda existem algumas definições divergentes devido à tradução. As definições adotadas para a realização deste trabalho sobre defeito, falha, erro, são as definidas por Vincenzi(2004) que são:

- Defeito (fault) – É um passo, processo ou definição de dados incorretos, uma instrução ou comando incorreto.
- Falha (failure) – É o resultado da ação de um defeito quando produz um resultado incorreto, ou seja, o resultado gerado é diferente do esperado.
- Erro (error) – É uma diferença entre um valor corrente e o valor esperado.

2.2 – Teste de software

A atividade de teste de software [PRESSMAN, 2005] é um elemento crítico de garantia de qualidade de software e representa a última revisão de

especificação, projeto e codificação. O teste de software é uma atividade da equipe de teste, que através de técnicas e ferramentas executam o software de forma controlada conforme Figura 2.1, um defeito é encontrado se a saída gerada for diferente da saída esperada.

Casos de teste são a combinação de uma entrada para o programa e a saída esperada segundo especificação. Através da execução do software utilizando como entrada o caso de teste, um resultado é gerado. A comparação do resultado esperado com o resultado gerado tem o objetivo de revelar defeitos. Os casos de teste são elaborados de acordo com as técnicas de teste que são: teste funcional e teste estrutural [CRESPO et al., 2004]. De acordo com PRESSMAN (2005) um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um defeito ainda não descoberto, e um teste bem sucedido é aquele que revela um defeito ainda não descoberto.

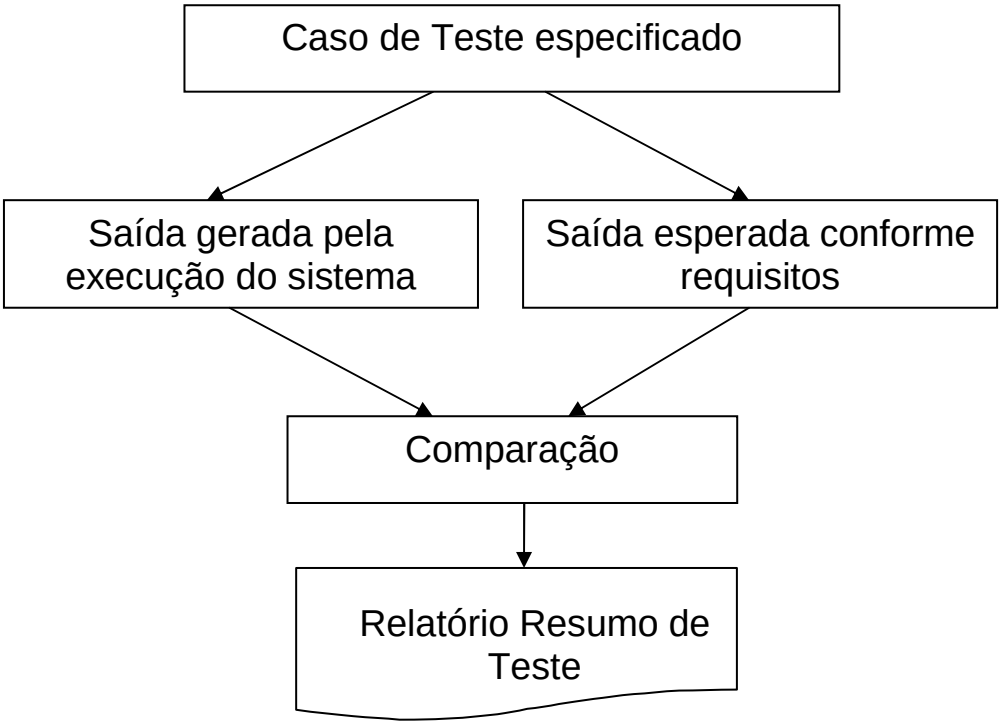

FIGURA 2.1 – Atividade de teste de software Adaptado de Crespo et al. (2004)

O relatório resumo de teste deve ser elaborado de forma resumida o resultado de cada caso de teste.

A Figura 2.2 mostra um processo de teste de software. Para projetar casos de teste, algumas políticas internas devem ser adotadas, como todas as declarações devem ser executadas pelo menos uma vez [SOMMERVILLE, 2003].

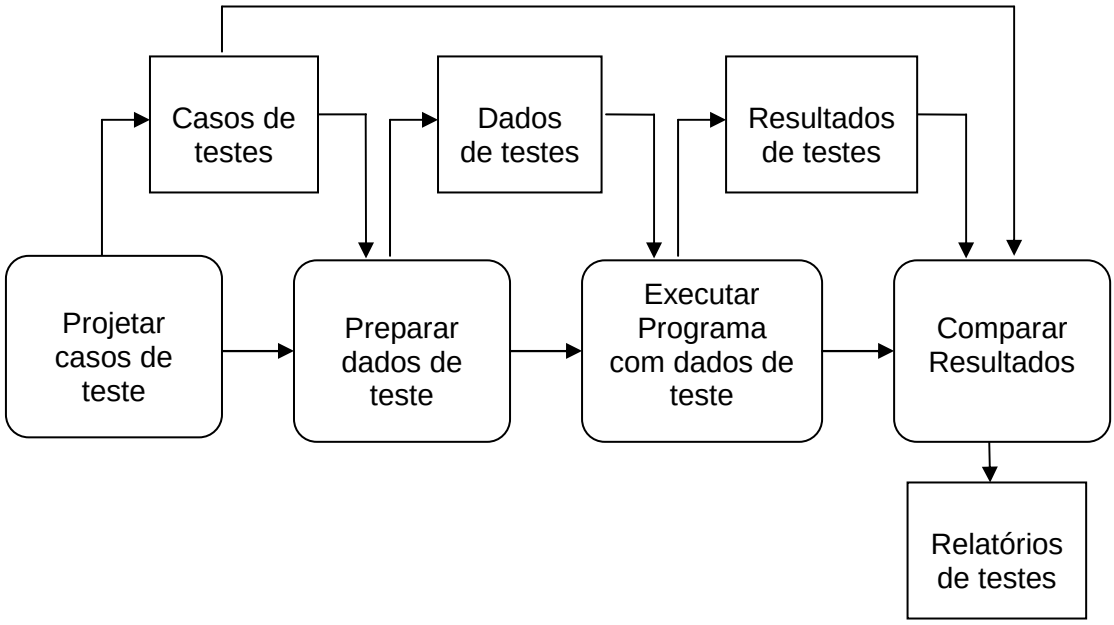

FIGURA 2.2 – Processo de teste para detecção de defeitos [SOMMERVILLE, 2003]

2.3 – Planejamento e Cronograma de teste de software

O planejamento de teste se ocupa em estabelecer padrões de teste, em vez de descrever teste de produtos [SOMMERVILLE, 2003]. O planejamento de teste auxilia no gerenciamento das atividades de teste. Na fase de planejamento são gerados os casos de teste e feito o levantamento e avaliação de riscos do software.

O cronograma de teste é elaborado com base nos requisitos do software, onde serão observadas, as prioridades de teste, bem como os recursos que serão utilizados. Nessa fase serão definidos a equipe, os critérios, as ferramentas, os prazos e roteiros de teste. O cronograma de teste é elaborado de acordo com as políticas organizacionais da empresa, buscando atender o processo de melhoria da qualidade do software. Para atender o processo de melhoria da qualidade, todo processo deverá ser documentado, e a utilização de métricas para medir e comparar o teste de forma a garantir a qualidade do teste de acordo com prazos, custos, e nível de confiança [SOMMERVILLE,2003] exigido pelo software.

A norma IEEE 829 define um conjunto de 8 documentos para a elaboração do planejamento, especificação, execução, e relatórios de teste de software conforme Figura 2.3.

É importante observar que muitas vezes são executados apenas testes caixa preta ou teste funcional (não utiliza o código fonte), dependendo da criticidade do sistema e da qualidade exigida. Alguns softwares têm um ciclo de vida [SOMMERVILE,2003] curto, como é o caso de softwares desenvolvidos para pesquisas na Internet, e o custo ou perda no caso da ocorrência de uma falha é mínimo, muitas vezes a empresa prefere assumir o risco.

2.4 – Metodologia de teste de software norma IEEE 829

A norma IEEE 829, 1983, revisão em 1998, estabelece procedimentos para documentação de teste de software. A utilização é voluntária, sendo que a norma reúne um amplo conhecimento sobre o assunto, a cada 5 anos está prevista uma revisão. Conforme a norma, a fase de teste pode ser dividida em etapas que são preparação do teste, execução do teste, emissão de relatórios de teste, no caso de um defeito será emitido o relatório de incidente de teste e encaminhado para avaliação e correção da equipe de teste. Após a correção o teste deverá ser repetido.

2.4.1 – Preparação de teste de software

A preparação do teste, Figura 2.3, de acordo com norma IEEE 829 é composta por quatro documentos:

- Plano de teste é a elaboração de um cronograma de teste, observando a especificação do projeto, recursos disponíveis, prazo para a execução do teste, e procedimentos de teste.
- Especificação do projeto de teste para elaboração deste procedimento, será considerada a especificação do procedimento de teste e especificação dos casos de teste. A Figura 2.3 mostra o relacionamento entre estas especificações.

- Especificação de casos de teste é elaborada de acordo com a especificação do projeto de teste, dos requisitos e pelas declarações de funções do software. O Caso de teste descreve uma entrada de dados de teste para uso no processo de teste de software, onde será comparado com uma saída esperada. Os casos de teste têm a finalidade de gerar dados de entrada de teste, normalmente são para testar os limites das sentenças do software e estas entradas podem ser válidas ou inválidos sendo os resultados do teste comparados com uma saída esperada.
- Especificação de procedimentos de teste é a especificação dos critérios de teste ou ainda um guia que enumera passo a passo a técnica requerida para executar o conjunto de casos de teste.

2.4.2 – Relatórios de teste de software

Os relatórios de teste de software são elaborados conforme norma IEEE 829, conforme Figura 2.3 e é composta por quatro documentos:

- Relatório diário de teste é o relatório onde é reportado cada caso de teste em ordem cronológica, e anotadas as observações relevantes. Relatório de Incidente de Teste ou *backlog* de defeitos é o registro ou documentação dos defeitos reportados durante o processo de teste, indicando o caso de teste e qual o defeito encontrado. No caso de um grande numero de *backlog* de defeitos poderá impedir a continuidade do processo de teste [KAN,2003]. Este relatório necessita de uma nova análise e correção pela equipe de programação. Muitas vezes um defeito pode gerar a necessidade de análise e revisão dos requisitos.
- Relatório resumo de teste – reporta de forma resumida a execução do teste.
- Relatório de encaminhamento de item de teste – reporta o encaminhamento dos itens de teste, muito útil para o gerenciamento de processos no caso de várias equipes de desenvolvimento.

Segundo Crespo et al. (2004) a aplicação de normas em Engenharia de Software exige um esforço adicional para adaptação e implantação.

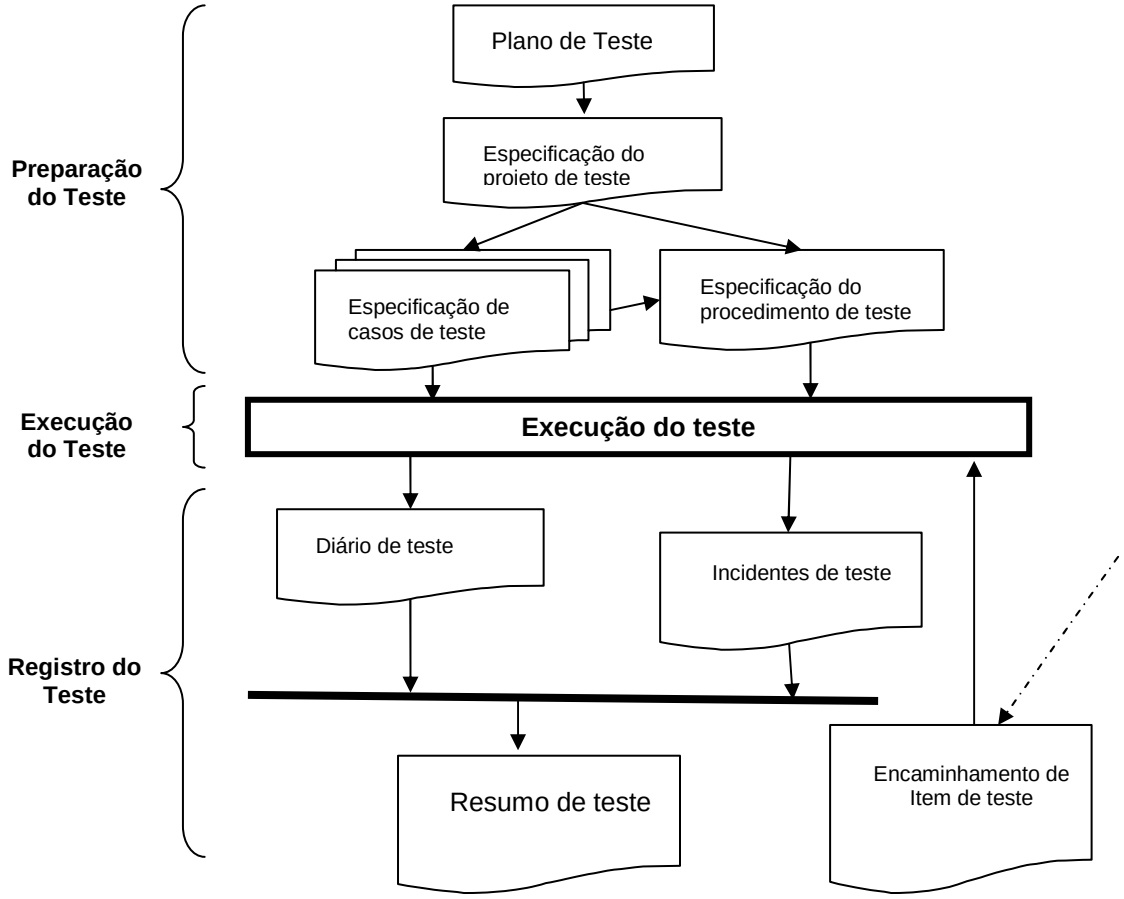

Figura 2.3 – Documentos de teste de software adaptado de Crespo et al. (2004)

2.5 – Conceitos de Confiança de Software

A confiança de um sistema está relacionada com a qualidade e integridade do sistema. Estes atributos são garantias de que o sistema não falhará. Não existe uma formula para a medição numérica dos atributos de confiança do software, por isso foram utilizados atributos que podem variar de não confiável, confiável, muito confiável, e ultra confiável [SOMMERVILLE,2003]. A confiança de um software poderá estar relacionada com o nível de abstração e confiabilidade da linguagem de programação, bem como, os métodos e técnicas de teste aplicadas, e estão relacionados ao uso, quando um sistema apresenta muitas falhas, os usuários normalmente classificam como um sistema não confiável, quando apresentam poucas falhas, a classificação é muito confiável e quando não apresentam falha a classificação é ultra-confiável (o nível de confiança está relacionado ao custo, quanto maior o nível de segurança maior será o custo conforme Figura 2.4. Assim é

impossível desenvolver e testar um sistema e garantir que este é totalmente seguro, pois o custo tenderia ao infinito). Um sistema de controle de navegação, controle de reator nuclear [SOMMERVILE,2003], são exemplos de sistema com elevado custo de desenvolvimento e teste, visto que são sistemas difíceis de determinar o valor ou prejuízo causado pela ocorrência de uma falha quando o software está em produção ou operação.

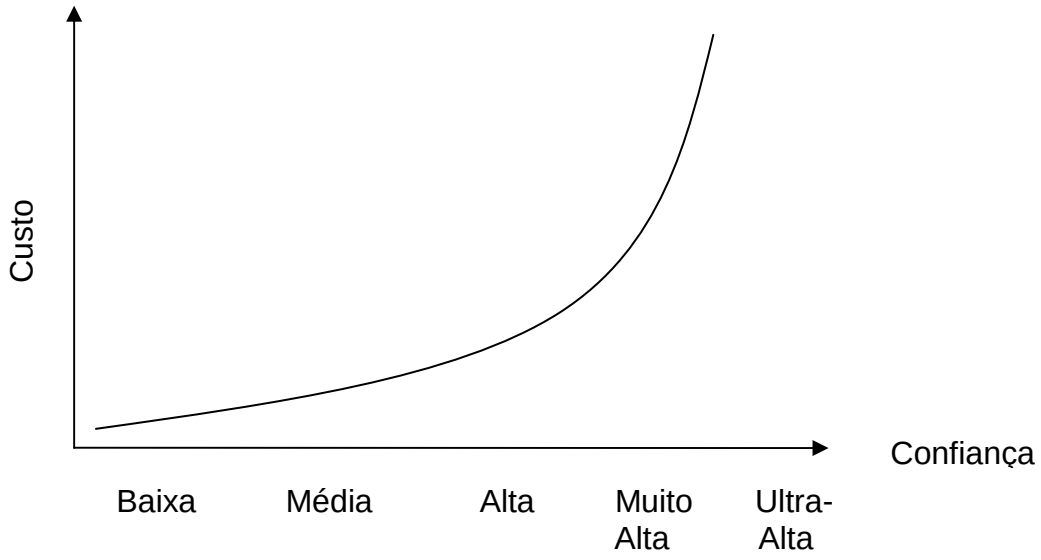

Figura 2.4 – Comparativo entre Confiança e Custos [SOMMERVILLE,2003]

Para uma análise comparativa entre custos, prazos de entrega e confiança devem ser observados a criticidade do software, pois muitas vezes o sistema não admite falha, devido ao custo ou perda no caso de uma falha ocorrer. Nos software que exigem um nível de ultra-segurança poderá ocorrer à dificuldade da revelação de falhas. Essa dificuldade ocorre no ponto em que os testes são realizados e nenhum defeito é revelado. Esse ponto é chamado de saturação de teste [HORGAN, MATHUR, PASQUINI, REGO, 1995], muitas vezes esses testes geram um custo alto, e demandam tempo, sendo que teste de software nesse ponto é feito em sistemas críticos que exigem um nível de confiança muito alto e ultra-alto.

