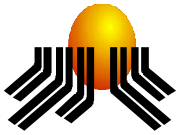


UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

VISUALIZAÇÃO DE INFORMAÇÃO COMO APOIO AO PLANEJAMENTO DO TESTE DE SOFTWARE

PIRACICABA, SP

2009

UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

VISUALIZAÇÃO DE INFORMAÇÃO COMO APOIO AO PLANEJAMENTO DO TESTE DE SOFTWARE

FABIANE PIFER FURLAN

ORIENTADOR: PROF.DR. PLÍNIO ROBERTO SOUZA VILELA

Dissertação apresentada ao Mestrado 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

VISUALIZAÇÃO DE INFORMAÇÃO COMO APOIO AO PLANEJAMENTO DO TESTE DE SOFTWARE

Autor: Fabiane Pifer Furlan
Orientador: Prof.Dr. Plínio Vilela

Dissertação de Mestrado defendida e aprovada em 29 de abril de 2009, pela Banca Examinadora constituída dos Professores:

_____

Prof. Dr. Plínio Roberto Souza Vilela
UNIMEP

_____

Profa. Dra. Simone do Rocio Senger de Souza
USP – São Carlos

_____

Profa. Dra. Ana Estela Antunes da Silva
UNIMEP

_____

Prof. Dr. Luiz Eduardo Galvão Martins
UNIMEP

A

Deus por tudo o que fez em minha vida

Aos

Meus pais, Jacir e Elvidge, pelo apoio e compreensão

À

Minha irmã Cristiane pelo incentivo

AGRADECIMENTOS

Ao Professor Dr. Plínio R. S. Vilela, que sempre me apoiou com muita paciência e compreensão, pela sua orientação, incentivo e oportunidades de aprendizado.

A todos os Professores do curso de mestrado da UNIMEP, que, além de grandes doutores, são grandes pessoas e serão lembrados por toda a minha vida.

Ao amigo Carlos de Abreu pelo inestimável apoio em implementação e em idéias que me forneceram durante os dois anos que estudamos juntos.

Agradeço também aos amigos e colegas que fiz no mestrado, pelos valorosos momentos de trabalho compartilhados.

Agradeço aos professores que compõem a banca de avaliação desta dissertação, por aceitarem o convite de analisarem este trabalho.

Há ainda dois agradecimentos muito importantes a fazer:

A minha família, a quem no tempo deste mestrado, e a quem nesta terra devo minha felicidade atual e vindoura. A esta bela família, devo infinitas doses de compreensão, paciência, gratidão e amor, pelos momentos que eram nossos e que foram abdicados em prol da elaboração desta dissertação. A você, pai, mãe, irmã, sobrinha e avó, todo o amor e carinho que meu coração pode dar.

E a Deus, fonte suprema de sabedoria, de quem sinto o amor, respeito e apoio em cada etapa da minha vida, protegendo-me das tempestades, iluminando meus conhecimentos e direcionando-me a Seus objetivos.

“A real viagem da descoberta não consiste
em buscar novas paisagens mas
ter olhos novos.” (Proust).

RESUMO

Dentre as atividades de garantia da qualidade de software, a atividade de teste é fundamental. Como a aplicação do teste exaustivo é impossível de ser realizada, uma forma de contornar esse problema é a aplicação mais intensa de teste nas partes mais críticas do software. Este trabalho de pesquisa se insere exatamente nessa forma de ação, pretende-se apoiar o processo de decisão do gerente de teste de software fornecendo a ele informações para que ele escolha as partes do software que deverão ser mais testadas, essas informações são apresentadas graficamente durante a fase de planejamento do teste de software.

Para tal tarefa foi desenvolvido um protótipo para auxiliar o gerente de teste na elaboração do planejamento de teste, através da representação gráfica exibida por ele. Desta forma, os gerentes podem tomar decisões importantes para selecionarem os trechos do código que merecem prioridade a serem testados, por apresentarem maior probabilidade de possuírem defeitos segundo as métricas de software disponibilizadas.

O que se pode concluir a partir dos dados levantados é que através da análise visual dos dados exibidos na representação gráfica houve uma contribuição para apoiar os gerentes de teste, tornando mais eficaz esta etapa da engenharia de software e que, por conseqüência, traz melhoria à qualidade do software avaliado.

PALAVRAS-CHAVE: Engenharia de Software, Teste de Software, Métricas de Software e Scrum

ABSTRACT

Software testing is one of the fundamental activities related to software quality assurance. Since exhaustive software testing is not practical, researchers have been studying techniques to try and isolate critical parts of the software that have to be tested the most. This research intends to propose a software visualization technique aimed at supporting test managers in the task of planning the software testing phase. The goal is to support the test manager in her decision making process.

A software prototype has been developed, in order to display certain software development related information in a graphical way, which helps the managers to work with a greater amount of information and expedite their decision making process. The strategy implemented assumes that certain parts of the software are more fault-prone than others and this relation is based on software metrics that are readily available.

What can we conclude from the data colected is that through the visual analysis of the data displayed in the graphic represetation there was a contribution to support the test managers, turning more efficient this stage of the software engineering and that, for consequence, improving the quality of the evaluate software.

KEYWORDS: Software Engineering, Software Test, Software Metrics and Scrum

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

1D	Unidimensional ou uma dimensão
2D	Bidimensional ou duas dimensões
3D	Tridimensional ou tridimensional
FTP	*File Transfer Protocol*
GQS	Garantia da Qualidade de Software
LOC	Linhas de Código
PF	*Function Points*
PFNA	Pontos de Função Não Ajustados
RGB	Modelo de Cor, componentes vermelho,verde e azul
RUP	*Rational Unified Rational*
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UAW	*Unajusted Actor Weight*
UC	Caso de Uso – *Use Case*
UCP	*Use Case Points*
UML	*Unified Modeling Language*
USP	*Use Case Size Points*
UUCW	*Unajusted Use Case Weight*

LISTA DE TABELAS

1 INTRODUÇÃO

Nas últimas décadas a indústria de software tem colocado um esforço substancial na melhoria da qualidade de seus produtos. Esse tem sido um trabalho difícil, já que o tamanho e a complexidade do software aumentam rapidamente e os usuários estão se tornando cada vez mais exigentes. O custo das falhas propagadas por defeitos no software incentivam a procura por estratégias de melhoria contínua da qualidade. Apesar de resultados encorajadores com várias características de melhoria de qualidade, a indústria de software está longe de produzir software livre de defeitos.

A atividade de teste, dentre os diversos aspectos da qualidade, tem sido uma das técnicas mais utilizadas pelos desenvolvedores de software antes da sua entrega ao usuário final. O teste de software em si não garantir que todos os defeitos sejam encontrados e que uma falha na execução do software não venha a ocorrer.

Apesar dessa restrição, a indústria mundial de Tecnologia de Informação tem investido, cada vez mais, em atividades de testes. Carreiras de especialistas em testes vêm sendo criada em maiores escalas. Grandes indústrias têm utilizado times independentes de teste no intuito de melhorar seu processo e qualidade do software produzido (SIT, 2002).

Teste é fundamental para a avaliação do software. Entretanto, testar software não é uma tarefa trivial e exige conhecimento, habilidades e infra-estrutura específicas. Considerando, ainda, o fator econômico, a meta do teste de software é encontrar os defeitos o mais cedo possível no processo de seu desenvolvimento.

O processo de teste é composto por uma sequência de atividades, são elas: planejamento, projeto, implementação, execução e análise dos resultados, verificação de término, e por fim, o balanço final. Este trabalho está centrado na primeira atividade do processo de teste que é o planejamento.

Planejamento é a distribuição racional no tempo dos recursos disponíveis para a realização de alguma atividade. É um trabalho que antecede a

execução dos testes e são elaborados pelos gerentes de teste. Planejar é, de modo geral, decidir antecipadamente o que deve ser feito e quando deve ser feito, ou seja, é traçar uma linha de ação. É justamente nesta etapa que será descrito o escopo dos testes, identificando métodos que serão empregados, recursos necessários, cronograma de atividades, pessoal necessário, itens que serão testados, itens que não serão testados, características dos itens testados e responsabilidades. Como resultado final desta etapa, teremos o documento plano de teste (Pressman, 2005).

Consideremos a tarefa de planejamento de teste de um software qualquer. Consideremos ainda um gerente de teste encarregado de executar o planejamento de teste, o qual deseja efetuar análises sobre um determinado conjunto de dados desta fase de planejamento. Essas análises têm como objetivo revelar uma série de informações sobre um subsistema que fazem parte de um sistema maior e que serão submetidos ao processo de teste, para que seja possível identificar quais as partes apresentam maior risco e que merecem maior atenção.

Em posse destas informações, o gerente terá a possibilidade de priorizar o teste de acordo com a política de teste da empresa, ou seja, a visão da empresa em relação ao teste de software, o que inclui quais os objetivos do teste, as retrições econômicas, prazos, controle de qualidade da atividade de teste, ferramentas, técnicas e treinamento. Desta forma, o gerente de teste tem condições de priorizar o processo de teste das partes que merecem maior atenção.

Com o objetivo de facilitar o planejamento de teste foi definida uma técnica, apoiada na visualização de informação. Para a aplicação desta técnica foi desenvolvido um protótipo, denominado TestPlan, que exibe, de forma gráfica, informações a respeito da fase de planejamento do teste de software. Pretende-se, com esse protótipo, auxiliar o gerente de teste na tomada de decisão, exibindo dados relevantes sobre o sistema que se encontra em desenvolvimento, enfatizando as áreas mais críticas, com base em métricas de software.

Desta forma, analisando os dados sobre esta situação, o gerente de software tem como objetivo obter uma melhor compreensão das informações contidas nestes subsistemas para tomar decisões que efetuem possíveis melhorias no processo de teste de software. Visto que as visualizações gráficas têm

demosntrado grande enfiência em relação facilidade de intepretação e compreensão de um grande conjunto de informações (Fayyad *et. al.*, 1996). Assim, por meio das representações visuais, pretende-se que o TestPlan forneça apoio cognitivo através de mecanismos que explorem as vantagens da percepção humana, acelerando o processamento visual.

Para o gerente de teste efetuar as análises desejadas, com o auxílio do computador, é preciso que ele informe os dados a serem analisados, para que este processe e retorne ao gerente de teste um resultado que o auxilie nas análises. Sendo assim, o problema a ser abordado neste projeto de pesquisa pode ser descrito da seguinte forma: como transformar um conjunto de métricas e informações sobre o sistema e seus subsistemas em uma visualização gráfica para apoiar o gerente de teste no processo de planejamento do teste de software.

Como proposta para solucionar este problema, o presente trabalho propõe uma técnica que pode apoiar o gerente de teste a conseguir respostas às suas análises, por intermédio do desenvolvimento de um protótipo que permite aos gerentes de teste consultar uma base de dados com informação sobre os subsistemas escolhendo alguns atributos das tabelas de dados. Através da escolha desses atributos, o TestPlan apresenta o resultado em uma estrutura visual, apoiada na visualização de informação, aos gerentes de teste com o intuito de apoiá-lo na análise e compreensão destes resultados a fim de obter a resposta desejada. Ao visualizar os dados o gerente de teste tem a possibilidade de priorizar as partes que merecem maior atenção, ou seja, os que estão mais propensos a riscos. Para isso, é preciso utilizar uma lista com os itens mais prioritários e com o tempo estimado para realizar a tarefa, para chegar a este resultado utilizaremos o *Product Backlog*, artefato da metologia Scrum (Schwaber, 2004). Inserido neste contexto, esta dissertação apresenta a aplicação do Scrum na fase de planejamento do teste de software.

A utilidade das métricas deve ser traçada antes do início de sua implantação para a avaliação de um software (Pressman, 2005). Há várias características associadas com o emprego das métricas de software. Sua escolha requer alguns requisitos, inserido no contexto deste trabalho as métricas

selecionadas foram escolhidas para subsdiar o processo de tomada de decisão, com o objetivo final de melhorar a qualidade do planejamento do teste de software.

Modelos que possam estimar o esforço das atividades de teste contribuem para uma prática mais sistematizada, pode ser incorporada em planos de teste e auxiliar nas atividades de garantia da qualidade de software (Maldonado, 1992).

Os subsistemas contêm uma série de informações sobre medidas de software relacionadas a eles e que serão abordadas nesta dissertação, são elas: medida de tamanho – obtida através da quantidade de linhas de código utilizando a métrica LOC (*Lines of Code*) (Pressmam, 2005); medida de complexidade de código – métrica de Halstead (Halstead, 1977); e por fim, medida da exposição ao risco (Pritchard, 1997). Sendo assim, essas métricas podem fornecer meios para os gerentes de software priorizarem os testes nas partes do código que apresentam maiores valores nas métricas.

Entretanto, o desenvolvimento de software orientado a objetos encontra-se bastante difundido e utilizado em diversas empresas desenvolvedoras de software. O mesmo pode-se dizer da utilização dos principais modelos de UML (*Unified Modeling Language*). Dentre os vários modelos UML, destaca-se o modelo de caso de uso (UC). Esses, normalmente, produzido nas fases iniciais do projeto o que torna possível estabelecer métricas baseadas em casos de uso. O que possibilita que essas, além das citadas anteriormente, sejam utilizadas na fase de planejamento do teste de software.

Dentre as métricas baseadas em caso de uso, destacam-se as métricas: Pontos por Caso de Uso (*Use Case Points* – UCP) Karner (1993), Pontos por Tamanho de Caso de Uso (*Use Case Size Points* – USP) (Braz, 2004) e, Pontos por Tamanho de Caso de Uso *Fuzzy* (*Fuzzy Use Case Size Points* – USPF)(Braz, 2004).

Existem na literatura muitos modelos para estimar o custo e esforço de desenvolvimento de software. São geralmente métricas relacionadas ao código fonte tais como LOC, métricas da Ciência de Software de Halstead ou em funcionalidade tais como Ponto de Função (Albrect, 1979).

Os modelos acima têm sido adaptados a software orientado a objetos (Fetcke, 1997), entretanto, o uso de métricas baseadas em casos de uso têm se mostrado bastante promissor, principalmente porque elas estão disponíveis nos primeiros estágios do desenvolvimento.

Desta forma, este trabalho além de propor métricas relacionadas ao código fonte ele ainda acrescenta métricas que não depedem da estrutura de um código de sistema ou de sua representação gráfica, mas a partir da estrtura interna de sua especificação. O teste, portanto, passa a ser de mais alto nível, no qual os elementos a serem acionados consitem nos elementos estruturais de uma especificação no software.

Um dos maiores problemas identificados nos projetos atuais é o grande dinamismo e complexidade dos negócios. Os sistemas se tornaram cada vez mais complexos e o tempo para o desenvolvimento tornou-se cada vez menor. Além desses problemas, temos também as frequentes mudanças nas especificações de um sistema, que deverão ser alteradas durante o seu desenvolvimento ou após a sua finalização. Sem contar com o surgimento de novas tecnologias, tanto de software quanto de hardware, que surgem a cada dia. Esse panorama intensifica ainda mais a premissa que a atividade de teste de software deve ser inicada o mais cedo possível, criticando o modelo cascata, no qual a atividade de teste está presente somente na fase final do projeto.

Então, o que se propõe é um modelo que não seja tão rígido como o cascata e sim um modelo no qual erros, riscos e inconsistências possam se tornar mais evidentes, o quanto antes. Sendo assim, o TestPlan trabalhará com sistemas que são desenvolvidos seguindo a metologia do RUP (*Rational Unified Process*) (Rational, 2008), ou seja, sistemas que no seu processo de desenvolvimento fazem uso de iterações para evitar ao máximo o impacto de mudanças no projeto e assim enfatizam os pontos de maior risco, o mais cedo possível nos projetos.

Visando analisar a solução proposta, esta implementação possibilitou efetuar um experimento, envolvendo um software de aplicação comercial, desenvolvido em Java, a escolha se deu pelo software ter o código fonte e a documentação funcional disponível.

Cada capítulo, seção e subseção deste texto discute aspectos relevantes para o desenvolvimento da proposta apresentada. Desta forma, o Capítulo 2 trata da revisão bibliográfica que está dividida da seguinte forma: a Seção 2.1 apresenta a referência sobre teste de software, a Seção 2.2 aborda aspectos relacionados à visualização de informação, a Seção 2.3 expõe as métricas de software que serão caracterizadas no protótipo, a Seção 2.4 apresenta a metodologia Scrum.

O Capítulo 3 apresenta o desenvolvimento do protótipo. O Capítulo 4 o estudo de caso. O Capítulo 5, a conclusão sobre o trabalho e posteriomente a estes capítulos são apresentadas as Referências Bibliográficas relacionadas ao texto.

2 REVISÃO DA LITERATURA

Neste capítulo é apresentado um referencial teórico sobre os temas relevantes para a proposta deste trabalho. Os principais conceitos são sobre teste de software, visualização de informação, métricas de software, RUP e Scrum que serão aplicados no protótipo e no estudo de caso.

2.1 TESTE DE SOFTWARE

Devido à crescente utilização de sistemas computacionais, atuando praticamente em todas as áreas da atividade humana, há uma forte exigência por software com alta qualidade e produtividade, motivo pelo qual a Engenharia de Software cresceu significantemente nos últimos anos, firmando técnicas, critérios, métodos e ferramentas.

A busca incessante da área de Garantia da Qualidade de Software (GQS) por novas técnicas e ferramentas que auxiliem na verificação e validação dos sistemas, faz com que o teste seja visto como uma atividade de grande importância, possibilitando a descoberta de erros introduzidos durante o desenvolvimento do software (Pressman, 2005).

2.1.1 OBJETIVOS DO TESTE DE SOFTWARE

Com a evolução da tecnologia e a intensa utilização do computador para executar as mais diversas tarefas, amplia-se a necessidade de desenvolver sistemas mais sofisticados e complexos, sendo que os mesmos deverão ser confiáveis, exigindo que o processo de desenvolvimento de sistemas seja organizado e planejado apropriadamente para garantir a qualidade do produto final.

Como resultado deste crescimento e a importância desta atividade no ciclo de vida de um software, houve a necessidade que profissionais da área se especializassem em detectar problemas que possam não ter sido localizados nas demais etapas de desenvolvimento do sistema. Esse profissional deve ser capaz de

reconhecer e antever uma solução para o problema, em qualquer etapa do desenvolvimento.

O teste é uma das atividades fundamentais da etapa de desenvolvimento de um software, sendo de extrema importância na busca incessante da garantia da qualidade do mesmo.

A sua importância é tanta que para muitas empresas de desenvolvimento de software, a etapa de teste é vista e tratada como uma das mais necessárias, dispondo, em muitos casos, de cerca de 30 a 50% de custos e tempo total do projeto (Pressman, 2005 ; Sommerville, 2003).

O teste de software surge como uma atividade fundamental visando a qualidade do software, ou seja, que este esteja de acordo com as especificações do usuário e que esteja funcionalmente correto, pois o objetivo do teste de software é detectar a presença de defeitos (Myers, 2004)**,** com isso, possibilitando-se chegar a um software o mais correto possível ao final desta etapa.

O objetivo da atividade de teste pode ser definido da seguinte maneira (Myers, 2004): a atividade de teste é o processo de executar um programa com a intenção de detectar a presença de defeitos. Desta forma, um bom teste de software é aquele que apresenta uma alta probabilidade de revelar algum defeito que ainda não foi descoberto. Se o teste revelar um defeito que ainda não foi descoberto diz se que o teste foi bem sucedido.

Segundo Bartié (2002), quanto mais procedimentos de testes forem empregados em software, maior será o nível de validação obtido do produto o que resultará em um maior nível de qualidade do software desenvolvido.

O ponto de partida para qualquer projeto de teste de software é uma metodologia de teste consistente e adequada ao ambiente de desenvolvimento da empresa (Molinari, 2006).

Quanto mais erros forem encontrados num sistema, numa quantidade mínima de tempo e de esforço, maior a qualidade do teste de software, pois pode-se lidar com as falhas do sistema antes que o cliente o encontre. Para que isso ocorra, é necessário conduzir sistematicamente, durante todas as etapas de

desenvolvimento do sistema, casos de testes usando técnicas disciplinadas, e não somente ao término do sistema, como normalmente ocorre.

Na seção seguinte são apresentadas algumas terminologias, segundo o padrão IEEE (IEEE, 1990), que são fundamentais para o entendimento deste trabalho.

2.1.2 DEFINIÇÃO DE ENGANO, DEFEITO, ERRO E FALHA

Os termos descritos a seguir são baseados no padrão IEEE 620.12 (IEEE, 1990). No contexto deste trabalho, engano (*mistake*) é uma ação humana que introduz um defeito no software. O defeito (*fault*) em um programa de computação é um passo, processo ou definição de dados incorretos, como por exemplo, um comando ou instrução incorreta que ao ser executado pode gerar uma falha. O erro (*error*) é um estado incorreto, intermediário ou final, de execução do software que pode produzir um resultado incorreto, pode levar a uma falha no software. A falha (*failure*) é a ocorrência de uma discrepância entre o resultado observado do software e o resultado prescrito pelos requisitos.

Assim, um engano introduz um defeito no software que quando ativado pode produzir um erro que se propagado até a saída do software constitui uma falha, como esboça a Figura 1. É muito importante diferenciar estes conceitos, uma vez que os testes de software detectam os defeitos através das falhas geradas pela execução do software. Se nenhuma falha ocorrer, o teste não revela os defeitos. Embora um sistema tenha passado por um bom teste de software, nunca será possível garantir que ele esteja isento de conter novas defeitos (Myers, 2004).

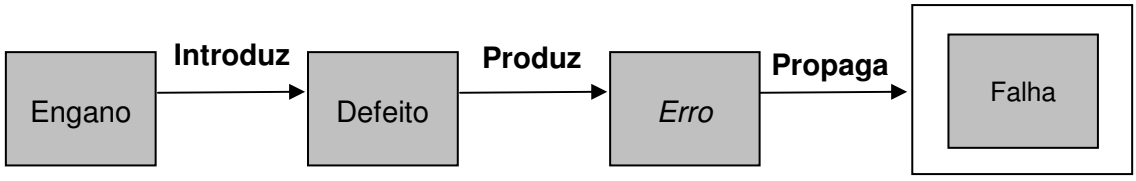


Figura 1: Diferenças entre Engano, Defeito, Erro e Falha.

2.1.3 JUSTIFICATIVAS DOS TESTES DE SOFTWARE

Uma das principais razões que levam um software a ter defeitos é por envolver atividades humanas na quais as oportunidades de enganos humanas são inúmeras (Rocha, 2001). As pessoas possuem limitações em executar tarefas e possuem dificuldades para se comunicarem com perfeição, ou seja, o ser humano é falível, o que pode proporcionar grandes danos ao processo de desenvolvimento. Esses enganos podem ocorrer logo na fase inicial do processo de criação do software bem como durante o desenvolvimento e em fases posteriores.

Desta forma, quanto mais cedo a presença de um defeito for revelada, menor será o custo com a sua correção e maior será a probabilidade de corrigi-lo corretamente (Rocha, 2001). Há uma estimativa que se um defeito for descoberto no início, o custo de sua correção está por volta de dez centavos de dólar, já quando descoberto na fase de *release*, a correção chega a ultrapassar cem dólares (Molinari, 2006). A detecção prévia dos defeitos é de suma importância e pode ser revelada na análise de Myers (2004) que emprega a denominada “regra de 10” aos custos da correção dos defeitos. A regra refere que quanto mais tarde o defeito for revelado mais caro torna-se a sua correção numa proporção de 10 vezes por fase de do ciclo de desenvolvimento (Bastos, 2006).

O teste de software pode ser aplicado devido a várias questões, que variam de acordo com os princípios e objetivos das organizações. Uma atividade de teste pode ser empregada por questões de qualidade, economia, segurança, confiabilidade ou negócio.

Por questões de qualidade, a empresa possui preocupações quanto à satisfação do cliente e se o produto está em conformidade com os requisitos do software. Segundo Bartié (2002), os esforços de teste devem ser dirigidos pelo risco existente e pelo impacto que a falta de qualidade provocará ao ambiente interno e externo da organização.

Por questões de economia, a ênfase é dada para os altos gastos em retrabalho, devido à manutenção corretiva causada por falhas de especificação,

falhas de projetos e falhas de programação. Estatísticas indicam que, em média, 50% do tempo de um analista é consumido por atividades de teste (Pressman, 2005).

Por questões de segurança, se o software gerencia informações delicadas, ou provoca ações que possam prejudicar ou beneficiar pessoas, constitui um alvo para o acesso impróprio ou ilegal. Desta forma, um bom teste de segurança busca verificar se todos os mecanismos de proteção embutidos no sistema protegem de fato a acessos indevidos.

Por questões de confiabilidade, quando exigida para um sistema de software pode ser aumentada através da realização de testes e eliminação dos defeitos encontrados. A confiabilidade é a probabilidade de que o software não falhe num período de tempo de execução e em um ambiente especificado.

Por questões de negócio, não uma questão de qualidade. Desta forma, testa-se um software para avaliar o risco de se liberar um produto de software, não para encontrar todos os defeitos do produto. Nessa perspectiva, a equipe de teste deve, em primeiro lugar, entender os riscos associados ao negócio, e então desenvolver estratégias de testes que abordem esses riscos. Os riscos mais altos devem receber maiores recursos de teste.

2.1.4 CUSTOS IMPLICADOS POR NÃO SE TESTAR UM SOFTWARE

A falta de um plano de teste pode ocasionar prejuízos diretos às empresas e/ou a perda de confiança do público e, como conseqüência, à perda de mercado. Esses fatos ocorrem particularmente em sistemas que apresentam um nível crítico de segurança, em geral, em aplicações que exigem um alto grau de confiabilidade, como por exemplo: sistemas que apresentem risco de morte (avião, metrô) e transações bancárias.

Até mesmo em sistemas que não lidam com aplicações críticas, mas que estejam diretamente relacionados com o público, erros podem ainda ter sérias implicações comerciais para uma organização, alguns exemplos são: aplicações na Internet, software de demonstração e software livre.

2.1.5 DIFICULDADES EM TESTAR UM SOFTWARE

Há varias questões relacionadas com a dificuldade em se testar um software. Para se testar um software com eficiência, deve-se a *priori* conhecer os profundamente os sistemas e a dificuldade também está relacionada com o fato dos sistemas não serem simples nem fáceis de entender (Hetzel, 1987). Desta forma, é compreensível que a atividade de teste não é uma tarefa comum devido às próprias características do software, entre elas, está o fato do software ser complexo e as pessoas possuírem limitações para compreender a sua complexidade. À medida que o software vai sendo desenvolvido, aumenta a dificuldade de compreensão da equipe do sistema como um todo. Outro ponto a considerar é o fato do software ser um bem intangível, imponderável, diferentemente de outras engenharias no qual o processo de desenvolvimento pode ser observado e avaliado o seu processo de desenvolvimento. Por fim, o processo de teste envolve pessoas com perfis diferenciados, como desenvolvedores, gerentes e usuários.

Segundo Bartié (2002), quando se discute sobre atividade de teste os problemas mais comuns relatados são:

- Alto custo do processo;
- Não há conhecimento sobre a relação entre custo/benefício na implementação de testes;
- Falta de profissionais especializados na área o que dificulta a eficiência do processo;
- Dificuldade em implantar um processo de teste;
- Falta de conhecimento sobre procedimentos e técnicas de testes adequados;
- Desconhecimento de como planejar a atividade de teste e preocupação com teste somente na fase final do projeto ou quando estiver parcialmente pronto;

- Ausência de procedimento de testes automatizados exigindo grande esforço e tempo quando executados manualmente.

Analisando-se os problemas descritos acima, observa-se que há uma relação direta com a ausência ou limitação das atividades de planejamento e controle de testes de software.

A indisponibilidade de tempo e recursos, bem como a falta de ferramentas apropriadas para a execução de teste, são os principais problemas encontrados pelas equipes de teste (Rocha, 2001). Porém, dadas todas essas dificuldades em se testar um software, o que se observa é que as grandes empresas vêm demonstrando interesse pelo teste de software, já que o custo de não testar implica em sistemas falhos, inseguros e de baixa qualidade o que repercutem na perda de mercado.

2.1.6 LIMITAÇÕES DO TESTE DE SOFTWARE

Uma das grandes limitações do processo de teste está relacionada às características específicas do teste de software, pois a atividade de teste em si não garante a ausência de defeitos no software; o teste pode apenas indicar que defeitos estão presentes e não se pode afirmar que um software testado não contém defeitos.

As limitações do teste estão relacionadas principalmente com dois aspectos: tempo e domínio dos dados de entrada (Rocha, 2001). É importante salientar que o software não pode ser exaustivamente testado, devido ao domínio de entrada ser muito grande, a ponto de ser impossível testar todas as possibilidades.

2.1.7 FASES DA ATIVIDADE DE TESTE

O processo de desenvolvimento de software envolve uma série de atividades e para cada fase de desenvolvimento do software existe uma fase de teste correspondente (Pressman, 1995). O que se propõe é um processo de desenvolvimento em “V” mostrado na Figura 2, abrangendo tanto as fases de desenvolvimento quanto às fases de teste. O modelo apresenta a importância do

teste de software desde o início do desenvolvimento do software, enfatizando a idéia que as atividades de teste não representam apenas uma fase, mas sim, uma parte que completa o ciclo de desenvolvimento do software (Hetzel, 1987).

O processo de teste inicia com os testes de unidades, que visam verificar se cada módulo ou unidade satisfaz à sua especificação, estabelecida no projeto detalhado. Após testar separadamente cada módulo, estes são agrupados para compor os subsistemas, conforme a arquitetura do sistema definida no projeto preliminar, sendo esta a fase de testes de integração; o objetivo deste é revelar falhas de interação entre os módulos e subsistemas.

Os testes de validação, segundo Pressman (1995), visam validar o software, ou seja, determinam se o software satisfaz aos requisitos especificados na fase de análise. De acordo com a nomenclatura IEEE (IEEE, 1990), esses testes são denominados testes de aceitação, sendo realizados pelos clientes para determinar se aceitam ou não o produto.

Finalmente os testes de sistema visam exercitar o sistema como um todo, incorporando todos os componentes tanto os componentes tanto de hardware quanto de software para determinar se o sistema completo satisfaz à sua especificação.

Os testes de regressão constituem uma categoria especial de testes e visam assegurar que alterações em partes do sistema não afetaram partes inalteradas. Assim, há a necessidade de se guardar casos de teste que foram aplicados, para que se possa reaplicá-los a cada alteração. Esses testes são aplicados principalmente em fase de manutenção, em que alterações são introduzidas após o sistema entrar em fase de operação, com o intuito de apontar correções, ou adaptar o sistema a novas plataformas ou ainda, para introduzir novos requisitos. Esses teste também são úteis durante os testes de integração pois a cada nova unidade integrada deve-se verificar se as partes já testadas não foram afetadas.

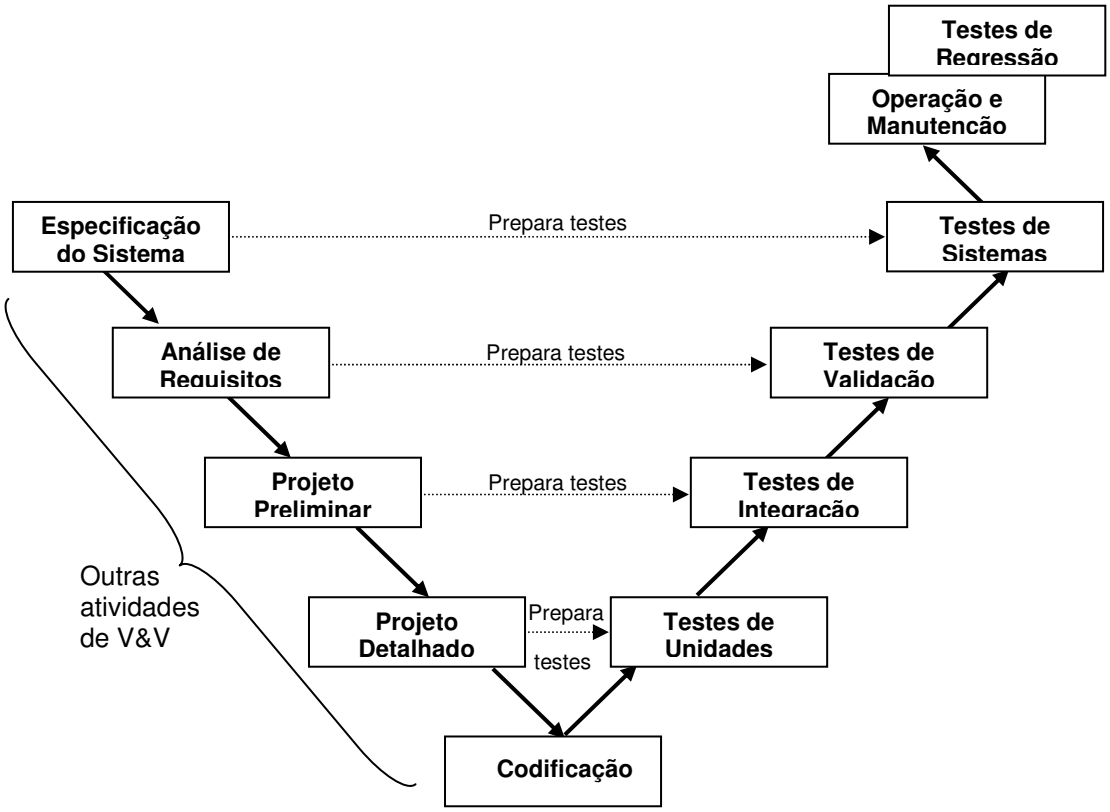


Figura 2: Fases de teste no processo de desenvolvimento do software (Fonte: adaptado de Molinari, 2006).

Existem diferente fases de testes, tais como: testes de unidades, testes de integração, testes de sistemas. Nos itens que seguem serão apresentadas cada uma delas.

TESTE DE UNIDADE

Os testes de unidades têm por objetivo verificar cada unidade que compõe o software, isoladamente, para determinar se cada uma delas realiza o que foi especificado. (Pressman, 1995). Os testes de unidade geralmente são executados pelo próprio programador, pois requerem o conhecimento da estrutura interna. Segundo McConnell (2004) os testes de unidade detectam de 15 a 50% dos defeitos.

TESTE DE INTEGRAÇÃO

As unidades testadas vão sendo integradas para formar os módulos e estes para formar os subsistemas, conforme a arquitetura definida no projeto preliminar. Os testes realizados nesta fase em que o sistema está sendo construído são os chamados testes de integração.

O teste de integração pode ter sua execução iniciada assim que alguns componentes ficarem prontos. No contexto do ciclo de desenvolvimento, um componente pronto significa um componente implementado e com os testes de unidade aprovados, ou seja, componentes individuais funcionando corretamente e atingindo os seus objetivos (Pfleeger, 2004).

Os testes de integração têm por objetivo encontrar falhas de integração entre as unidades, e não mais em testar a funcionalidade das mesmas. Procuram-se, portanto falhas que não possam ser encontradas pelos testes de unidades. Segundo McConnell (2004) os testes de integração detectam de 25 a 40% dos defeitos.

TESTE DE SISTEMA

Após a integração das unidades devidamente testada, deve-se realizar o teste de sistema que tem como objetivo determinar se o software e demais elementos que compõem o sistema, tais como hardware e banco de dados, atendem aos requisitos especificados. Os tipos de testes realizados e que são de interesse para o software são (Pressman, 1995):

- Testes de Segurança: têm por objetivo determinar se os mecanismos de proteção de um sistema contra intrusões (situações de erro provocadas intencionalmente por humanos funcionam adequadamente).
- Teste de Estresse: visam testar o sistema nos limites de sua capacidade; por exemplo: o que acontece se o número máximo de interrupções que o sistema pode atender simultaneamente é excedido? Ou se o limite máximo de memória permitido (ou outros recursos) é excedido?
- Teste de Desempenho: têm por objetivo determinar se o desempenho do sistema integrado é adequado, de acordo com os requisitos do

sistema. Testes de desempenho podem ser realizados ao longo do desenvolvimento, mas somente quando todos os componentes do sistema são finalmente integrados é que se pode ter uma medida real do desempenho do sistema. Esse tipo de testes geralmente requer alguma instrumentação do sistema para monitorar uso de recursos, tempos de resposta, entre outros.

TESTE DE ACEITAÇÃO

O teste de aceitação avalia o comportamento do sistema em relação aos requisitos do cliente, que define tarefas específicas para determinar se seus requisitos foram atendidos.

TESTE DE REJEIÇÃO

Testes de regressão são necessários cada vez que são feitas mudanças no software e consistem na re-execução dos casos de testes já aplicados para determinar se as alterações não produziram nenhum efeito indesejado.

A re-execução dos testes pode ser feita manualmente ou através do uso de ferramentas. Que permitem capturar os casos de teste (para que eles possam ser reaplicados posteriormente) e os resultados observados (para que eles possam ser comparados aos resultados obtidos posteriormente).

2.1.8 CASOS DE TESTE

O caso de teste descreve como testar uma parte do sistema. Essa descrição é determinada por um conjunto de entradas, condições de execução e os resultados esperados na execução do teste.

Segundo o RUP (2008), “um caso de teste é um conjunto de passos que descrevem um cenário de teste bem definido cuja principal função é comparar as respostas dos estímulos gerados pelos passos com um resultado esperado”.

Um conjunto de casos de teste tem como objetivo revelar a maior quantidade de defeitos existentes no software a ser testado.

2.1.9 TÉCNICAS E CRITÉRIOS DE TESTE

Existem basicamente duas formas de se testar um software, de maneira *ad hoc* ou sistemática. Na primeira, os casos de teste são derivados somente da intuição da pessoa que irá realizar o teste. Na segunda forma, os casos de teste utilizam um conjunto de técnicas, métodos e critérios para conduzir e avaliar a qualidade do teste de software e reduzir o tempo e os custos associados à atividade de teste.

Segundo Pressman (1995), dentre as técnicas de verificação e validação, o teste de software é uma das atividades mais empregadas e também a mais onerosa, em alguns casos, podendo consumir 40% dos custos de desenvolvimento do software. Portanto, empregar técnicas e critérios de teste, de forma sistemática, tem contribuído para reduzir o tempo e os custos associados à atividade de teste e ainda apresentar, de uma formar eficiente, os defeitos existentes no software.

Os critérios de teste podem ser classificados em duas técnicas: funcional, estrutural. A diferença entre essas técnicas está na origem da informação utilizada na avaliação e construção dos conjuntos de casos de teste (Rocha, 2001).

É importante enfatizar que as técnicas de teste devem ser aplicadas em conjuntos, devido à grande diversidade de critérios de teste existentes e por poderem revelar diferentes tipos de defeitos. Segundo Rocha (2001), nenhuma técnica é totalmente completa e nenhuma delas empregadas isoladamente é suficiente para garantir a qualidade da atividade de teste.

2.1.10 TÉCNICA FUNCIONAL

O teste funcional também é conhecido por teste da caixa preta e recebe esta denominação pelo fato de tratar o software como uma caixa cujo conteúdo é desconhecido e só é possível visualizar o lado externo (Pressman, 1995). Desta forma, a técnica funcional utiliza métodos para garantir que os requisitos do sistema são plenamente atendidos pelo software que foi construído, não se preocupando em verificar como ocorrem internamente os processamentos no software.

A técnica funcional testa o sistema do ponto de vista do usuário, não considerando a estrutura interna do código, apenas testando as funcionalidades do software sem a necessidade de se ter o código fonte. Restringe-se na análise da funcionalidade do programa e estabelece-se unicamente na especificação dos requisitos para indicar que tipos de saídas são esperados para um determinado conjunto de entradas (Becker, 2003).

O teste funcional, por se basear na especificação do software, surge como uma técnica capaz de reduzir os custos inerentes ao processo de teste uma vez que casos de teste podem ser obtidos conjuntamente ao seu desenvolvimento. Em função disto, muitas pesquisas estão sendo desenvolvidas com o objetivo de produzir técnicas efetivas para a derivação de casos de teste a partir da especificação dos casos de uso de um sistema.

Vale ressaltar que a aplicação da técnica funcional não exclui a aplicação da técnica estrutural. Essas técnicas são complementares e não exclusivas. O que significa a obtenção de uma maior qualidade no software se ambas forem aplicadas.

2.1.10.1 CRITÉRIO DE TESTE BASEADO EM CASO DE USO

A análise de requisitos está relacionada ao processo de adquirir os objetivos do cliente com o sistema que ele deseja que seja desenvolvido, um requisito é uma característica ou propriedade que deve ser implementada no software, ou seja, quais são as operações que o sistema deve realizar e quais são as restrições que existem sobre elas. A captura de requisitos do software é uma atividade fundamental para o desenvolvimento do mesmo (Larman, 2004).

Uns dos grandes problemas parar capturar os requisitos é comunicar com clareza o que realmente é necessário para o cliente e os membros da equipe de desenvolvimento. Os casos de uso são instrumentos de grande valia para manter esse processo simples e compreensível para todos os interessados no sistema, e por isso são amplamente utilizados para compreender e validar os requisitos (Larman, 2004).

Os casos de uso podem ser empregados para detectar o comportamento pretendido do sistema, sem ser preciso especificar como esse comportamento é implementado. O modelo de especificação de software utilizado, nesse caso, consiste no diagrama de caso de uso da UML (OMG, 2008) um modelo de análise de requisitos funcionais responsável por dirigir todo processo de desenvolvimento do software. O teste baseado neste diagrama torna possível validar os requisitos de software já na primeira fase do desenvolvimento do software. A vantagem de se desenvolver casos de testes justamente nessa fase consiste na possível identificação de erros, omissões, inconsistências e redundâncias em uma fase cujo custo da remoção de erros é bem menor do que nas fases posteriores do desenvolvimento (Ryser e Glinz, 1999). Isso permite com que a especificação seja melhorada antes do programa ser escrito.

A descrição de um caso de uso deve ser constituída de várias informações, tais como: o seu nome, uma curta descrição referente aos seus objetivos, uma descrição textual dos fluxos de eventos que podem ocorrer durante sua execução, uma descrição das restrições que necessitam ser satisfeitas para que o caso de uso tenha início, e uma descrição das restrições satisfeitas quando a execução do caso de uso é finalizada (Sommerville, 2003).

Dentre as informações descritas acima, para a geração de casos de testes, a parte mais importante é a dos fluxos de eventos. A descrição de um caso de uso é composta por um fluxo básico e por alguns fluxos alternativos. O fluxo básico descreve as ações que acontecem em uma execução normal do caso de uso. Já os fluxos alternativos descrevem tanto ações alternativas que podem acontecer durante a execução do caso de uso, como ações que encadeiem na geração de falhas durante a sua execução. Dependendo do tipo de ação que um fluxo alternativo execute, faz com que eles também sejam denominados de fluxos secundários e de fluxos de exceção.

A Figura 3 representa a estrutura típica destes fluxos de eventos. A seta preta representa o fluxo básico de eventos, e as curvas representam fluxos alternativos. Note que alguns fluxos alternativos retornam ao fluxo básico, enquanto os fluxos alternativos podem retornar ao fluxo básico (fluxos alternativos 1 e 3), podem originar-se de outro fluxo alternativo (fluxo alternativo 2), ou podem terminar

o caso de uso sem retornar ao fluxo básico (fluxos alternativos 2 e 4) (Hermann, 2008).

Figura 3: Fluxo de eventos de um caso de uso
(Fonte: Heumann, 2008).

Os casos de teste, para o teste funcional, são derivados de casos de uso. É fundamental criar casos de teste para cada cenário do caso de uso. Os cenários de caso de uso são obtidos através da descrição dos caminhos que percorrem o fluxo básico e os fluxos alternativos, desde o início até a finalização do caso de uso. Um cenário de caso de uso pode ser descrito como uma instância do caso de uso. Como a execução de um caso de uso pode ocorrer por vários caminhos. Cada caminho identificado corresponde a um cenário. Para facilitar o entendimento, os cenários são inseridos em uma única tabela, como ilustra a Tabela 1. Desta forma é possível agrupar todas as informações pertencentes a um caso de uso específico em uma tabela (Heumann, 2006).

Tabela 1: Cenários para o caso de uso da Figura 3

Cenário	**Fluxo Inicial**	**Alterações**		
Cenário 1	Fluxo básico			
Cenário 2	Fluxo básico	Fluxo alternativo 1		
Cenário 3	Fluxo básico	Fluxo alternativo 3	Fluxo alternativo 2	
Cenário 4	Fluxo básico	Fluxo alternativo 3		
Cenário 5	Fluxo básico	Fluxo alternativo 3	Fluxo alternativo 1	
Cenário 6	Fluxo básico	Fluxo alternativo 3	Fluxo alternativo 1	Fluxo alternativo 2
Cenário 7	Fluxo básico	Fluxo alternativo 4		
Cenário 8	Fluxo básico	Fluxo alternativo 3	Fluxo alternativo 4	

Heumann (2008) propõe uma forma de derivação de casos de teste a partir de casos de uso que faz uso dos diferentes cenários obtidos de um caso de uso para derivar casos de teste. A metodologia de Heumann para criar casos de teste a partir dos casos de uso é obtida através de três passos:

1) para cada caso de uso, gerar uma lista de cenários de casos de uso;

2) para cada cenário, identificar, ao menos, um caso de teste e as condições que o farão ser executado;

3) para cada caso de teste, identificar os dados que são utilizados para o teste.

No primeiro passo, deve-se ler a descrição textual do caso de uso e identificar o fluxo principal e os fluxos alternativos criando uma tabela com todos os cenários possíveis para o caso de uso, como ilustrado na Tabela 2.

No segundo passo, os casos de teste para cada cenário devem ser identificados. Para isso, deve-se avaliar os cenários, revisar a descrição do caso de uso e então derivar o caso de teste. Cada cenário deve originar pelo menos um caso de teste. Os casos de teste são agrupados em uma tabela para facilitar a visualização e aumentar a organização do processo. Porém, nesta tabela não conterá dados reais de entrada, por ser uma etapa intermediária e importante para demonstrar claramente as condições que estão sendo testadas para cada caso de

teste. Os valores que esta tabela assumirá serão: *V*, *I* e *N/A* que significam *válido*, *inválido* e *não se aplica*, respectivamente.

Tabela 2: Caso de teste para o caso de uso (adaptado de Heumann, 2008)

Cenário	Dados de Entrada	Resultado Esperado
[*Identificador do cenário do caso de uso*]	[*V, I ou N/A*]	[*Descrição da saída esperada para o cenário*]

No terceiro e último passo, deve-se revisar e validar os casos de teste e identificar casos de teste redundantes ou faltantes. Depois de aprovados, os dados de teste devem ser identificados. Deve ser criada uma tabela que represente as condições, os dados de entrada e os resultados esperados para que os casos de teste possam ser executados., ou seja, devem ser identificados os dados que efetivamente serão usados para implementar e executar os testes. Isto deve ser feito substituindo os *V*s e *I*s da tabela gerada no segundo passo deste processo, por entradas para os casos de teste. A Tabela 3 mostra um exemplo dos casos de teste com as suas respectivas entradas.

Tabela 3: Caso de teste com entradas para o caso de uso (adaptado de Heumann, 2008)

Cenário	Dados de Entrada	Resultado Esperado
[*Identificador do cenário do caso de uso*]	*[Informações das condições de entrada para a execução do caso de teste]*	[*Descrição da saída esperada para o cenário*]

2.1.7 TÉCNICA ESTRUTUTAL

O teste estrutural, também, conhecido como teste da caixa branca ou caixa de vidro, é baseado na estrutura interna do software. Os testes abordados pela técnica estrutural utilizam métodos que objetivam identificar defeitos exercitando as estruturas internas do software (Bartié, 2002).

É importante salientar que o teste estrutural é aplicado para complementar o teste funcional (Pressman, 1995), visto que em algumas situações o

teste funcional não é capaz de detectar alguns defeitos, pois preocupa somente em comparar se as saídas produzidas pelo sistema estão de acordo com o esperado, não levando em consideração a funcionalidade interna do software. Desta forma, podem existir procedimentos internos que não afetam as respectivas saídas e que não serão detectados se for aplicado apenas o teste funcional.

A Figura 4 ilustra o esquema do teste estrutural, no qual a estrutura de controle do sistema é absolutamente necessária. Desta forma, torna-se necessário ter o código fonte do software para a aplicação dos casos de teste.

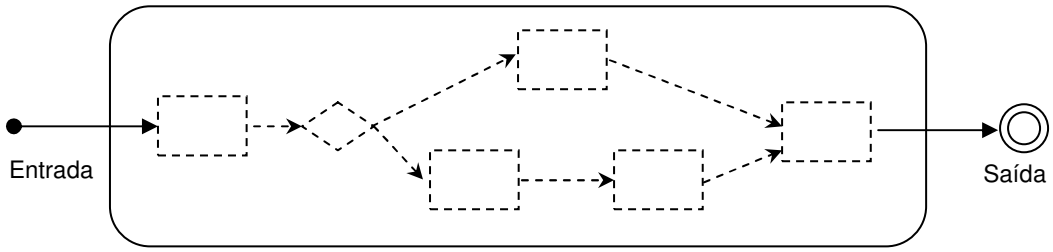


Figura 4: Técnica Estrutural.

O teste estrutural visa detectar os seguintes tipos de erros: comandos incorretos, estruturas incorretas, variáveis não definidas e erros de inicialização e finalização de *loops*.

A maioria dos critérios de teste dessa técnica utiliza o grafo de fluxo de controle (GFC) ou grafo de programa para representar o programa. Em um grafo de fluxo, cada estrutura de controle de um programa (seqüência, seleção e repetição) pode ser representada por um símbolo correspondente, como ilustra a Figura 5.

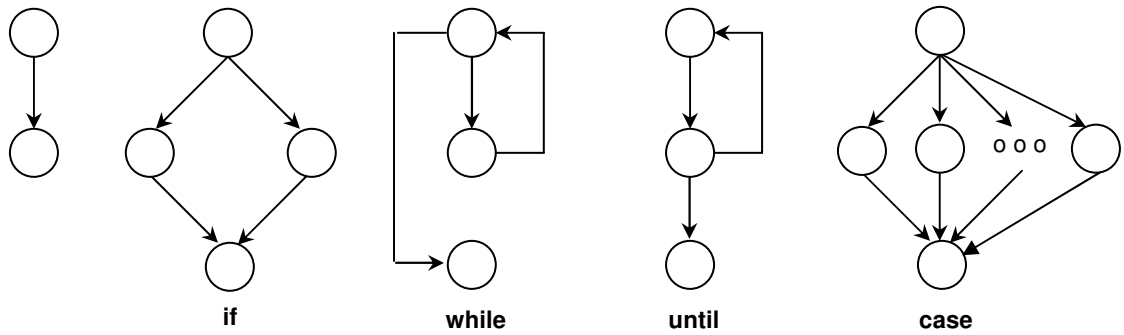


Figura 5: Representação das estruturas de controle de um programa em grafos de fluxo de controle (Fonte: Hetzel, 1987)

Por meio do grafo podem-se definir alguns conceitos importantes: os nós representam conjuntos de comandos executáveis, os arcos representam as estruturas de controle, a área delimitada pelos arcos é chamada região, um nó predicado é um nó contendo uma condição e um caminho de um programa é representado por uma seqüência de nós, como ilustra a Figura 6.

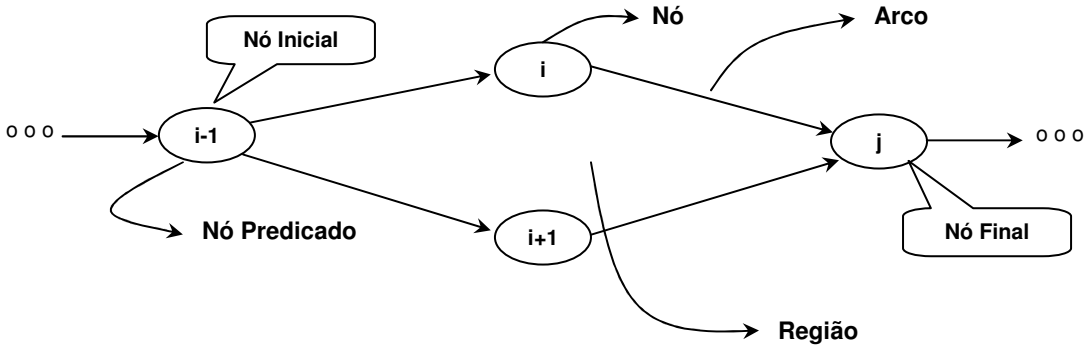


Figura 6: Grafo de Fluxo de Controle.

Um grafo de fluxo de controle é, portanto um grafo dirigido, com um único nó inicial e um único nó final, no qual cada nó representa um ou mais comandos de execução e cada aresta representa um possível desvio de um bloco para outro.

A aplicação de um caso de teste corresponde à execução completa do programa partindo do nó inicial indo até o nó final, percorrendo um caminho no grafo de fluxo do programa. Um caminho completo é uma seqüência de nós, começando no nó inicial e terminando no nó final.

2.1.11.1 COBERTURA DE COMANDOS (TODOS-NÓS)

Os critérios baseados em fluxo de controle usam características baseadas no controle de execução do programa, como comandos ou desvios, para determinar os casos de teste (Pressman, 1995).

A cobertura de comandos direciona o testador a construir casos de teste de tal forma que cada comando do programa seja executado pelo menos uma vez (Myers, 2004). Todos os nós do grafo de fluxo de controle deverão ser testados.

É um critério fácil de satisfazer, pois é um dos critérios menos exigentes no sentido de exercitar o programa.

2.1.12 PLANEJAMENTO DO TESTE DE SOFTWARE

O processo de teste é dividido em duas grandes fases: preparação e realização dos testes. Na fase de preparação é produzido o plano de teste e são projetadas as especificações para cada teste. Já, durante a realização, é a execução dos testes que foram especificados na fase de preparação, os defeitos são revelados e corrigidos e por fim, os relatórios de teste são escritos.

Como toda atividade de produção de software, o teste requer uma preparação cuidadosa que precede sua execução, incluindo os seguintes itens: planejamento de teste, projeto de teste e especificação dos casos de teste. Os resultados da execução do teste incluem as concordâncias e discordâncias entre as saídas esperadas e saídas obtidas, bem como eventuais incidentes observados.

O processo de teste é composto por seis atividades: planejamento, projeto, implementação, execução e análise de resultados, verificação de término e balanço final. Essas atividades são baseadas na norma para teste de unidade do IEEE (Pádua, 2001 *apud* IEEE Std. 1008 – *Std. For Sotfware Unit Testing*, 1987). Apesar de serem propostas para testes de unidade, são aplicáveis para todas as fases de teste. A Figura 7 mostra esse processo.

Figura 7: O processo de teste (Fonte: IEEE Std 1008).

Uns dos documentos gerados nesta fase é o plano de teste, como produto da fase de planejamento, dentro do processo de desenvolvimento e execução de teste, deve ser visto como a base de sustentação para a realização dos testes. Segundo Molinari (2006), “planejamento de teste é um processo” e “plano de testes é o resultado prático desse planejamento, é o seu fruto e artefato”. O planejamento solícito auxilia a projetar e organizar testes de um sistema de forma a garantir a sua adequação.

A finalidade do plano de teste de software é agrupar todas as informações necessárias ao planejamento e ao controle do esforço de teste referente a uma iteração específica. Ele faz a descrição da abordagem dada ao teste do software e é o plano de nível superior gerado e usado pelos gerentes para coordenar o esforço de teste. Ele prescreve o escopo, a abordagem, os recursos, a composição das tarefas, a ordem da execução das tarefas e o cronograma relacionados as atividades de teste a serem executadas (Sommerville, 2003; Pfleeger, 2004). Sua elaboração é de responsabilidade do gerente de teste.

O planejamento dos testes enfoca os aspectos gerenciais do processo, o que inclui:

- Identificação das áreas de risco que devem ser testadas;
- Determinação das fontes a serem utilizadas para a elaboração de casos de testes;
- Determinação dos itens a testar;
- Estabelecimento das condições de completeza dos testes para cada item a testar;
- Especificação das condições de término dos testes (término normal e anormal);
- Determinação dos recursos necessários, tais como: pessoas, software, hardware, ferramentas de teste;
- Determinação do cronograma de teste.

Os planos de teste não têm apenas interesse para a gerência do projeto, mas também são úteis para a equipe de teste e de desenvolvimento. (Sommerville, 2003). Para a equipe de desenvolvimento esses planos os ajudam a ter uma visão geral dos testes e a situar o seu trabalho nesse contexto. Os planos de teste também fornecem informações para o pessoal responsável por assegurar que os recursos necessários estejam disponíveis para a equipe de teste.

Planejamento não é uma tarefa trivial, mas que gera conseqüências negativas por esse motivo o planejamento de teste não é uma atividade muito praticada pelos gerentes de teste. As alegações fornecidas por gerentes de teste que não praticam o planejamento de teste normalmente são (Molinari, 2006):

- “prazo curto”;
- “não é possível testar todas as entradas”;
- “não é possível testar todas as combinações das entradas”;
- “não é possível testar todos os caminhos”;
- "não é possível testar todas as falhas potenciais, sejam de interface ou de requerimento incompletos;
- “prazo longo demais para teste”;
- “temos muitos problemas políticos internos, que nos impedem de focar em testes”;
- “temos muito problema de relacionamento pessoal; gerente que não fala com analista, analista que não fala com gerente, gerente que não fala com gerente”.