Sistemas críticos [SOMMERVILLE, 2003] são sistemas que necessitam de um nível de confiança muito alto e ultra-alto. A probabilidade de ocorrer uma falha em sistemas críticos deve ser mínima, pois estão diretamente relacionados a perdas que podem ser:

- Financeiras, sistemas de controle de contas de bancos.
- Danos à natureza, sistemas de controle de produtos químicos.
- Vida humana, sistemas de controle de navegação de aeronaves.

2.6 – Ciclo de vida de software

O ciclo de vida [SOMMERVILLE, 2003] compreende desde a definição inicial do enfoque, passando pela definição dos requisitos, planejamento, desenvolvimento do software, teste de software, entrada em operação, manutenção, até a desativação do software. A Figura 2.5 mostra o ciclo de vida clássico também conhecido como modelo cascata [PRESSMAN, 2005]. O modelo cascata é uma seqüência para desenvolvimento de software conforme segue:

- Engenharia de Sistemas. Esta é a fase inicial da seqüência do modelo cascata, onde são definidos os requisitos em alto nível, focalizando mais o ambiente, interfaces, hardware, pessoas e banco de dados.
- Análise é a fase de especificação do software, onde são documentados os requisitos e revistos com o cliente.
- Projeto é a fase onde é documentada a estrutura de dados, arquitetura do software, de acordo com os requisitos.
- Codificação é a transformação do projeto em um software.
- Teste é a fase onde o software é executado de forma controlada, e são revistos as funcionalidades do software de acordo com requisitos, verificando se as entradas previstas estão iguais as geradas durante a execução do software.
- Manutenção é a fase onde novas funcionalidades podem ser incorporadas a pedido do cliente ou adaptações necessárias a continuidade do software em operação.

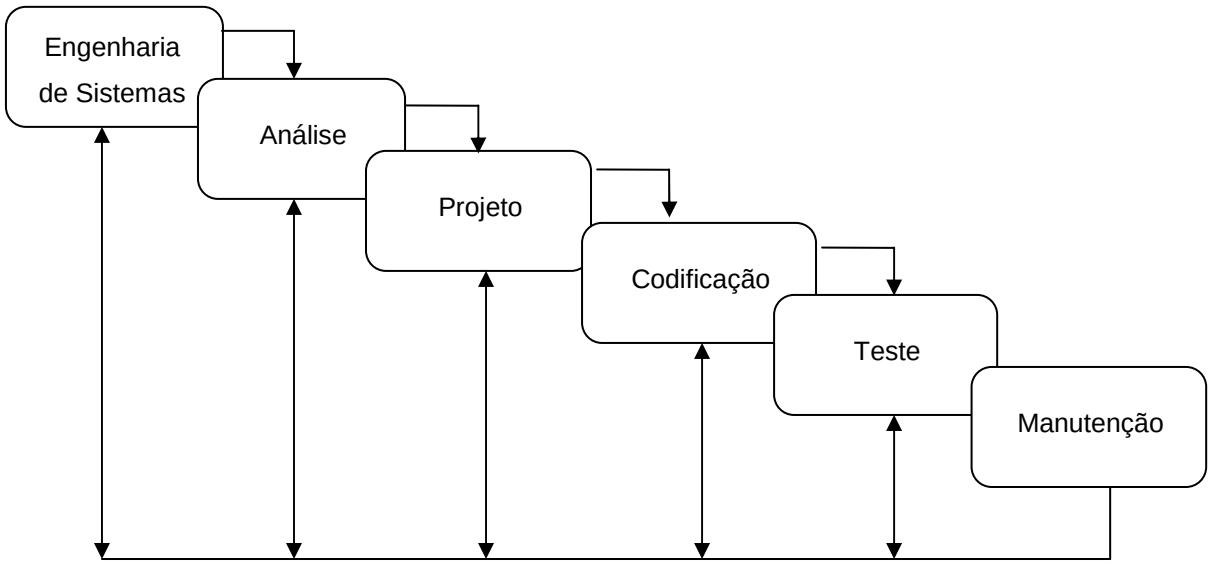

Figura 2.5 – Ciclo de vida clássico [PRESSMAN, 2005]

2.7 – Rational Unified Process (RUP)

O processo de desenvolvimento de software utilizando RUP é um processo para transformar requisitos em software utilizando uma seqüência de fases com interações seqüenciais, e consiste em gerenciar atividades e dividir tarefas e responsabilidades com a equipe de desenvolvimento [MANZONI, PRICE, 2003]. A meta do processo RUP é a de construir software de alta qualidade dentro de um cronograma, atendendo as necessidades do cliente e custos previstos [RATIONAL,2009].

As atividades para desenvolvimento do software são divididas em dois eixos, X e Y. No eixo X o processo é dividido em quatro fases, concepção, elaboração, construção e transição e no eixo Y, a modelagem de negócios é dividida em requisitos, análise e projeto, implementação, teste e distribuição. Conforme Figura 2.6 é possível verificar o esforço requerido em cada fase do processo, bem como demonstra que a fase de teste de software inicia desde a fase de consepção.

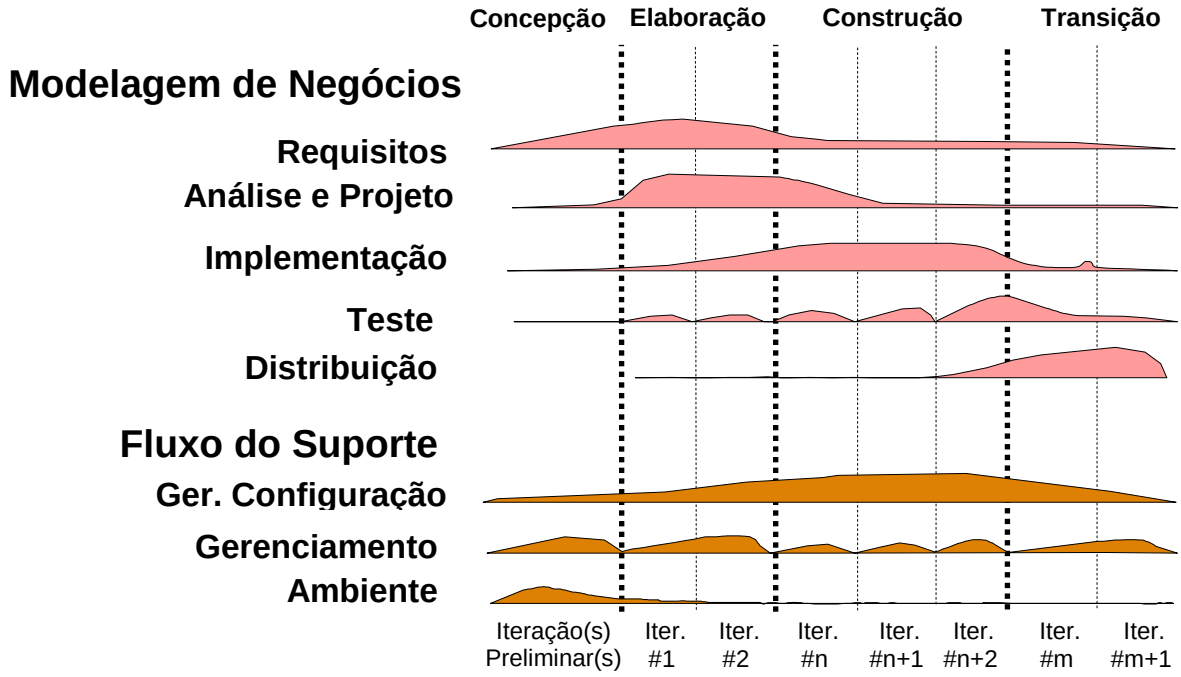

Figura 2.6 – Rational Unified Process (RUP) [RATIONAL, 2009]

2.8 – *Goal Question Metrics* (GQM)

O paradigma *Goal Question Metrics* (GQM) [BASILI, CALDIERA, ROMBACH, 1994], é usado para selecionar um conjunto de métricas de acordo com uma necessidade clara de acordo com cada meta ou objetivo. Por exemplo:

- Interesse, objetivo ou meta: de acordo com os interessados verificou-se a necessidade de reduzir defeitos e aumentar produtividade.
- Questões: as questões têm por objetivo responder se as metas serão atendidas. Por exemplo: Qual a taxa de defeito atual? Qual a taxa de defeito após a implantação do processo?
- Após as definições das métricas, novas questões serão geradas: Que dados serão necessários? Quais formatos? Como Coletar? Onde armazenar e como Utilizar?

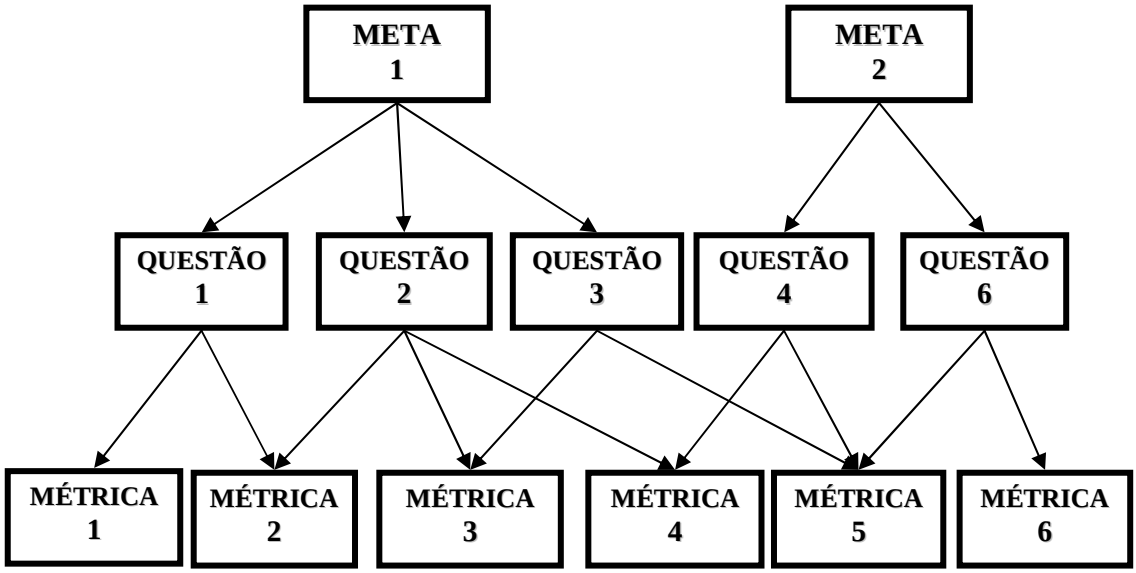

Figura 2.7 – *Goal Question Metrics* (GQM)
[BASILI, CALDIERA, ROMBACH, 1994]

O processo GQM de estrutura hierárquica [BASILI, CALDIERA, ROMBACH, 1994] conforme Figura 2.7. É iniciado com uma meta, onde são feitas várias questões para solucionar a meta, e geradas as métricas seletivamente para responder as questões. Uma mesma métrica poderá responder uma ou mais questões.

2.9 – Técnicas e critérios de teste de software

As técnicas de teste de software são adotadas para facilitar a revelação de defeitos durante a fase de teste. Quando o teste de software utiliza-se exclusivamente da especificação do software para derivar os requisitos de teste é classificada como teste funcional ou caixa preta, quando se utiliza de informações sobre a estrutura de implementação do software, como, por exemplo, o código fonte, para derivar os casos de teste, ele é classificado como teste estrutural ou caixa branca. Cada técnica de teste de software tem uma finalidade, desde testar as estruturas interna do software, testar as funcionalidades, testar a segurança entre outros.

2.9.1 – Teste Caixa Preta

O teste **caixa preta** (black box) [PRESSMAN, 2005] é também conhecido como teste funcional. Os casos de teste são elaborados baseando-se nos requisitos, onde não há necessidade do código fonte. O objetivo é testar funções, componentes, interfaces externas e demonstrar a integração do software. Outras técnicas de caixa preta podem e devem ser utilizadas de acordo com as necessidades ou restrições tecnológicas, entre elas o teste de desempenho, teste de usabilidade, teste de carga, teste de stress, teste de confiabilidade, e teste de recuperação.

2.9.2 – Teste Caixa Branca

O teste **caixa branca** (white box) [PRESSMAN, 2005] é também conhecido como teste estrutural, onde os casos de testes são elaborados de acordo com a estrutura ou código fonte do software. Os casos de teste caixa branca procuram exercitar cada condição, cada caminho, cada laço, cada estrutura de dados interna. Exige conhecimento da linguagem de programação, do código fonte do software, e quanto maior a complexidade maior será a dificuldade de gerar casos de teste. O teste caixa branca define os requisitos de teste a partir da estrutura do software, na realidade, mesmo sendo feito cuidadosamente. Em um exemplo apresentado por Pressman (2005), que poderia ser um programa PASCAL de 100 linhas e um único laço que pode ser executado não mais do que 20 vezes conforme Figura 2.8 há aproximadamente 10^{14} caminhos que podem ser executados. Para testar este programa, utilizando teste exaustivo, seria impraticável, devido à quantidade de casos de teste necessários e o tempo necessário para o teste. O teste exaustivo pode ser aplicado selecionando um número limitado de caminhos lógicos, levando em consideração a importância [PRESSMAN, 2005].

Pode ser feito uma combinação entre tipos de teste caixa banca com teste caixa preta procurando testar a interface com o software visando garantir o funcionamento correto do software.

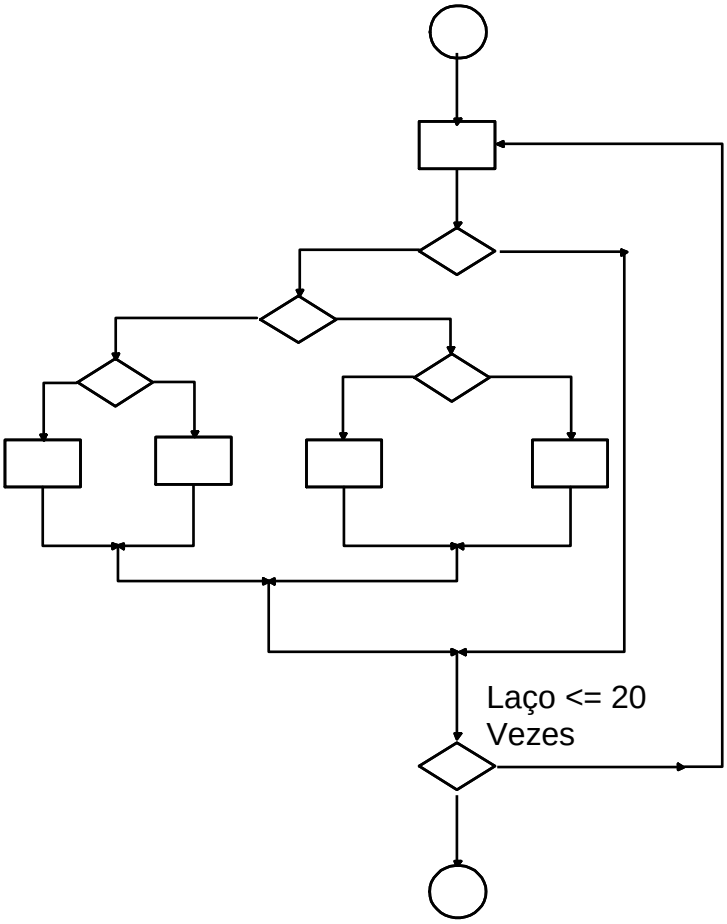

Figura 2.8 – Problemas com Testes Exaustivos [PRESSMAN,2005]

2.9.3 – Teste de Unidade

O teste de Unidade (caixa branca) é um tipo de teste cujo objetivo é testar a menor unidade [PRESSMAN, 2005] ou subsistema do software, observando a estrutura interna. O teste é elaborado baseado nos limites dos caminhos, visa revelar defeitos de lógica e de implementação. O Teste de Unidade permite o teste de uma classe, pequenos clusters, componentes, isoladamente. Para a execução do Teste de Unidade é necessário o desenvolvimento de *drivers e stubs.* Os *drivers* são usados para fazer chamadas das unidades testadas e enviar os parâmetros necessários para execução dessas unidades. Os *stubs* são as sub-rotinas chamadas pelas unidades testadas, essas sub-rotinas devem ser simuladas durante o teste de unidade, para garantir o retorno de valores corretos para cada chamada. Possibilitando o isolamento total da unidade testada.

2.9.4 – Teste de Integração

Teste de Integração complementa o teste de unidade, pois, seu objetivo é testar a integração entre as unidades testadas. De acordo com Pressman (2005), a maioria dos casos de teste gerada para o teste de integração é do tipo teste funcional ou caixa preta, pois não necessitam do código fonte. Quando um defeito é revelado nesta fase, o custo da correção é maior, pois é um defeito encontrado na integração de uma unidade já testada, necessitando muitas vezes revisar os requisitos.

2.9.5 – Teste de Usabilidade

Teste de Usabilidade é um teste do tipo caixa preta, pois não necessitam dos códigos fonte do software e de acordo com norma ISO 9241, têm por objetivo testar a aceitação, facilidade de aprendizagem, e interação entre o usuário e a interface do software. O teste verifica a qualidade observando a facilidade de uso, aprendizagem, produtividade, baixa taxa de erros cometidos pelos usuários, e satisfação dos usuários.

2.9.6 – Teste de Carga ou Teste de Stress

Teste de carga ou teste de stress (caixa preta) tem o objetivo de sobrecarregar o sistema até a ocorrência de uma falha para verificar o comportamento do software perante uma falha. Analisar a probabilidade de corromper a base de dados, a disponibilidade do software quando utilizado por muitos usuários, e elaborar planos de recuperação.

2.9.7 – Teste de Desempenho ou Teste de Performance

Teste de Desempenho ou teste de Performance, deverá simular um ambiente real [BARTIÉ,2002] onde o software será executado. Este tipo de teste verifica o comportamento do software comparando com os requisitos, quanto tempo de resposta, execução de um volume de transações simultâneas, quantidade de usuários e tráfego na rede.