É inegável que o gerente de teste representa um papel fundamental. Ele é responsável por coordenar, supervisionar todos os processos e toda a atividade de teste. Segundo Hetzel (1987), são muitas as atividades que um gerente de teste deve desempenhar com responsabilidade pela definição e adequação de todas as atividades de teste e medição da qualidade. Por esses motivos é que o gerente torna-se um ponto chave na preparação e execução dos testes.

Segundo Bartié (2002), ferramentas de planejamento de testes devem auxiliar a maneira de operar o planejamento dos testes, demarcando os escopos, abordagens, recursos, agendando reuniões e elaboração do programa de atividades. Essas ferramentas também devem prestar auxílio no processo de documentação inicial, tornado possível planejamentos mais organizados que estabeleçam padrões e na preparação da avaliação de tempo e custos necessários. Quanto à análise de criticidade, essas ferramentas devem auxiliar o processo de priorização de sistemas, identificando quais devem ser testados primeiramente. Através deste esquema se obtém um direcionamento dos esforços de uma maneira bem mais eficiente, o que propicia conseqüências positivas, em um curto espaço de tempo. Outro ponto a considerar é que essas ferramentas devem apoiar na identificação das áreas mais complexas e, por conseguinte, as quais possuem maiores risco de induzir novos erros.

A realização do planejamento apresenta diversos benefícios para uma empresa. Entre elas estão:

- Papéis e responsabilidades bem definidos, auxiliando o gerenciamento desta atividade e como conseqüência as estimativas de custo, prazo e esforço;

- Objetivos dos testes detalhados, tornando mais simples a compreensão sobre os diferentes tipos de teste que serão aplicados em um sistema e suas razões, além de apoiar a identificação de possíveis riscos para a atividade;

- Documentação dos testes bem definida, fornecendo uma padronização entre documentos e facilidade a leitura desses documentos em diferentes projetos, e;

- Facilidade de comunicação entre a equipe de teste e de desenvolvimento criando um meio de comunicação formal entre essas equipes, definido assim uma linguagem comum entre elas.

Em resumo, o planejamento é a distribuição racional no tempo dos recursos disponíveis para a realização de alguma atividade. Planejar é, de modo geral, decidir antecipadamente o que deve ser feito e quando deve ser feito. É justamente nesta etapa que será descrito o escopo dos testes, identificando métodos que serão empregados, recursos necessários, cronograma de atividade, pessoal necessário, itens que serão testados, características dos itens testados e responsabilidades. Como resultado final desta etapa, teremos o documento Plano de Teste.

A primeira parte de um plano de teste envolve determinar quais os objetivos e qual estratégia de teste usar. O plano também deve abordar os recursos utilizados, os processos a serem seguidos, as técnicas e critérios a serem aplicados, o tempo estimado e outros aspectos.

O teste de software por ser uma das práticas mais custosas do processo de desenvolvimento faz se necessário ter um bom planejamento com o propósito de se evitar perdas de recursos e atrasos no cronograma, ou até mesmo, não contribuir para a avaliação do software, não ser finalizado, dentre outras possibilidades. Por isso, a realização de testes em software demanda recursos adequados, e o emprego efetivo desses recursos requer um bom planejamento e controle (Mcgregor e Sykes, 2001). O planejamento assegura que o critério de teste seja especificado e o conjunto de casos de testes sejam determinados antes do fim da fase de implementação do software, e assim evita os testes que avaliam somente características do software das quais os desenvolvedores já têm conhecimento sobre o seu funcionamento.

A realização de testes de software sem um planejamento pode ser comparada ao desenvolvimento de um projeto qualquer sem a elaboração de um plano para ele, e isto ocorre, geralmente, pela mesma razão: grande pressão para iniciar a fase de execução o mais rápido possível.

2.2 VISUALIZAÇÃO DE INFORMAÇÃO

As representações gráficas são utilizadas nas mais variadas formas e no mais diversos domínios, quer na criação pela arte, no simples registro fotográfico, no desenho, na pintura, na gravura, em qualquer forma visual de expressão, com o propósito de expressar uma idéia já existente. Entretanto, há uma segunda abordagem que tira proveito da percepção visual humana para a solução de problemas, essa diferentemente da primeira, consiste em empregar as representações gráficas para descobrir a idéia.

Seguindo a primeira abordagem, as representações visuais estão veiculadas através de anúncios publicitários colocados em evidência em edifícios ou impressas em jornais e revistas, em cartazes afixados em muros ou veículos de transporte e, por fim, em toda exibição em telas de cinema e televisão. A segunda abordagem está envolvida com as inovações advindas da evolução tecnológica dos meios de comunicação, dos diversos mecanismos de obtenção de dados, imagens e sinais e da computação gerando um volumoso conjunto de informações procedentes dos mais variados meios e formatos.

Das duas abordagens apresentadas, a segunda merece destaque em especial, denominado como a área de visualização de informação, a qual, atualmente é explorada por muitos pesquisadores que objetivam a construção de representações visuais de dados abstratos, reunindo milhares de dados em uma única figura (Card *et al*., 1999). Com objetivo final de facilitar o seu entendimento, apoiar na descoberta de novas informações (idéias) e revelar padrões ocultos inseridos nas mesmas.

Essa área sofreu grandes inovações advindas da evolução computacional, retratando o seguinte panorama: sistemas computacionais cada vez mais inseridos em diferentes áreas das atividades humanas, coletando e armazenando grandes conjuntos de dados (das mais diversas formas, dimensões e complexidade), o que dificulta em potencial a exploração de toda essa informação. Porém, conta-se hoje com tecnologias computacionais que estão sendo desenvolvidas e aplicadas, que visam apoiar o armazenamento, a busca e a recuperação desse volumoso conjunto de dados.

A visualização de informação refere-se a dados abstratos como relacionamentos ou informações inferidas a partir de dados mensurados, ou seja, os dados não possuem informações inerentes de geometria, ou seja, não existe uma representação visual espacial (Spence, 2001). Esse é o ponto chave que as comunidades científicas investigam, o de encontrarem metáforas visuais para representar os dados e quais os mecanismos de análise estas representações sustentam.

Na visualização de informação é absolutamente necessário ao projetista gerar meios para converter dados em uma representação gráfica. Essa representação deve exprimir as mais importantes propriedades dos dados e também expressar como os itens estão relacionados (Luzzardi, 2003). Sendo assim, todas as técnicas buscam expor, em representações gráficas, as informações; explorando o máximo a aptidão da percepção humana, induzindo a interpretações mais sólidas e compreensíveis.

Como na visualização de informação os tipos de dados tratados possuem características diversas, os sistemas comerciais desenvolvidos para a área de visualização de informação não são de um contexto muito amplo. São desenvolvidos para realizarem tarefas específicas, ou seja, estão centrados em um determinado tipo de dado. Sendo assim, há poucos sistemas de uso genérico, pois há uma grande variedade de dados abstratos, cada qual com suas particularidades, o que resulta no desenvolvimento de sistemas dedicados a assuntos específicos.

As técnicas que estão sendo propostas para auxiliarem na visualização e análise dos dados analisam fundamentalmente características da percepção humana com o objetivo final de ampliar a cognição, através de recursos computacionais (Card *et al.*, 1999). É nesse sentido que a área de visualização de informação se ostenta como um campo de estudo de grande serventia, reunindo técnicas que facultam na compreensão das informações através de representações visuais.

Outra característica sobre a visualização é que ela envolve o sentido humano, o qual possui uma ampla capacidade de captação da informação por unidade de tempo. A percepção visual é um sentido rápido e paralelo capaz de dar

ênfase a um objeto de interesse, sem perder de vista o que está acontecendo ao seu redor. É nesse sentido que com a visualização de informação é possível condensar um grande volume de dados em uma simples visualização.

Devido a estes fatos, há a necessidade de ferramentas gráficas e de auxílio computacional nas áreas de pesquisa, desenvolvimento e produção como apoio ao processo de interpretação das informações geradas, ou seja, a tradução de informações para um formato gráfico é de uma importância considerável para que o ser humano processe os dados de forma mais rápida. Essa interpretação da informação em um formato mais eficaz para a mente humana tem como conseqüência o aumento na sua produtividade (Oliveira, 1997).

2.2.1 CONCEITOS BÁSICOS

A visualização torna-se muito mais do que representações gráficas de informações. Ela exerce funções como uma ferramenta cognitiva, fundamentalmente limitada à mente humana, o que a torna um artefato para a edificação do conhecimento, empregando as capacidades perceptivas e cognitivas do ser humano. Inserindo no contexto computacional, a visualização de informação é bem explorada na definição dada por Card *et al.*, que a define como sendo “o emprego de representações visuais de informações abstratas, sustentadas pelo computador e interativas por ampliar a cognição” (Card *et al.*, 1999 p. 7).

Pode-se deduzir que o ato de processar a informação em um formato visual está interligado com a modificação de qualquer coisa abstrata em representações visuais, ou seja, em imagens as quais os seres humanos têm a possibilidade de visualizá-las; com o propósito de apoiar a compreensão sobre um dado assunto, sem a qual, despenderia um maior empenho para ser compreendido. Porém, demonstrar algo abstrato em apenas uma figura que sintetiza e reduz uma grande quantidade de informação e ainda não se desprende da idéia central é uma tarefa difícil, tornando-se assim, o principal desafio dos pesquisadores da área de visualização de informação.

2.2.2 MODELO DE REFERÊNCIA DE VISUALIZAÇÃO

Todo processo de tratar, recuperar e disseminar a informação é comparado ao processo que a visualização de informação propõe em transformar em formas gráficas representativas um conjunto de dados brutos. Esse processo é apresentado na Figura 8, e é reconhecido como um modelo de referência do processo de mapeamento visual apresentado por Card *et al.* (1999). Esse modelo apresenta a divisão do processo de geração de representação gráfica para um conjunto de dados em três estágios e a função do usuário nessas transformações, tendo as suas práticas descritas nas subseções a seguir.

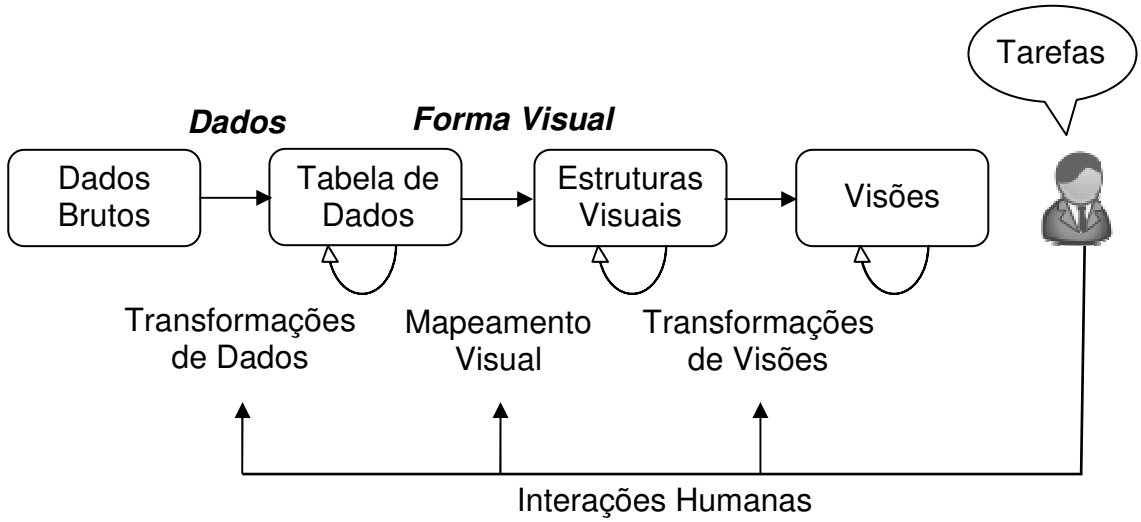


Figura 8: Modelo para Visualização de Dados (Fonte: Card *et al.*, 1999).

DADOS BRUTOS

Nesse modelo uma pessoa tem em mente uma tarefa de examinar uma situação qualquer e obter informações importantes sobre ela. Para que isso seja concretizado é preciso primeiramente obter dados sobre a situação que deseja. Essa obtenção dos dados pode ser feita de forma manual pela própria pessoa ou através de sistemas computacionais, interligados ou não por algum dispositivo de captação de dados, como por exemplo, sensores ou câmera. Com esse conjunto de dados brutos coletados as informações nele contidas podem ser de naturezas

diversas, ou seja, os dados podem ser heterogêneos, como por exemplo, datas, medidas, códigos, símbolos, imagens, vídeos, entre outros.

TABELA DE DADOS

Dada essa heterogeneidade dos dados, para recuperar informação inicialmente é preciso que esses dados sejam organizados em estruturas que possuam entidades similares. Esse processo é o primeiro estágio denominado de Transformação de Dados (habitualmente conhecida como pré-processamento), que nada mais é do que organizar e processar um conjunto de dados brutos em representações lógicas mais estruturadas, normalmente na forma de tabelas (Card *et al.*, 1999). Quando esses dados estão armazenados em sistemas gerenciadores de banco de dados (SGDB), essa transformação torna-se desnecessária, pois o banco já disponibiliza tais organizações (Silva, 2007).

Na visualização de informação, a decisão de qual técnica será utilizada para uma dada aplicação é um fator extremamente importante, ainda mais quando tratamos de dados abstratos, que são individualizados pela carência de uma geometria implícita aos dados.

A caracterização dos dados seria a consideração inicial tornando-se a etapa inicial na determinação de uma técnica de visualização, pois para Luzzardi (2003) informações descrevem fenômenos ou processos ou entidades os quais são objetos de análise ou estudo. Assim, o autor expõe que informações são equivalentes a atributos os quais são caracterizados conforme diferentes critérios.

Para se obter sucesso no desenvolvimento de sistemas de visualização os projetistas devem levar em consideração a forma de mapear as informações para uma representação gráfica que facilite ao máximo a sua interpretação. Visando o auxílio de dar indicações que permitem situar aplicações em técnicas, alguns autores propõem certas classificações para análise e visualização dos dados. Diferentes autores (Card *et al.*, 1999; Spence, 2001; Ware, 2004) a classificam de forma semelhante à semântica de atributos ou variáveis, embora com uma nomenclatura um pouco diferenciada, porém com o mesmo sentido:

- Nominais ou categóricos: um dado conjunto de elementos sem uma ordem característica;

- Ordinais: um dado conjunto de elementos que ostentam um relacionamento de ordem entre si;

- Quantitativos: um dado conjunto de elementos que ostentam um escopo numérico, na qual há a possibilidade de por em prática operações aritméticas sobre os mesmos.

Para a simplificação no presente trabalho, os valores assumidos pelas variáveis podem ser classificados em dois formatos: nominal e quantitativo. Sendo que o primeiro assume valores evidentemente distintos, discretos e enumeráveis. O segundo apresenta valores numéricos, contínuos, sobre os quais podem ser efetuadas operações aritméticas.

ESTRUTURAS VISUAIS

Após essa fase, na qual os dados já estão organizados em uma tabela de dados, vem o processo de Mapeamento Visual dos dados. Nesse estágio o que se procura fazer é determinar como cada atributo será representado em uma forma visual, ou seja, para cada variável da tabela de dados deve-se associar a uma propriedade gráfica ou espacial. Em outras palavras, tendo os dados em tabelas é preciso selecionar uma estrutura visual que melhor o represente. Para Card et al. (1999) uma estrutural visual é definida como um conjunto de elementos visuais que representam um conjunto de dados. Esses elementos visuais são chamados de marcas. São objetos presentes no espaço gráfico, como pontos, linhas, áreas e volumes. Spence (2001) adiciona ícones multidimensionais a esse conjunto de objetos Cada marca é caracterizada por propriedades gráficas que são: cores formas, texturas, orientação entre outros, ou seja, são elementos gráficos utilizados para representar os dados. A Figura 9 exibe alguns exemplos de marcas que podem ser utilizadas para a representação visual e a Figura 10 apresentas algumas propriedades gráficas da marca que são atributos visuais que as caracterizam (Nascimento e Ferreira, 2005)

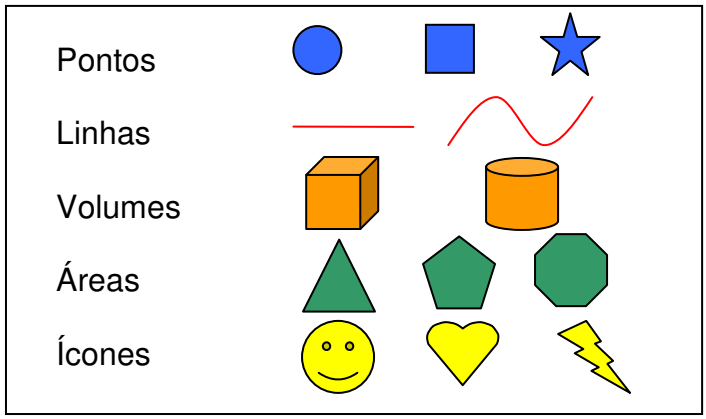


Figura 4: Tipos de marcas

(adaptado de Nascimento e Ferreira, 2005)

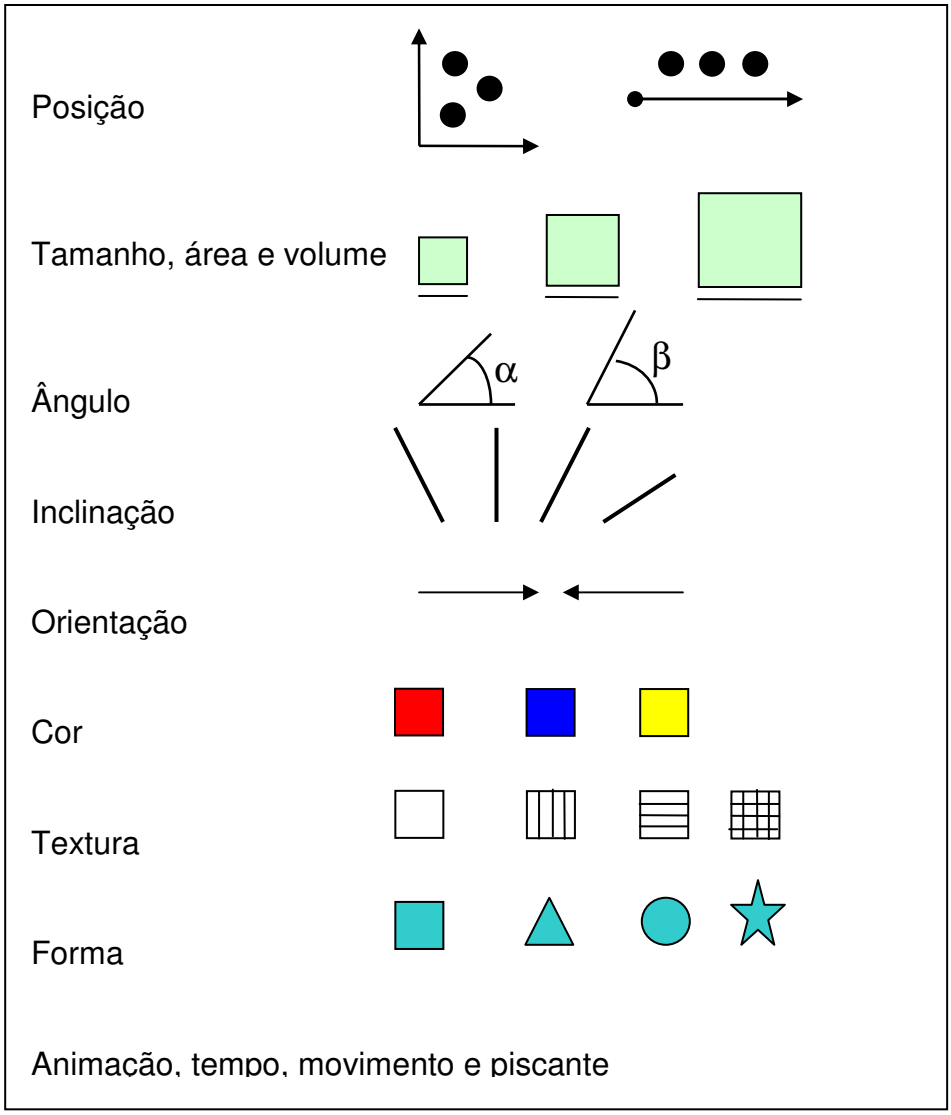


Figura 5: Propriedades gráficas das marcas

(adaptado de Nascimento e Ferreira, 2005)

As marcas também apresentam propriedades espaciais que caracterizam o espaço para a visualização, denominado por Card *et al.* (1999) de substrato espacial, normalmente representada por eixos de um gráfico, como os eixos X e Y do plano cartesiano. Segundo Nascimento e Ferreira (2005), existem quatro tipos elementares de eixo: eixo não estruturado, ou seja, não há a utilização de eixo; eixo nominal, nesse há uma região dividida em sub-regiões; o eixo ordenado, nesse também há uma região dividida em sub-regiões, no qual a ordem das mesmas tem relevância; e por fim, o eixo quantitativo, nesse a região possui uma métrica.

Nessa etapa é preciso designar para cada atributo como ele será representado, quais as propriedades gráficas podem ser empregadas para a sua representação. Esse é uma das etapas mais árduas para a definição de um mapeamento visual, pois a quantidade de propriedades gráficas é bastante ampla, tais como cores (o que agrega tonalidade, saturação e brilho), formas (diversificada formas geométricas e ícones), textura, tamanho, orientação, entre outros. É necessário selecionar, no meio dessas possibilidades de combinar propriedades, um mapeamento que expresse de maneira efetiva a informação.

EXPRESSIVIDADE E EFETIVIDADE

Segundo Mackinlay (1986), há duas características que merecem destaque ao se fazer um mapeamento visual: a expressividade e a efetividade. Um mapeamento é descrito como expressivo se todos os dados da tabela de dados e apenas os dados são expostos nas estruturas visuais. Tem-se a efetividade do mapeamento quando esse é interpretado de maneira rápida. Para se obter a melhor representação gráfica (mapeamento) deve-se considerar a estrutura gráfica que melhor o representa, a qual é dependente do tipo de dado a ser representado, e do usuário ao qual se destina.

O mapeamento deve ser efetivo o que está completamente relacionado com a percepção dos seres humanos. A visualização de informação é totalmente dependente das habilidades perceptuais que os humanos possuem. É através da percepção visual que é possível aumentar a informação do mundo visual por unidade de tempo, os seres humanos são dotados para analisarem, observarem,

reconhecerem e recordarem de imagens muito rapidamente. É por isso que os fenômenos da percepção devem merecer destaque no mapeamento de sistemas para a exploração e visualização de informações (Romani, 2000).

VISÕES

A última etapa é a de Transformações Visuais a qual consiste em executar operações sobre as representações visuais com o intuito de obter informações adicionais sobre os itens do conjunto de dados, criando novas visões da estrutura visual de acordo com as exigências do usuário. Estas transformações são classificadas em três categorias: exploração, controle e distorção.

A técnica de exploração é muito utilizada para apresentar informações adicionais sobre uma determinada marca (reta, ponto, ícone, etc.), aqui pode se fazer o uso de uma janela menor sobreposta (*popup window*) com os dados adicionais sobre os dados. Controle e distorção podem ser encontradas em Card *et al.* (1999).

2.2.3 GRÁFICO DE DISPERSÃO DE DADOS (*SCATTERPLOT*)

Gráficos de dispersão de dados é uma técnica de visualização bem conhecida e popularizada. É utilizada para mapear dados bidimensionais (2D), tridimensionais (3D) e multidimensionais (nD) utilizando coordenadas (Feketc e Plaisant, 2002).

Está técnica de visualização projeta no plano espacial os relacionamentos dos atributos da tabela de dados representados pela coordenadas X e Y no caso de mapeamento de dados 2D.

Segundo Kosara *et al.* (2007) é através dos pontos projetados num plano que os itens de um conjunto de dados são representados pelos eixos. E o objetivo desta visualização é denotar os dados de uma forma experimental com o intuito de indicar os pontos de concordância. Sendo assim, esta técnica torna-se muito eficaz para denotar se há uma relação, um padrão ou uma tendência entre os dados.

Outra questão importante sobre esta técnica é que ela admite a adição de propriedades visuais, tais como: cor, forma, tamanho, orientação entre outros. Possibilitando o aumento do número de atributos que podem ser representados. Um exemplo desta visualização pode ser visto na Figura 11.

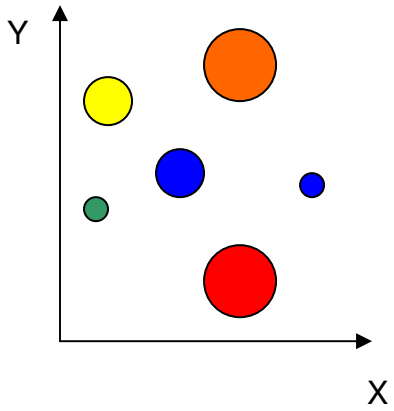


Figura 11: Eixos X e Y da visualização de dispersão
de dados bidimensionais – os dados são representados pelas esferas.

Além dos gráficos convencionais de pontos no espaço, há algumas técnicas para representação de dados diversos a qual veremos na subseção seguinte o mapeamento por cores.

2.2.4 MAPEAMENTO POR CORES

Por ser uma técnica bem comum, o mapeamento por cores resume-se em associar dados escalares a cores, exibindo-as como indicação dos valores. Para isto, é preciso ter uma tabela de cores (*color lookup table*) na qual o mapeamento será feito através da indexação, ou seja, os valores escalares são usados como índices para esta tabela.

Existe também outra forma de tabelas de dados, conhecida como função de transferência. Nesse caso, a função de transferência é uma expressão qualquer que mapeia um valor escalar especificando uma cor, normalmente em RBG e um valor de opacidade (Adaime, 2005).

2.2.5 INTERAÇÃO HOMEM-COMPUTADOR

Uma visualização de modo estático não possibilita uma análise concreta e eficiente de um grande conjunto de dados. Normalmente esta limitação é contornada utilizando-se mecanismos que possibilitem ao usuário explorar diversas ações em diferentes níveis da visualização. Estas ações caracterizam-se pela modificação da representação visual, possibilitando que novos aspectos dos dados sejam observados e interpretados.

As técnicas de visualização além, de ter em si uma representação visual elas devem dispor de um grupo de técnicas de interação para que se torne possível usuários explorarem o conjunto de informações, obtendo maiores detalhes para o entendimento dos dados nas representações visuais. Desta forma, estratégias de interação para fornecerem uma visão geral ou detalhada do conjunto de dados.

Esse conjunto de técnicas é importante, pois ele tem como objetivo suprir as eventuais desvantagens e pontos fracos das técnicas de visualização, essencialmente as técnicas que possuem desorganização visual e sobreposição dos objetos, propiciando para aos usuários meios para manipulação de grandes conjuntos de dados (Oliveira e Levkowitz, 2003).

2.3 MÉTRICAS DE SOFTWARE

Como todo processo de desenvolvimento deve ser controlado, o desenvolvimento de software também merece este cuidado. Isso é possível através da utilização de medições de software. A medição deve estar inserida no processo de desenvolvimento do software por permitir a previsão e monitoramento dos custos, controle de qualidade e por fim, para se obter uma melhora na própria compreensão e validação do processo de desenvolvimento.

As métricas são utilizadas para medir cada estágio do desenvolvimento do software bem como vários aspectos do produto. De acordo com Pressman (2005), esse processo é medido com o intuito de melhorá-lo e o produto mensurado com o intuito de aumentar a sua qualidade.

De acordo ainda com Pressman (2005) há várias razões que levam a um software a ser medido, entre elas:

- Apontar a qualidade;
- Determinar a produtividade das pessoas envolvidas com a produção do produto;
- Estimar as benfeitorias (relacionados com a qualidade e produtividade) advindas de ferramentas de software e de novos métodos;
- Dar argumentos para fundamentar a solicitação de novas ferramentas e de treinamentos complementares.

2.3.1 LINHAS DE CÓDIGO - LOC

A medida de tamanho mais comum de software é a contagem de linhas de código (LOC), que é uma medida do tamanho físico do programa. Embora seja aparentemente simples e óbvia, para capturar o volume de código, quanto à forma de contagem de linhas que deverão ser tratadas, é uma questão muito discutida na literatura. Para alguns autores a contagem de linhas de código não devem considerar as linhas de comentários e nem as linhas em branco, pois essas

só têm o objetivo de documentar e organizar o código, não influenciando a funcionalidade do software.

Segundo Peters (2001) a métrica de LOC é uma medida direta e muito simples de calcular e ela está muito passível a organização do código, podendo obter uma grande variação nas LOC produzidas de acordo com o arranjo do código.

Por possuir características simples isto a torna uma das suas principais vantagens, pois é uma métrica fácil de coletar e verificar. Porém, essa sua simplicidade torna a LOC tecnologicamente dependente de linguagem de programação, ou seja, quando se utiliza linguagens de programação diferentes torna-se difícil as comparações de LOC. Desta forma, esta forte vinculação da métrica com a linguagem de programação torna impossível o uso de dados históricos em softwares que não foram codificados na mesma linguagem.

2.3.2 MEDIDA DE CIÊNCIA DO SOFTWARE - HALSTEAD

Halstead (1977) propôs um conjunto de medições de software conhecido como Ciência do Software, este conjunto de medições tem o propósito de determinar valores quantitativos da complexidade direta de operadores e operandos em um módulo ou blocos de construções de um programa de computador (código – composto de operadores e operandos) (Peters, 2001).

As medidas primitivas da Ciência de Software de Halstead são (Kan, 2003):

n_1 – o número de operadores distintos que aparecem em um programa;

n_2 - o número de operandos distintos que aparecem em um programa;

N_1 – o número total de ocorrência de operadores

N_2 – o número total de ocorrência de operandos

Segundo Kan (2003), Halstead utiliza medidas primitivas para desenvolver expressões para o comprimento global do programa, o volume mínimo

potencial para um algoritmo, o volume real (número de bits exigido para especificar um programa), o nível de linguagem (uma constante para determinada linguagem) e outras características tais como esforço de desenvolvimento, o tempo de desenvolvimento e até mesmo o número projetado de falhas no software. As principais equações de Halstead são exibidas na Tabela 4.

Tabela 4: Medidas para cálculo de complexidade de Halstead (adaptado de Kan, 2003)

Vocabulário (*n*)	$n = n_1 + n_2$
Tamanho (*N*)	$N = N_1 + N_2$
Volume (*V*)	$V = N \log_2 (n)$
Nível (L)	$L = V^* / V$
Dificuldade (*D*)	$D = V / V^*$
Esforço (*E*)	$E = V / L$

2.3.3 MÉTRICA EM GERÊNCIA DE RISCO

Os gerentes de projeto sempre estão preocupados em concluir seus projetos no tempo previsto e de acordo com as restrições de esforço e custo e embora essas tarefas serem importantes deve estar atento à ocorrência de eventos indesejáveis durante o desenvolvimento ou a manutenção do software (Pfleeger, 2004).

Prever os riscos que podem prejudicar a qualidade de um software em desenvolvimento e precaver-se para evitar esses riscos é uma tarefa muito importante (Sommerville, 2003). Mas o que se entende por um risco? Segundo Pleeger (2004) “um risco é um evento indesejável que tem conseqüências negativas”. Já Sommerville (2003) define risco “como um probabilidade de alguma circunstância adversa realmente venha a ocorrer”. Desta forma, os envolvidos com o desenvolvimento e manutenção de software devem-se empenhar na análise de riscos para evitá-los ao máximo as conseqüências negativas (Pleeger, 2004).

Nesta perspectiva, as pessoas envolvidas com a atividade de teste devem avaliar o risco para o projeto e descrevê-lo para o gerente de teste para que

se possa elaborar estratégias que tratem esses riscos na atividade de teste e priorize os riscos mais altos.

Os riscos podem ocorrer por diversos fatores, entre eles:

- Adição ou alteração de funcionalidade no software;
- Adição ou mudança na tecnologia;
- Requisitos mal definidos;
- Rotatividade de pessoal;
- Experiência do programador;
- Complexidade do software;
- Dependência entre módulos e/ou sistemas.

Há na literatura uma métrica para gerência de risco e priorização proposta por Boehm (1984) que merece destaque e é denominada de exposição ao risco.

A exposição ao risco é obtida a partir da seguinte fórmula:

*ER = PROB * IMP*

Onde, *ER* é a exposição ao risco, *PROB* é a probabilidade de um risco ocorrer e *IMP* é o valor do impacto associado ao risco. Segundo Pleeger (2004); entende-se por impacto de risco a perda associada a um evento em uma circunstância de algo negativo vir a ocorrer no projeto, ou seja, a ocorrência de risco. Já a probabilidade é ter uma estimativa se o evento irá ocorrer.

2.3.4 PONTOS POR FUNÇÃO (*FUNCTION POINTS*)

A contagem de pontos de função como métrica para determinar o tamanho funcional de um sistema foi proposto em 1979, por Albrecht (1979), sua abordagem para definir o tamanho de um sistema é de acordo com a quantidade de requisitos funcionais, ponderando a complexidade de cada um deles.

Para se definir a quantidade de pontos de função de um sistema, inicialmente são calculados os “Pontos de Função Não Ajustados” (PFNA) e procede-se com a aplicação da formula de ajuste, a qual é influenciada pelos “Fatores Técnicos” do projeto, para finalmente determinar o resultado.

Os Pontos de Função Não Ajustados são calculados a partir da divisão do sistema em: “entradas do usuário”, “saídas do usuário”, “consultas do usuário”, de acordo com a sua complexidade. Em seguida, os recursos de dados são classificados em: “arquivos”, pertencentes ao sistema, e “interfaces externas”, externos ao sistema, e, da mesma forma cada arquivo e interface externa recebe um peso. Finalmente, todos os pontos de cada classificação são somados, como mostra a Figura 12.

			Fator de Ponderação			
Parâmetro de medida	**Contagem**		**Simples**	**Médio**	**Complexo**	
Número de entradas do usuário		x	3	4	6	=
Número de saídas do usuário		x	4	5	7	=
Número de consultas do usuário		x	3	4	6	=
Número de arquivos		x	7	10	15	=
Número de interfaces externas		x	5	7	10	=
Contagem – total ----------→						

Figura 12: Computando a métrica ponto por função (Pressman, 2005)

O próximo passo é determinar a influência dos “Fatores Técnicos” para se determinar o resultado da estimativa. Um questionário deve ser respondido com valores no intervalo de zero a cinco, em que zero significa sem influência no projeto e cinco, essencial. O questionário a ser respondido é mostrado na Tabela 5.

Tabela 5: Computando os pontos por função (Pressman, 2005)

Pontue cada fator numa escala de 0 a 5:	
0 | Sem influência	1 | Incidental	2 | Moderado	3 | Médio	4 | Significativo	5 | Essencial
***F_i*:**	
1.	O sistema requer *backup* e recuperação confiáveis?
2.	São exigidas comunicação de dados?
3.	Há funções de processamento distribuídas?
4.	O desempenho é crítico?
5.	O sistema funcionará num ambiente operacional existente, intensamente utilizado?
6.	O sistema requer entrada de dados *on-line*?
7.	A entrada de dados *on-line* exige que a transação de entrada seja elaborada em múltiplas telas ou operações?
8.	Os arquivos-mestres são atualizados *on-line*?
9.	A entrada, saída, arquivos ou consultas são complexos?
10.	O processo interno é complexo?
11.	O código foi projetado de forma a ser reusável?
12.	A conversão e a instalação estão incluídas no projeto?
13.	O sistema é projetado para múltiplas instalações em diferentes organizações?
14.	A aplicação é projetada de forma a facilitar mudanças e o uso pelo usuário?

Para a finalização do caçulo dos pontos, a fórmula de ajuste que se utiliza dos “Pontos de Função Não Ajustados” e das respostas do questionário de “Fatores Técnicos” do projeto. A equação da fórmula de ajuste é mostrada a seguir:

$$FP = \text{contagem total} \times [\, 0{,}65 + 0{,}01 \times \sum (F_i) \,]$$

Onde, FP é a quantidade de pontos de função, contagem total é a quantidade de Pontos de Função Não Ajustados e F_i (i = 1 a 14) é a soma da resposta dos Fatores Técnicos.

2.3.5 PONTOS POR CASO DE USO

Hoje em dia, nos projetos de desenvolvimento de software, a especificação dos requisitos através do intermédio de casos de uso é uma realidade. Desta forma, houve a necessidade de empregar uma métrica que atendesse esta demanda. Foi provavelmente essa carência que conduziram Karner (1993) a desenvolver a métrica Pontos por Caso de Uso (*Use Case Points* – UCP) em 1993 (Karner, 1993).

A métrica UCP é uma derivação da métrica Ponto de Função (Albrecht, 1979) para o emprego de casos de uso (UCs), a maioria das características da métrica PF está inserida na métrica UCP, como a contagem de pontos não ajustados e a utilização de fatores técnicos.

Um dos seus benefícios é sem dúvida a rapidez que a análise pode ser iniciada na medida que os casos de usos forem criados.

O cálculo do UCP é feito atendendo a avaliação e classificação de cada caso de uso pertencente ao sistema. Todos os atores pertencentes nos casos de uso também devem ser classificados. O número do UCP é calculado levando em consideração a avaliação do peso de todos os atores, o peso de todos UCs e nos fatores de ajuste.

Os passos a seguir devem ser seguidos (Freire, 2003):

1) Classificação dos atores do sistema:todos os atores pertencentes nos UCs do sistema devem ser classificados conforme a sua complexidade. A pontuação dos pesos dos atores é realizada utilizando-se a Tabela 6.

Tabela 6: Pesos dos Atores por Complexidade (Freire, 2003)

Tipo de Ator	**Peso**	**Exemplo**
Ator simples	1	Outro sistema acessado através de uma API de programação.
Ator médio	2	Outro sistema interagindo através de um protocolo de comunicação, como TCP/IP ou FTP.
Ator complexo	3	Um usuário interagindo através de uma interface gráfica (*stand-alone* ou *web*).

2) Contagem dos pesos dos atores: o peso dos atores não ajustados (*Unajusted Actor Weight* – UAW) é calculado somando-se os produtos do número de atores de cada tipo pelo seu respectivo peso.

3) Classificando os UCs: depois do cálculo do UAW, todos os UCs serão avaliados. De forma similar aos atores, os UCs são divididos segundo a sua complexidade. A classificação dos UCs pode ser realizada através da identificação do número de entidades pertencentes do caso, conforme pode ser observado na Tabela 7. Compreende-se por entidade cada conceito de negócio (objeto do mundo real) encontrado no UC. Um conceito pode ser uma idéia, um objeto ou uma coisa. Por exemplo, saque, saldo, banco.

Tabela 7: Pesos dos UCs por número de Entidades (Freire, 2003)

Tipo de UC	**Número de entidades**	**Peso**
Simples	5 ou menos	1
Médio	5 a 10	2
Complexo	Mais de 10	3

4) Contagem do peso dos UCs: o peso dos casos de uso não ajustados (*unajusted use case weight* – UUCW) é realizado da mesma forma que o cálculo dos atores não ajustados (UAW), soma-se o produto do número de UCs de cada tipo pelo respectivo peso. O valor do UUCP (UCP não ajustado) é dado pela soma do peso dos atores (UAW) com o peso dos casos de uso (UUCW)

5) Cálculo do USP: o peso dos caso de uso não ajustados são então ajustados de acordo com os Fatores de Complexidade Técnica (TCF), exibidos na Tabelas 8 e Fatores Ambientais (EF), conforme mostra a Tabela 9. Fatores ambientais e seus pesos foram importados da teoria de Pontos de Função, e os fatores de Complexidade Técnica foram propostos por Karner (1993). Assim, UCP = UUCP * TCF * EF.

Tabela 8: Fatores de Complexidade Técnica (Karner, 1993)

Descrição	**Peso**
Sistemas Distribuídos	2,0
Desempenho da aplicação	1,0
Eficiência do usuário final (on-line)	1,0
Processamento interno complexo	1,0
Reusabilidade do código em outras aplicações	1,0
Facilidade de instalação	1,0
Usabilidade (facilidade operacional)	0,5
Portabilidade	0,5
Facilidade de manutenção	1,0
Concorrência	1,0
Características especiais de segurança	1,0
Acesso direto para terceiros	1,0
Facilidades especiais de treinamento	1,0

Tabela 9: Fatores Ambientais (Karner, 1993)

Descrição	**Peso**
Familiaridade com o Processo de Desenvolvimento de Software	1,5
Experiência na Aplicação	0,5
Experiência com OO, na Linguagem e na Técnica de Desenvolvimento	1,0
Capacidade do Líder de Projeto	0,5
Motivação	1,0
Requisitos estáveis	2,0
Trabalhadores com dedicação parcial	-1,0
Dificuldade da Linguagem de Programação	-1,0

Uma restrição do caso da métrica UCP é que só se pode adquirir um valor para o sistema como um todo. Como os UCs só são avaliados através de um peso, não se pode realizar a obtenção de medidas para cada UC isoladamente (Braz, 2004).

A aquisição do tamanho dos UCs separadamente pode possibilitar a avaliação de prioridade e de viabilidade por parte da equipe de gerentes de projetos (Braz, 2004). Não se tendo uma medida do tamanho de cada UC em separado, não é possível se obter as estimativas para cada UC, nem tampouco se tomar decisões fundamentadas na métrica.

2.3.6 PONTOS POR TAMANHO DE CASO DE USO

A métrica Pontos por Tamanho de Caso de Uso (USP – *Use Case Size Points*) é proposta para reduzir as limitações da métrica Pontos por Caso de Uso – UCP (Braz, 2004).

Como métrica pode ser aplicada para cada caso de uso isoladamente. Depois de se obter o USP de um UC, torna-se possível a aquisição de estimativas do tempo e custo para o desenvolvimento desse UC em particular e não apenas do sistema por completo.

O tamanho da funcionalidade é determinado do USP por meio da estrutura e das seções pertencentes de um UC, computando-se o número e pesos dos cenários, pré e pós-condições, atores entre outros.

O número de entidades pertencentes numa seção é a mais importante informação a ser considerada pelo analista na classificação de uma seção do UC.

O cálculo do USP pode ser feito para o sistema por completo ou por apenas um UC (Braz, 2004). Para isso, é preciso analisar quais UCs deseja avaliar.

Para calcular o USP de cada UC, é preciso conferir o número e o tamanho de cada seção contida no mesmo. Desta forma, é preciso avaliar cada seção separadamente, para finalmente chegar a um número final para o UC. Para isso é preciso seguir alguns passos (Braz, 2004):

1) Classificação dos atores: cada ator possui uma complexidade (CA) determinada conforme a quantidade de informações que transmite ou que recebe do UC, sendo classificado conforme a Tabela 10.

Tabela 10: Complexidade dos Atores (Braz e Vergilio, 2006)

Complexidade	**Número de informações**	**Qtde. USP**
Simples	<=5	2
Médio	6 a 10	4
Complexo	>10	6

O total de pontos da seção Atores, o TPA, é dado pela fórmula:

$$TPA = \sum_{i=1}^{i=n} CAi$$

Onde n é o número de atores do caso de uso.

2) Classificação das pré-condições:cada pré-condição do UC possui sua complexidade (CPrC) determinada de acordo com o número de expressões lógicas testadas, de acordo com a Tabela 11. A necessidade de testar as pré-condições pode acrescentar significantemente complexidade ao UC.

Tabela 11: Complexidade das Pré-condições (Braz e Vergilio, 2006)

Complexidade	**Número de informações**	**Qtde. USP**
Simples	1	1
Médio	2 ou 3	2
Complexo	>3	3

Depois, devem ser somadas as complexidades de todas as pré-condições do UC, obtendo o TPPrC (Total de Pontos das Pré-condições), dado pela seguinte fórmula:

$$TPPrC = \sum_{i=1}^{i=n} CPrCi$$

Onde n é o número de pré-condições do UC.

3) Classificação do cenário principal: a complexidade do cenário principal do UC deve ser classificada conforme a quantidade de entidades que o mesmo possui e o número de passos essenciais indispensáveis para a conclusão do cenário, as duas quantidades devem ser somadas. O resultado deve classificar o cenário conforme a Tabela 12. O número de pontos do cenário principal é dado pela sigla PCP.

Tabela 12: Complexidade dos Cenários (Braz e Vergilio, 2006)

Complexidade	**Número de informações**	**Qtde. USP**
Muito simples	<=5	4
Simples	6 a 10	6
Médio	11 a 15	8
Complexo	16 a 20	12
Muito complexo	>20	16

4) Classificação dos cenários alternativos: é feita de forma análoga a classificação computada para o cenário principal. Para isso, cada um dos cenários alternativos pertencentes no UC devem ser classificados conforme a sua complexidade. Cada cenário alternativo recebe um número de pontos (PCA). A complexidade dos alternativos comporta-se da mesma maneira que a complexidade do cenário principal, como mostra a Tabela 12. O total de pontos dos cenários alternativos é dado pela seguinte fórmula:

$$TPCA = \sum_{i=1}^{i=n} PCAi$$

Onde n é o número de cenários alternativos do UC.

5) Cada exceção pertencente ao UC: também deve ser classificada conforme a sua complexidade (CE). Calculada através do número de expressões lógicas testadas para detectar a ocorrência da exceção. O total de pontos acrescentados pelas exceções é dada pela seguinte fórmula:

$$TPE = \sum_{i=1}^{i=n} CEi$$

Onde n é o número de exceções do UC.

A complexidade das exceções varia de acordo com a Tabela 13.

Tabela 13: Complexidade das Exceções (Braz e Vergilio, 2006)

Complexidade	**Número de expressões testadas**	**Qtde. USP**
Simples	1	1
Médio	2 ou 3	2
Complexo	>3	3

6) Classificação das pós-condições: necessidade de se deixar o sistema no estado determinado pelas pós-condições pode acrescentar complexidade ao UC. A complexidade das pós-condições varia conforme a Tabela 14.

Tabela 14: Complexidade das Pós-condições (Braz e Vergilio, 2006)

Complexidade	**Número de informações**	**Qtde. USP**
Simples	<=3	1
Médio	4 a 6	2
Complexo	>6	3

Continuamente, devem ser somadas as complexidades de todas as pós-condições do UC, determinando o TPPoC (Total de Pontos das Pós-Condições), dado pela seguinte fórmula:

$$TPPoC = \sum_{i=1}^{i=n} CPoCi$$

Onde n é o número de pós-condições do UC.

7) Logo após de determinar o número de pontos de cada seção, deve-se obter o número de pontos por tamanho não ajustados (UUSP – *unajusted Use Case Size Points*)

O total de pontos é calculado através do somatório do número de pontos encontrado em cada uma das seções avaliadas. Esse total é dado pela fórmula:

UUSP = TPA + TPPrC + PCP + TPCA + TPE + TPPoC

Através desse cálculo já pode se obter uma idéia do tamanho geral do sistema. O número de pontos não ajustado representa o tamanho da funcionalidade, desconsiderando as facilidade e dificuldades encontradas no desenvolvimento, que podem prover dos fatores técnicos e ambientais pertencentes no software e na organização.

8) Fatores de Ajuste Técnico: os fatores de ajuste técnico representam a influência que algumas características técnicas (existentes no

software e inerentes para todos UCs do sistema). Cada fator de ajuste exibido na Tabela 8 recebe um valor entre zero e cinco de acordo com a influência no UC, onde cinco representa grande influência. O fator de ajuste é calculado pela seguinte fórmula:

$$FTA = 0{,}65 + (0{,}01 * \sum_{i=1}^{14} Ii$$

9) Fatores de Ajuste Ambiental: os fatores ambientais apresentam algumas características existentes no ambiente de desenvolvimento que pode influenciar o custo do software. Cada fator, exibido na Tabela 15, recebe um valor e o Fator de Ajuste de Ambiente (FAA) é dado pela seguinte fórmula: :

$$FAA = (0{,}01 * \sum_{i=1}^{5} Ii$$

Tabela 15: Fatores Ambientais (Braz e Vergilio, 2006)

Fator	Descrição	Influência
E1	Existência formal de processo de desenvolvimento	I1
E2	Experiência adquirida com a aplicação a ser desenvolvida	I2
E3	Experiência do time com o uso de tecnologia	I3
E4	Presença de um analista experiente	I4
E5	Requerimentos estáveis	I5

10) Concluindo o cálculo do valor final para cada UC: é dado pela seguinte:

$$USP = UUSP * (FPA – FAA)$$

2.3.7 PONTOS POR TAMANHO DE CASO DE USO *FUZZY*

Ainda que o USP seja derivado da métrica UCP e expõe algumas inovações, ele da mesma maneira que seu predecessor e que o PF, também emprega uma classificação de complexidade funcional. Desta forma, os fontos fortes conseqüentes da aplicação da teoria *Fuzzy* sobre a métrica FP podem ser

adquiridas com a extensão do modelo original do USP. Esse modelo estendido do USP é denominado Pontos por Tamanho de Caso de Uso *Fuzzy* (*Fuzzy use case size points* – USPF) (Braz, 2004).

Braz (2004) sugere calcular o USPF seguindo os seguintes passos:

1) Fuzificação dos termos lingüísticos;

2) Desfuzificação dos valores dos termos lingüísticos;

A fuzificação dos termos lingüísticos transforma as tabelas de classificação de complexidade em uma classificação contínua. Isso torna-se possível através da geração de um número *Fuzzy* trapezoidal para cada faixa de complexidade dispostas nas variadas tabelas de classificação de complexidade. Desta forma, cada tipo de seção de UC (atores, pré e pós-condições, entre outras) será classificado através de um gráfico contendo um número *Fuzzy* trapezoidal para cada faixa de complexidade pré-existente.

Para gerar o gráfico, que é um número trapezoidal, as seguintes variáveis são calculadas, para cada categoria nas tabelas de classificação ($1 \leq i \leq n$, e n é o número de termos lingüísticos na tabela de classificação que está sendo analisada). Logo:

- m_i: valor inferior do termo lingüístico Ti na tabela de classificação;
- n_i: $(m_i + m_{i+1}) / 2$
- a_i: $n_i - 1$
- b_i: $m_i + 1$

1) Fuzificação:

Se o número a ser classificado estiver entre os valores m_i e n_i do número *Fuzzy* correspondente. Nesse caso o valor em pontos será o mesmo que o valo USP convencional forneceria;

Se o número a ser classificado estiver entre os valores n_i e b_i do número *Fuzzy* correspondente. Nesse caso será necessário calcular o grau de pertinência do número para cada um dos números *Fuzzy* correspondentes.

Para finalizar o cálculo de fuzificação, somam-se os pontos obtidos do UC, da seguinte maneira:

UUSPF = TPA + TPPrC + PCP + TPCA + TPE + TPPoC

Obtendo-se o USPF não ajustado.

A Tabela 16 exibe os valores das variáveis para as tabelas de classificação do USP de complexidade dos atores, pré-condições, cenários, exceções e pós-condições. O gráfico obtido dos números *Fuzzy* relativo à tabela de classificação de atores, pré-condições, cenários, exceções e pós-condições é exibida pela Figura 13, Figura 14, Figura 15, Figura 16 e Figura 17, respectivamente.

Tabela 16: Valores para os termos de fuzificação (adaptado de Braz e Vergílio, 2004)

Tabela	Atores	Cenários	Exceções	Pré-Cond.	Pós-Cond.
m_1	1	1	1	1	1
n_1	3,5	3,5	1,5	1,5	2,5
a_1					
b_1	6	6	2	2	4
m_2	6	6	2	2	4
n_2	8,5	8,5	3	3	5,5
a_2	3,5	3,5	1,5	1,5	2,5
b_2	11	11	4	4	7
m_3	11	11	4	4	7
n_3		13,5			
a_3	8,5	8,5	3	3	5,5
b_3		16			
m_4		16			
n_4		18,5			
a_4		13,5			
b_4		21			
m_5		21			
n_5					
a_5		18,5			
b_5					

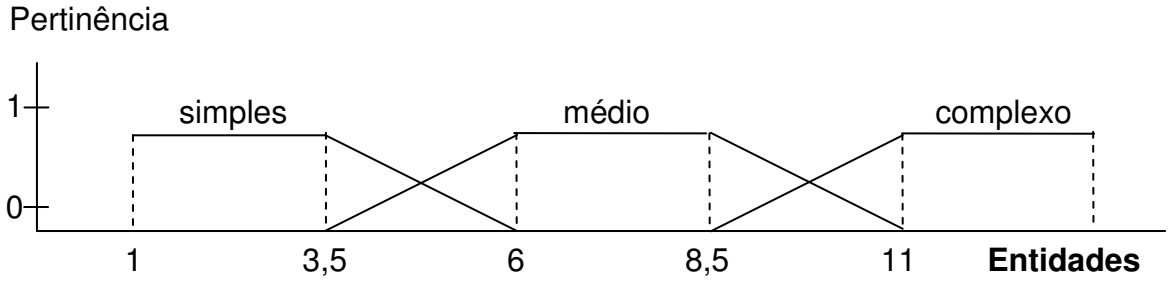


Figura 13: Números *Fuzzy* correspondentes à tabela de classificação de atores (Fonte: adaptado de Braz e Vergílio, 2006).

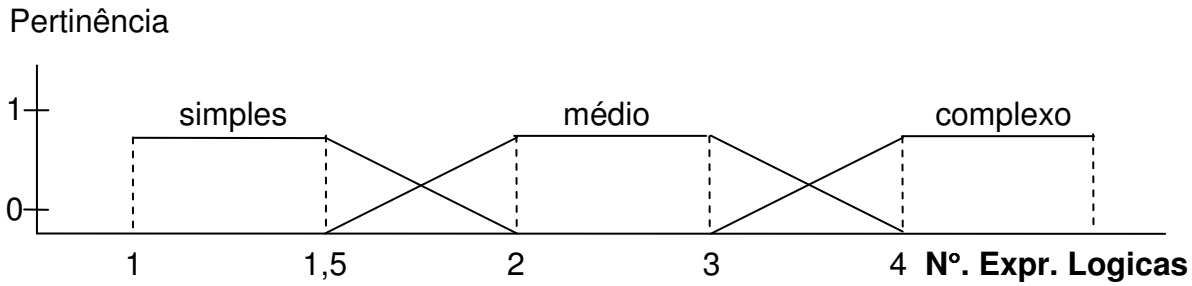


Figura 14: Números *Fuzzy* correspondentes à tabela de classificação de pré-condições (Fonte: adaptado de Braz e Vergílio, 2006).

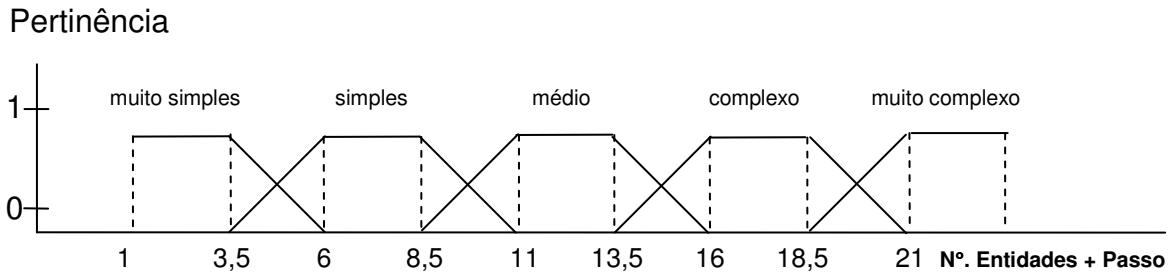


Figura 15: Números *Fuzzy* correspondentes à tabela de classificação de cenários (Fonte: adaptado de Braz e Vergílio, 2006).

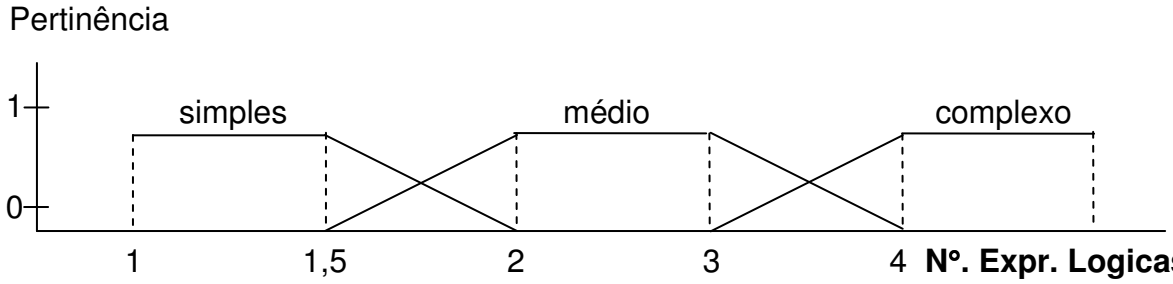


Figura 16: Números *Fuzzy* correspondentes à tabela de classificação das exceções (Fonte: adaptado de Braz e Vergílio, 2006).

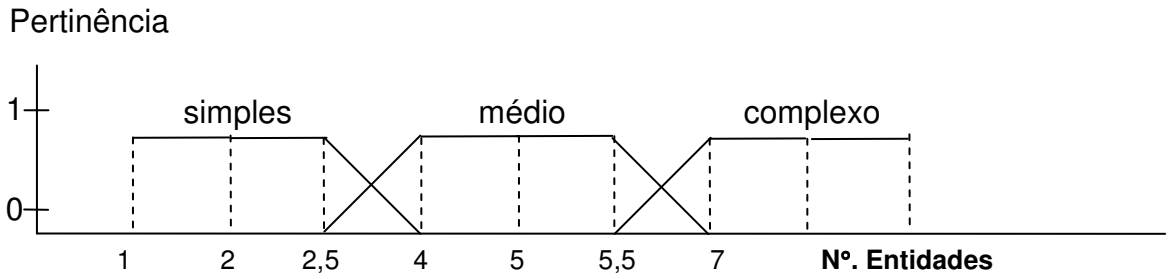


Figura 17: Números *Fuzzy* correspondentes à tabela de classificação de pós-condições (Fonte: adaptado de Braz e Vergílio, 2006).