2.9.8 – Teste de Recuperação

Teste de recuperação verifica a capacidade do software em permanecer operacional após a ocorrência de uma falha, ou condição anormal [BARTIÉ,2002], ou ainda o comportamento do software quando uma transação é interrompida, neste caso é verificado o retorno da transação ao estado inicial. Alguns softwares exigem segurança crítica, neste caso o sistema deverá permanecer operacional mesmo com a presença de defeitos de qualquer natureza, até que o defeito seja corrigido.

2.9.9 – Teste de Mutantes

O Teste de Mutantes [DeMillo, 1978] é a técnica de teste caixa branca que através de alteração do software original gera um mutante. Assim um mutante é uma versão do software original contendo uma pequena alteração. As alterações são feitas utilizando os operadores de mutação. Os operadores de mutação são operadores baseados em troca de variáveis ou operadores condicionais com o objetivo de gerar modificações sintáticas que são defeitos comuns gerados pelos programadores [VINCENZI, 2004]. O teste consiste em executar o software original utilizando um caso de teste gerando uma saída S1 e executar o mutante utilizando o mesmo caso de teste gerando uma saída S2. Se S1 for diferente de S2 o mutante é morto, se S1 igual a S2 o mutante permanece vivo. No caso do mutante permanecer vivo, isto é, S1 igual S2, o conjunto de casos de teste necessita ser melhorado ou pode se decidir que o mutante é equivalente ao software original. No caso de equivalência o mutante poderá ser desconsiderado. Se o mutante não for equivalente será necessário uma análise detalhada da estrutura interna do software onde ocorreu a mutação e analisar o motivo de S1 ser igual a S2,

2.9.10 – Efeito Saturação

Quando um tipo de teste é executado exaustivamente e a probabilidade de revelar defeitos é quase nula, ocorre o Efeito Saturação [HORGAN, MATHUR, PASQUINI, REGO,1995]. Mesmo assim, alguns recursos podem não ser testados, e não ser suficientes para garantir a qualidade do software testado, neste

caso pode ser feito uma nova etapa de teste utilizando outra técnica, até sua saturação, dependendo da criticidade do software, e assim sucessivamente conforme Figura 2.9. O efeito saturação ocorre em cada fase de teste, onde a dificuldade para revelar defeitos é maior, e conseqüentemente o custo para revelar defeito quando um determinado tipo de teste está saturado é elevado. A Figura 2.4 Comparativo entre Confiança e Custo, o ponto de saturação ocorre quando a confiança é muito alta e ultra-alta, pois aumentando a confiança pode ocorrer de nenhum defeito ser revelado.

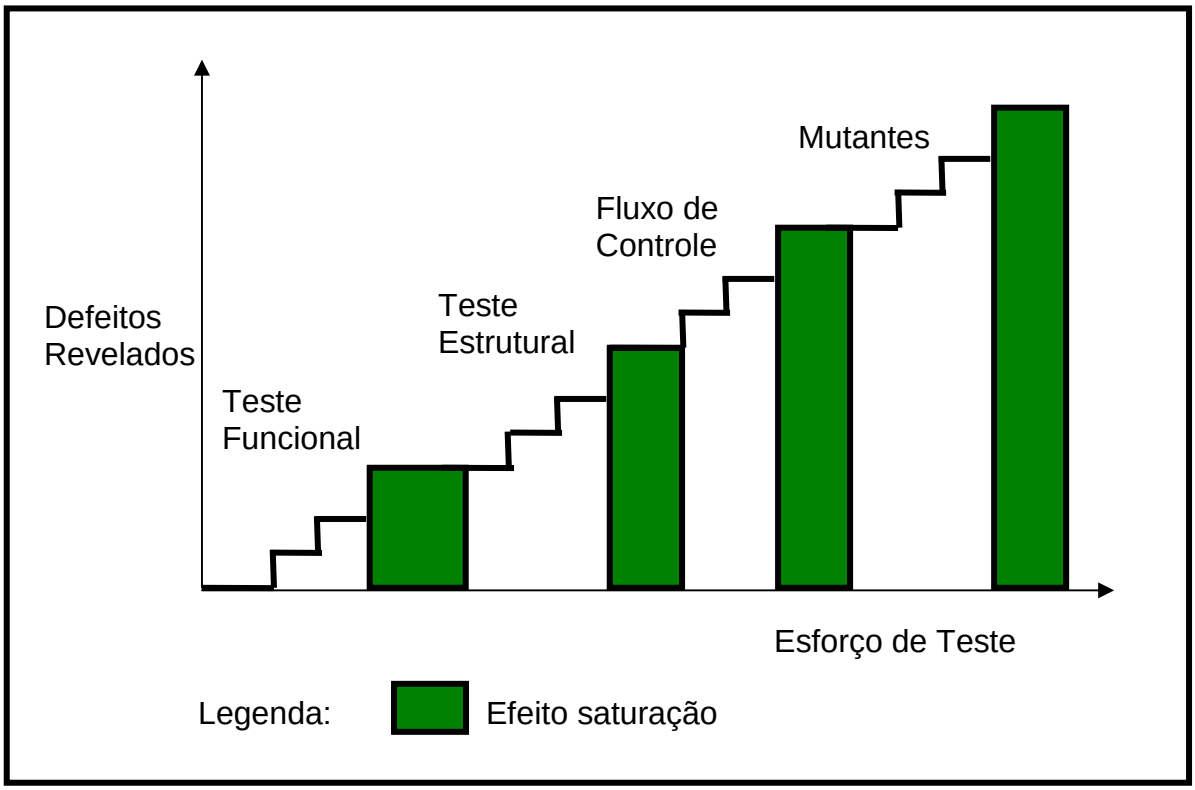

Figura 2.9 – Efeito Saturação [HORGAN, MATHUR, PASQUINI, REGO, 1995]

2.10 – Métricas

As Métricas de software tiveram origem na dificuldade e necessidade de medir e acompanhar a produtividade das equipes de desenvolvimento e teste de software. Para a utilização das métricas é necessário saber qual é objetivo para coletar os dados de acordo com este objetivo [HORGAN, MATHUR, PASQUINI, REGO,1995].

2.10.1 – Métrica Linhas de Código (LOC)

A abordagem sobre a métrica Linhas de Código (LOC) teve origem para medir a produtividade das equipes de desenvolvimento. As linguagens utilizadas eram Assembler, Cobol, Fortran e utilizavam cartão perfurado. Era fácil utilizar esta métrica, pois era só contar o número de cartões [SOMMERVILLE,2003].

Alguns estudos conforme citado por SOMMERVILLE (2003) contestam LOC, pois existe software com uma grande quantidade de linhas de código, e outro com menor quantidade de linhas de código executando as mesmas funcionalidades, assim LOC penaliza programas bem projetados [PRESSMAN,2005]. A linguagem de programação é outro fator que interfere nesta métrica, comparando o nível da linguagem de programação. Conforme o nível da linguagem de programação, um mesmo software desenvolvido com um número de linhas de código bem menor que outro com as mesmas funcionalidades mais com uma linguagem de programação de nível inferior.

2.10.2 – Ciência de software Halstead´s

A abordagem de Halstead (1977) *apud* KAN (2003) e PRESSMAN (2005) demonstra a diferença entre a ciência do software e a ciência do computador. A ciência do software é um conjunto de operadores e operandos que são distinguidos pelo compilador, assim é possível medir um software da seguinte forma:

N1	Número de operadores de um programa
N2	Número de operandos de um programa
N1	Número de ocorrências do operador
N2	Número de ocorrências do operando

Através das definições de Halstead desenvolveu-se um conjunto de fórmulas que permitem medir o tamanho do software, o volume, a complexidade, e até uma previsão do número de defeitos, entre outras. As principais fórmulas desenvolvidas, conforme Tabela 2.1, são:

Tabela 2.1 – Fórmulas conforme Halstead

Vocabulário (n)	n = n1 + n2
Tamanho (N)	N = N1 + N2 $N = n1 \log_2(n1) + n2 \log_2(n1)$
Volume (V)	$V = N \log_2(n)$ $V = N \log_2(n1 + n2)$
Nível (L)	L = V* / V L = (2/n1) * (n2/N2)
Dificuldade (D)	D = V / V* D = (n1/2) * (N2/n2)
Defeitos (B)	B = V / S*

Onde V* é o volume mínimo que um software, e S* é a constante que representa a probabilidade de erros (conforme Halstead S* = 3000), desta forma os defeitos quanto à dificuldade estão relacionados com o volume do software. V é o volume do software em bits e está relacionado com a linguagem de programação. De acordo com Halstead pode ser definido um volume mínimo para um programa de acordo com a linguagem de programação gerando um programa em sua forma compacta, medindo o volume, e depois gerando o programa real, e medindo o volume real.

As linguagens de programação conforme Halstead (1977) *apud* PRESSMAN (2005) podem ser classificadas em níveis (L) e são constantes para cada linguagem. Para as linguagens da Tabela 2.2, os valores dos níveis foram testados empiricamente.

Tabela 2.2 – Nível das linguagens de programação

Linguagem	Média L
Inglês Falado	2,16
PL/1	1,53
ALGOL/68	2,12
FORTRAN	1,14
Assembler	0,88

De acordo com a Tabela 2.2 [PRESSMAN,2005] quanto maior o nível da linguagem de programação maior será o nível de abstração em relação a linguagens de baixo nível como exemplo a linguagem Assembler.

2.10.3 – Análise de Pontos por função (APF)

A métrica Pontos por Função proposto por Albrescht (1979) e Albrescht e Gaffyei (1983) *apud* SOMMERVILLE (2003), tem a finalidade de medir as funcionalidades do ponto de vista do usuário. Desta forma o resultado desta medida pode gerar o mesmo valor, independentemente da linguagem de programação ou experiência do desenvolvedor, pois mede o que o software faz. Um software desenvolvido em uma linguagem A e outro desenvolvido em uma linguagem B, onde ambos possuem as mesmas entradas e mesmas saídas, mesmas interfaces, mesmos relatórios, eles possuem a mesma quantidade de Pontos por Função, mais com o número de linhas de código (LOC) diferentes. Este é um dos motivos que diversos estudos contestam o uso de LOC. Outro motivo é que LOC depende da experiência do desenvolvedor.

O método de cálculo da métrica pontos por função utiliza funcionalidades do tipo:

- Entradas e saídas externas;
- Interações com usuários;
- Interfaces externas;
- Arquivos utilizados pelo sistema.

O cálculo dos pontos por função não ajustado (unadjusted function-point count UFC) é a somatória de cada contagem bruta pelo peso da contagem.

$$UFC = \sum (\text{contagem bruta}) * (\text{peso})$$

2.10.4 – Métrica complexidade ciclomática de McCabe

A complexidade ciclomática de McCabe (1976) é a métrica de software definida pelo número de caminhos independentes em um subsistema do software. A

complexidade ciclomática de McCabe pode ser usada para comparar a complexidade dos subsistemas do software e para comparação de complexidade entre softwares [KAN,2003]. Um software com uma baixa complexidade ciclomática de McCabe é um software de fácil compreensão e consequentemente maior será a facilidade de manutenção. A Figura 2.10 mostra um exemplo de gráfico de um programa simples, onde as letras representam as sentenças (nós), e as setas os arcos. A fórmula para cálculo da complexidade ciclomática de McCabe é:

M = V(G) = e – n + 2p
e = número de arcos
n = número de nós
p = número de partes desconexas do grafo

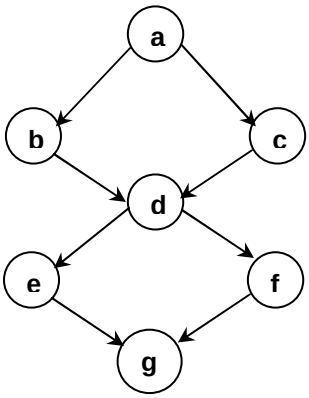

Figura 2.10 – Complexidade ciclomática de McCabe.

A complexidade de ciclomática de McCabe de um grupo de gráficos é igual à soma de cada gráfico [KAN,2003], sendo assim, a complexidade de um software é igual à soma da complexidade de cada subsistema do software.

No exemplo de um gráfico com duas sentenças IF, sete nós e uma parte desconexa temos:

M = V(G) = e – n + 2p
e = 8, n = 7, p = 1
M = 8 – 7 + 2 * 1
M = 3

Para uma boa facilidade de compreensão e facilidade de teste McCabe recomenda-se que a complexidade ciclomática V(G) não ultrapasse o valor 10. O uso da métrica de complexidade ciclomática vem ganhando aceitação e praticantes [KAN, 2003], por ser um recurso com facilidade de aplicação. Troster (1992) segundo KAN (2003) encontrou uma grande relação entre a complexidade ciclomática de McCabe e o número de defeitos encontrados. A Tabela 2.3 compara a complexidade ciclomática de McCabe com a avaliação de risco de software [VANDOREN; SCIENCES; SPRINGS, 2000]. Para complexidade entre 1 e 10 o programa é simples, sem risco, e fácil de testar, e quanto maior a complexidade de McCabe maior será o risco, e o programa será mais complexo. Quanto maior a complexidade de McCabe [BARTIÉ,2002], maior será a dificuldade de teste, e maior será a dificuldade de garantir a qualidade e confiabilidade do software. Portanto, é aconselhável procurar produzir software com a complexidade menor possível, pois software com complexidade ciclomática alta pode ter uma elevada dificuldade de teste ou até a impossibilidade de teste.

Tabela 2.3 – Avaliação da complexidade ciclomática de McCabe [KAN, 2003]

Complexidade Ciclomática	Avaliação de Risco
1 – 10	Programa Simples, sem muito risco.
11 – 20	Mais complexo, risco moderado.
20 – 50	Complexo, programa de alto risco.
Maior que 50	Programa intestável (risco muito alto)

2.11 – Tomada de decisão durante a fase de teste

As empresas estão cada vez mais buscando a qualidade, e assim buscam construir software com a probabilidade de falhas cada vez menor, procurando evitar que falhas possam interferir no bom funcionamento do software.

Na fase de teste de software o processo de revisão envolve uma série de decisões relacionadas à qualidade do software. Essas decisões podem sofrer alterações visto que a atividade de teste é uma atividade que se repete enquanto o software estiver com seu ciclo de vida ativo, pois a cada execução do software um

novo conjunto de dados é testado [PRESSMAN,2005], assim sendo, após a entrada em operação a atividade de teste de software é transferida para o cliente.

A quantidade de falhas reveladas em relação ao tempo de teste de software é mostrada na Figura 2.11. A quantidade de falhas reveladas diminui com o aumento do tempo de teste até estabilizar [PRESSMAN,2005], assim, a decisão de quando parar de testar é determinada pelo nível de confiança do software que determina a quantidade de falhas aceitáveis para não interferir na qualidade do software.

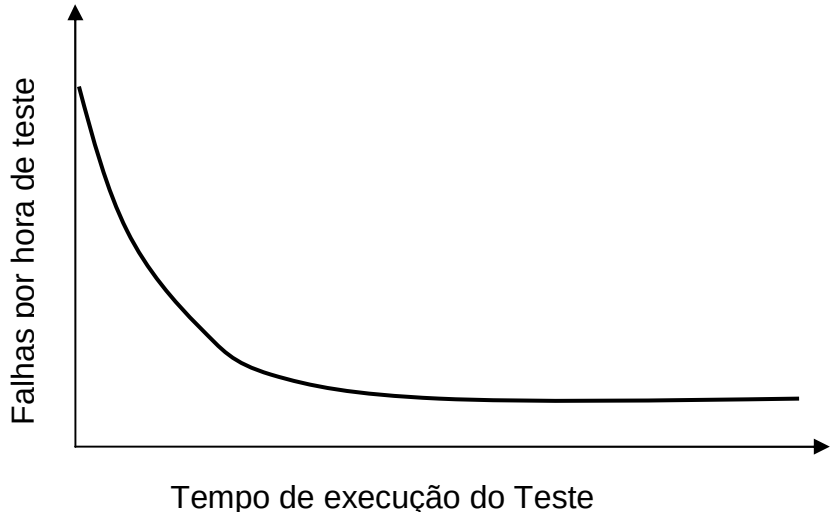

Figura 2.11 – Intensidade de Falhas como uma função do tempo de execução [PRESSMAN, 2005]

2.12 – Avaliação de riscos

A avaliação de riscos [SOMMERVILLE, 2003] é uma forma encontrada para avaliar a probabilidade de uma falha ocorrer e sua gravidade, e pode ser classificada em níveis de aceitação como segue:

- Intolerável para sistemas que não admitem falhas e se uma falha ocorrer não resulte em um acidente.
- ALARP (as low as reasonably pratical) tão baixo quanto for razoavelmente prático para sistemas projetados com redução da probabilidade de falhas e que na ocorrência de uma falha seja avaliado o custo e prazo de entrega.

- Aceitável para sistemas projetados para reduzir a probabilidade de falhas, levando em consideração o prazo de entrega, custos e outros atributos não funcionais do sistema.

Para gerenciar e analisar os riscos pode ser feito um *checklist* [BOEHM, 1991] onde deverão constar os riscos de requisitos, risco da equipe, riscos técnicos (ferramentas e ambiente) e para cada risco constar o impacto no caso de uma falha ocorrer. Conforme Boehm (1991) existem três estratégias para controlar os riscos:

- Prevenção de riscos, através de revisões no projeto procurando evitar o risco.
- Transferência de risco, transferir a responsabilidade pelos riscos para alguém (Clientes, fornecedores, etc.). Transferindo inclusive custos dos riscos.
- Aceitação de risco, admitir e assumir os riscos e definir, projetar ações corretivas no caso de risco.

2.13 – Quando parar de testar

Uma das maiores dificuldades durante a fase de teste de software é saber quando parar de testar [EHRLICH *at all*,1993]. Se parar de testar muito cedo, poderão ocorrer falhas, e gerar insatisfação do cliente, e o custo de recuperação e correção poderá ser muito alto. No entanto se o teste continuar por um período longo, o custo será elevado, bem como o prazo de entrega, podendo representar a diferença entre o sucesso e o fracasso do software.