2) Desfuzificação:

O processo de desfuzificação do termo lingüístico tem como objetivo transformar os termos lingüísticos do USPF em um valor em UUSPF, não sujeito a grandes mudanças de classe existentes no USP convencional.

A desfuzificação é realizada através da aplicação de duas regras simples. Depois disto, o UUSPF (USPF não ajustado) pode ser calculado. O processo envolve o cálculo da função de pertinência *f* (x), que representa o quanto o elemento “x” (um número real) pertence ao conjunto em questão.

Cada uma das regras é aplicada a um tipo de situação, a primeira situação acontece quando o número obtido pertence a um único número *Fuzzy* e a segunda é quando o valor está entre dois números *Fuzzy* (em uma região de transição) (Braz e Vergilio, 2006).

1) se o número a ser classificado (expressões lógicas, entidades, entre outros) estiver entre os valores m_i e n_i do número *Fuzzy* correspondente. Nesse caso o valor em pontos será o valor em pontos da faixa a que pertence este número. Isso ocorre porque ele pertence à base superior do número trapezoidal. Neste caso, o valor da função de pertinência *f* (x) é igual a um, ou seja, o mesmo valor que o USP convencional forneceria.

2) se o número x a ser classificado (expressões lógicas, entidades, entre outros) estiver entre os valores de n_i e b_i do número *Fuzzy* correspondente, ou seja, está entre a faixa de valores comum a dois número *Fuzzy*, será necessário calcular o grau de pertinência do número para cada um dos números *Fuzzy* correspondentes. Para tal, aplica-se a fórmula:

$$f(x) = (b_i - x)/(b_i - n_i)$$

$$\overline{f}(x) = 1 - f(x)$$

$$dUUSPF(x) = f(x).UUSP_i + f(x).UUSP_{i+1}$$

onde: dUUSPF(x) é o valor obtido após a desfuzificaçao do número "x"; $UUSP_i$ é o valor de UUSP para a faixa "i"; e $UUSP_{i+1}$ é o valor de UUSP para a faixa subseqüente.

2.3.8 Relação Entre Métrica De Tamanho e Complexidade E Número de Defeitos

Podemos citar dois experimentos realizados que utilizaram métricas de tamanho (LOC) e complexidade de código (Ciência de Halstead).

O primeiro experimento realizado e que inclui as métricas citadas foi elaborado (Vilela *et. al.*, 2004) no qual todas as métricas atingiram um grau de correlação acima de 95% em relação ao número de defeitos do software o que se conclui que tanto a métrica de tamanho como a de complexidade são boas para predizer o número de defeitos de um software.

Outro experimento (Lucca *et. al.*, 2004) tinha como meta analisar se existe uma relação entre a presença de defeitos e métricas de tamanho e complexidade. Para tal, foi utilizado o programa Cal.c. O qual exibe um calendário no

console e aceita algumas número de parâmetros que controlam qual mês e/ou ano, deverá ser exibido. Foi selecionado esse programa por ele possuir uma série de defeitos de software, todos eles documentados (4 módulos e 142 LOC), dada a dificuldade em encontrar softwares reais que possuam seus defeitos documentados. Desta forma, foram selecionadas algumas métricas de complexidade (Dificuldade, Volume e Esforço todas de Halstead e a Complexidade Ciclomática de McGabe) e tamanho (LOC, Comprimento e Vocabulário, essas duas últimas também de Halstead) para que se medissem cada segmento do código com o número correspondente de defeitos que ele continha. Com esses dados, foi calculado o coeficiente de correlação da métrica ao número de defeitos para cada um das métricas selecionadas. Como resultado final obteve-se que as métricas de tamanho e complexidade são boas para predizer o número de defeitos de um software e a métrica LOC foi considerada a melhor métrica para esta finalidade.

Considerando este fato, deve-se dar mar ênfase nas partes maiores ou mais complexas. Desta forma, pode-se obter uma melhor estimativa de tempo e orçamento nos trechos de código segundos estas métricas.

2.4 SCRUM

Antes de discutir o assunto propriamente dito devemos primeiramente introduzir outro assunto: o desenvolvimento ágil de software.

O desenvolvimento ágil de software caracteriza-se como sendo uma maneira “não tradicional” de desenvolver software e que segundo o manifesto *Agil Software Development* (Agile, 2008) o objetivo principal concentra-se na satisfação do cliente através da entrega antecipada e contínua do software.

Quebrando o paradigma das metodologias mais burocráticas as quais têm causado freqüentes frustrações em empresas devido as pesadas e rígidas especificações e documentações, muita vezes exigidas pelas normas e critérios de maturidade.

Dentre as várias metodologias ágeis existentes, uma das mais conhecidas é o Scrum. Uma recente pesquisa sobre o uso de metodologias ágeis (Versionone, 2008) mostra que 71% dos entrevistados usam Scrum combinado com outras metodologias e 49% usam apenas o Scrum.

O Scrum ostenta-se como uma metodologia extremamente ágil e flexível. Tem como objetivo revelar um processo interativo e incremental de desenvolvimento de produtos ou gestão de projetos. Cria um conjunto de funcionalidades mais próximas do objetivo final no terminar de cada iteração (*Sprint*), geralmente com duração de 15 ou 30 dias. Centrado no trabalho em equipe, torna melhor a comunicação e aumenta a cooperação, consentindo que cada membro faça o seu melhor e se sinta bem com o que faz, o que mais tarde se reflete num aumento de produtividade.

Scrum aplica-se a projetos tanto pequenos como grandes. O seu principal objetivo é obter uma avaliação correta do ambiente em evolução, adaptando-se constantemente as necessidades.

Abrangendo processos de engenharia de software, este método não pretende nem provê qualquer método ou técnica específica para a fase de desenvolvimento do software. O Scrum apenas institui conjunto de regras e práticas de gestão que devem ser adotadas para garantir o sucesso de um projeto.

2.4.1 PAPÉIS

O Scrum é dividido em três papéis, cada um são agentes importantes no desenvolvimento do produto, cabe a cada um a suas respectivas responsabilidades conforme demonstrado abaixo (Larman, 2003):

- *Product Owner:* Responsável pela visão do produto e como retorno deve aprovar ou não no final os resultados do produto desenvolvido;
- *Scrum Master*: Elemento da equipe responsável pela administração do projeto e chefiar as *Scrum Meetings*, são geralmente engenheiros de software ou da área de sistemas. Embora sendo gestor da equipe não possui domínio sobre os demais membros da equipe. É motivada a auto-gestão.
- *Scrum Team*: A equipe de desenvolvimento de uma atividade do projeto (*Sprint*). São auto-gerenciáveis e responsáveis por preparar suas próprias atividades e geralmente são voltados completamente ao projeto.

2.4.2 ARTEFATOS

São declarações escritas durante o projeto que servem para auxiliar a definir objetivos e ministrar estes objetivos conforme apresentados a seguir (Larman, 2003):

- *Product Backlog*: é o ponto incial do Scrum, sendo considerada a prática responsável pela coleta das atividades. Nesta prática, através de reuniões com todos os membros da equipe envolvidos no projeto, são apontados os itens com todas as necessidades do negócio, ou seja, o *Product Backlog* é uma lista de todas as funcionalidades a serem desenvolvidas durante o projeto. Deve

ser bem definida e detalhada, bem como ser ordenada por prioridade de execução;

- *Sprint Backlog*: Trabalho a ser desenvolvido num *Sprint* de modo a criar um produto a apresentar ao cliente. Deve ser desenvolvido de forma incremental, relativo ao *Product Backlog* anterior. Com o *Product Backlog* priorizado o time seleciona os itens que acham possível de executar durante o *Sprint*. As dúvidas são esclarecidas e ao final tem-se então o *Sprint backlog*, que são os itens do *Backlog* priorizado para o *Sprint*. Pars cada item, o time inicia o detalhamento de suas atividades, estimando em horas, a duração de cada uma delas. Uma vez que todas as tarefas foram estimadas, o time questiona se realmente consegue assumir o compromisso de realizar as tarefas dentro do *Sprint* e finalizar o item selecionado. Uma vez decido o item, o time passa para o próximo e esse processo continua até que todos os itens do *Sprint Backlog* sejam avaliados. Nesse momento são alocados os recursos para as tarefas; apenas se estabelce as estimativas em horas para cada uma. Após a estimativa refinada, pode-se calcular o total de horas necessário para realizar todas as tarefas. É importante deixar sempre uma folga, já que mesmo detalhando a estimativa sempre podem aparecer surpresas. Uma vez ajustados os valores e com o time se comprometendo com a execução das tarefas, existe um ambiente completo para a produção do resultado final do *Sprint*. O próximo passo é iniciar a execução do *Sprint*.

- *Burdown Chart*: é um gráfico feito diariamente que identifica e estima as tarefas pendentes.

2.4.3 CERIMÔNIAS

Cada uma das três cerimônias tem tempos determinados para sobrevir possuindo tempo e argumentos bem definidos conforme descritos a seguir (Larman, 2003):

- Reunião de Planejamento do Sprint (*Sprint Planning Meeting*): como o próprio nome diz é uma reunião para planejar um *Sprint*. É nessa reunião que a equipe decide, baseado no tamanho do grupo, quanto trabalho será necessário para cada tarefa e quais os trabalhos serão realizados no *Sprint*.

- Reunião de Scrum Diária (*Daily Scrum Meeting*): durante o ciclo de S*print* diariamente é feita uma reunião para avaliar o estado do Scrum.

- Reunião de Revisão do Sprint (*Sprint Review Meeting*): é a reunião feita no final de cada *Sprint*

2.4.4 REGRAS DO SCRUM

A fim de produzir software corretamente, pelo método ágil Scrum, é importante seguir algumas regras de execução, designadamente aos aspectos de *Product Backlog*, *Sprint* e *Scrum Meeting*.

Ao que diz respeito ao *Product Backlog*, deve ser discutido pela equipe todos os pontos que devem completar a lista de funcionalidades da aplicação, sendo de responsabilidade única do *Scrum Master* a ordenação da lista por prioridade de execução.

Quanto ao *Sprint*, deve ser realizado num período de 15 ou 30 dias, não sendo superior a isto, e ter uma equipe de 6 a 9 pessoas. Deve também ter um objetivo bem claro, baseado no *Product Backlog*. O *Product Backlog* não deve ser modificado durante a realização do Sprint, com exceção de novas funcionalidades que, segundo o *Scrum Master*, tenham influência essencial durante o projeto e que possam ser completadas dentro do período destinado ao Sprint. Se o *Sprint* estiver

tomando uma direção indesejada, é possível decompor o Sprint e começar um novo, fundamentado este num novo *Sprint Backlog*.

As *Scrum Meetings* são de suma importância no desenvolvimento do projeto. É nessas reuniões que o *Scrum Master* deve atualizar-se do decorrer do projeto e buscar identificar as mais importantes barreiras ao desenvolvimento, tendo assim a possibilidade de atuar de uma maneira eficaz na sua eliminação.

As reuniões durante um *Sprint* devem ocorrer diariamente, sempre na mesma hora e local e não devem ultrapassar 30 minutos.

Toda a discussão é limitada às respostas dos membros da equipe às perguntas postas pelo *Scrum Master*, sendo elas:

1) O que desenvolveu desde a última reunião?

2) Que dificuldades encontrou durante o seu trabalho?

3) O que planeja desenvolver até a próxima reunião?

Todos os membros da equipe devem responder as perguntas e com base nas respostas o *Scrum Master*, caso necessário, deve instantaneamente tomar decisões se alguma restrição impeça o bom desenvolvimento do trabalho.

2.4.5 O FUNCIONAMENTO DO SCRUM

A primeira tarefa a fazer para iniciar o processo do Scrum é definir a equipe. Essa equipe não deve ter mais de 6 a 9 membros. Caso houver um número maior de membros deve-se separam em várias equipes Scrum e cada equipe deverá focar numa área específica do trabalho.

Outro ponto a considerar é sobre os materiais. Esses deverão ser proporcionados a cada equipe e são eles: marcadores para os quadros, *post-its*, impressões do *Product Backlog* do produto para que todos os participantes tenham acesso a todas as informações do processo Scrum em qualquer ponto do desenvolvimento.

A próxima etapa é designar o *Scrum Master*, já que é o responsável para gerenciar as *Scrum Meetings*, tomar decisões, remover barreiras do caminho para não interromper a execução da *Sprint* em pontos críticos. O *Scrum Master* fica encarregado, como citado anteriormente, de perguntar a todos da equipe as três questões expostas.

Para não estender o tempo da reunião o *Scrum Master* deve hábil para tomar decisões imediatamente e achar a solução de todos os impedimentos instantaneamente, para não alongar o tempo da reunião.

Compete ao *Scrum Master* identificar o *Product Backlog*, primeiramente é preciso listar todo o trabalho necessário e reuni-lo em incrementos que não devem ultrapassar 30 dias. Se houverem áreas de trabalho que não possam ser definidas em 30 dias, deve ser instituído um incremento para um tempo conhecido. Depois desta fase deve-se listar todo o trabalho relevante e definir prioridades para todos os itens listados. Terminado este estágio o *Product Backlog* deve ser assinado pelos membros da equipe.

Para dar continuação a execução é preciso dar início a distribuição de trabalhos e dirigir as reuniões de Scrum diariamente, na qual as equipes se encontram e se atualizam sobre o processo. As reuniões devem ser feitas sempre no mesmo horário e local, evitando desta forma atrasos. Durante esta fase o *Scrum Master* vai cumprindo o seu papel e qualquer pendência sobre as três questões mencionadas deverá ser adiada para a próxima reunião.

Ao final de cada *Sprint*, é preciso ser feito uma revisão para registrar um balanço final da *Sprint* em questão, para tal, é preciso responder algumas questões:

1) Qual o valor acrescentado neste incremento?

2) O que foi completado do nosso *Sprint Product Backlog*?

3) Qual o *feedback* por parte do cliente do produto?

4) O que aconteceu de relevante no grupo durante o *Sprint*?

5) Como é que cada um se sentiu?

6) O que podemos concluir disso?

7) O que pode ser posto e prática para melhor o próximo *Sprint*?

A Figura 18 sintetiza o funcionamento do Scrum.

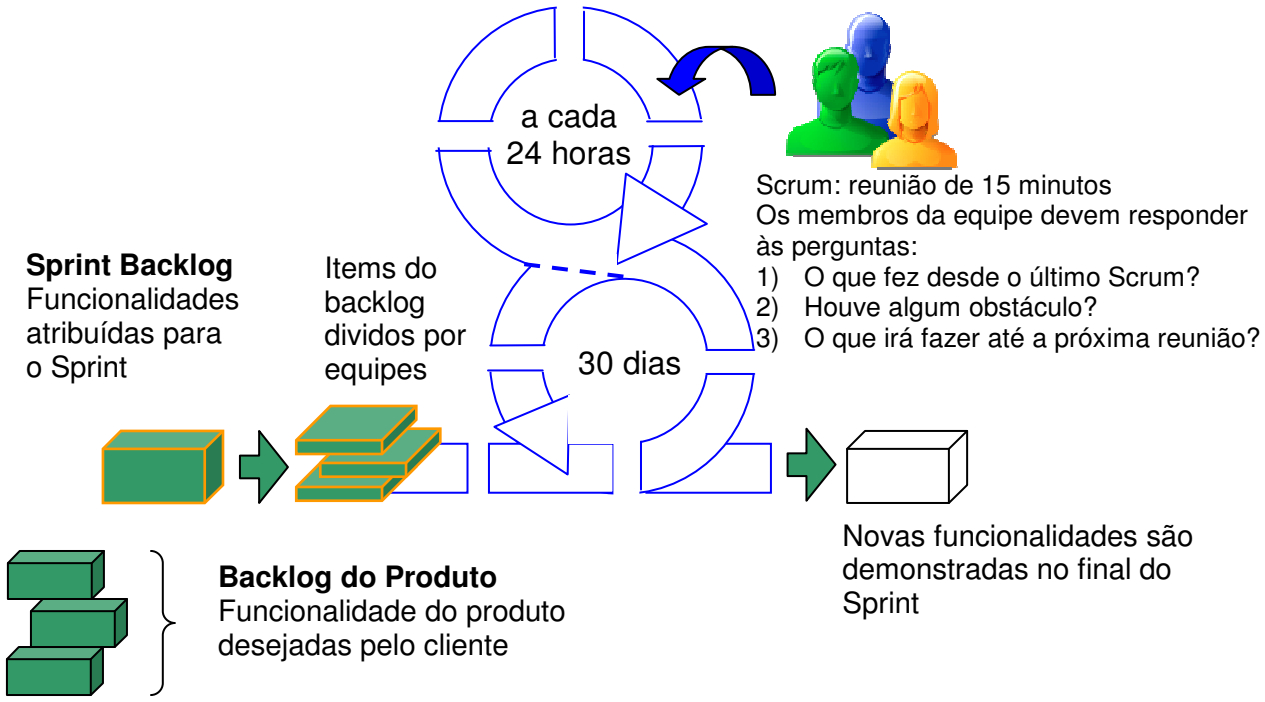


Figura 18: Descrição do processo Scrum (Fonte: adaptado de Scrumnet, 2008).

2.5 RUP (*Rational Unified Process*`)

O RUP é um processo de desenvolvimento baseado no "*Unified Process*" desenvolvido pela Rational. É um processo de Engenharia de Sofware que fornece uma disciplinada abordagem por todo ciclo de vida para especificar tarefas e responsabilidades dentro de uma organização (Kruchten, 2003).

O ciclo de vida está organizado em:

- Fases: concepção, elaboração, construção e transição.
- *Workflows* (Disciplinas): requisitos, análise, projeto, implementação e teste.

O RUP propõem uma abordagem apoiada em disciplinas para conceder tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é caucionar a produção de software de alta qualidade.

Como mostra a Figura 19, a estrutura do RUP é organizada em duas dimensões. O eixo vertical apresenta as principais disciplinas que compõem o processo, elas agrupam as atividades de maneira lógica, por natureza. O eixo horizontal representa o tempo e representa o ciclo de vida do processo à medida que se desenvolve.

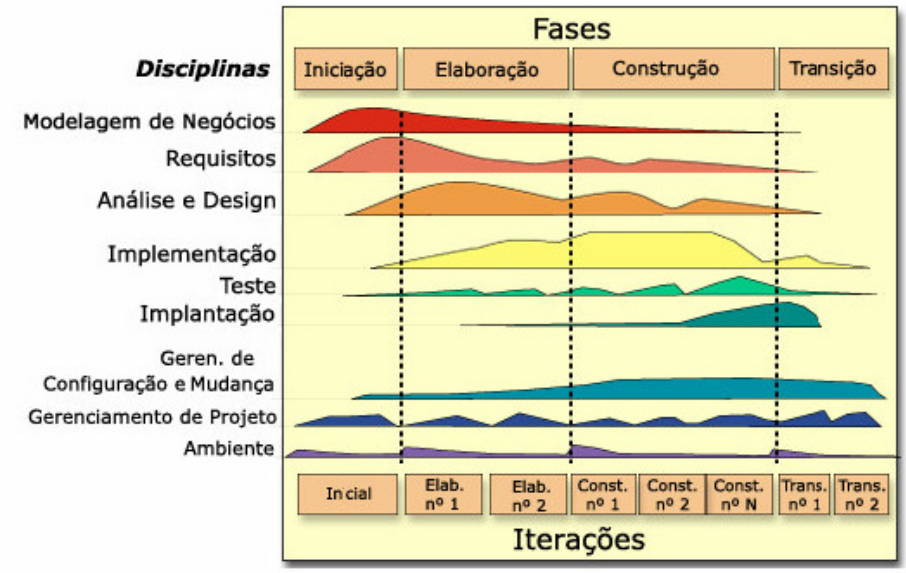


Figura 19: Gráfico do RUP (Rational, 2008).

A primeira dimensão representa o aspecto dinâmico do processo quando ele é aprovado e é expressa em termos de fases, iterações e marcos. Cada fase pode ser dividida em um ou mais iterações e termina com um maior ou menor *milestones* respectivamente. Cada ciclo tem como conseqüência uma nova *release* do sistema, e cada release é um produto pronto para a entrega para o usuário final do sistema.

A segunda dimensão representa o aspecto estático do processo, como ele descrito em termos de componentes, atividades, fluxo de trabalho e papéis do processo.

No decurso do ciclo de vida do projeto deve haver pontos de verificação visando à qualidade do produto. O desenvolvimento e a verificação geram um processo repetitivo (iterativo e incremental) no qual o produto vai sendo avaliado e, caso necessário, melhorado até estar maturado, pronto para ser disponibilizado para o usuário final. Durante estas fases, nos pontos de verificação, podem ser executados testes, para afirmar a conformidade do produto.

A estrutura dinâmica do RUP representa o desenvolvimento do processo no decorrer do tempo. O processo é caracterizado pelo desenvolvimento iterativo e por uma evolução incremental e com foco na redução dos riscos. O ciclo de vida de um projeto utilizando o RUP é dividido em uma sucessão de pequenos ciclos do modelo clássico, em cascata, chamado de modelo iterativo (Kruchten, 2003).

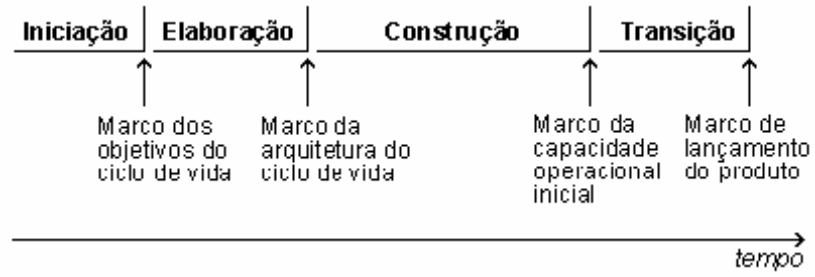


Figura 20: Fases do processo iterativo e
seus marcos (Krutchten, 2003).

Em cada iteração o processo passa por todas as etapas do modelo clássico realizando as atividades para uma pequena parte do projeto. A cada nova iteração uma nova parte do projeto é desenvolvida e integrada ao que já foi realizado em iterações anteriores, evoluindo o projeto de maneira incremental.

Como forma de acompanhar e monitorar o progresso do projeto no desenvolvimento iterativo e incremental são definidos pontos no tempo em que o andamento do projeto é avaliado.

Esses pontos, chamados de marcos, definem se o desenvolvimento irá prosseguir ou não, ou se será necessário alguma mudança. Há quatro marcos principais no RUP que dividem as iterações em quatro fases: concepção, elaboração, construção e transição. Cada fase é finalizada quando um dos marcos principais é realizado, como mostrado na Figura 20 (Kruchten, 2003). A cada iteração do processo iterativo incremental atividades como levantamento de requisitos, análise, projeto, implementação e testes são realizadas em um ciclo de vida em cascata. No entanto, em cada fase do processo a ênfase em cada atividade muda, como podemos observar no gráfico da Figura 17. Abaixo detalhamos um pouco mais as quatro fases do RUP:

1) **Concepção**: define o escopo do projeto, incorpora o estudo de viabilidade e uma parte da análise de requisitos. Concentra-se em delimitar o escopo do sistema proposto; descrever ou esboçar a arquitetura do sistema (principalmente as partes do sistema que são novas, de risco ou apresentam dificuldades); identificar os riscos críticos; construir um protótipo do sistema proposto com as idéias básicas do novo sistema. É nessa fase pode-se decidir por continuar ou abandonar o sistema.

2) **Elaboração**: incorpora a maior parte da análise de requisitos, a analise de domínio e o projeto. Concentra-se em elaborar a arquitetura básica do sistema proposto; identificar e detalhar os casos de uso; desenvolver um plano de projeto e eliminar os elementos de maior risco para o projeto.

3) **Construção**: corresponde à codificação e testes é o desenvolvimento do produto propriamente dito. Concentra-se em implementar, testar e integrar os

minis projetos para compor o sistema como um todo; complementar o desenvolvimento dos casos de uso; disponibilizar a versão beta.

4) **Transição**: em linhas gerais consiste na instalação e manutenção do sistema. É considerada uma espécie de período de análise pelos usuários do produto desenvolvido. Concentra-se em implantar o produto no ambiente; adequar as características desses ambientes; proporcionar treinamento aos usuários; oferecer assistência técnica.

Sendundo Kruchten, o processo do RUP possui várias características, entre elas (Kruchten, 2003):

1) **Dirigidos a casos de uso**: a identificação de casos de uso é a atividade que dirige todo o processo de desenvolvimento, desde a análise de requisitos até ao teste do sistema final. Os casos de uso representam uma funcionalidade do sistema, utilizando protótipos de interface gráfica ajudam na comunicação com os clientes, mas apenas mostram o que o sistema faz, e não como. Os casos de uso servem para a criação da arquitetura, teste, definição das iterações e documentação para o usuário;

2) **Centrado na arquitetura**: um sistema de arquitetura é usado como um artefato primário para conceituação, construção, gerenciamento e evolução do sistema no processo de desenvolvimento;

3) **Processo iterativo e incremental**: cada parte do projeto passa por todas as fases de desenvolvimento (concepção, elaboração, construção e transição). Cada *workflow* pode ser repetido (iteração) até que se atinjam as necessidades do projeto. Em cada nova iteração os riscos são identificados e analisados. O ciclo de vida iterativo divide o projeto em partes menores tornando o gerenciamento mais fácil. Além disso, todos os envolvidos começam a trabalhar mais cedo, pois os testes de integração são realizados desde o início, os riscos mais críticos são resolvidos mais cedo e é maior o *feedback* com o usuário. E por fim, promove a definição inicial de uma de uma arquitetura de software robusta, que posteriormente facilitará o desenvolvimento em paralelo, a sua reutilização e manutenção;

4) Orientado a Objetos: é sustentado em UML, linguagem para a equipe pode se comunicar entre si e com os clientes.

As iterações da metodologia RUP vencem os pontos fracos do modelo cascata, a seguir são apresentadas algumas vantagens do modelo RUP:

- Riscos são amenizados antecipadamente;
- As modificações são melhor gerenciáveis;
- Existe um maior grau de reuso;
- A equipe do projeto pode aprender ao longo do processo;
- O produto é de melhor qualidade.

3 TESTPLAN

Ao passo que os capítulos anteriores trataram de obter uma visão abrangente, principalmente, sobre teste de software, visualização de informação, métricas de software, RUP e Scrum, este capítulo tem como objetivo apresentar a solução proposta nesta dissertação.

O processo de teste é caracterizado por um conjunto de atividades que são executadas ao longo de todo o ciclo de desenvolvimento do software. Essas atividades estão agrupadas em etapas bem definidas que são: planejamento, projeto, implementação, execução e análise dos resultados, verificação de término e, por fim, o balanço final. Se analisarmos, especificamente, a atividade de planejamento do processo de teste de software, veremos que ela é fundamental para o sucesso de um projeto de teste. A subseção seguinte tem como objetivo caracterizar o problema abordado por esta dissertação, enfocando a fase de planejamento do teste de software.

3.1. DESCRIÇÃO DO PROBLEMA

Consideremos a tarefa de planejamento do teste de um software qualquer. Consideremos ainda um gerente de teste encarregado de executar o planejamento de teste, este deseja efetuar análises sobre um determinado conjunto de dados desta fase de planejamento. Essas análises têm como objetivo revelar uma série de informações sobre itens de um subsistema que faz parte de um sistema maior e que serão submetidos ao processo de teste, para que seja possível identificar quais itens apresentam maior risco e que merecem maior atenção.

De posse dessas informações, o gerente de teste terá a possibilidade de determinar uma ordem, ou seja, poderá priorizar os itens que merecem maior atenção de acordo com a política de teste da empresa, ou seja, a visão da empresa em relação ao teste de software, o que inclui quais os objetivos do teste, as restrições econômicas, prazos, controle de qualidade das atividades de teste, ferramentas, técnicas e treinamento. Desta forma, o gerente de teste tem condições de priorizar o processo de teste nos itens que merecem maior atenção.

Assim, analisando os dados sobre essa situação, o gerente de software tem como objetivo obter uma melhor compreensão das informações contidas nesses itens para tomar decisões que efetuem possíveis melhorias no processo de teste de software.

Para o gerente de teste efetuar as análises desejadas, com o auxílio do computador, é preciso que ele informe os dados a serem analisados, para que este processe e retorne ao gerente de teste um resultado que o auxilie nas análises. Desta forma, o problema desta dissertação pode ser abordado da seguinte forma: como transformar um conjunto de métricas e informações sobre o sistema e seus subsistemas em uma visualização gráfica para apoiar o gerente de teste no processo de planejamento do teste de software.