2.14 – Definições de Visualização Gráfica

O avanço da ciência e comércio tem sido caracterizado por invenções que tem permitido que antigos pensamentos sejam vistos de novas maneiras [CARD,1999]. As representações visuais tais como mapas, diagramas estatísticos, gráficos são exemplos. No entanto a visualização pode ser usada de forma interativa com o objetivo de aumentar a cognição ou conhecimento humano [CARD,1999]. Segundo CARD (1999), “***Uma imagem vale dez mil palavras***” pensando assim, a visualização gráfica poderá ser usada como uma ferramenta para

perceber, tornar conhecido, e resolver problemas lógicos. A visualização detém uma grande promessa para a ciência computacional e engenharia, fornecendo de imediato e em longo prazo ferramentas para usuários e ferramentas para tomada de decisão [CARD,1999].

2.14.1 – Atividades e capacidades da mente humana

.

Através da multiplicação podemos analisar as atividades da mente humana, é fácil de entender, basta observar que para multiplicar os números 34 X 72 mentalmente pode demorar um longo tempo. O fato é que a mente humana tem dificuldades em guardar resultados parciais [CARD,1999]. Utilizando um lápis e um papel para anotar os resultados parciais, esta tarefa poderá ser resolvida em um menor tempo. O fato de anotar estes resultados parciais pode ser considerado uma forma de visualização dos resultados para executar o cálculo ou tarefa. O lápis e o papel são os dispositivos ou ferramentas de representação visual.

2.14.2 – Ferramentas de Exploração Visual

As ferramentas de exploração visual de informações possuem as seguintes funcionalidades [CARD, 1999]:

- Atributos – são recursos que permitem a utilização dos atributos visuais tais como: forma, cor, tamanho, posição, orientação, curvatura, etc. para facilitar a interpretação.
- Navegação – possibilita a aproximação, rotação, reposicionamento de determinada área da visualização gráfica.
- Interação – possibilitam o uso de mecanismo de seleção de atributos visuais.
- Controle – possibilita o controle seletivo dos dados utilizados para gerar a visualização.

O modelo apresentado na Figura 2.11 mostra como é possível interagir com uma base de dados bruta e gerar um novo subconjunto de dados, através de filtros ou algoritmos, possibilitando gerar várias visualizações diferentes de acordo com o parâmetro do filtro selecionado, o que facilita a cognição humana

[CARD,1999] para a realização de uma tarefa. A transformação de dados ocorre na geração da nova tabela, quando são aplicados cálculos, ordenação de dados, algoritmos ou qualquer forma onde uma nova tabela derivada da original é gerada. A tabela gerada com novas variáveis deverá representar os dados originais. A transformação envolve perda ou ganho de informações [CARD,1999].

O mapeamento visual é a transformação da tabela em estruturas visuais. As estruturas visuais são atributos de visualização que representam a tabela. De acordo com Mackinlay (1986) *apud* CARD (1999) para uma boa estrutura visual é importante preservar os dados. As estruturas visuais poderão facilitar ou dificultar o entendimento ou a interação humana da visualização. O usuário utiliza os recursos de interação para obter uma visualização gráfica que facilite a realização de sua tarefa. Quando um controle é selecionado, uma nova visualização é gerada.

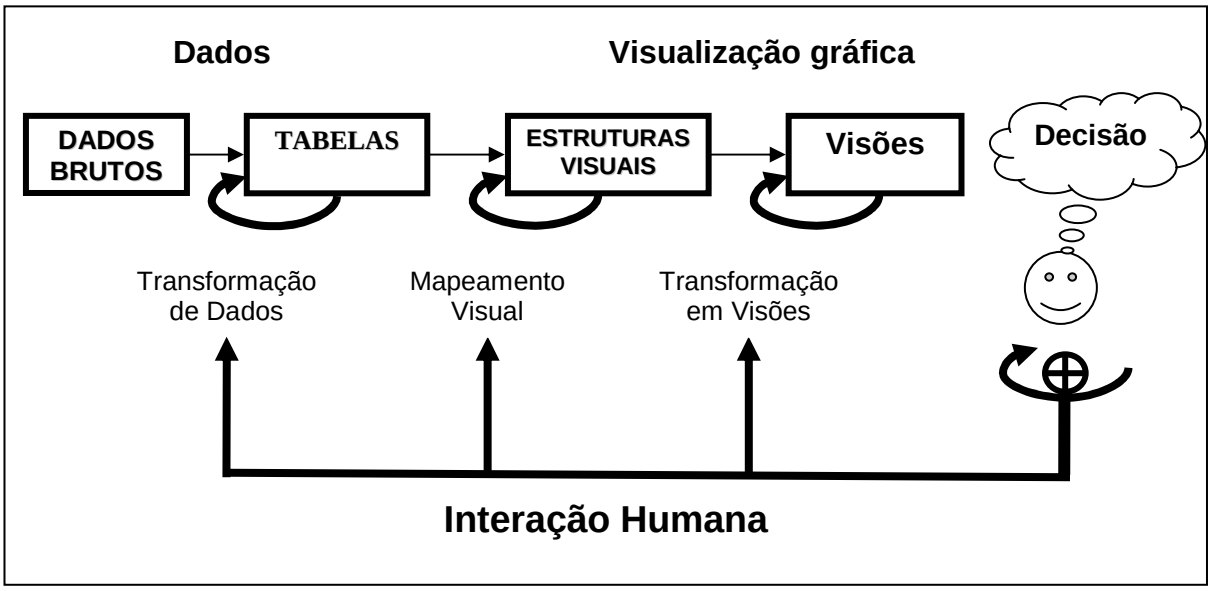

Figura 2.11 – Modelo adaptado de referência para visualização [CARD,1999].

2.14.3 – Filtragem Interativa

A técnica de filtragem interativa possibilita explorar uma base de dados seletivamente. A seleção pode ser feita por meio de filtragem de atributos, gerando um novo subconjunto de dados, que pode ser usado para localizar dados em tabelas, revelarem um padrão em uma base de dados ou selecionar argumentos para transformação dos dados. A filtragem não altera a base de dados original [CARD,1999], e facilita a comparação entre os dados.

2.14.4 – Percepção visual humana e a capacidade cognitiva

A percepção visual humana pode rapidamente perceber um padrão escondido em uma base de dados através da visualização gráfica, limitando-se a sua memória e atenção, devida sua capacidade cognitiva ou capacidade de conhecimento. O uso da visualização gráfica pode acelerar a percepção dos dados [CARD,1999].

Uma variedade de mecanismos de processamento de informações pode ser utilizada para interagir com as informações [CHI;CARD,1999]. Para obter a compreensão das informações através da percepção visual, podem ser realizadas operações sucessivas conforme Figura 2.12. A seqüência de mecanismos e operações para interagir com as informações são:

1. Definindo o problema: Quando um problema é encontrado, primeiro será necessário definir o problema, através da formulação de questões. As perguntas são formuladas como hipóteses.
2. Definindo os dados: Depois de formuladas as questões, será necessário o registro dos dados que responderão as questões em uma tabela de dados brutos.
3. Escolhendo uma análise abstrata: Os dados deverão ser registrados em uma estrutura de dados de fácil percepção para consultas posteriores. A seleção dos dados é realizada através de algoritmos que formarão uma nova tabela.
4. Escolhendo uma visualização abstrata: Dependendo da estrutura analítica e das técnicas de visualização disponíveis, a escolha pode ser feita em uma variedade de visualizações e abstrações. A visualização de muitas informações em um gráfico poderá ser difícil, no entanto algumas técnicas podem ser adotadas para formar uma nova visualização abstrata.
5. Adotando uma linguagem de transformação: Esta etapa será adotada operadores apropriados para executar a tarefa.
6. Transcrevendo os valores em nova visão: Esta etapa gera a visualização, permitindo ao usuário ter uma visão das respostas das hipóteses geradas no item 1.

7. Processando valores ou visões: Através dos controles interativos como rotação, redimensionar (zoom), filtros, o usuário poderá alterar características visuais com o objetivo de chamar a atenção.
8. Interpretando e decidindo: O processo cognitivo ocorre quando o usuário analisa as visões procurando obter conhecimento através de características visuais obtidas através das visões.
9. Interagindo: Depois de analisar as visões, e obter conhecimento através das visões, novas perguntas poderão surgir, novas visões serão construídas para responder as questões. O processo se repete até o usuário obter domínio da situação através do conhecimento.

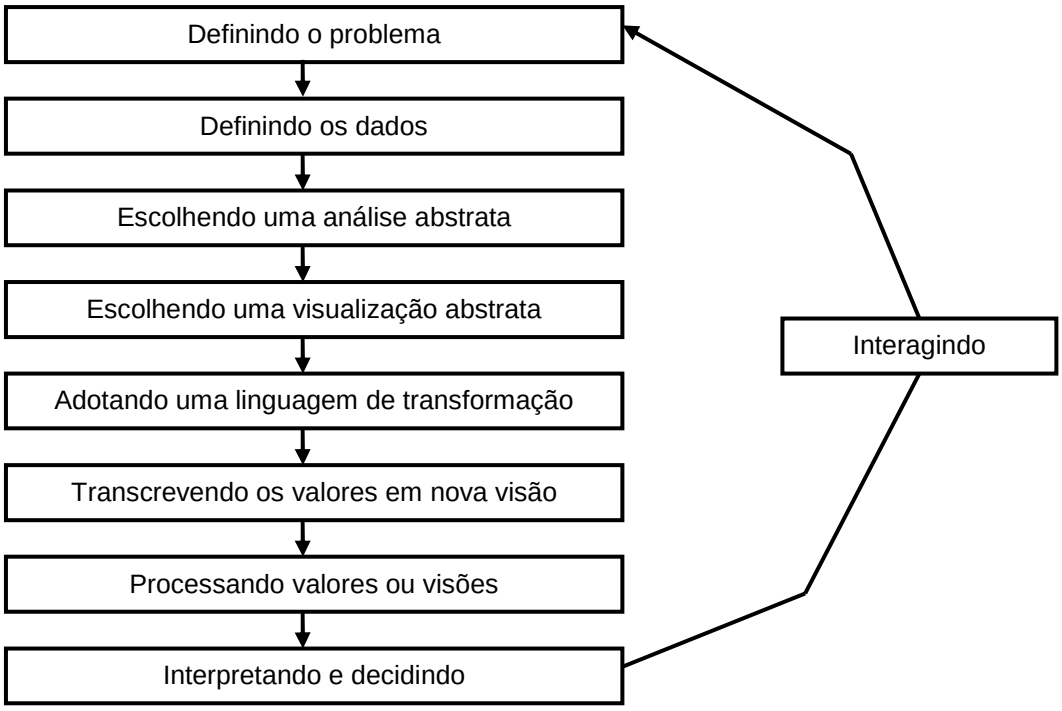

Figura 2.12 – Ciclo da Percepção Visual modelo adaptado [CHI;CARD,1999].

O objetivo do ciclo de interação é melhorar a percepção das informações contidas nos gráficos através da repetição do processo, alterando apenas controles e atributos. A cada ciclo uma nova visão é processada, e um novo conhecimento é descoberto.

3 – ESPECIFICAÇÃO DO PROJETO

O processo de tomada de decisão em teste de software é um processo que exige experiência do gerente de teste, e da equipe de teste. A tomada de decisão está diretamente relacionada com a qualidade e confiabilidade do software. Outro fato importante é a experiência das equipes de levantamento e análise de requisitos, projetos e implementação, bem como o nível da linguagem de programação. A maior dificuldade da tomada de decisão está no quando parar de testar, onde, o gerente de teste de software tem que levar em consideração, a criticidade do software, prazo de entrega e custos.

A ferramenta desenvolvida neste trabalho gera visões através de gráficos, seguindo a seqüência de operações definidas por CARD (1999), Figura 2.11, e utiliza os atributos cores, tamanho, altura, posição, controle seletivo, e controle interativo, sendo que esses mecanismos de controle e interação seguem as definições de CARD (1999). Utilizando esses conceitos, a ferramenta apóia um gerente de teste no processo de decisão, pois através das visões o gerente recebe informações e toma conhecimento de forma simples e clara do processo de teste de software, sendo que as visões representam as métricas selecionadas.

Conforme citado por Pressman (2005) o esforço de teste de software consome cerca de 40% do esforço total do projeto em teste. Ele cita também que no caso de testes de sistemas críticos que envolvem vidas humanas (por exemplo, controle de vôo, monitoração de reatores nucleares), o teste de software pode custar de três a cinco vezes mais que todos os outros passos de Engenharia de Software. Através dessa afirmação, o investimento na construção de ferramentas de apoio a tomada de decisão durante a fase de teste de software pode contribuir com o processo de teste de software gerando padrão de tomada de decisão, reduzindo custos e prazos de entrega.

Durante o processo de teste de software, podem ser adotados vários tipos ou critérios de teste. O ponto de saturação de teste [HORGAN, MATHUR, PASQUINI, REGO, 1995] ocorre quando um tipo de teste foi exercitado até o ponto em que praticamente esgotaram os casos de teste e não há mais defeitos a serem revelados. Neste ponto, o profissional deverá utilizar um novo tipo de teste até atingir a saturação.

Este trabalho procura investigar se é possível estabelecer padrões e critérios no processo de teste de software e formar uma base de conhecimento das métricas selecionadas. Através da base de conhecimento melhorar a qualidade das informações de teste de software, reduzindo prazos e custos. Sempre levando em consideração a visualização gráfica e investigando se a visualização gráfica pode apoiar o gerente de teste na tomada de decisão.
As medidas utilizadas no processo de visualização gráfica deste trabalho são:

- Métricas orientadas ao tamanho LOC.
- Métricas de Complexidade.
- Número de casos de teste planejados.
- Número de casos de teste executados.
- Número de casos de teste com defeitos detectados.
- Número de defeitos solucionado
- Prioridade do cliente.

3.1 – DESCRIÇÃO DO PROBLEMA

A Tabela 3.1 demonstra a facilidade de entendimento e análise de dados no caso onde a tabela contém poucas informações.

Tabela 3.1 – Tabela simples, fácil entendimento

Subsistemas	Complexidade ciclomatica
Subsistema 1	5
Subsistema 2	6
Subsistema 3	4
Subsistema 4	5
Subsistema 5	2
Subsistema 6	4

Para analisar uma tabela com várias colunas é fácil notar a dificuldade para o conhecimento das informações necessárias para a tomada de decisão no gerenciamento do processo de teste do software proposto conforme Tabela 3.2,

essa dificuldade vem da dificuldade da mente humana em armazenar subtotais. Além dessa dificuldade existe pode ser verificado que os valores quantitativos possuem pesos diferentes. O valor da complexidade ciclomática do Subsistema 1 é 5, O número de linhas de código do Subsistema 1 é 600. Pode ser notado que não existe uma relação entre as quantidades. A visualização gráfica desta tabela poderia levar a erros de interpretação. Para solucionar esta dificuldade foi necessário atribuir pesos para equilíbrio dos valores quantitativos. O valor dos pesos pode ser alterado para que uma determinada coluna possa chamar a atenção do gerente de teste conforme determinada situação. Como por exemplo, realçar a prioridade do cliente é somente aumentar o peso dessa métrica.

Tabela 3.2 – Tabela complexa, muitas informações

Subsistemas	Complexidade	LOC	Cliente	Previstos	Executados	*Backlog*	Correção
Subsistema 1	5	600	3	480	420	250	190
Subsistema 2	6	120	1	420	390	250	200
Subsistema 3	4	123	2	390	330	320	300
Subsistema 4	5	233	5	480	75	220	160
Subsistema 5	2	321	4	510	450	220	200
Subsistema 6	4	240	6	390	330	350	280

O objetivo principal do desenvolvimento da ferramenta é investigar se é possível utilizar visualização gráfica para apoiar um gerente de teste de software na tomada de decisão, sendo assim não foi considerada a importação de dados, bem como descrever todas as etapas e diagramas utilizados. Desta forma a ferramenta desenvolvida será utilizada para demonstrar que pode apoiar no entendimento da tabela facilitando e agilizando o processo de decisão durante a fase de teste. A ferramenta foi desenvolvida conforme modelo de referência de visualização gráfica de CARD (1999), Figura 2.11, e seqüência de funcionalidades conforme segue:

- Atualização dos dados brutos da tabela principal através das funcionalidades da ferramenta (Alterar, Incluir, Excluir).
- Os dados brutos da tabela principal serão filtrados de acordo com a data selecionada. Nesta fase são aplicados os algoritmos de filtro.

- Os dados serão processados utilizando-se de pesos e transformados para uma nova tabela.
- Os dados transformados, utilizando recursos interativos, e estruturas visuais, geram padrões visuais.
- Os padrões visuais se transformarão em visões que são os gráficos interativos.
- Através das visões o gerente de teste de software analisa e toma a decisão. Caso necessário, novas visões podem ser geradas alterando os atributos da ferramenta, com a finalidade de melhorar a qualidade da visualização.

3.2 – CASOS DE USO

A ferramenta de apoio a tomada de decisão foi desenvolvida conforme diagrama casos de uso Figura 3.1, da seguinte maneira:

1. Atualizar dados: A atualização de dados poderá ser feita pela equipe de teste de software ou pelo gerente de teste. Esta atividade consiste em:

    - Cadastrar métricas: O cadastro de métrica é utilizado para incluir, alterar e excluir métricas. Este cadastro possibilita cadastrar as métricas e medidas para acompanhamento e apoio ao teste de software.
    - Cadastrar subsistemas: Este cadastro é utilizado para incluir, alterar e excluir subsistemas.
    - Cadastrar diário de teste: Este cadastro é utilizado para incluir, alterar e excluir dados do diário de teste.

2. Gerar visualização gráfica: Esta funcionalidade possibilita selecionar opções processar visualização é a atividade do gerente de teste, onde poderá selecionar controles e filtros de seleção, bem como o tipo de visualização gráfica adequada para apoiá-lo na tomada de decisão.

O apoio a tomada de decisão vem do conhecimento das informações adquiridas através da percepção da visualização gráfica. A ferramenta possibilita gerar várias visões através dos mecanismos de filtros e controles.