Para solucionar este problema, o presente trabalho propõe uma forma que pode apoiar o gerente de teste a conseguir respostas às suas análises fornecendo ao gerente de software a possibilidade de visualizar, de forma gráfica e interativa, os resultados obtidos da consulta, com o intuito de apoiá-lo na análise e compreensão destes resultados a fim de obter a resposta desejada. Para alcançar este resultado, é preciso utilizar técnicas de representação gráfica, buscando uma forma de representar os dados de uma maneira que seja a mais adequada possível e essa tarefa seja de responsabilidade do próprio TestPlan e não do gerente de teste, e for fim, ao visualizar os dados o gerente de teste tem a possibilidade de priorizar os itens que merecem maior atenção, ou seja, os que estão mais propensos a riscos. Para isso, é preciso utilizar uma lista com os itens mais prioritários e com o tempo estimado para realizar a tarefa. Para chegar a este resultado utilizaremos o *Product Backlog*, artefato da metologia Scrum (Schwaber, 2004).

3.2 SOLUÇÃO DO PROBLEMA ABORDADO PARA O PLANEJAMENTO DO TESTE DE SOFTWARE

A medição do software tem a importância de informações que permitam ao gerente de teste planejar o seu processo de teste de forma adequada controlando todo o trabalho com maior exatidão e tornando seu conceito em relação ao sistema mais seguro e confiável do ponto de vista do cliente. Os subsistemas

contêm uma série de informações sobre medidas de software relacionadas a eles e que serão abordadas neste trabalho. São elas:

- Métrica orientada ao tamanho: LOC;
- Métricas de complexidade de código: Dificuldade de Halstead;
- Métricas de complexidade funcional: PF;
- Métrica gerencial: Exposição ao risco;
- Métricas baseadas em casos de uso: UCP, USP e USPF;

Essas métricas estarão armazenadas em um banco de dados, onde serão efetuadas consultas a essa base para a obtenção de um conjunto de dados, apresentados, normalmente, em forma textual aos seus utilizadores. O fato do resultado da consulta vir de forma textual dificulta o entendimento dos dados em um nível mais abstrato dos dados em análise, exigindo um maior processamento por parte do sistema visual humano e dificultando a observação de padrões e tendências relevantes nos dados.

Desta forma, empregar técnicas de visualização de informação como meio de apoiar essa tarefa de análise é uma forma vantajosa, com boas perspectivas em diferentes áreas da atividade humana. Isto torna-se possível porque as técnicas de visualização de informação reúnem características importantes para ampliar a cognição através do uso de representações gráficas que auxiliam na velocidade do processamento, devido às características da visão humana. Porém, seu emprego no planejamento de teste de software ainda é limitado. Com as dificuldades advindas do planejamento do teste de software, a utilização de técnicas de visualização de informação, com o propósito de melhorar o entendimento e a observação dos dados, torna-se um instrumento de grande valia.

Unindo recursos interativos do computador com recursos da percepção humana, dados provenientes dos subsistemas podem ser agrupados em uma estrutura tabular para posteriormente serem representados visualmente, o que tornará possível um melhor entendimento dos dados relevantes sobre o assunto.

Também são usados, nesse ambiente, recursos de interaividade: operações de consultas e detalhamento dos dados com o objetivo de facilitar a detecção de padrões e uma melhor compreensão da situação a ser analisada.

Dada a importância deste problema para os gerente de teste, o presente trabalho procura propor uma solução com base em conceitos de visualização de informação para exibir representações visuais que o auxiliam na atividade de planejamento do teste de software. Pois, se não houver um plano de teste bem detalhado, a perda de confiança em um software pode resultar na perda de mercado, gerando prejuízos a empresas. O processo de gerenciamento do teste deve ser o mais perfeito possível, para evitar ao máximo que os testes, as correções e atualizações tenham possibilidades de estar inadequadas e incompletas. Entretanto, quando bem planejado e implantado, poderá desempenhar um importante papel na atividade de teste, Para isso, é preciso que haja informações atualizadas e consistentes, o que é fundamental na realização de melhores estimativas, como exemplo: prazo, custo e satisfação das partes envolvidas. A solução proposta será apresentada na subseção seguinte.

3.3 APRESENTAÇÃO DA SOLUÇÃO

Com o propósito de solucionar o problema abordado pelo trabalho, apresentado na subseção 3.1, esta seção apresenta a proposta para facilitar a tarefa de adquirir, preparar, apresentar e analisar os dados dos subsistemas, com o apoio dos conceitos e teorias da visualização de informação apresentadas por Card *et. al.* (1999).

Como resultado foi desenvolvido um protótipo, denominado TestPlan, que permite aos gerentes de teste consultar uma base de dados com informações sobre os subsistemas escolhendo alguns atributos das tabelas desta base. Através da escolha desses atributos, o protótipo apresenta o resultado em uma estrutura visual aos gerentes de teste. Nessa estrutura visual, o protótipo fornece meios interativos que possibilitam detalhamento dos dados apresentados, transformando as representações visuais através de controle seletivo. Contribuindo, assim, para a obtenção de melhores estimativas de escopo, prazo e melhor mapeamento das informações contidas na atividade de planejamento de teste.

Para auxiliar no desenvolvimento do TestPlan foi desenvolvido o diagrama de caso de uso, com base nos requisitos funcionais do TestPlan presentes no Anexo A, a Figura 21 representa esse diagrama de caso de uso. O diagrama de classes também foi desenvolvido para auxiliar na codificação do TestPlan e está representado na Figura 22.

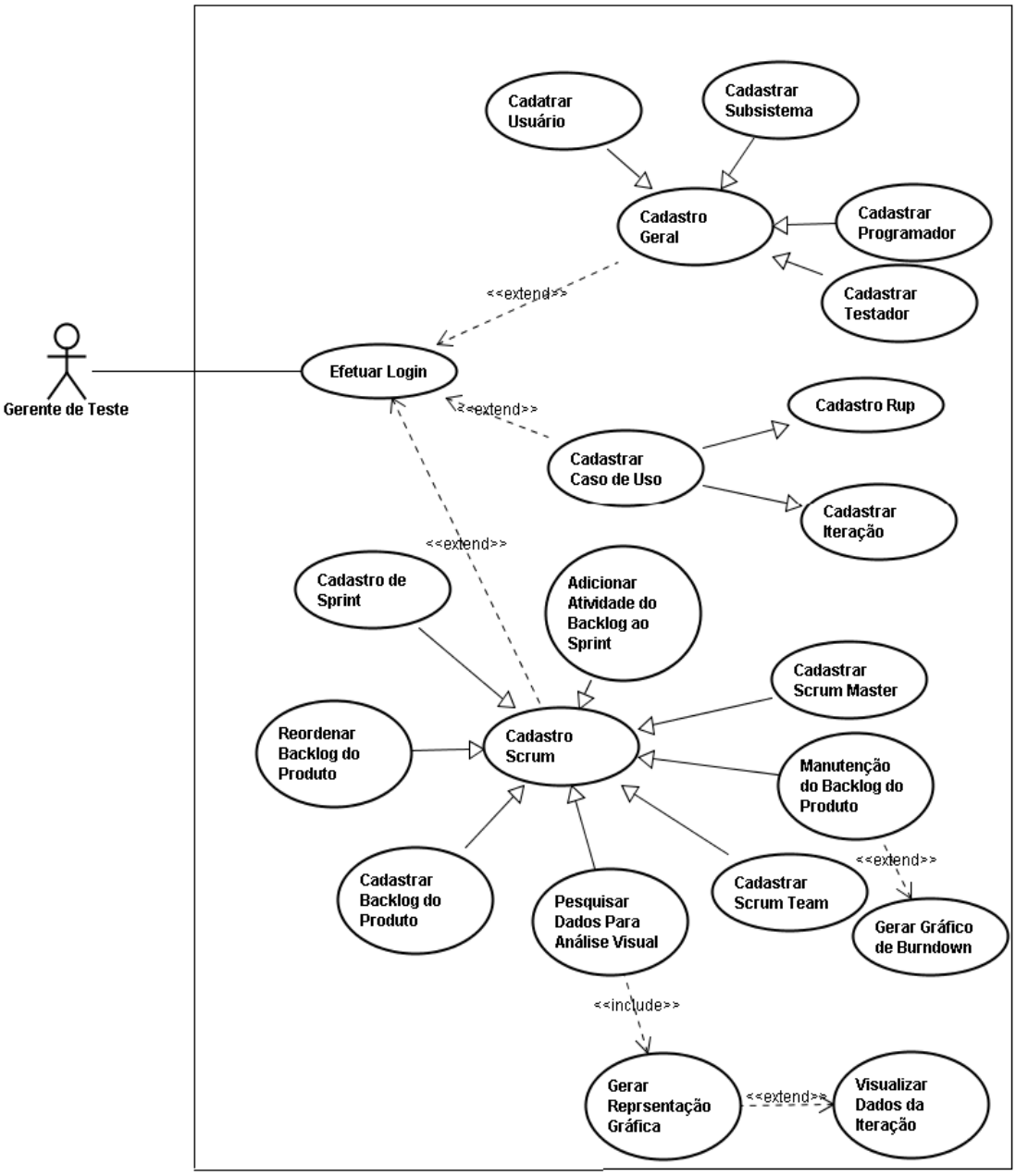


Figura 21: Diagrama de Caso de Uso do TestPlan.

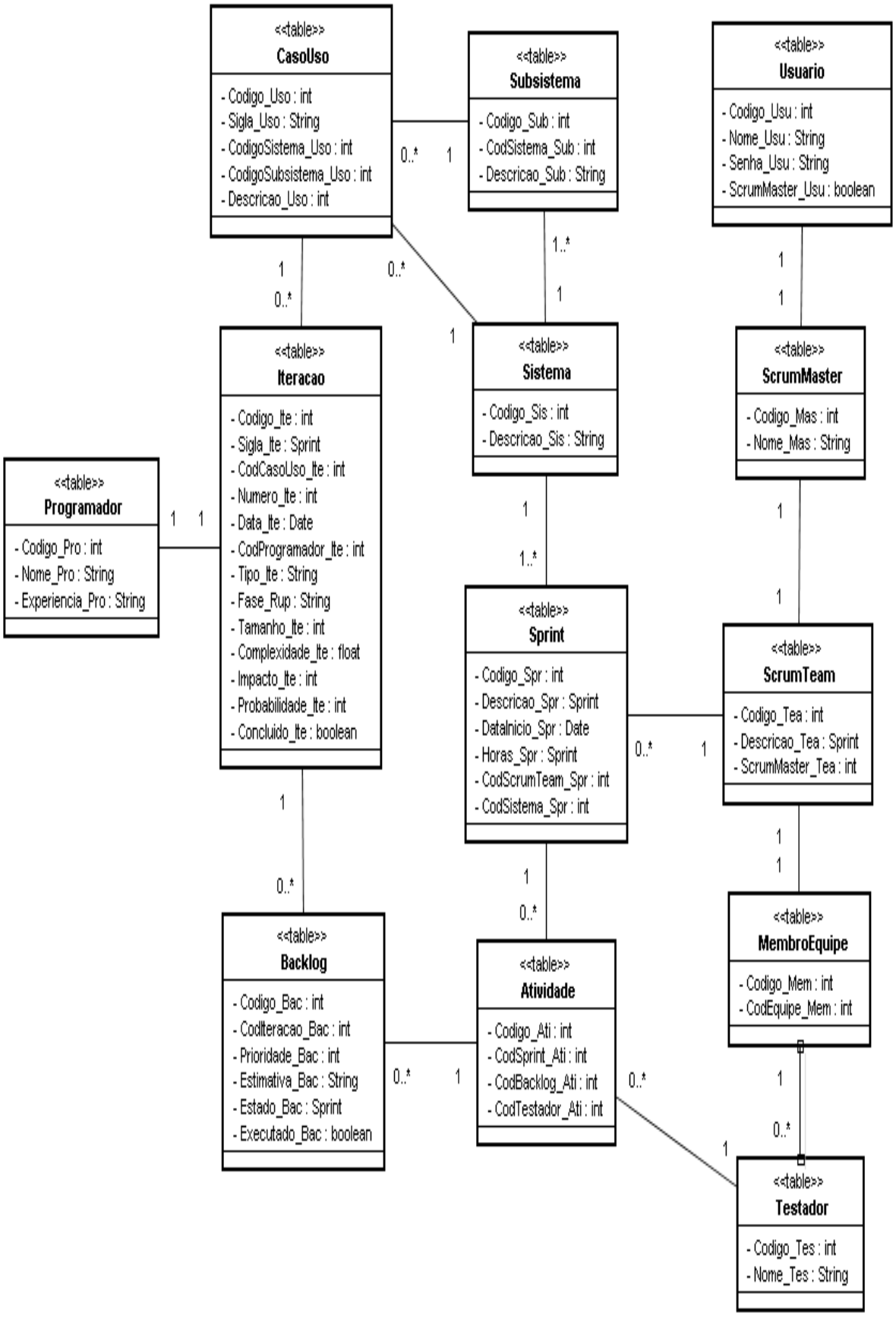


Figura 22: Diagrama de Classes para o TestPlan.

3.4 PADRÃO ADOTADO E FERRAMENTAS UTILIZADAS

A interface de usuário segue o padrão Windows XP para facilitar o aprendizado da utilização do TestPlan por parte dos gerentes de teste. O sistema de gerência de banco de dados utilizado foi o Microsoft Access, devido à sua simplicidade de uso, difusão no mercado e compatibilidade com o paradigma relacional.

O ambiente de desenvolvimento foi o Visual Studio 2005 versão Express (Microsoft, 2008), utilizando a linguagem de programação C#, pelos recursos de desenvolvimento rápido e suporte para a tecnologia orientada a objetos.

Para o desenvolvimento da representação gráfico foi utilizado a ferramenta visual Dundas Chart versão shareware distribuída gratuitamente (Dundas, 2008).

3.5 ARQUITETURA DO TESTPLAN

A partir dos requisitos funcionais, descritos no Anexo A, foi projetado uma solução arquitetural que atendesse a esses requisitos. Com isso, definiu-se um esquema da arquitetura do protótipo pode ser visto na Figura 23. A arquitetura está dividida em quatro módulos principais: controle de acesso, cadastro geral, cadastro Rup, cadastro Scrum.

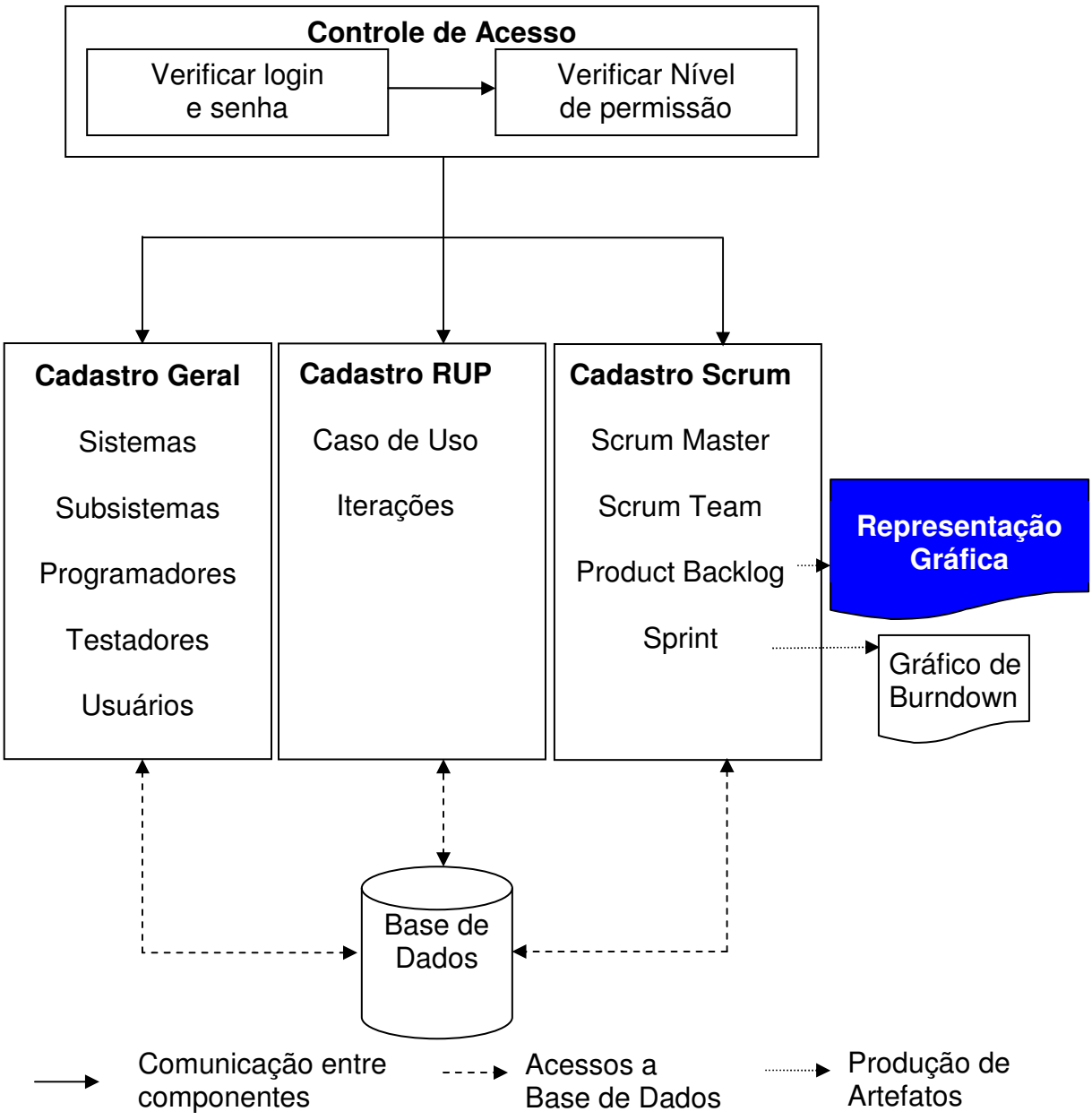


Figura 23: Arquitetura do TestPlan.

BANCO DE DADOS

Como ressaltado anteriormente, o protótipo utilizará uma base de dados, o que tornou necessário primeiramente definir quais serão os dados contidos nas tabelas de dados, os quais posteriormente serão submetidos à análise. Desta forma, foi feito um levantamento dos dados relevantes que necessitam estar

presentes na base de dados levando em consideração a sua importância para a tomada de decisão do gerente de teste.

Adotou-se que os dados seriam organizados em tabelas, por ser o tipo de dados base considerado no modelo de referência proposto por Card *et. al.* (1999) para a visualização de informação, além de uma forma comum de organização dos dados e por permitir um tratamento mais simples, uma vez que estes dados são interpretados sempre da mesma forma, isto é, a ocorrência de tuplas de valores relativos a um conjunto de variáveis.

CONTROLE DE ACESSO

Este componente é responsável por viabilizar o acesso à infra-estrutura pelos usuários, permitindo o acesso às suas funcionalidades somente por usuários previamente cadastrados na base de dados. Além disso, este componente verifica o nível de permissão do usuário que efetuar o *login*.

CADASTRO GERAL

O acesso ao cadastro geral é feito após a realização do *login* do usuário. Este módulo permite ao usuário realizar atividades que antecedem a visualização gráfica e a priorização das iterações. O usuário pode efetuar operações de cadastros e manutenções de sistemas, subsistemas, programadores, testadores e usuários, os quais são muito importantes para o funcionamento do protótipo.

CADASTRO RUP

É neste módulo que a implementação da metodologia do RUP está disponível. Desta forma, este módulo permite que os casos de uso e suas respectivas iterações sejam cadastradas. As iterações, aqui, merecem um destaque especial, pois a partir das informações pertencentes a elas será gerado o gráfico contendo os dados das iterações o que apoiará a tomada de decisão do gerente de teste.

CADASTRO SCRUM

É neste módulo que a implementação da metodologia Scrum está disponível. Além de possibilitar o cadastro e manutenção do Scrum Máster e Scrum Team, este módulo possibilitará a montagem da lista priorizada das atividades para o teste de software como artefato. Outra característica pertencente ao módulo é o controle do sprint que gera o gráfico burndown como artefato.

3.6 EXPERIMENTO DO TESTPLAN

Para a realização do experimento do TestPlan os dados cadastrados no banco de dados são simulados. Esses dados foram simulados para tornar possível a execução do TestPlan e apresentar os recursos que ele disponibiliza. Para a veracidade dos dados, no próximo capítulo, serão apresentados dois estudos de caso com dados reais. Porém, esses dados simulados não impedirá e nem comprometerá o experimento, pois estão sendo empregados para exemplificar a seqüência de operações que são necessárias para tornar o TestPlan em operação.

3.6.1 ENTRADA DE DADOS NO TESTPLAN

O primeiro passo refere-se aos cadastros: de sistema, subsistema, programadores, testadores e usuários. Para cada cadastro foi desenvolvida um tela e implementada uma solução conforme os requisitos funcionais listados no Anexo A.

Não serão apresentados os para cada funcionalidade por não ser o foco deste trabalho. Entretanto o cadastramento de cada requisito citado acima são fundamentais para a execução do TestPlan.

É possível utilizar ferramentas e/ou recursos para adiquir métricas diretamente do código fonte do sistema ou através dos cenários dos casos de uso. Porém, o cadastramento das métricas será feito manualmente para cada iteração ou caso de uso que seja utilizado por não ser o foco deste trabalho.

Por possibilitar a utilização de métricas obtidas através do código fonte ou da estrutura interna do software e as obtidas através dos casos de uso o TestPlan exige que seja informada qual das duas alternativas será empregada. Isto

está relacionado a etapa de desenvolvimento que o sistema em análise se encontra. As métricas são inseridas no cadastro de casos de uso ou no cadastro de iterações dependendo da alternativa selecionada. Assim, se o sistema se encontra na fase de codificação ou posteriores as métricas relacionada ao código fonte ou a estrutura interna do software deverão ser inseridas no “Cadastro de Iterações”. Caso o sistema esteja na fase inicial, no qual não há código fonte, as métricas baseadas nos requisitos funcionais deverão ser inseridas no cadastro de “Casos de Uso”.

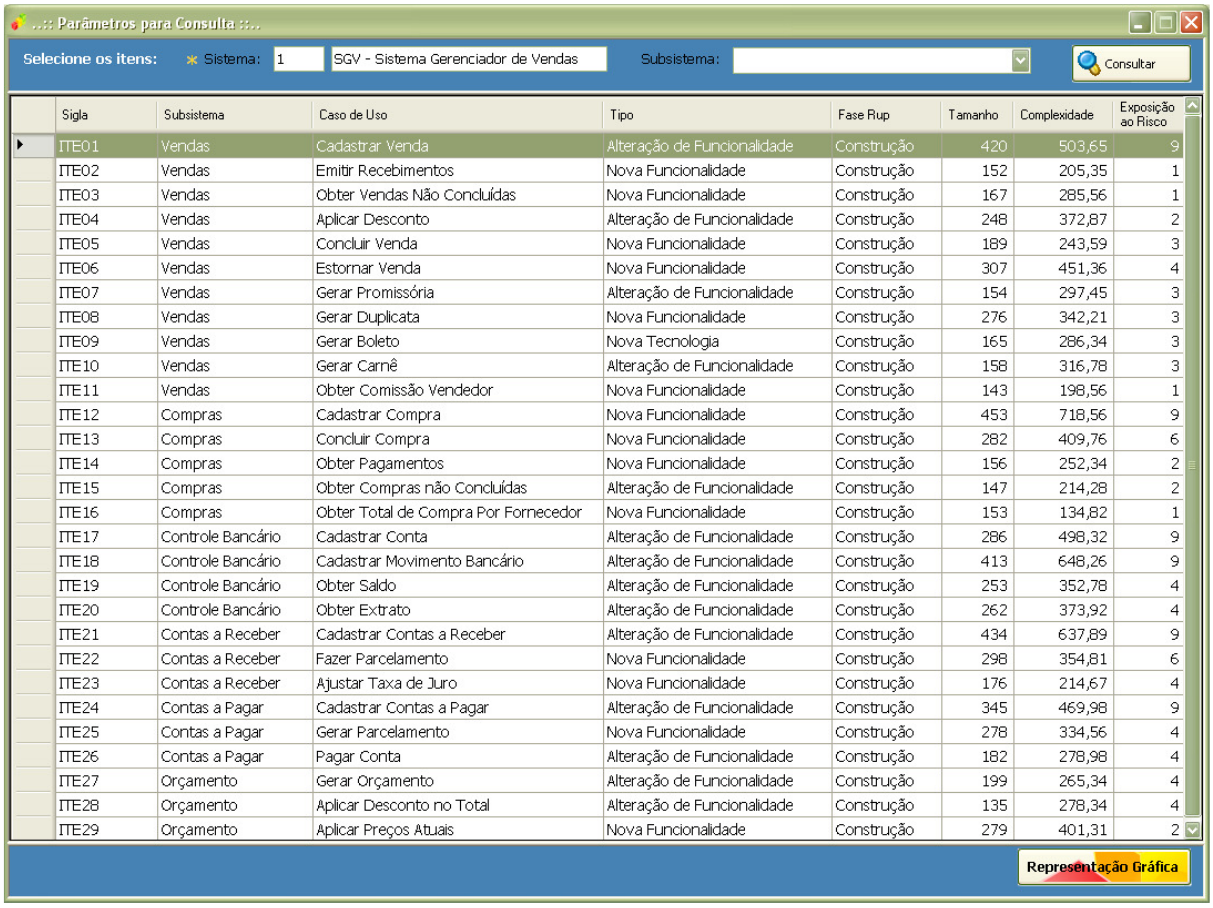
Seguindo o modelo de referência apresentado por Card *et. al.* (1999), após a etapa de transformar dados brutos: informações sobre sistema e subsistemas (casos de uso, iterações, métricas entre outras) em estruturas com entidades similares (tabelas). Neste caso, ao realizar os cadastros foi concluído o primeiro estágio de “Transformação de Dados”.

3.6.2 REPRESENTAÇÃO GRÁFICA

Sem dúvida um dos pontos mais importantes do TestPlan é o artefato de representação gráfica das métricas gerado no módulo Scrum, pois trata da solução proposta pela dissertação. É nessa etapa que estão empregados os conceitos de visualização de informação estudados na revisão bibliográfica e implementadas no TestPlan.

Para a geração da representação gráfica, primeiramente, é preciso que o gerente de teste efetue uma consulta a base de dados para o refinamento das informações que ele deseja analisar. Desta forma, houve a necessidade de desenvolver uma tela de consulta antes da representação gráfica.

A tela de consulta foi desenvolvida semelhante a um formulário, no qual são apresentados alguns parâmetros para a definição da consulta. Após definidos os parâmetros comandos SQL são enviados à base de dados e os dados retornados da pesquisa são tratados e apresentados no centro da tela em um formato tabular, conforme exibe a Figura 24.



Sigla	Subsistema	Caso de Uso	Tipo	Fase Rup	Tamanho	Complexidade	Exposição ao Risco
ITE01	Vendas	Cadastrar Venda	Alteração de Funcionalidade	Construção	420	503,65	9
ITE02	Vendas	Emitir Recebimentos	Nova Funcionalidade	Construção	152	205,35	1
ITE03	Vendas	Obter Vendas Não Concluídas	Nova Funcionalidade	Construção	167	285,56	1
ITE04	Vendas	Aplicar Desconto	Alteração de Funcionalidade	Construção	248	372,87	2
ITE05	Vendas	Concluir Venda	Nova Funcionalidade	Construção	189	243,59	3
ITE06	Vendas	Estornar Venda	Nova Funcionalidade	Construção	307	451,36	4
ITE07	Vendas	Gerar Promissória	Alteração de Funcionalidade	Construção	154	297,45	3
ITE08	Vendas	Gerar Duplicata	Nova Funcionalidade	Construção	276	342,21	3
ITE09	Vendas	Gerar Boleto	Nova Tecnologia	Construção	165	286,34	3
ITE10	Vendas	Gerar Carnê	Alteração de Funcionalidade	Construção	158	316,78	3
ITE11	Vendas	Obter Comissão Vendedor	Nova Funcionalidade	Construção	143	198,56	1
ITE12	Compras	Cadastrar Compra	Nova Funcionalidade	Construção	453	718,56	9
ITE13	Compras	Concluir Compra	Nova Funcionalidade	Construção	282	409,76	6
ITE14	Compras	Obter Pagamentos	Nova Funcionalidade	Construção	156	252,34	2
ITE15	Compras	Obter Compras não Concluídas	Alteração de Funcionalidade	Construção	147	214,28	2
ITE16	Compras	Obter Total de Compra Por Fornecedor	Nova Funcionalidade	Construção	153	134,82	1
ITE17	Controle Bancário	Cadastrar Conta	Alteração de Funcionalidade	Construção	286	498,32	9
ITE18	Controle Bancário	Cadastrar Movimento Bancário	Alteração de Funcionalidade	Construção	413	648,26	9
ITE19	Controle Bancário	Obter Saldo	Alteração de Funcionalidade	Construção	253	352,78	4
ITE20	Controle Bancário	Obter Extrato	Alteração de Funcionalidade	Construção	262	373,92	4
ITE21	Contas a Receber	Cadastrar Contas a Receber	Alteração de Funcionalidade	Construção	434	637,89	9
ITE22	Contas a Receber	Fazer Parcelamento	Nova Funcionalidade	Construção	298	354,81	6
ITE23	Contas a Receber	Ajustar Taxa de Juro	Nova Funcionalidade	Construção	176	214,67	4
ITE24	Contas a Pagar	Cadastrar Contas a Pagar	Alteração de Funcionalidade	Construção	345	469,98	9
ITE25	Contas a Pagar	Gerar Parcelamento	Nova Funcionalidade	Construção	278	334,56	4
ITE26	Contas a Pagar	Pagar Conta	Alteração de Funcionalidade	Construção	182	278,98	4
ITE27	Orçamento	Gerar Orçamento	Alteração de Funcionalidade	Construção	199	265,34	4
ITE28	Orçamento	Aplicar Desconto no Total	Alteração de Funcionalidade	Construção	135	278,34	4
ITE29	Orçamento	Aplicar Preços Atuais	Nova Funcionalidade	Construção	279	401,31	2

Figura 24: Tela de Consulta com dados filtrados da Tabela de Dados Brutos.