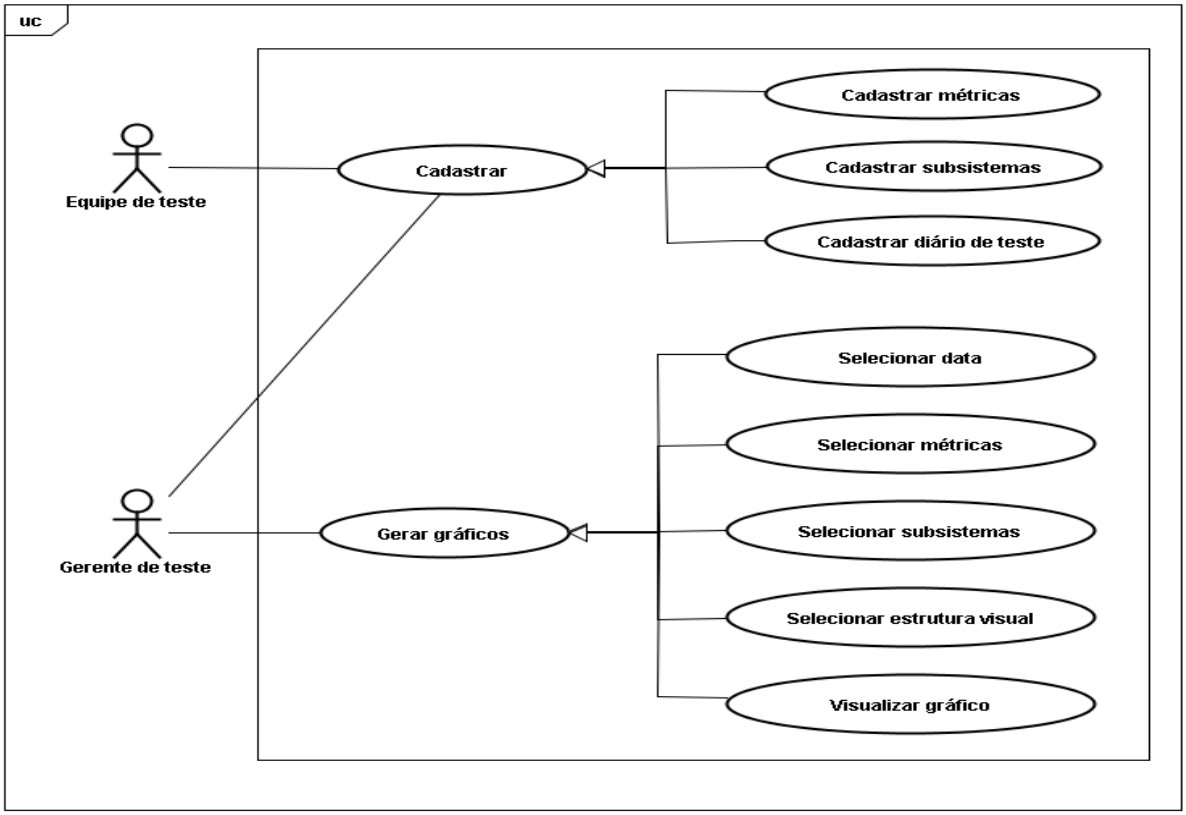

Figura 3.1 – Diagrama casos de uso

3.3 – DIAGRAMAS DE SEQÜÊNCIA

Os diagramas de seqüência foram elaborados utilizando UML 2.0 através da ferramenta JUDE Community freeware. Foi escolhida a modelagem Unified Modeling Language (UML) por ser uma modelagem de fácil comunicação entre as pessoas envolvidas com o projeto e facilitar a especificação, documentação e visualização de sistemas orientados a objeto.

Cada diagrama de seqüência tem como objetivo mostrar as interações cada caso de uso descrevendo a troca de mensagens entre os objetos, ou seja, como o sistema age internamente.

3.3.1 – DIAGRAMA DE SEQÜÊNCIA CADASTRAR MÉTRICAS

O cadastro de métricas é a funcionalidade que oferece a flexibilidade de seleção de métricas de acordo com o objetivo a ser analisado, permitindo ainda atribuir pesos a cada métrica. O objetivo do peso é possibilitar que valores pequenos possam ser representados graficamente.

O diagrama de seqüência Figura 3.2 descreve as funcionalidades e troca de mensagens entre os objetos conforme segue:

1. Pesquisar: permite selecionar métricas, se nada for digitado o sistema considera todas as métricas.
2. Incluir: permite incluir métricas.
3. Alterar: permite alterar métricas
4. Excluir permite excluir métricas

O acesso ao formulário métricas é feito pela equipe de teste ou gerente de teste. O formulário para esta funcionalidade é mostrado na Figura 4.1.

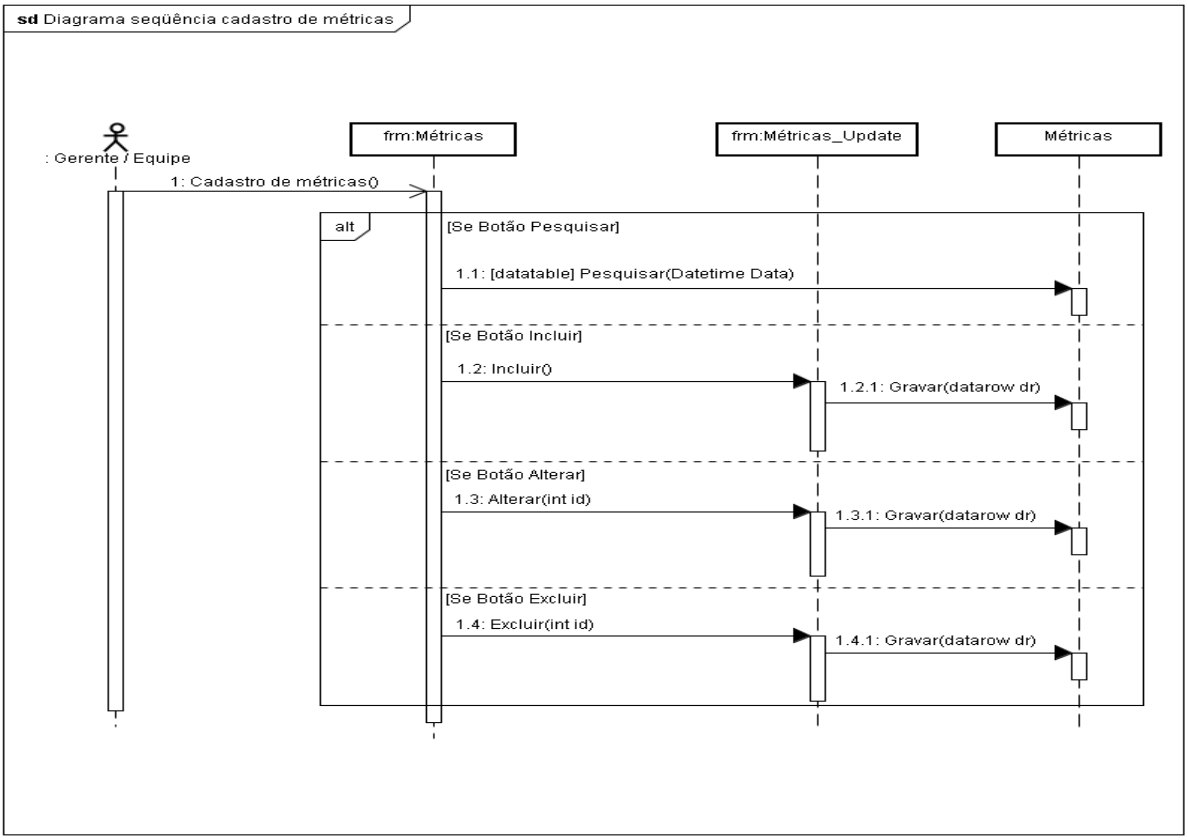

Figura 3.2 – Diagrama de seqüência cadastro de métricas

3.3.2 – DIAGRAMA DE SEQÜÊNCIA CADASTRAR SUBSISTEMAS

O cadastro de subsistemas é a funcionalidade que oferece a flexibilidade para manutenção no cadastro de subsistemas. Cada subsistema representa uma parte do sistema. O diagrama de seqüencia Figura 3.3 mostra as interações e troca de mensagens entre os objetos conforme segue:

1. Pesquisar: permite selecionar subsistemas, se nada for digitado o sistema considera todos os subsistemas.
2. Incluir: permite incluir subsistemas.
3. Alterar: permite alterar subsistemas.
4. Excluir permite excluir subsistemas.

O acesso ao formulário cadastro de subsistemas é feito pela equipe de teste ou gerente de teste. O formulário para esta funcionalidade é mostrado na Figura 4.3.

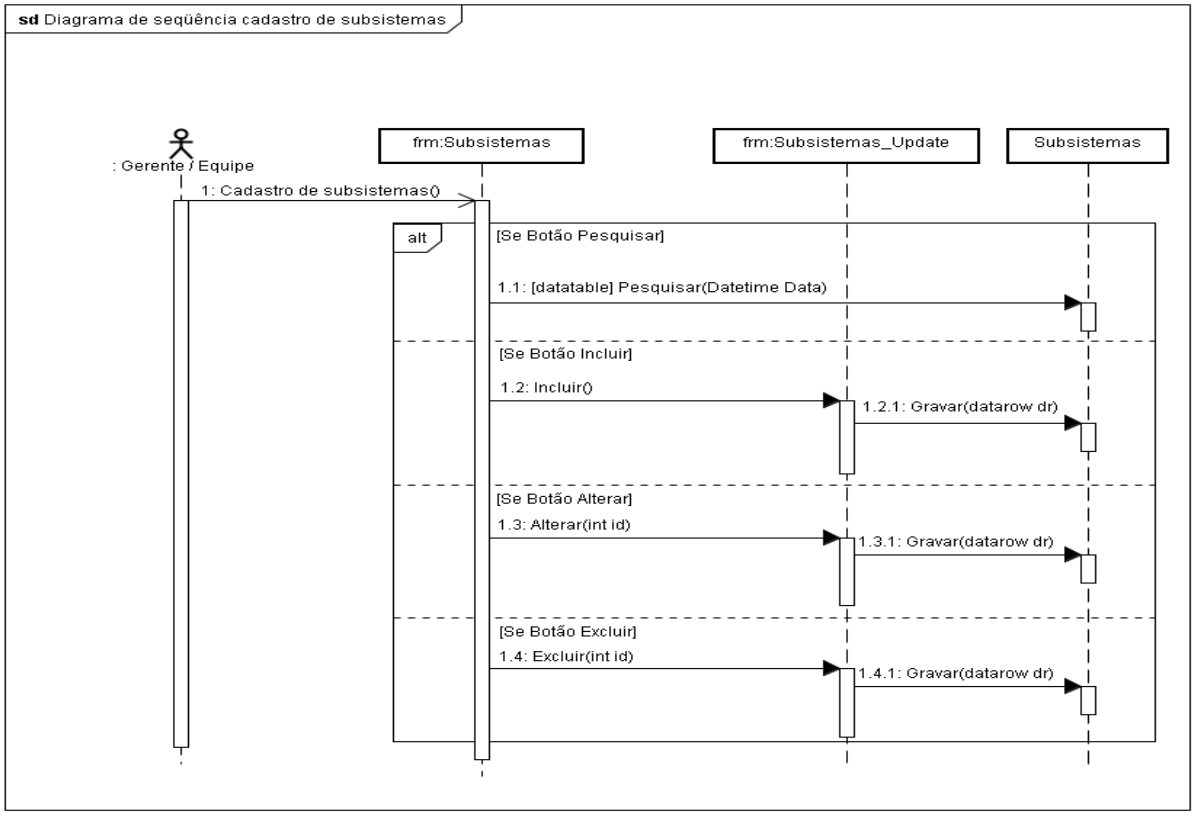

Figura 3.3 – Diagrama de seqüência cadastro de subsistemas

3.3.3 – DIAGRAMA DE SEQÜÊNCIA CADASTRAR DIÁRIO DE TESTE

O cadastro do diário de teste é a funcionalidade que oferece a flexibilidade para manutenção no cadastro diário de teste. O diário de teste é o registro das ocorrências diárias de teste, possibilitando o cadastro do resumo de teste. O diagrama de seqüencia Figura 3.4 mostra as interações e troca de mensagens entre os objetos conforme segue:

1. Pesquisar: permite selecionar a data do diário de teste.
2. Incluir: permite incluir diário de teste.
3. Alterar: permite alterar diário de teste.
4. Excluir permite excluir diário de teste.

O acesso ao formulário cadastro de subsistemas é feito pela equipe de teste ou pelo gerente de teste. O formulário para esta funcionalidade é mostrado na Figura 4.5.

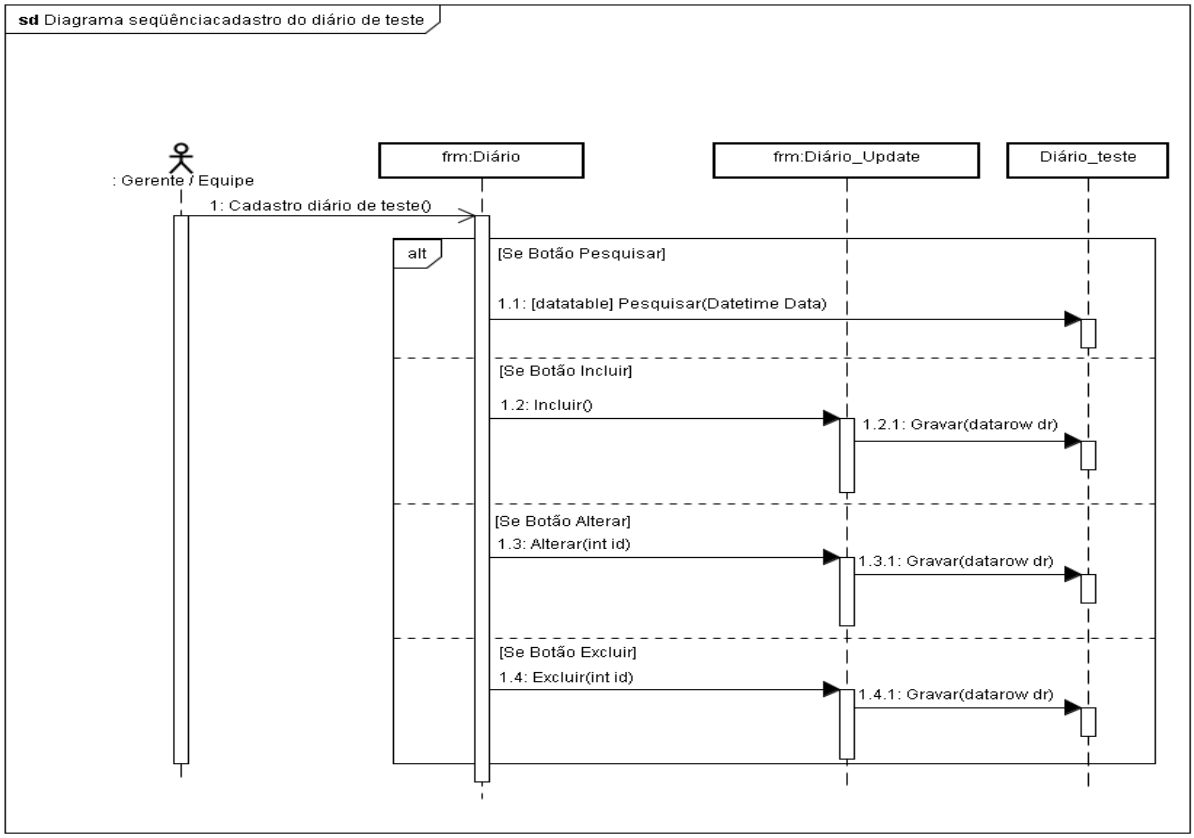

Figura 3.4 – Diagrama de seqüência cadastro de diário de teste de software

3.3.4 – DIAGRAMA DE SEQÜÊNCIA GERAR VISUALIZAÇÃO GRÁFICA

O diagrama de seqüência da Figura 3.5 foi adaptado da Figura 2.11, onde a seqüencia é a mesma utilizada na ferramenta desenvolvida conforme segue:

1. Usuário: Nesta classe o parâmetro data é selecionado para ser usado no algoritmo da classe número 2.
2. Dados Brutos: esta é a classe que executa o algoritmo de seleção de dados brutos, conforme parâmetro recebido da classe número 1, retornando uma tabela filtrada.
3. Dados transformados: classe que recebe a tabela filtrada e adiciona pesos conforme previsto, para equilibrar os dados, para melhorar as informações, representando os dados reais.
4. Estruturas visuais: classe de seleção dos atributos visuais. Permitindo selecionar a medida que representará o eixo Y, a medida que representará o tamanho, a medida que representará a altura.
5. Visão: Nesta classe a visualização gráfica é mostrada. Se ocorrer a interação com a visualização executar a tarefa, senão, voltar à tarefa número 1.

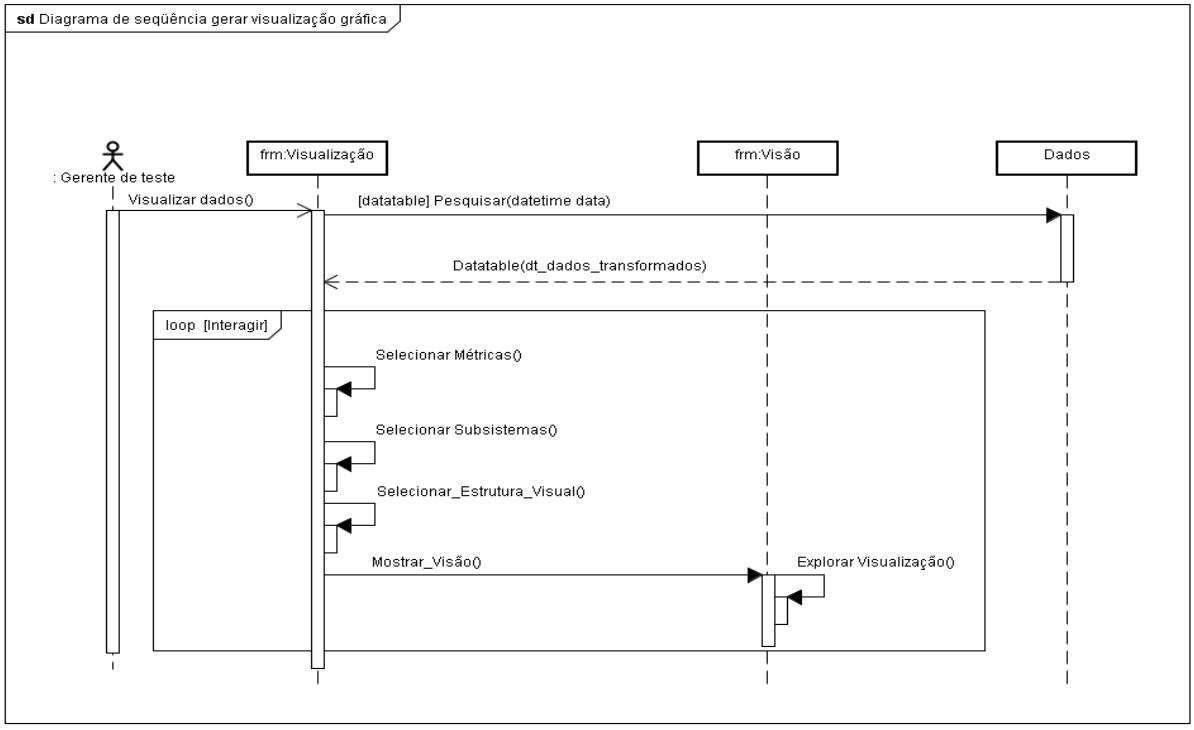

Figura 3.5 – Diagrama de seqüência gerar visualização gráfica

3.4 – DIAGRAMA DE ESTADO DA FUNCIONALIDADE GERAR GRÁFICOS

O diagrama de estado mostra os eventos, condições de guarda e ações necessárias para gerar a visualização gráfica conforme Figura 3.6. Essa seqüência é descrita a seguir:

1) Selecionando uma data: O usuário seleciona a data para processamento.
2) Localizando dados: Nesta etapa os dados são filtrados de acordo com o algoritmo selecionado.
3) Transformando dados: Os dados selecionados são transformados através de pesos para distribuir os tamanhos de forma proporciona, em uma nova tabela. Nesta fase o usuário poderá definir as medidas que serão representadas pelo eixo Y, tamanho e altura na visualização.
4) Mostrando Visões: Nesta etapa poderão ser mostradas várias visões, de acordo com os controles selecionados, conforme segue:
    - Selecionar Métricas ou Medidas.
    - Selecionar Subsistemas.
    - Selecionar gráfico do tipo *pie ou doughnut.*
    - Mostrar visualização. O usuário poderá volta à fase I desta etapa, e selecionar novos controles, repetindo a operação.

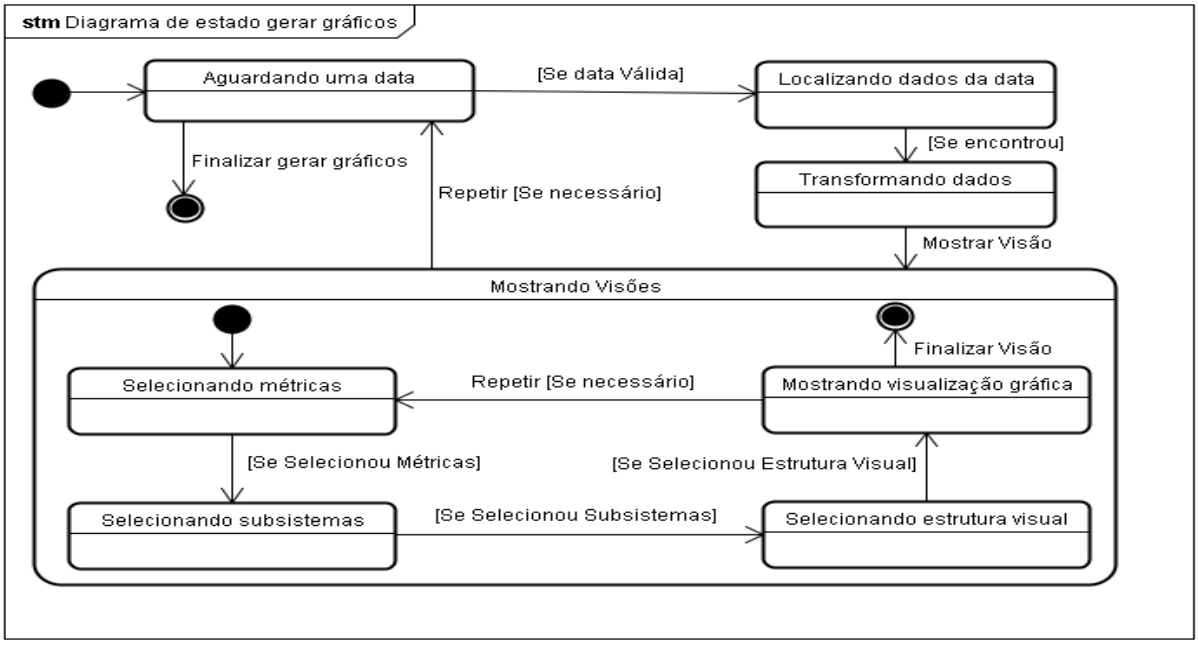

Figura 3.6 – Diagrama de Estado gerar gráficos

3.5 – DIAGRAMA DE CLASSES MODELO IMPLEMENTAÇÃO

O diagrama de classes modelo de implementação conforme Figura 3.7 representa o cadastro de tabelas básicas, isto é: não estão incluídas as classes que filtram e geram tabelas dinâmicas para formar a visualização gráfica.

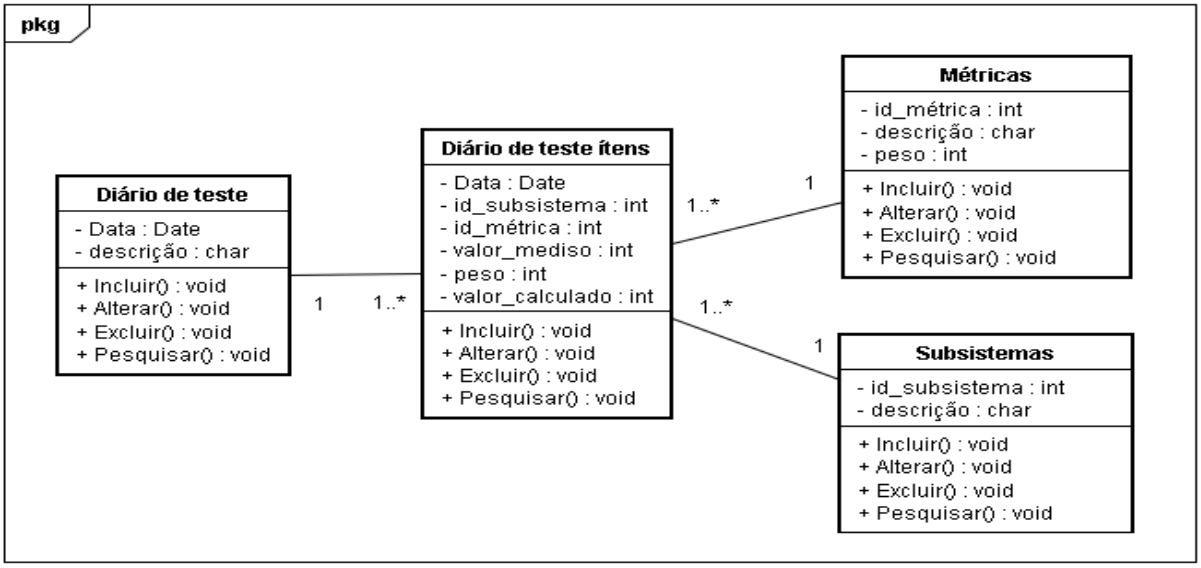

Figura 3.7 – Diagrama de classe modelo implementação

3.6 – RECURSOS UTILIZADOS

A ferramenta foi desenvolvida utilizando orientação a objetos com os seguintes recursos:

- Ferramenta JUDE Community versão freeware distribuído gratuitamente no site: http://jude.change-vision.com/jude-web/product/community.html, utilizado para elaboração dos diagramas casos de uso, diagramas de seqüência, diagramas de classes, diagramas de estado.
- Visual Studio 2005 da empresa Microsoft linguagem de programação C# versão Express que é uma versão gratuita distribuída no site: <http://www.microsoft.com/downloads>
- Gráficos Dundas Chart versão shareware distribuída gratuitamente em seu site: < http://www.dundas.com/Downloads/Index.aspx >
- O banco de dados utilizado ACCESS por não necessitar de instalação e o drive do Visual Studio possui os recursos necessários para manutenção.

4 – VALIDAÇÃO DA FERRAMENTA

A validação da ferramenta foi feita com dados simulados, devido a dificuldade em encontrar dados reais de teste de software utilizados na tomada de decisão no processo de gerenciamento de um teste de software real. Os dados foram simulados observando os valores quantitativos que podem representar dados reais. Exemplo: para simular dados que represente a métrica complexidade ciclomática de McCabe, foi observada a Tabela 2.3 (Avaliação da complexidade ciclomática de McCabe). Utilizando esse critério, os dados simulados podem indicar dados reais, tornando possível o experimento da ferramenta, e ajudar na indicação da veracidade das hipóteses levantadas.

4.1 – ENTRADA DE DADOS PARA VALIDAÇÃO DA FERRAMENTA

É possível criar recursos para capturar informações de métricas diretamente no Software que será testado, dependendo apenas do código fonte ou alguma forma de leitura da estrutura interna do software, que não é foco deste trabalho. A entrada de dados na tabela para validação da ferramenta foi feita manualmente.

4.2 – CADASTRO DE MÉTRICAS DE SOFTWARE E OUTRAS MEDIDAS

As métricas para a validação da ferramenta foram estão selecionadas de forma a representar um teste real e estão relacionadas na Tabela 4.1.

Tabela 4.1 – Métricas e medidas de teste de software

Descrição	Tipo	Peso
Complexidade de McCabe	Métrica de teste de software	100
LOC (linhas de Código)	Métrica de teste de software	1
Quantidade de casos de uso	Medida	15
Quantidade de casos de uso testados	Medida	15
Quantidade de defeitos encontrados	Medida	30
Quantidade de defeitos corrigidos	Medida	30
Prioridade do cliente	Medida	100

O formulário Figura 4.1 e Figura 4.2 são utilizadas para o cadastro de métricas e fornece ao usuário a possibilidade de incluir, alterar, excluir e pesquisar métricas, esta funcionalidade possibilita selecionar as métricas que possam melhor representar o teste software. O objetivo do acompanhamento por métricas será o acompanhamento das métricas selecionadas e a comparação do número de falhas reveladas durante o teste de software. Através de vários experimentos poderá ser possível avaliar como exemplo se a complexidade de McCabe está relacionada com a quantidade de defeitos revelados. Através dessa relação poderá ser feito uma previsão do número de defeitos a ser encontrado no teste de software e ainda poderá ser feito uma recomendação de treinamento para a equipe de desenvolvimento, visando a redução do número de defeitos durante a fase de desenvolvimento do software.

O valor do ajuste ou peso foi inserido de forma aleatória para equilibrar a formação do gráfico. Considerando a complexidade de valor igual a cinco poderá ser igual a um LOC de duzentos, o peso foi usado para mostrar na visualização as duas métricas com a mesma proporção.

Figura 4.1 – Cadastro de métricas e medidas

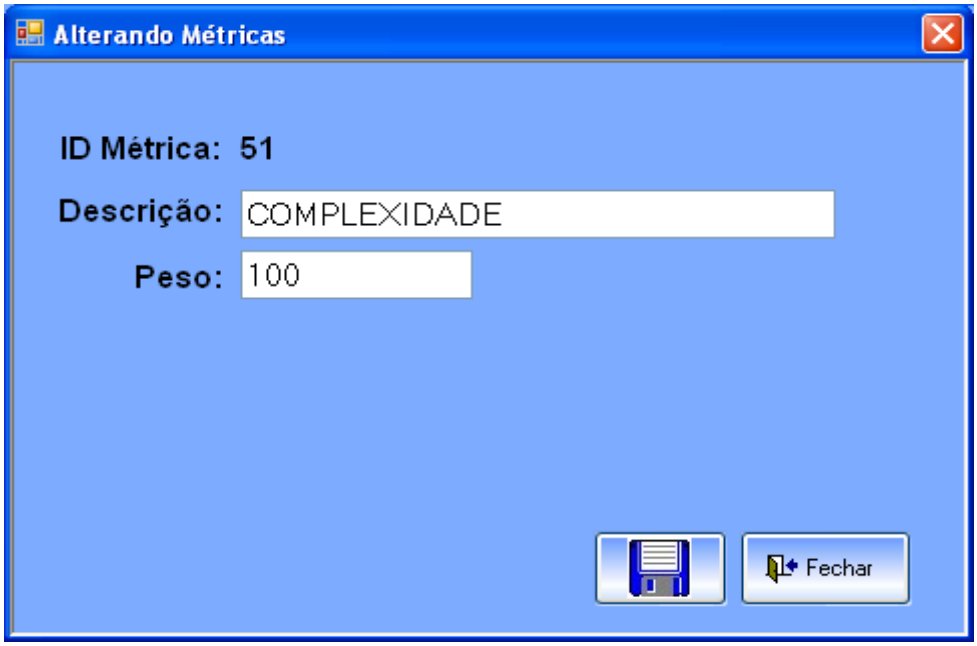

Figura 4.2 – Cadastro de métricas de software update

4.3 – CADASTRO DE SUBSISTEMAS

A tomada de decisão pode ser diferente para cada subsistema. Conforme Tabela 4.2 cada subsistema poderá ter complexidade e criticidade diferente, sendo assim, a quantidade de casos de teste e os tipos de teste podem ser diferentes para cada subsistema. É importante conhecer estas informações pois estes requisitos de qualidade envolvem análise de custos, prazo de entrega e responsabilidade pelos riscos (Clientes, fornecedores, etc.). A tolerância a falhas segundo SOMMERVILLE (2003) pode ser classificada em: intolerável; baixa; aceitável, e a confiança: baixa; média; alta; muito alta; ultra-alta.

Os requisitos de tolerância a falhas e confiança poderão ser transformados em valores quantitativos, possibilitando a visualização gráfica.

Tabela 4.2 – Qualidade requerida por subsistemas

Subsistema	Tolerância a falhas	Confiança
Subsistema 1	Intolerável	Alta
Subsistema 2	Baixa	Média
Subsistema 3	Aceitável	Baixa
Subsistema 4	Aceitável	Baixa
Subsistema 5	Aceitável	Baixa
Subsistema 6	Aceitável	Baixa

O cadastro de subsistemas conforme Figura 4.3 e Figura 4.4 possibilitam incluir, alterar, excluir e pesquisar subsistemas. Cada subsistema representa uma parte do sistema. Cada subsistema poderá ser testado por uma equipe diferente com suas métricas e decisões diferentes.

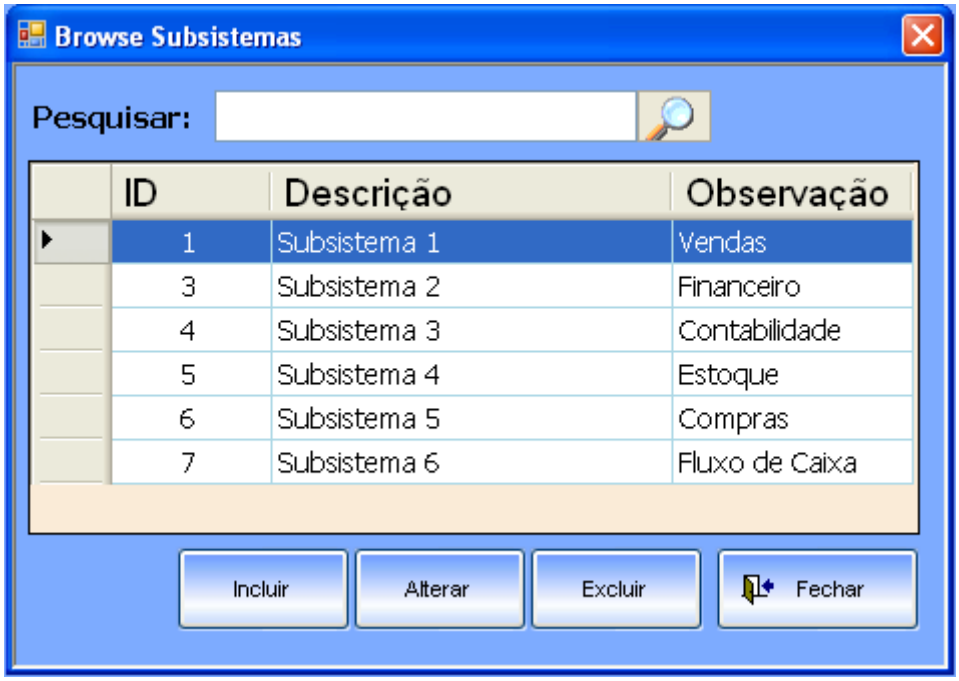

Figura 4.3 – Cadastro de subsistemas

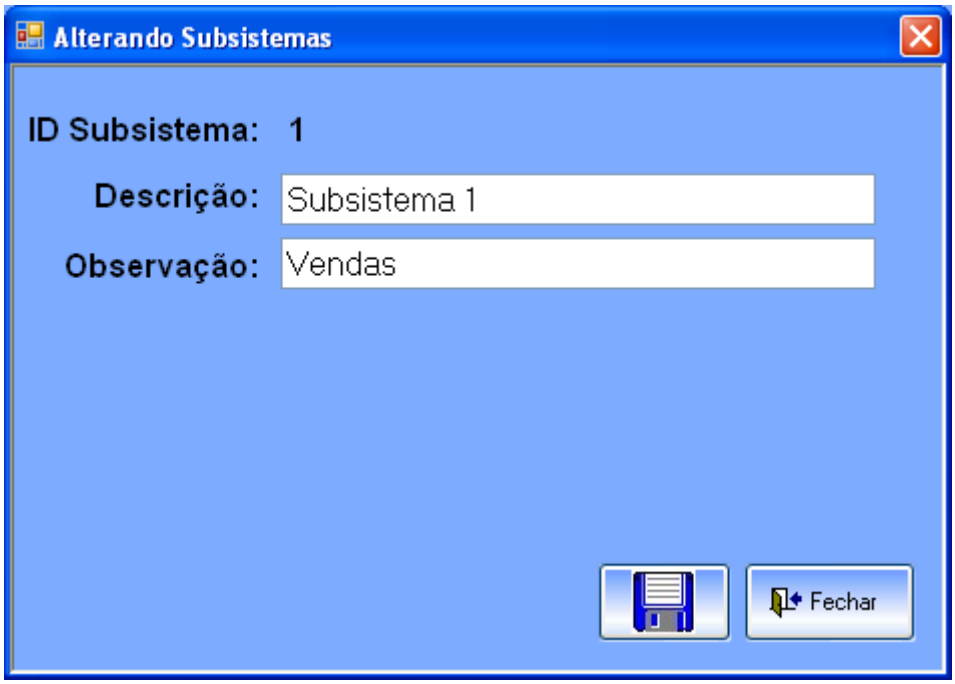

Figura 4.4 – Cadastro de subsistemas Update

4.4 – CADASTRO DO DIÁRIO DE TESTE

O cadastro do diário de teste é atualizado conforme Figura 4.5 até Figura 4.8. A atualização diária dos dados do resumo de teste para efeito de validação da ferramenta é feita manualmente embora possa ser importada de outras bases de dados geradas por outras ferramentas de gerenciamento de teste de software, o que facilitará o uso da ferramenta.