A quantidade de registros retornados pela consulta dependerá do sistema que sobre análise. A facilidade de entendimento e análise dos dados é proporcional a esta quantidade. No caso de analisar um sistema com vários subsistemas integrados, esta análise exige um maior esforço e tempo necessário para interpretá-la. Por esse motivo a o TestPlan possui a característica de refinar a consulta por subsistemas, selecionando um subsistema específico, pretendendo amenizar este problema.

A Figura 25 ilustra este refinamento selecionando o subsistema “Vendas”. Desta forma, os dados consultados podem oscilar de acordo com o controle de seleção.

Sigla	Subsistema	Caso de Uso	Tipo	Fase Prep	Tamanho	Complexidade	Exposição ao Risco
ITE01	Vendas	Cadastrar Venda	Alteração de Funcionalidade	Construção	420	503,65	9
ITE02	Vendas	Emitir Recebimentos	Nova Funcionalidade	Construção	152	205,35	1
ITE03	Vendas	Obter Vendas Não Concluídas	Nova Funcionalidade	Construção	167	285,56	1
ITE04	Vendas	Aplicar Desconto	Alteração de Funcionalidade	Construção	248	372,87	2
ITE05	Vendas	Concluir Venda	Nova Funcionalidade	Construção	189	243,59	3
ITE06	Vendas	Estornar Venda	Nova Funcionalidade	Construção	307	451,36	4
ITE07	Vendas	Gerar Promissória	Alteração de Funcionalidade	Construção	154	297,45	3
ITE08	Vendas	Gerar Duplicata	Nova Funcionalidade	Construção	276	342,21	3
ITE09	Vendas	Gerar Boleto	Nova Tecnologia	Construção	165	286,34	3
ITE10	Vendas	Gerar Carnê	Alteração de Funcionalidade	Construção	158	316,78	3
ITE11	Vendas	Obter Comissão Vendedor	Nova Funcionalidade	Construção	143	198,56	1

Figura 25: Refinamentos de dados utilizando o controle de seleção de subsistemas.

Entretanto, mesmo aplicando filtros de seleção há uma melhora na apresentação dos dados o que reduz o tempo de análise, mas devido a limitação do ser humano em memorizar dados parciais o tempo de análise ainda é considerado longo (Card *et. al.*, 1999).

O próximo passo consiste em selecionar uma estrutura visual que melhor represente os dados contidos nas tabelas. A estrutura visual escolhida para apresentar os subsistemas que fazem parte de um sistema que necessitam recebe prioridade a serem testados segundo as métricas de software, foi a representação através da técnica de gráfico de bolhas. Essa escolha levou em consideração a quantidade de informações a serem exibidas, a manipulação dessas informações e os efeitos no processo cognitivo dos gerentes de teste, considerando que a representação visual será efetiva e eficiente para a tarefa em questão.

Dispondo dos dados retornados pela consulta o TestPlan irá gerar o mapeamento visual, Figura 26. Como resultado será apresentado um gráfico que contará com dois eixos: x para a horizontal e y para a vertical. Utilizando, desta forma, o substrato espacial como discutido na seção 2.2.2 da revisão bibliográfica.

O eixo x terá representação nominal, informando os casos de uso ou as iterações. O eixo y terá representação quantitativa, informando os valores de uma métrica.

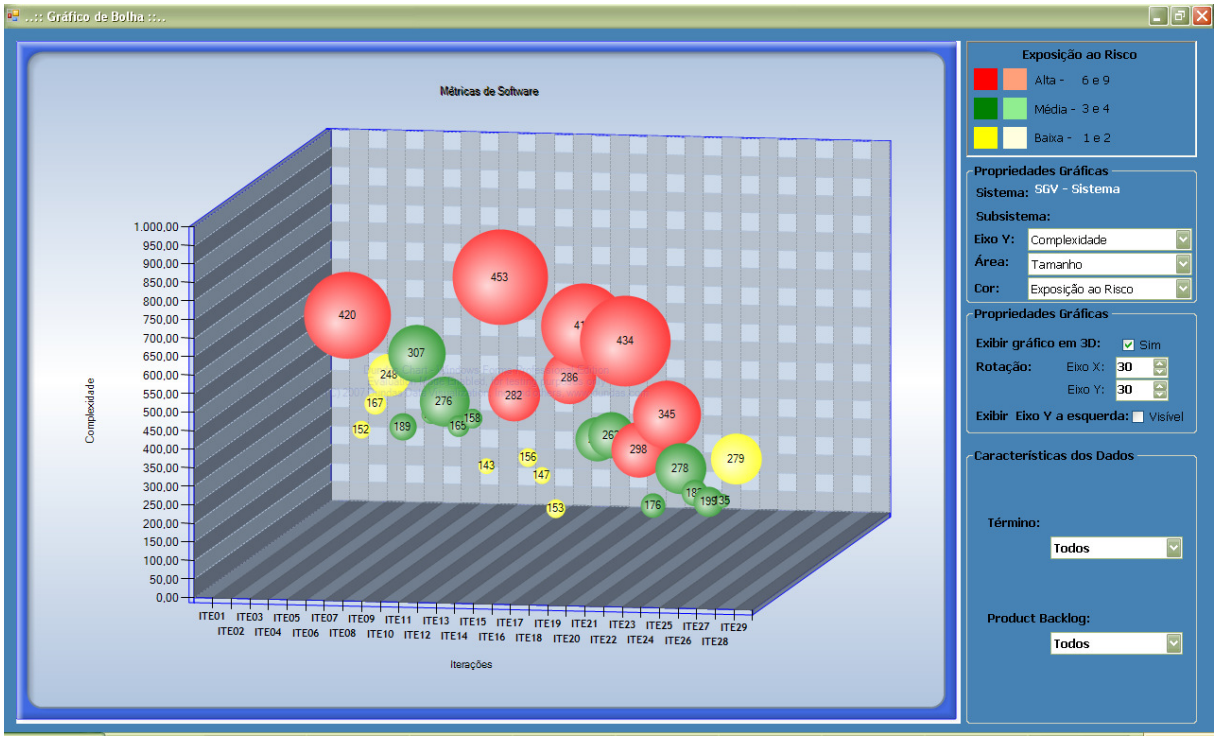


Figura 26: Visualização Gráfica do Sistema SGV em 3D.

Os pontos gerados pelas coordenadas x e y serão os casos de uso ou as iterações do sistema em análise representadas por bolhas. Esse elemento visual representa uma marca, como exemplificado na seção 2.2.2 da revisão bibliográfica. Por ser uma marca é caracterizada por propriedades gráficas, neste contexto, são utilizadas as propriedades: cor, forma e área.

Cada bolha terá uma cor associada a ela, o que servirá para demonstrar uma métrica. Da mesma maneira as bolhas terão dimensões diferentes que representarão outra métrica. No caso da Figura 26 o eixo y está representando a métrica de complexidade de código, portanto, as bolhas com maior complexidade estão mais ao topo do gráfico. A métrica de tamanho (LOC) está representada pela área das bolhas, portanto, as bolhas com áreas maiores apresentam trechos de código mais extensos. As cores estão indicando a exposição ao risco da funcionalidade implementada no trecho de código. Com auxílio da legenda pode-se notar que o risco é alto quando a bolha for vermelha, médio quando a bolha for verde e pequeno quando a bolha for amarela. E, por fim, o número associado a cada bolha representa a área, neste caso, os valores são o número de linhas codificadas da iteração.

Estas são informações tiradas instantaneamente analisando a representação visual. A primeira conclusão, neste caso, mostra que a maior bolha “ITE12” apresenta a maior complexidade, o maior tamanho e alto grau de exposição ao risco, isto é, este trecho de código merece atenção em especial segundo as métricas de software, ou seja, ele seria o primeiro a ser testado, pois dependerá de maior tempo para o teste de software, um testador experiente, entre outras questões.

Outra característica importante que foi empregada no TestPlan é o mapeamento por cores. Através da utilização de cores há uma distinção bem maior se comparada com a utilização de escala de cinzas. Então a solução é determinar uma escala de cor que diferencie bem entre si e que seja natural para os gerentes de teste. Para auxiliá-lo haverá uma legenda para especificar qual o grau de exposição ao risco (alta, média e baixa) representando cada cor, caso seja essa a métrica selecionada.

3.6.3 GERANDO NOVAS VISÕES

Uma vez que os dados da tabela de dados estão representados em uma estrutura visual, o TestPlan possibilita que os gerentes de teste interajam com essa estrutura, alterando o seu formato, mas mantendo o seu conteúdo, criando novas visões da estrutura visual de acordo com a necessidade dos gerentes de teste.

O TestPlan permite visualizar o conjunto de dados em diferentes dimensões: bidimendional (2D) e tridimensional (3D). Ao desmarcar o controle “Exibir o gráfico em 3D” a representação visual será representada em 2D. No caso de marcar o controle e obter uma representação 3D o TestPlan possibilita a rotação relativa ao eixo vertical e ao eixo horizontal.

Através dos controles seletivos “Eixo Y”, “Área” e “Cor” os gerentes podem alterar as métricas selecionadas, criando novas visões. Há a possibilidade de gerar três formatos distintos alterando os controles seletivos das métricas.

1) Determinando as métricas:

- Eixo Y = “Complexidade”;
- Área = “Tamanho” e
- Cor = “Exposição ao Risco”.

O gráfico ficará como mostra a Figura 27.

2) Determinando as métricas:

- Eixo Y = “Tamanho”;
- Área = “Exposição ao Risco” e
- Cor = “Complexidade”.

O gráfico ficará como mostra a Figura 28.

3) Determinando as métricas:

- Eixo Y = “Exposição ao Risco”;
- Área = “Complexidade” e
- Cor = “Tamanho”.

O gráfico ficará como mostra a Figura 29.

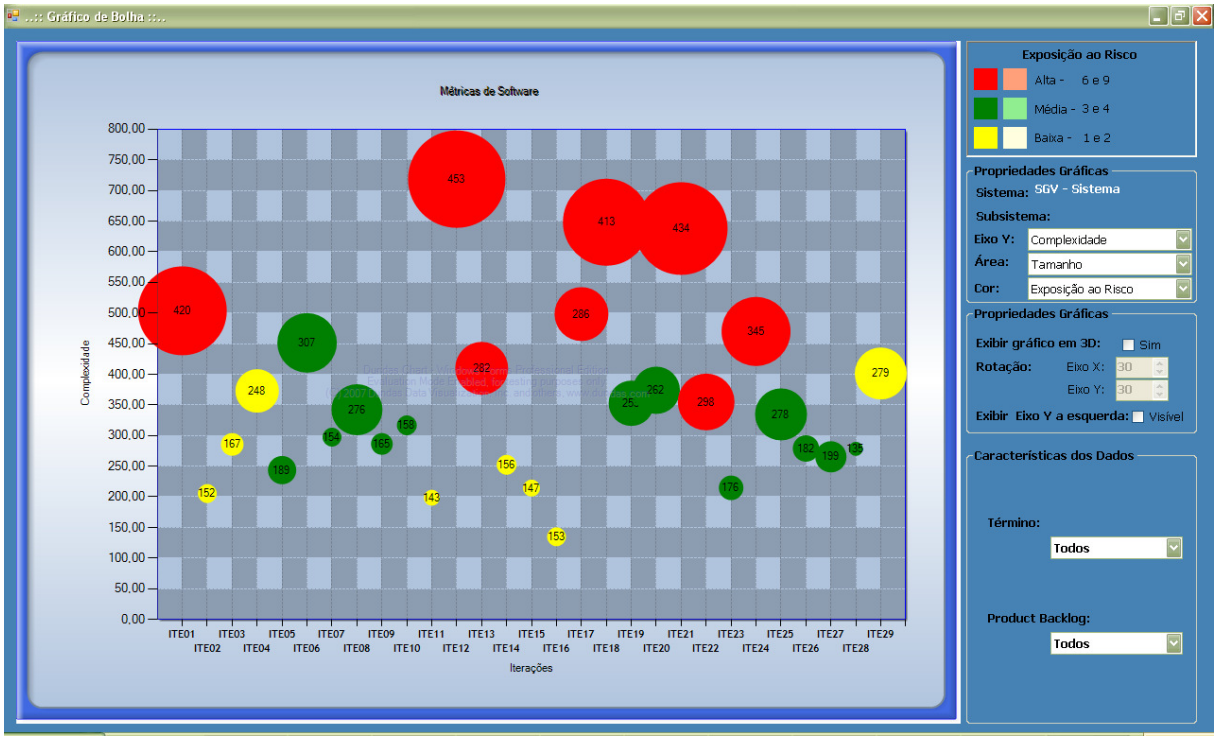


Figura 27: Visão: eixo y, área e cor representam complexidade, tamanho e exposição ao risco respectivamente.

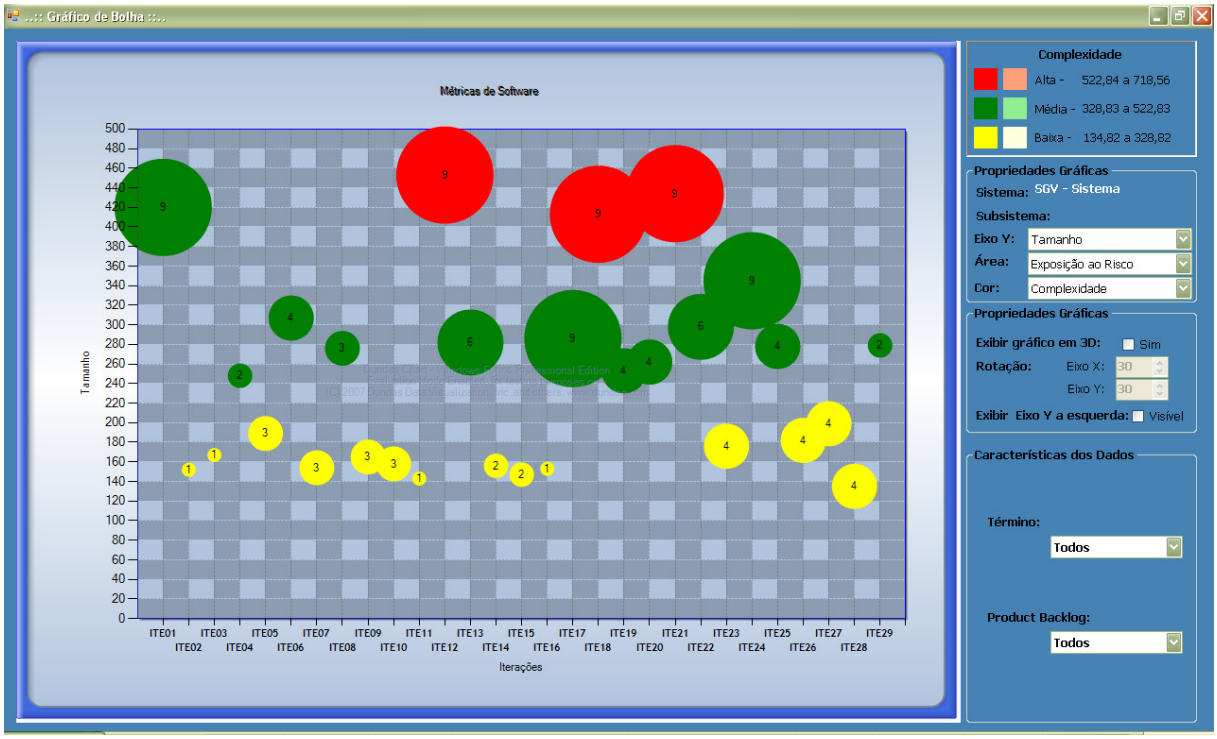


Figura 28: Visão: eixo y, área e cor representam tamanho, exposição ao risco e complexidade respectivamente.

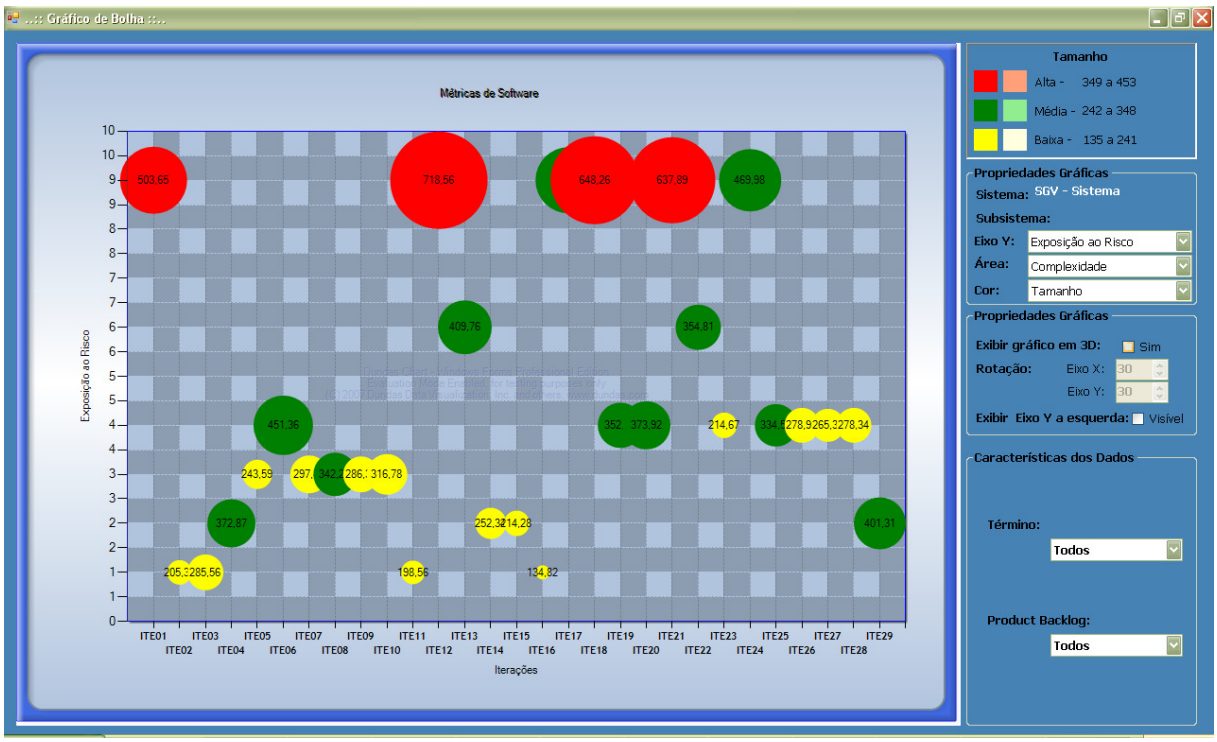


Figura 29: Visão: eixo y, área e cor representam exposição ao risco, complexidade e tamanho respectivamente.

1) Analisando a Figura 27:

As iterações que possuem maior complexidade de código estão mais ao topo do gráfico, os trechos de código mais extensos são os que representam maior área e o grau de exposição ao risco é alta para as iterações representadas pela bolha vermelha, média para as bolhas verde e baixa para as bolhas amarelas.

2) Analisando a Figura 28:

As iterações que possuem maior tamanho (trechos mais extensos) estão mais ao topo do gráfico, os trechos de código com maior grau de exposição ao risco são os que representam maior área e a complexidade de código é alta para as iterações representadas pela bolha vermelha, média para as bolhas verde e baixa para as bolhas amarelas.

3) Analisando a Figura 29:

As iterações que possuem maior grau de exposição ao risco estão mais ao topo do gráfico, os que apresentam maior complexidade de código são os que representam maior área e os trechos de código são mais extensos para as iterações representadas pela bolha vermelha, médio para as bolhas verde e baixo para as bolhas amarelas.

3.6.4 INTERAÇÃO COM TESTPLAN

Como o protótipo utilizará eixos com valores para representar atributos, para cada eixo, os quais possuem propriedades para apresentar distância entre as informações deparamos com duas possibilidades. A primeira delas é que se as características dos itens exibidos forem bem distintas, irá gerar regiões no gráfico de acordo com a probabilidade da distribuição dos valores de distância, pelo motivo de não serem uniformes, podendo gerar, por exemplo, grandes espaços sem preenchimento. Por outro lado, podem ocorrer a sobreposição dos itens, caso haja dados com características semelhantes, esse sim é um fator que pode ocasionar dúvidas para a análise dos dados. Propondo solucionar esse problema, utilizaremos a interação do gerente de teste com o TestPlan, o qual pode clicar com o mouse

sobre a bolha e o TestPlan exibe maiores informações e o problema de ocasionar alguma dúvida pode ser solucionado.

Além disto, existem outras informações que estão ocultas ao gerente de teste e também tem sua importância para a tomada de decisão, como por exemplo, a experiência do programador que codificou o trecho de código. Para contornar essa questão o gerente de teste deverá posicionar o mouse sobre a bolha, o cursor mudará do formato de seta para uma mão com o dedo indicador, e pressionará o mouse. Após realizar este evento uma nova tela se abrirá com informações adicionais sobre o trecho de código. A Figura 30 mostra a tela de informações adicionais.

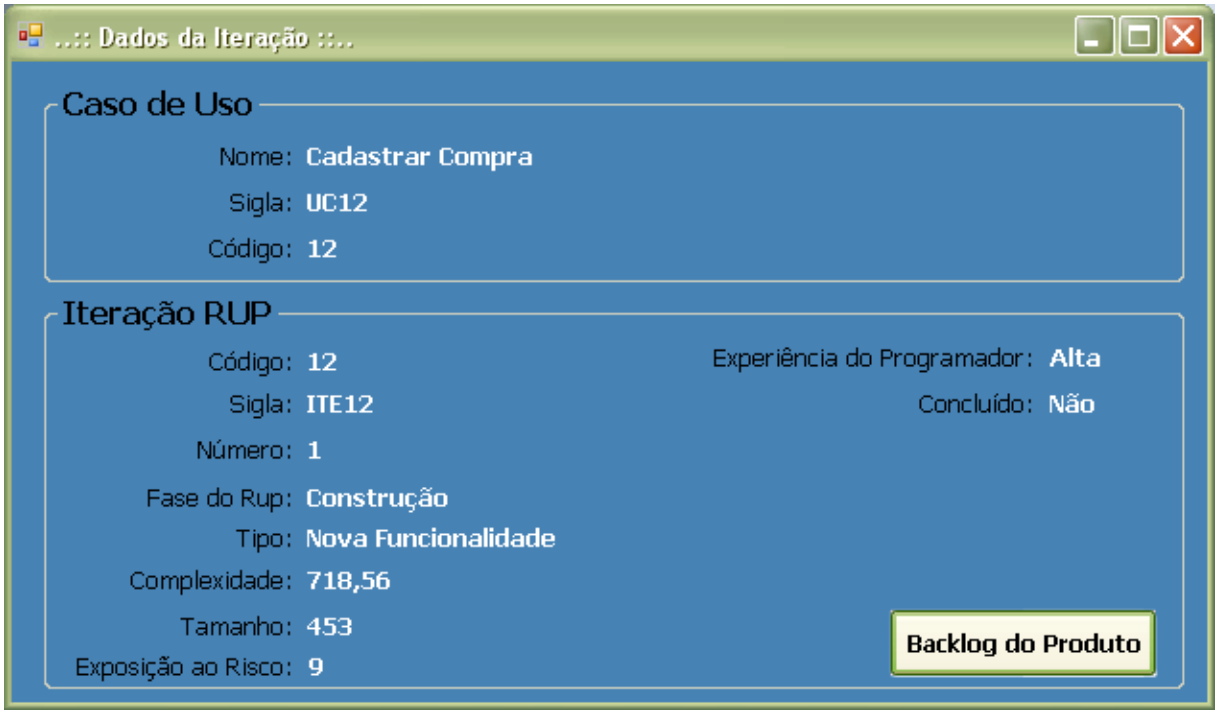


Figura 30: Dados adicionais sobre a iteração “ITE12”.

3.6.5 APLICAÇÃO DO SCRUM NO TESTPLAN

É a partir do gráfico gerado que o gerente de teste analisará cada iteração (bolhas) e poderá gerar a lista priorizada para o teste de software. Este é uns dos objetivos desta dissertação, a partir de representação gráfica o gerente de teste busque a sua meta: priorizar os trechos de código que merecem maior atenção. Para isso utilizamos vários conceitos da metodologia Scrum para que esta tarefa pudesse ser concretizada. Uma questão que merece destaque aqui é que a

tomada de decisão no processo de planejamento será de responsabilidade do gerente de teste, o gráfico oferece a ele informações, mas a decisão de priorizar o teste só cabe a ele.

A partir da tela “Dados da Iteração”, Figura 30, o gerente de teste tem a possibilidade de montar a lista ordenada das atividades para o teste de software, ou seja, o *backlog* do produto. Essa tarefa deve ser feita respeitando a ordem dos itens a testar. A cada item adicionado ao *backlog* do produto é necessário estimar em horas o quanto cada atividade requer para ser executada. A Figura 31 exibe a tela de cadastro do backlog do produto.

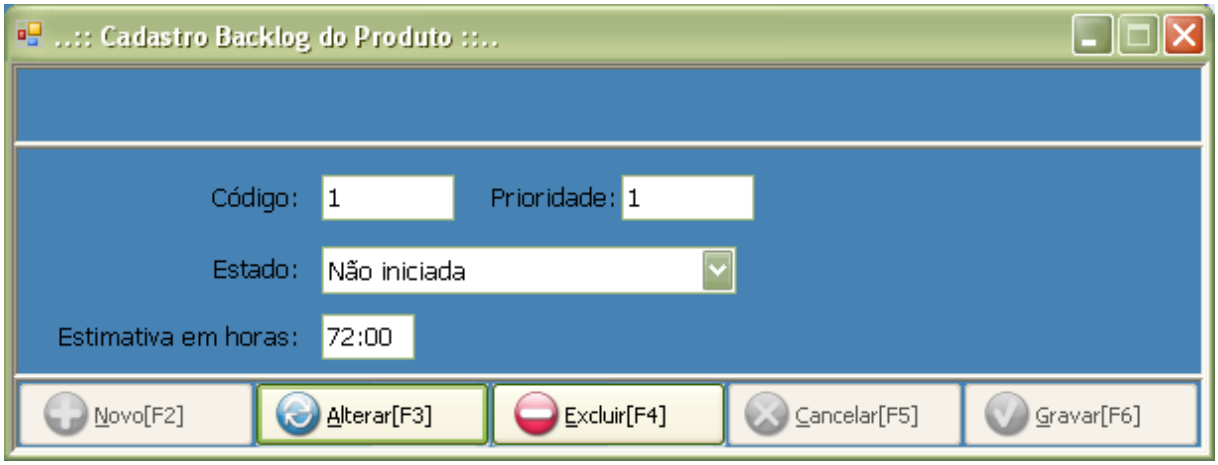


Figura 31: Cadastro do *Backlog* do Produto.

A este ponto o gerente de teste já tem a possibilidade de ir adicionando as iterações no *Backlog* do Produto, vale ressaltar, que a ordem é um fator muito importante neste passo. As iterações devem ser adicionadas na ordem que deseja que sejam executadas no teste. A cada iteração adicionada na lista priorizada a bolha que a representa no gráfico terá a sua cor alterada para um tom de opacidade. Com isso, tem-se mais uma característica para auxiliar o gerente de teste informando que aquela iteração já está presente na lista, como exibe a Figura 32.