Figura 4.5 – Cadastro do diário de teste

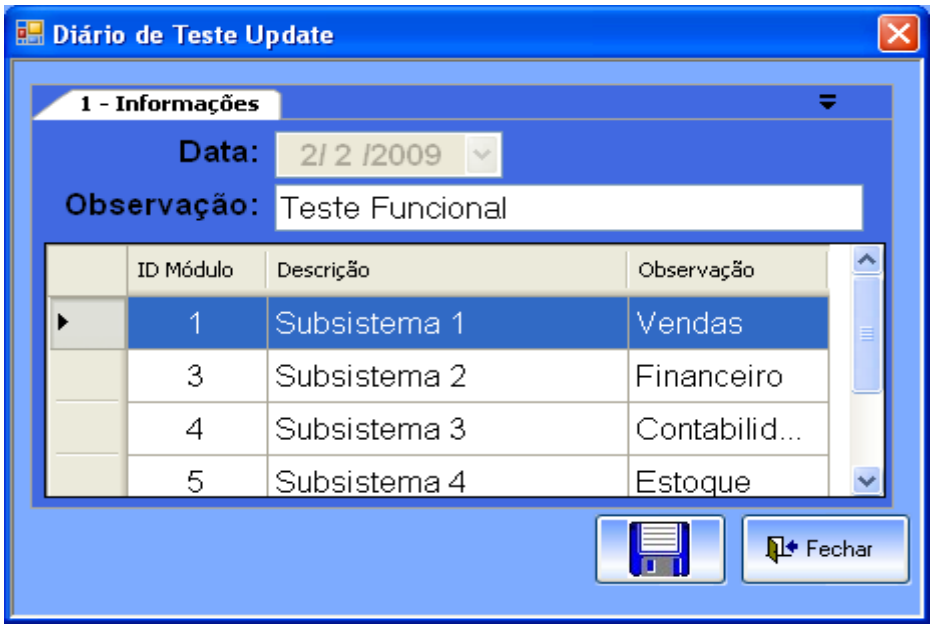

Figura 4.6 – Cadastro do diário de teste subsistemas

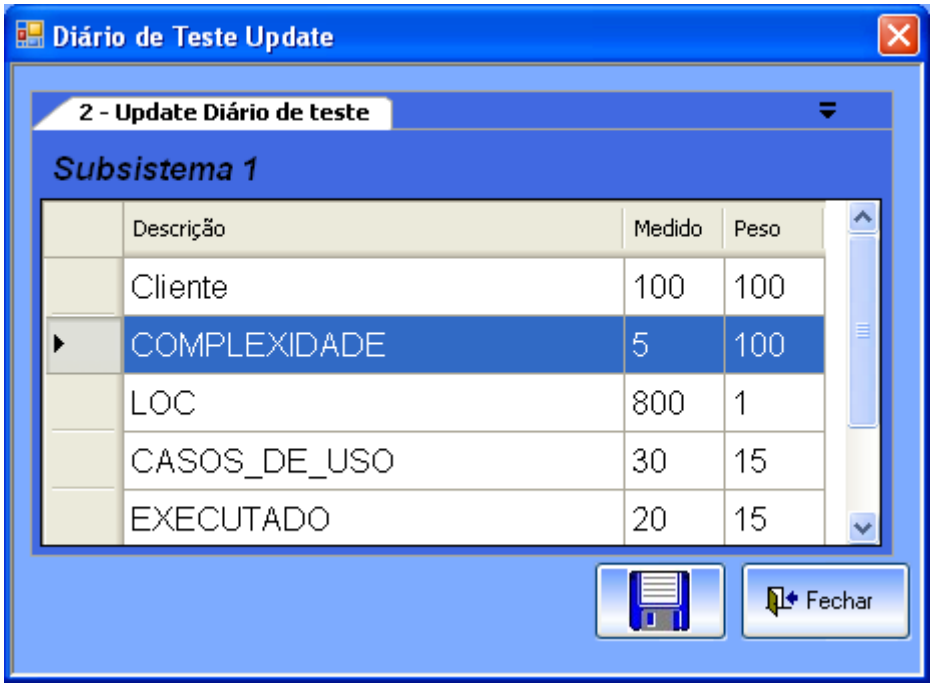

Figura 4.7 – Cadastro do diário de teste métricas

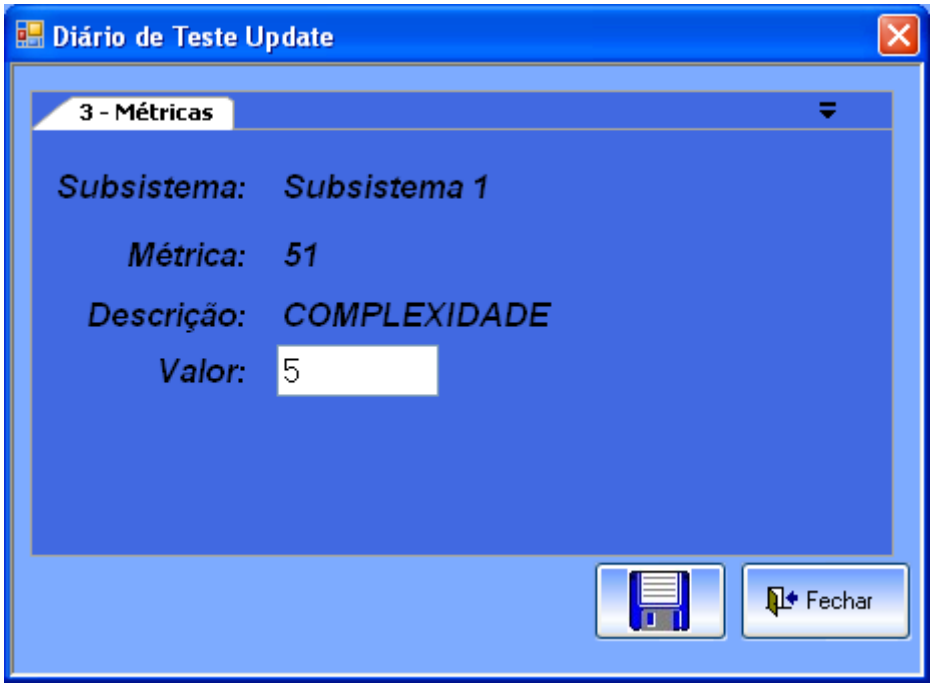

Figura 4.8 – Cadastro diário de teste métricas update

4.5 – ANÁLISE DE TABELAS DE DADOS

Uma tabela contém uma grande quantidade de dados, no caso simulado estão os dados dos subsistemas 1 ao subsistema 3, apenas do dia 02/02/2009. Neste caso existe uma grande dificuldade de análise, mesmo que a tabela represente poucos dados. A Tabela 4.3 representa os dados filtrados da tabela de dados brutos.

Tabela 4.3 – Tabela de dados filtrados, simulação do dia 02/02/2009

Subsistema	Setor	Métrica	Medido	Peso	Valor
Subsistema 1	Vendas	Cliente	100	100	10000
Subsistema 1	Vendas	COMPLEXIDADE	5	100	500
Subsistema 1	Vendas	LOC	800	1	800
Subsistema 1	Vendas	CASOS DE USO	30	15	450
Subsistema 1	Vendas	EXECUTADO	20	15	300
Subsistema 1	Vendas	*BACKLOG*	10	30	300
Subsistema 1	Vendas	CORREÇÃO	5	30	150
Subsistema 2	Financeiro	Cliente	80	100	8000
Subsistema 2	Financeiro	COMPLEXIDADE	6	100	600
Subsistema 2	Financeiro	LOC	500	1	500
Subsistema 2	Financeiro	CASOS DE USO	50	15	750
Subsistema 2	Financeiro	EXECUTADO	35	15	525
Subsistema 2	Financeiro	*BACKLOG*	16	30	480
Subsistema 2	Financeiro	CORREÇÃO	12	30	360
Subsistema 3	Contabilidade	Cliente	60	100	6000
Subsistema 3	Contabilidade	COMPLEXIDADE	4	100	400
Subsistema 3	Contabilidade	LOC	400	1	400
Subsistema 3	Contabilidade	CASOS DE USO	20	15	300
Subsistema 3	Contabilidade	EXECUTADO	18	15	270
Subsistema 3	Contabilidade	BACKLOG	5	30	150
Subsistema 3	Contabilidade	CORREÇÃO	5	30	150

A Tabela 4.4 representa a tabela de dados transformada e gerada dinamicamente, pois o número de colunas e linhas pode oscilar de acordo com os controles de seleção, a tabela representa apenas um dia. Pode-se observar que a tabela transformada possui uma melhora na apresentação dos dados brutos, possibilitando redução no tempo de análise. Mas mesmo assim o tempo é longo devido a capacidade de guardar resultados parciais do ser humano ser limitada [CARD,1999].

Tabela 4.4 – Tabela de dados transformados, simulação do dia 02/02/2009

SUBSISTEMA	Cliente	COMPLE-XIDADE	LOC	CASOS DE USO	EXECUTADO	BACKLOG	CORREÇÃO
Subsistema 1	10000	500	800	450	300	300	150
Subsistema 2	8000	600	500	750	525	480	360
Subsistema 3	6000	400	400	300	270	150	150
Subsistema 4	7500	500	750	750	600	450	390
Subsistema 5	4000	200	300	375	300	180	180
Subsistema 6	7300	400	550	285	270	300	270

4.6 – CONTROLES DA FERRAMENTA VISUALIZAÇÃO GRÁFICA

Para uma boa visualização gráfica a ferramenta apresenta uma série de controles, conforme Figura 4.9 e Figura 4.10, os quais permitem selecionar métricas, subsistemas, e os atributos visuais (tamanho e altura), que serão visualizados, as principais funcionalidades da ferramenta são:

- Seleção de subsistemas.
- Seleção do tipo de gráfico que pode ser *doughnut* ou *pie*.
- Seleção da métrica que será mostrada no eixo Y.
- Seleção da métrica que será representada pelo tamanho.
- Seleção da métrica que será representada pela altura.
- Seleção das métricas ou medidas que formarão cada mini gráfico da bolha representado pelo atributo cor, conforme Figura 4.10.
- Botão Visualizar que gera o gráfico.

Figura 4.9 – Controle de funcionalidades

Figura 4.10 – Controle de métricas para visualização

4.7 – ANÁLISE DA VISUALIZAÇÃO GRÁFICA

O gráfico gerado é um gráfico de bolhas, onde o eixo X representa os subsistemas, e o eixo Y a prioridade do cliente. E para facilitar e agilizar o entendimento o atributo tamanho representa a complexidade, o atributo altura representa LOC. É fácil notar que a maior prioridade do cliente é o subsistema 1, a menor o subsistema 5, o subsistema 1 possui maior LOC, o subsistema 2 maior complexidade, o subsistema 5 menor complexidade.

A legenda mostra os atributos de cor que representam a complexidade, LOC, casos de uso, executado, *backlog* e correção, de forma que todas as variáveis estão representadas de forma gráfica. Comparando a Tabela 4.4 com o gráfico da Figura 4.11, pode ser notado que são iguais, representam as mesmas medidas, sendo que a cognição é muito maior na Figura 4.11. Esse estudo de caso contribui para demonstrar a veracidade das hipóteses de que a visualização gráfica realmente pode apoiar na tomada de decisão no processo de teste de software. Utilizando a ferramenta de visualização, os gráficos podem ser gerados diversas vezes, selecionando parâmetros diferentes, facilitando ainda mais a compreensão do contexto.

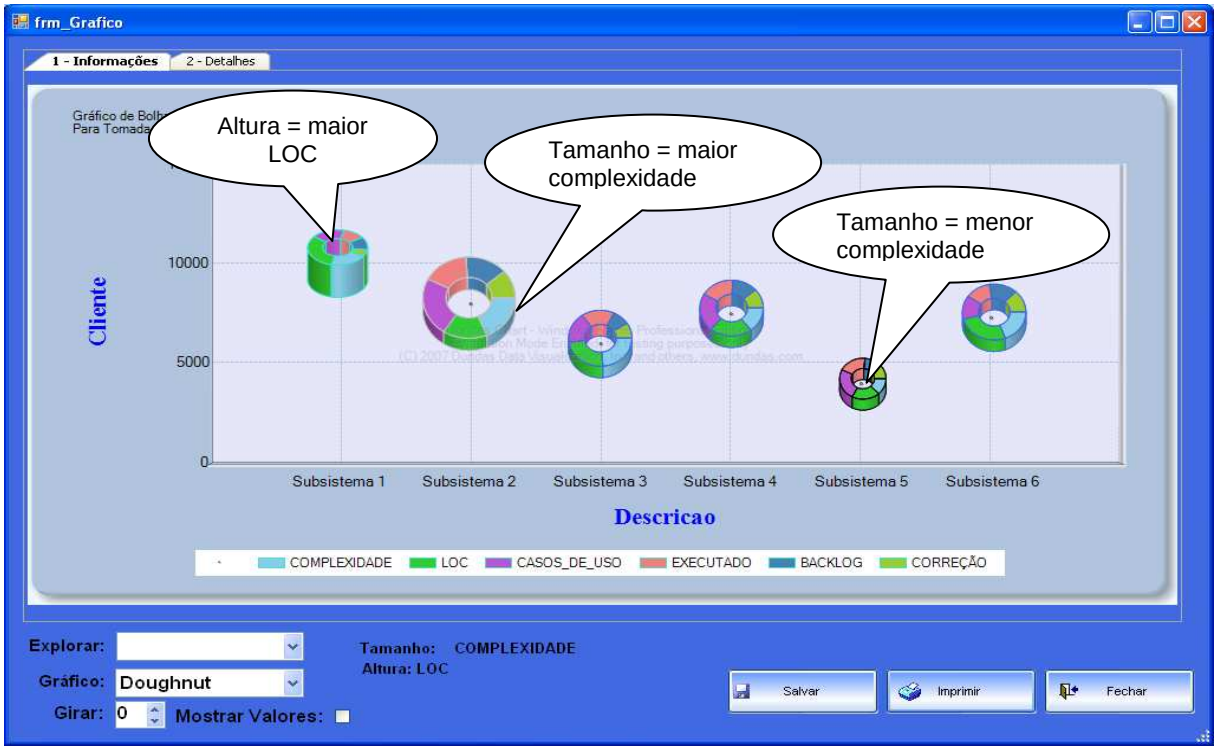

Figura 4.11 – Gráfico *doughnut*

4.8 – ANALISANDO VISÕES

Analisando a Figura 4.12, Figura 4.13 e Figura 4.14, pode ser verificado que ambas representam a mesma posição, somente a formação da visualização é diferente. O eixo Y representa o Gráfico de Bolhas e pode ser analisado como segue:

1. Analisando a prioridade do cliente:

    - Figura 4.12 a prioridade do cliente é o subsistema 1, e é representada pelo eixo Y, onde a bolha está na posição mais alta, a menor prioridade é o subsistema 5.
    - Figura 4.13 a prioridade do cliente é o subsistema 1, sendo representada pelo tamanho e o menor tamanho é o subsistema 5.
    - Figura 4.14 a prioridade do cliente é o subsistema 1, sendo representado pela altura.

2. Analisando a complexidade:

    - Figura 4.12 a maior complexidade está no subsistema 2, e é representada pelo tamanho, a menor complexidade é o subsistema 5.
    - Figura 4.13 a maior complexidade é o subsistema 2, sendo representada pelo eixo Y e a menor complexidade é o subsistema 5.
    - Figura 4.14 a maior complexidade é o subsistema 2, sendo representado pelo tamanho.

3. Analisando LOC:

    - Figura 4.12 o LOC está no subsistema 1 e é representada pelo tamanho.
    - Figura 4.13 a maior LOC é o subsistema 1 sendo representado pelo tamanho.

- Figura 4.14 a maior LOC é o subsistema 1 sendo representado pelo eixo Y e o menor LOC está no subsistema 5.

Uma das funcionalidades da ferramenta é possibilitar ao gerente de teste formar várias visualizações, alterando as métricas ou medidas que serão mostradas no eixo Y, tamanho e altura.

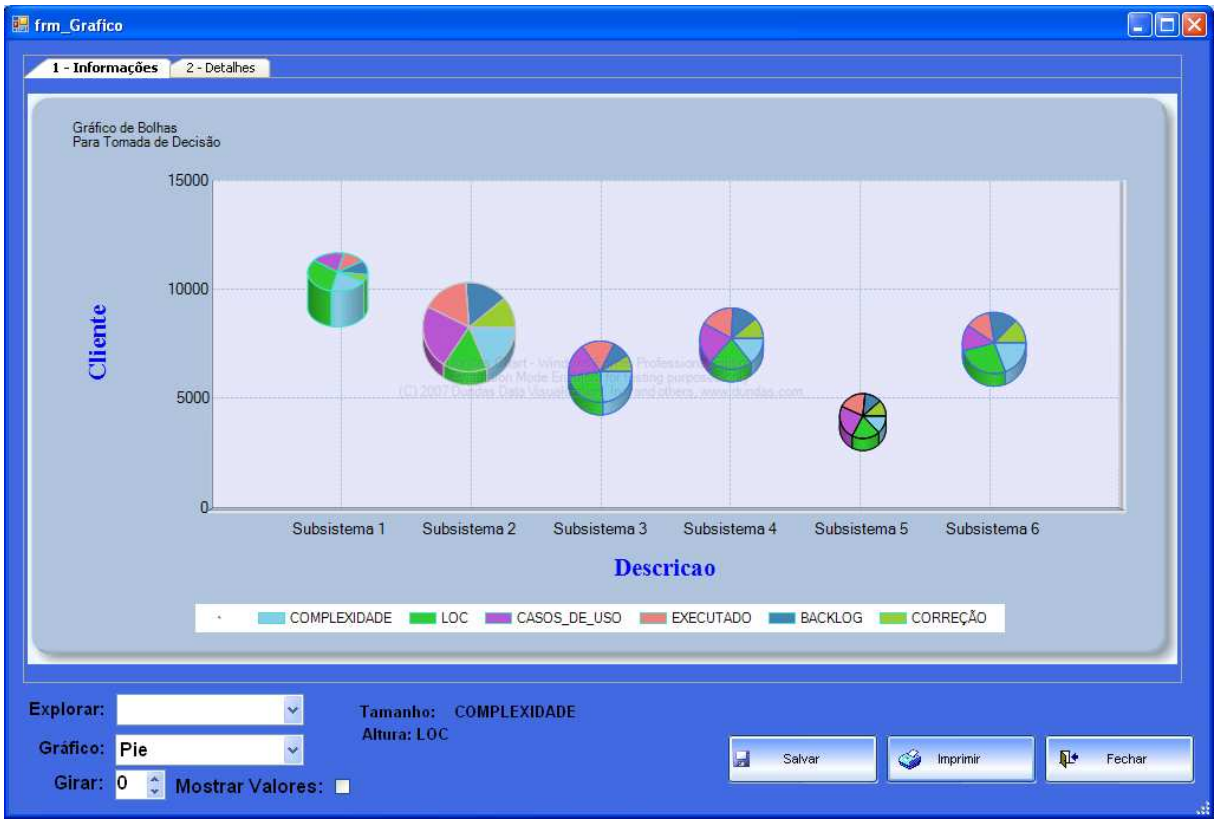

Figura 4.12 – Gráfico eixo Y representa a prioridade do Cliente

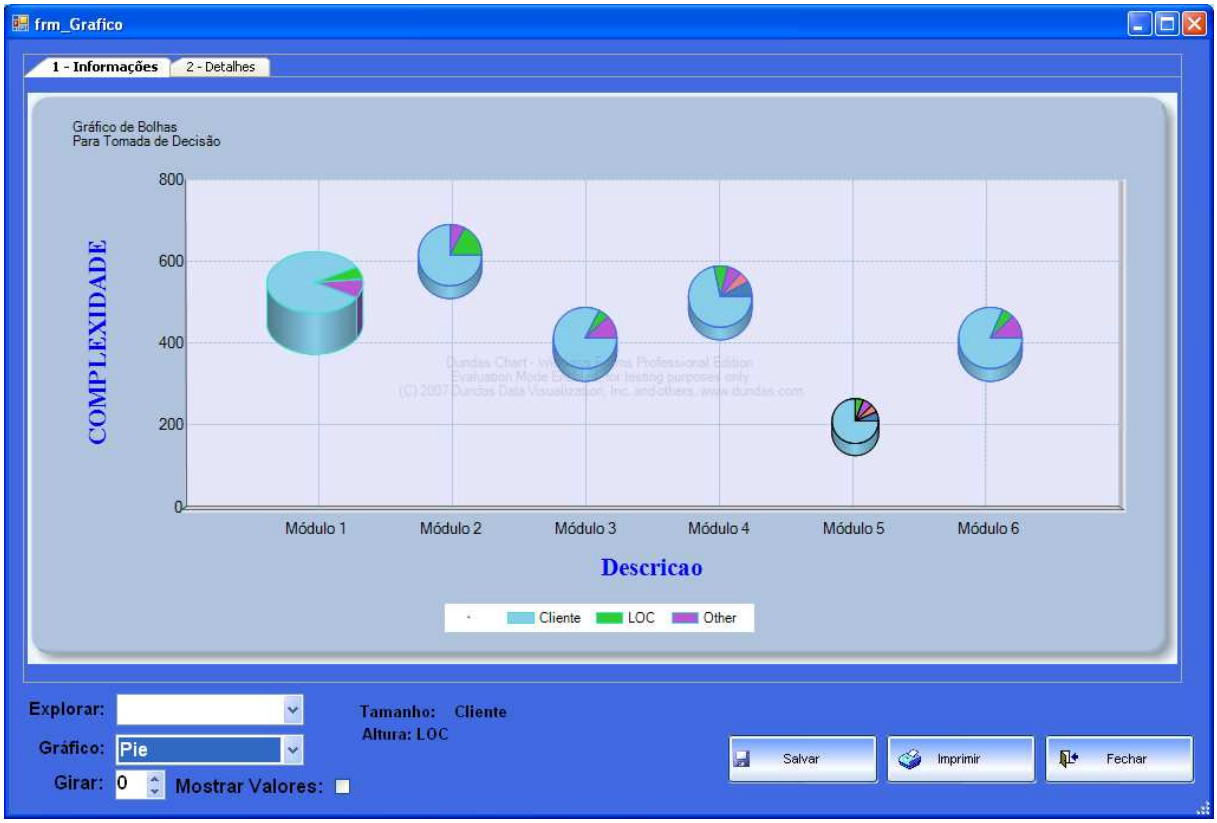

Figura 4.13 – Gráfico eixo Y representa a Complexidade de McCabe

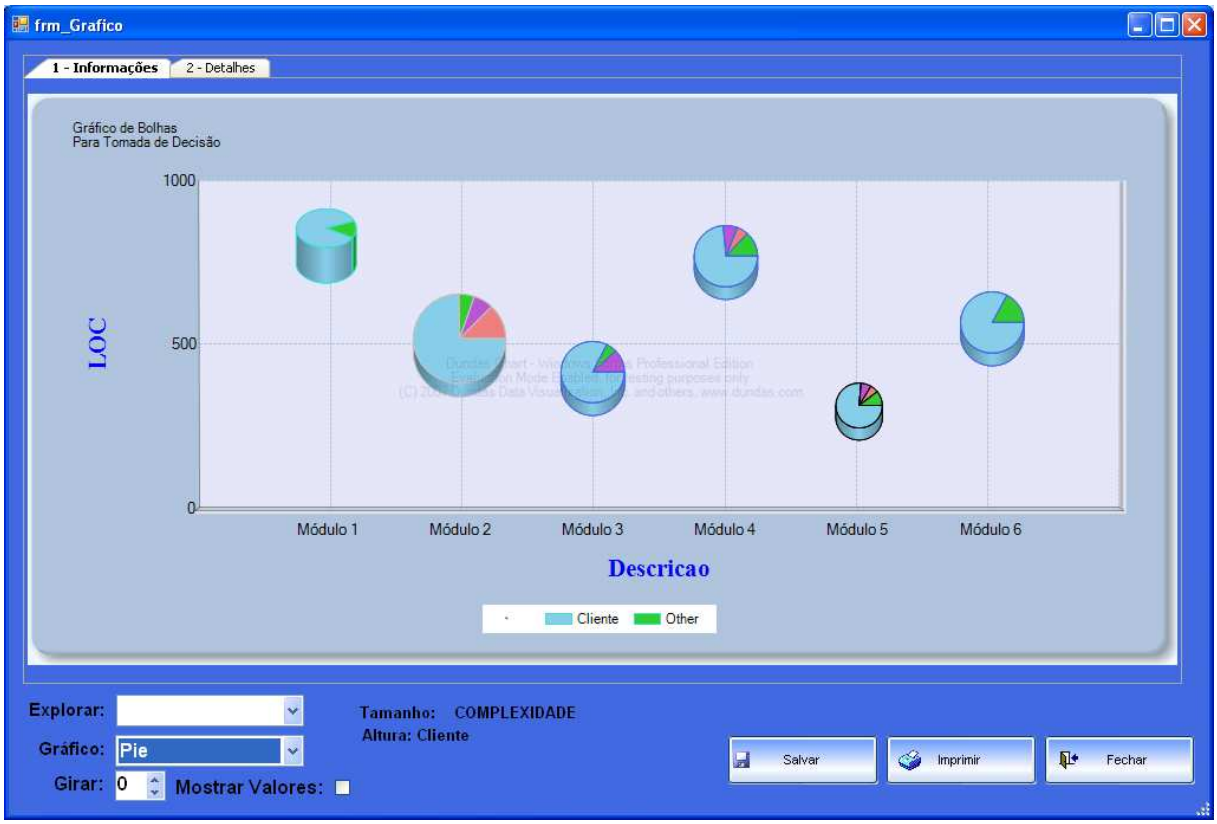

Figura 4.14 – Gráfico eixo Y representa LOC

4.9 – DIFICULDADES E LIMITAÇÕES ENCONTRADAS

Para desenvolver esta ferramenta, uma serie de dificuldades surgiram, entre elas estão:

- Não foram localizados trabalhos relacionados com visualização gráfica envolvendo teste de software.
- Não foram encontrados trabalhos relevantes a esta investigação, sobre decisão durante a fase de teste de software.
- Dificuldade para encontrar uma driver de visualização gráfica compatível com Visual Studio para tornar possível gerar gráficos interativos, utilizando todos os atributos de visualização gráfica.
- Dificuldade em gerar vários gráficos dentro de um mesmo gráfico, isto é uma série de mini gráficos.

As Limitações encontradas estão relacionadas ao tempo e a dados reais de teste de software, pois as empresas de desenvolvimento consideram o software como propriedade intelectual não compartilhando informações ou ainda não gera uma base de conhecimento registrando as informações e resultados obtidos durante a fase de teste de software.

5 – CONCLUSÃO

A contribuição principal desta abordagem é a investigação se a visualização gráfica pode apoiar um gerente de teste na tomada de decisão durante a fase de teste de software. Para realizar essa investigação foram realizados estudos sobre métricas de software, teste de software, tomada de decisão durante o teste de software, qualidade e confiabilidade de Software e ferramentas e recursos de exploração visual. Embora não tenha sido encontrada abordagem com dados práticos e reais de tomada de decisão, os valores utilizados para esta investigação simulam dados próximos aos dados reais, sendo assim, os gráficos gerados podem ser considerados para investigação de veracidade das hipóteses levantadas.

A ferramenta desenvolvida é um protótipo para investigar as hipóteses levantadas. Para melhor utilização novas rotinas devem ser desenvolvidas visando importação de dados de outros sistemas de acompanhamento do processo de teste de software.

Na simulação de uso da ferramenta, os dados foram digitados manualmente. Durante a simulação é possível gerar uma série de interações gerando representações visuais, em uma mesma seleção de dados, e com formatos diferentes, com o objetivo cognitivo, e foi possível demonstrar a veracidade das hipóteses levantadas demonstrando que a visualização gráfica realmente pode apoiar um gerente de teste de software na tomada de decisão no processo de teste de software.

A tomada de decisão em teste de software é fundamental para o processo, sendo polêmica, envolvendo uma série de situações, não existe um padrão ou uma norma de procedimentos. Se parar de testar muito cedo, poderão ocorrer falhas, e gerar insatisfação do cliente, e o custo de recuperação e correção poderá ser muito alto. No entanto se o teste continuar por um período longo, o custo será elevado, bem como o prazo de entrega, podendo representar a diferença entre o sucesso e o fracasso do software [EHRLICH *at al*,1993]. A ferramenta de visualização é um item de apoio do gerente de teste, o objetivo é apoiar a tomada de decisão sendo que este apoio vem da facilidade da interação com a base de dados reduzindo significativamente o tempo de análise da situação.

TRABALHOS FUTUROS

Uma nova pesquisa poderá ser realizada, melhorando algumas funcionalidades da ferramenta, tais como importação de dados de outras ferramentas de acompanhamento de teste de software, bem como aplicar em um projeto real desde o inicio da fase de teste. Como o acompanhamento do processo de teste é diário, poderá ser investigado se existe uma relação entre o número de falhas reveladas e o tempo de teste, levando em consideração a complexidade e LOC. Poderá também ser verificado se a ferramenta poderá auxiliar na determinação do ponto de saturação de teste, observando se existem indícios de que a complexidade está relacionada com o número de defeitos revelados.

Através do uso de dados reais, uma base de dados de conhecimento será formada e poderá gerar informações suficientes para formar padrões de tomada de decisão em teste de software facilitando a tomada de decisão e auxiliando a formação de novos gerentes de teste de software.

REFERÊNCIAS BIBLIOGRÁFICAS

BARTIÉ, A. **Garantia da Qualidade de Software: adquirindo maturidade organizacional.** Editora Campos, Rio de Janeiro, Terceira Reimpressão, 2002.

BASILI, V.R.; CALDIERA, G.; ROMBACH, H.D. **THE GOAL QUESTION METRIC APPROACH**. Encyclopedia of Software Engineering – 2 Volume Set.
Page 528-532. John Wiley & Sons, 1994.

BOEHM, B. W. **Software Risk Management: Principles and Practices. IEEE Software**, Volume 8, Number 1, January 1991.

CARD, S.K; MACKINLAY, J. D.; SHNEIDERMAN, B. **Readings in Information Visualization: Using Vision to Think.** Morgan Kaufman Publishers, 1999.

CHI, E.H.; CARD, S.K. **Sensemaking of Evolving Web Sites Using Visualization Spreadsheets**. Publication Date: Oct 1999. Disponível em: < http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=801853 >
Acessado em 12 Feb. 2009.

CRESPO, A.N.; SILVA, O.J.; BORGES, C.A.; SALVIANO, C.F.; ARGOLLO, M.T.J,; JINO, M. **Uma Metodologia para teste de software no Contexto da Melhoria de Processo.** UNICAMP, Campinas SP. In: Simpósio Brasileiro de Qualidade de Software, maio de 2004. Disponível em:
<http://www.sbc.org.br/bibliotecadigital/download.php?paper=254>.
Acesso em 27 Mai. 2008.

DeMillo, R.A.; Lipton, R.A.; Sayaward, F.G.; **Hints on test data selection: Help for the practicing programmer**. IEEE Computer, v. 11, n.4, p.34-43, 1978.

EHRLICH, W.; PRASANNA, B.; STAMPFEL, J.; WU, J. **Determinating the cost of a stop-teste decision**. Publication Date: Mar 1993. Disponível em:
<HTTP://IEEEXPLORE.IEEE.ORG/XPL/FREEABS_ALL.JSP?ARNUMBER=199726>
Acessado em 12 Feb. 2009.

FAYYAD, U.M; PIATETSKY-SHAPIRO, G.; SMYTH, P.; UTHURUSAMY, R.; **Advances in knowledge Discovery and data mining**, 1996.

HORGAN,J.Robert; MATHUR, Aditya P.; PASQUINI, Alberto; REGO, Vernon. **PERILS OF SOFTWARE RELIABILITY MODELING**, February, 16, 1995. Disponível em ftp://ftp.cs.purdue.edu/pub/serc/tech-reports/By-School/Purdue/TR160P.PS.Z Acessado em 12 Mai. 2008.

MANZONI,L.V.; PRICE,R.T.; **Identifying Extensions Required by RUP (Rational Unified Process) to Comply with CMM (Capability Maturity Model) Levels 2 and 3**; IEEE Standard For Software Test Documentation, Feb. 2003. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1178058&isnumber=26465>

McCABE, T. J., **A Complexidade Measure**. IEEE Transaction on Software Engineering, Vol.SE-2, No. 4, Dec. 1976

KAN, Stephen H., Complexity Metrics and Models. **Metrics and Models in Software Quality Engineering**, Second Edition, 2003.

PFLEEGER, Shari Lawrence, Testando os Programas, **Engenharia de Software**, Segunda Edição, 2004; tradução Dino Franklin.

PRESSMAN, Roger S., **Engenharia de Software**, Editora Makron Books, 2005; Tradução José Carlos Barbosa dos Santos.

RATIONAL; **Rational Unified Process**; Visão Geral; Disponível em: < http://www.wthreex.com/rup/process/ovu_proc.htm> Acessado em Fev, 2009.

SOMMERVILLE, Ian, Planejamento de Verificação e Validação, **Engenharia de Software**, Sexta Edição, 2003; tradução André Mauricio de Andrade, segunda reimpressão 2005.

VANDOREN, Edmond; SCIENCES, Kaman; SPRINGS, Colorado. **Cyclomatic Complexity**. Updated 12 Jul 2000. Disponível em: <http://www.sei.cmu.edu/str/descriptions/cyclomatic.html>
Acessado em 12 Mai. 2008.

VINCENZI, A. M. R.; **Orientação a Objeto: Definição, Implementação e Análise de Recursos de Teste e Validação. Tese de Doutoramento**, USP, São Carlos, SP, 2004.

REFERÊNCIAS CONSULTADAS

IEEE Standard For Software Test Documentation, February 1983. Disponível em: <http://ieeexplore.ieee.org/iel1/4305/12389/00573169.pdf?tp=&arnumber=573169&isnumber=12389> Acessado em: 27 Mai. 2008.

IEEE Standard For Software Test Documentation, Dec. 1998. Disponível em: <http://ieeexplore.ieee.org/iel4/5976/16010/00741968.pdf?tp=&arnumber=741968&isnumber=16010> Acessado em: 27 Mai. 2008.