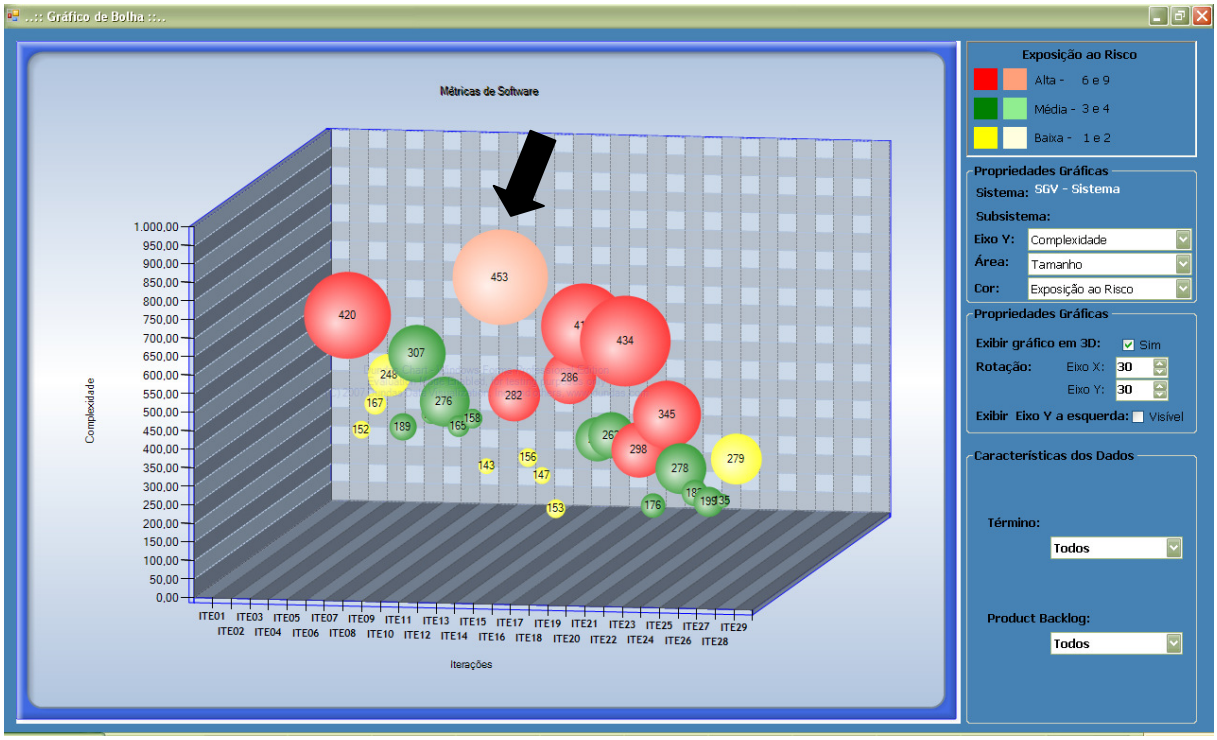


Figura 32: Iteração adicionada ao *backlog* do produto.

Outro aspecto a ser considerado é que o fato de uma funcionalidade ser mais simples do que outra não permite que ela seja dispensada do teste. Para se garantir a qualidade de um software, todas as funcionalidades devem ser bem testadas. Este aspecto remete a uma questão, que ficou fora do escopo do trabalho, que é o quanto cada iteração deve ser testada. Este aspecto está relacionado aos requisitos de teste e, conseqüentemente, aos casos de teste. Quanto maior o número de casos de testes, mais tempo será necessário para executar o teste. E está tarefa, de estimar o tempo, é de responsabilidade do gerente de teste.

Partindo do pressuposto que os trechos de código que apresentam as maiores métricas devem ser priorizados, a tarefa de adicionar a iteração (bolha) ao *backlog* do produto deve ser feita para todas as iterações passo a passo. Assim, a lista do *backlog* do produto pode ser completada e iniciar a definição do *sprint*. A Figura 33 exibe a tela de cadastro do sprint.

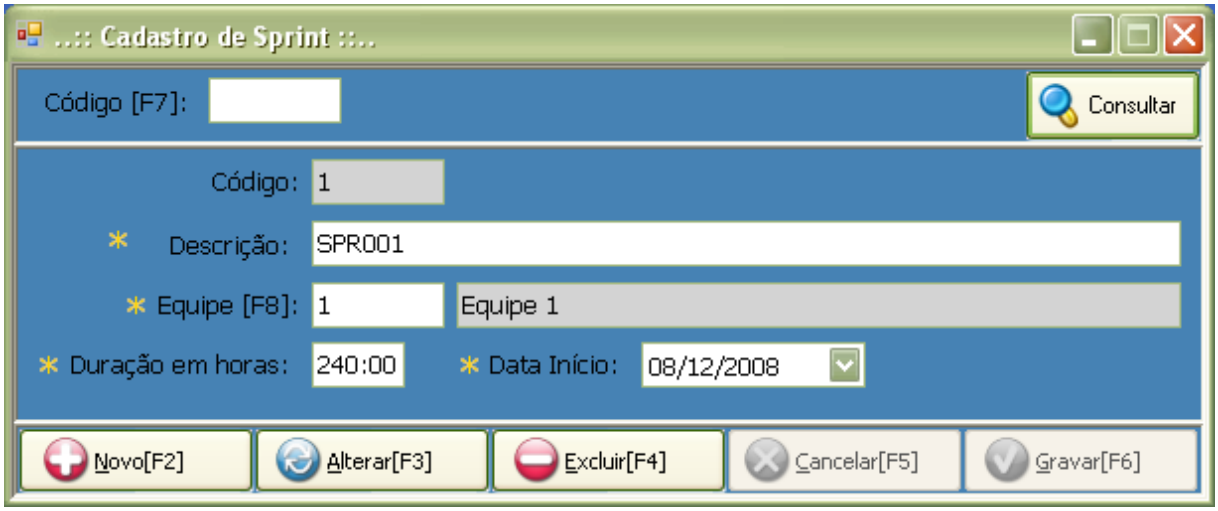


Figura 33: Cadastro do *sprint*.

Definido que o *sprint* terá duração de 30 dias, considerando que sejam trabalhadas 8 horas diárias o que resultaria em 240 horas. É o momento para definir as tarefas com seus respectivos responsáveis, neste caso, os responsáveis são os testadores. Funcionalidade implementada pelo protótipo como exibe a Figura 34.

Figura 34: Tela para adicionar atividades para os respectivos responsáveis.

Definidos os responsáveis por quais tarefas e dado início ao sprint, a medida que cada testador for executando as suas atividades eles devem ir atualizando o tempo gasto com cada uma. Este tempo gasto será acumulativo

conforme os dias trabalhados eles serão atualizados através da tela de manutenção do *sprint*, como mostra a Figura 35.

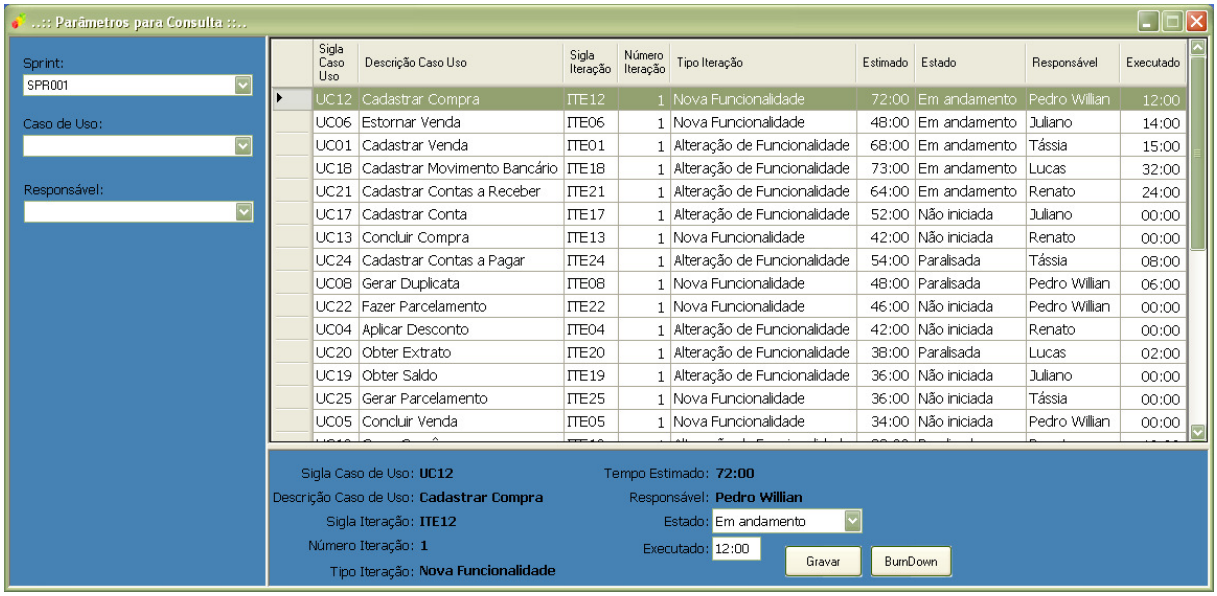


Figura 35: Tela de manutenção do *sprint.*

Com todas essas informações é possível gerar o gráfico de *burndown* que é um documento feito todo dia na qual a equipe do s*print* identifica e estima as tarefas que faltam para que o s*print* seja completado com sucesso. Com as tarefas especificadas o trabalho segue e o s*crum master* recalcula o trabalho restante para completar o s*print*. O *burddown chart* é um gráfico acumulativo do trabalho restante que tem como objetivo acompanhar a produção da equipe, quando o trabalho restante chega a zero então o Scrum foi completado. A Figura 36 exibe o gráfico de *burndown* gerado para o dia 30 de dezembro de 2008.

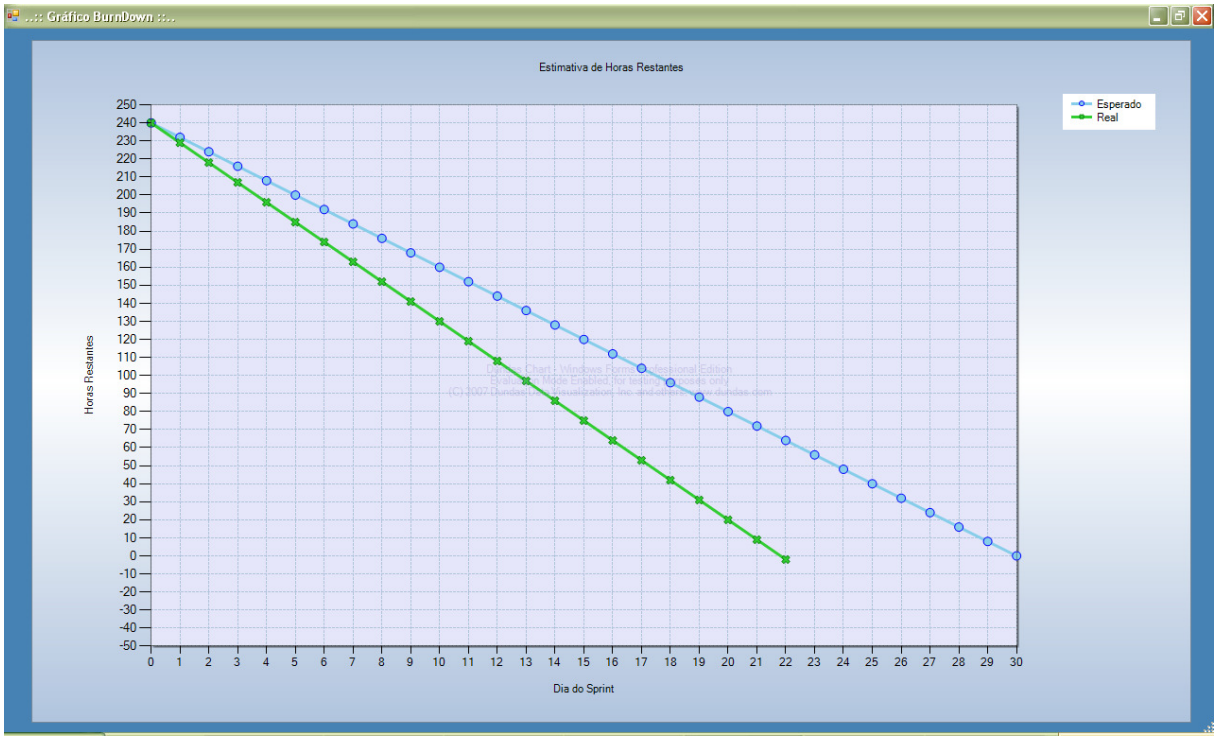


Figura 36: Gráfico de *burndown* do dia 30 de dezembro.

3.6.6 OUTROS RECURSOS DE INTERATIVIDADE

Como também é importante ressaltar, que as iterações que já passaram pela atividade de teste é já estão concluídas serão mapeadas para isso serão mapeadas com forma diferente (um “X”), como mostra a Figura 37.

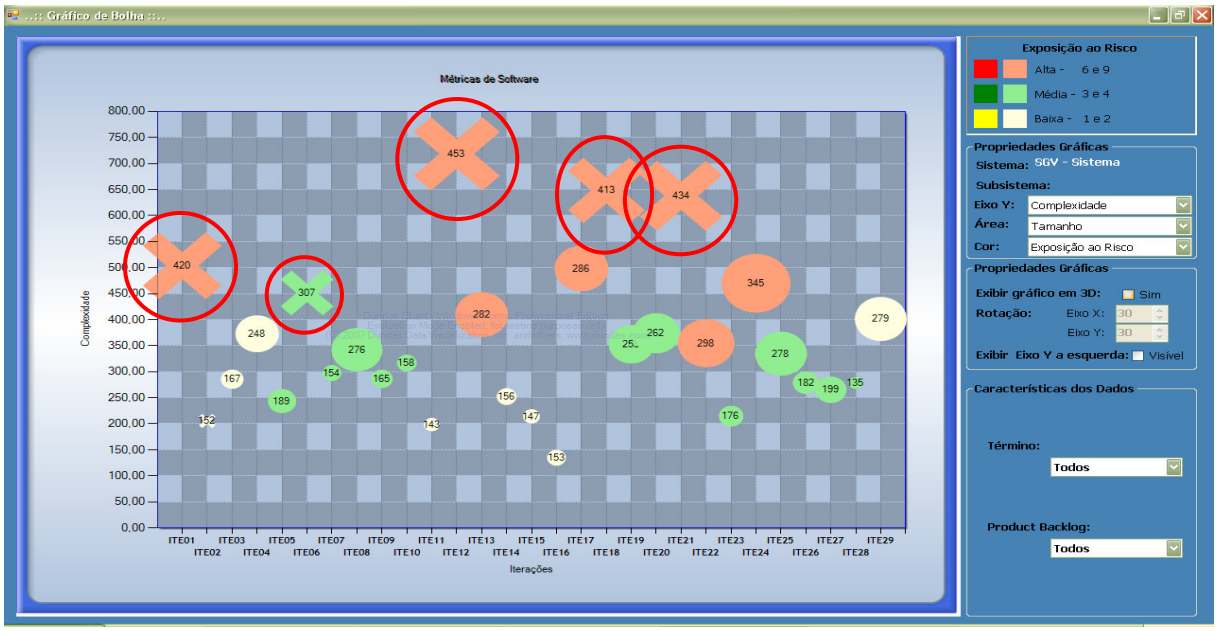


Figura 37: Iterações que já passaram pelo teste.

A utilização da interatividade no TestPlan possibilita que os gerentes de teste observem os pontos de maior relevância e escolha observar apenas os dados escolhidos através do uso dos controles seletivos: “Término” e “Product Backlog”. A Figura 38 mostra apenas as iterações que não estão concluídas. A Figura 39 mostra apenas as iterações que não foram adicionadas ao *Product Backlog.*

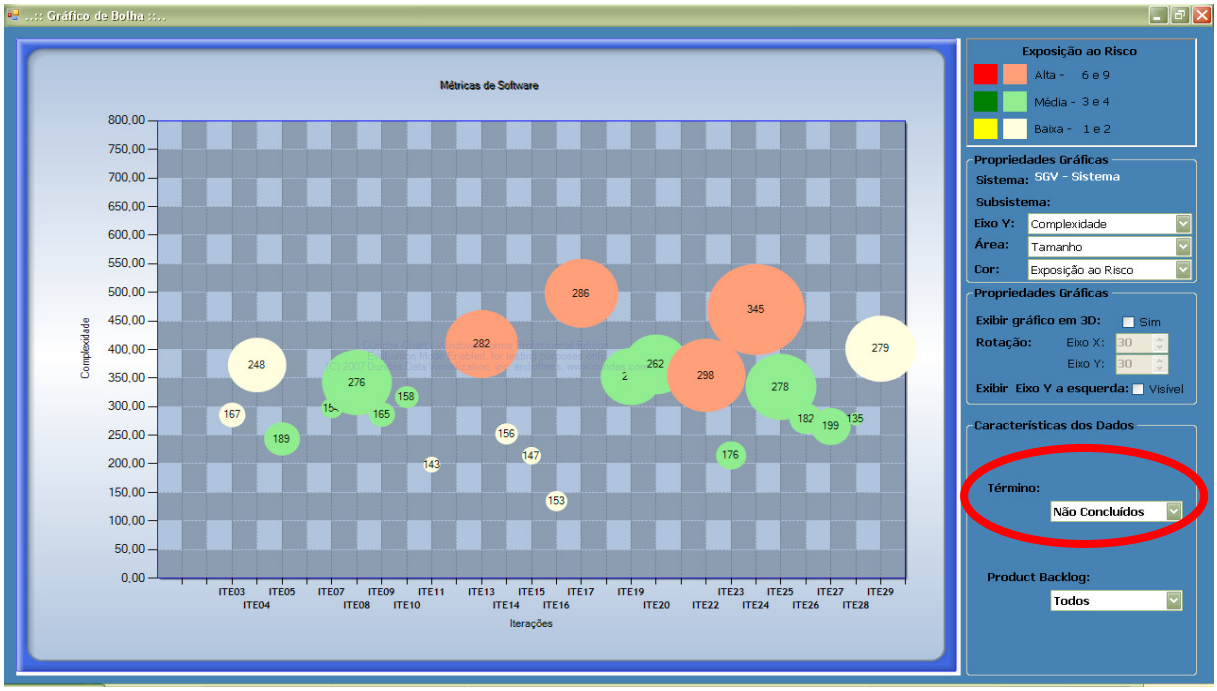


Figura 38: Mapeamento das Iterações não concluídas.

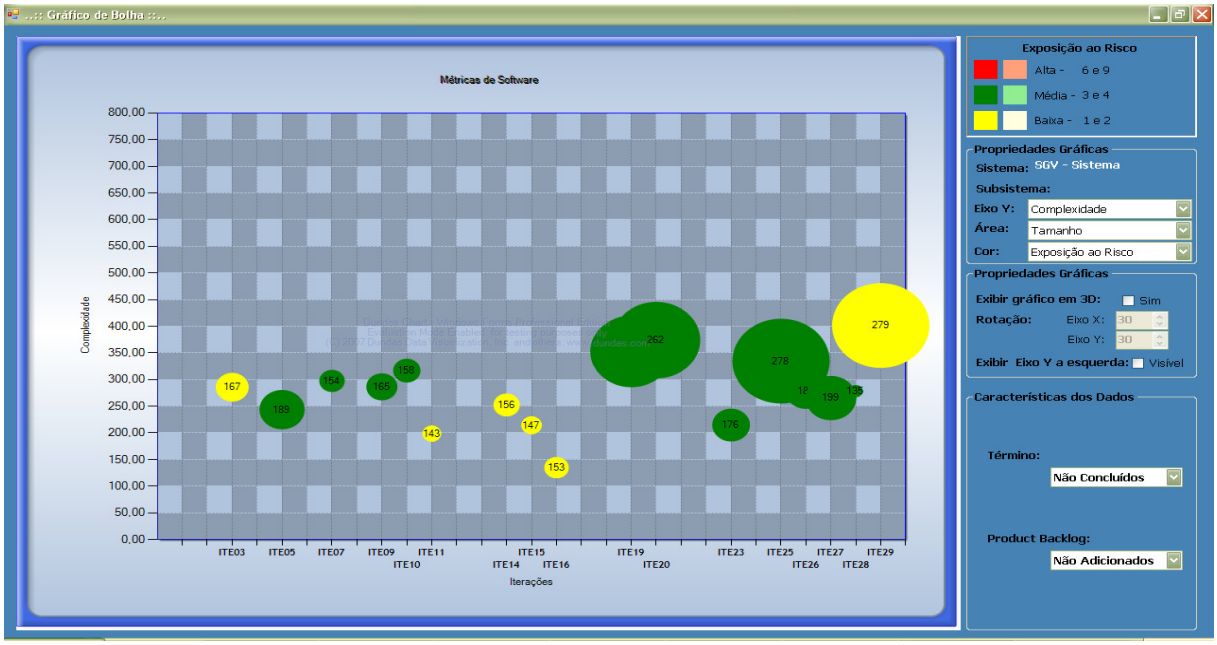


Figura 39: Mapeamento das Iterações não adicionadas ao *Product Backlog.*

3.7 SOLUÇÕES ALTERNATIVAS

Dada a solução proposta pela dissertação para o problema abordado, outras maneiras alternativas para a resolução do problema foram analisadas.

Uma das questões levantadas foi analisar as iterações através do nível de sua criticidade, levando em conta a opinião do cliente em relação ao produto. Assim, haveria uma priorização conforme a solicitação do mesmo, ou seja, para o desenvolvimento de uma nova funcionalidade, alteração de uma funcionalidade o qual poderia alegar vários motivos para tal, como por exemplo, uma falha ocorrida no software, dificuldade em operar o sistema, operações demoradas que deveriam ser otimizadas, entre outras. Desta forma, os casos de teste seriam priorizados seguindo a solicitação do cliente.

Outra maneira, porém mais complexa que a primeira por ter a necessidade de conhecer os problemas do software, seria com base no conhecimento dos defeitos do programa. Então, seleciona-se os casos de teste que os revelam. Assim, os casos de testes que mais revelem defeitos seriam priorizados e poderiam ser classificados por categorias ou pesos.

Ainda, fazer a priorização por cobertura de comandos, no qual mede-se por instrumentação a cobertura de código para cada caso de teste. Sendo assim, os casos de teste com maior cobertura seriam priorizados.

Essas foram algumas das formas de priorização levantadas, mas há inúmeras outras maneiras que também seriam válidas. A única que seria totalmente descartada, por desviar-se do escopo do trabalho, seria a priorização aleatória, sem critério prévio de ordenação, no qual os casos de testes seriam priorizados em qualquer ordem.

4 Estudo De Caso

4.1 SISTEMA DE GERENCIAMENTO DE MERCIARIA - MERCI

Após a implementação do protótipo visando verificar a sua contribuição para solucionar o problema identificado, foi realizado um estudo utilizando um software o qual disponibiliza o seu código fonte, suas especificações de requisitos e modelos documentados. O software em questão é denominado Merci desenvolvido para gerenciamento de mercearia e foi implementado para mostrar a aplicação do modelo de processos Práxis (Paula Filho, 2008).

O software Merci, desenvolvido em Java, foi escolhido por já existir um estudo sobre o software que avalia a relação entre o tamanho e a complexidade de um software e o número estimado de seus defeitos (Lucca, 2007). Desta forma, houve facilidade em obter a coleta de dados e o cálculo das métricas já disponibilizadas no estudo sem precisar preocupar-se de como estas métricas seriam coletadas. Porém, o estudo demonstra que a coleta de dados e os cálculos das métricas foram realizados através da documentação do software disponibilizado no site de onde além da documentação, o próprio software está disponível (Paula Filho, 2008).

As métricas escolhidas pelo estudo foram: LOC, como métrica de tamanho; Dificuldade e Esforço de Halstead, como métrica de complexidade de código; Erros de Halstead, como estimador de defeitos do software e Pontos de Função, como métrica de complexidade funcional. Foi utilizada a métrica de estimativa de feito ao invés do número real de defeitos, pois segundo Lucca são escasso programas com defeitos reais documentados. Mesmo atualmente tendo uma gama de software com código fonte disponível, a documentação de defeitos detectados durante as fases de unidade, de integração e de sistemas ainda não é encontrada (Lucca *et. al.,* 2007).

No presente estudo selecionaremos apenas as métricas LOC, Dificuldade de Halstead e Pontos por Função. Essas métricas foram calculadas através da ferramenta Counter.java, desenvolvida para fins acadêmicos, que conta o número de linhas de código e calcula as métricas de Halstead. A Tabela 2 exibe as

métricas coletadas e calculadas para cada caso de uso do software Merci. A Figura 40 exibe a “Tela de Consulta” com os dados do software Merci.

Tabela 16: Dados tabulados para o Sistema Merci (adaptado de Lucca, 2007).

Casos de Uso	Sigla	LOC	Dificuldade	PF
Gestão de Usuário	ITEMER01	670	579,27	23
Emissão de Relatórios	ITEMER02	461	396,69	20
Gestão Manual de Estoque	ITEMER03	267	169,58	6
Gestão de Mercadorias	ITEMER04	839	595,91	26
Gestão de Pedido de Compra	ITEMER05	1.234	922,84	36
Gestão de Fornecedores	ITEMER06	472	361,05	25
Operação de Venda	ITEMER07	912	563,23	8
Abertura e Fechamento do Caixa	ITEMER08	323	217,88	13
Emissão de Nota Fiscal	ITEMER09	304	166,16	8

Figura 40: Tela de Consulta para o software Merci.

Através dos controles seletivos “Eixo Y”, “Área” e “Cor” os gerentes podem alterar as métricas selecionadas, criando novas visões. Há a possibilidade de gerar três formatos distintos alterando os controles seletivos das métricas.

1) Determinando as métricas:

- Eixo Y = “Complexidade”;
- Área = “Tamanho” e
- Cor = “Pontos por Função”.

O gráfico ficará como mostra a Figura 41.

2) Determinando as métricas:

- Eixo Y = “Tamanho”;
- Área = “Pontos por Função” e
- Cor = “Complexidade”.

O gráfico ficará como mostra a Figura 42.

3) Determinando as métricas:

- Eixo Y = “Pontos por Função”;
- Área = “Complexidade” e
- Cor = “Tamanho”.

O gráfico ficará como mostra a Figura 43.

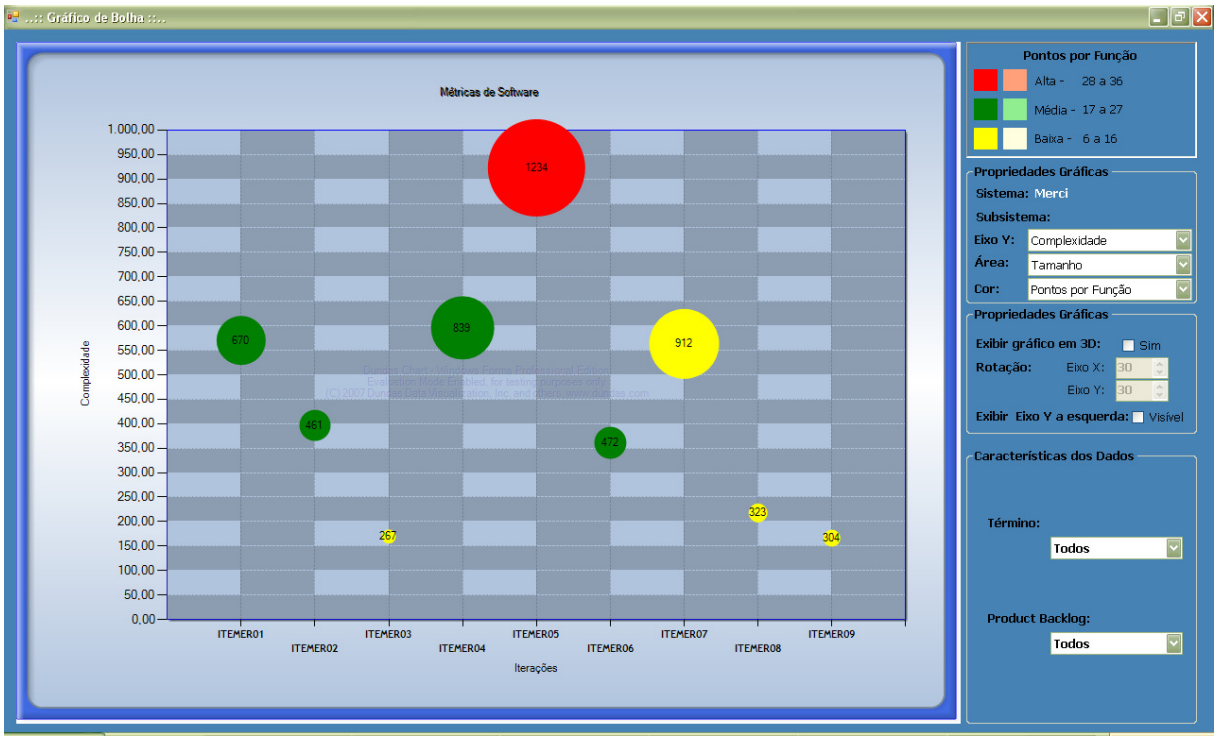


Figura 41: Visão: eixo y, área e cor representam complexidade de código, tamanho e pontos por função respectivamente.

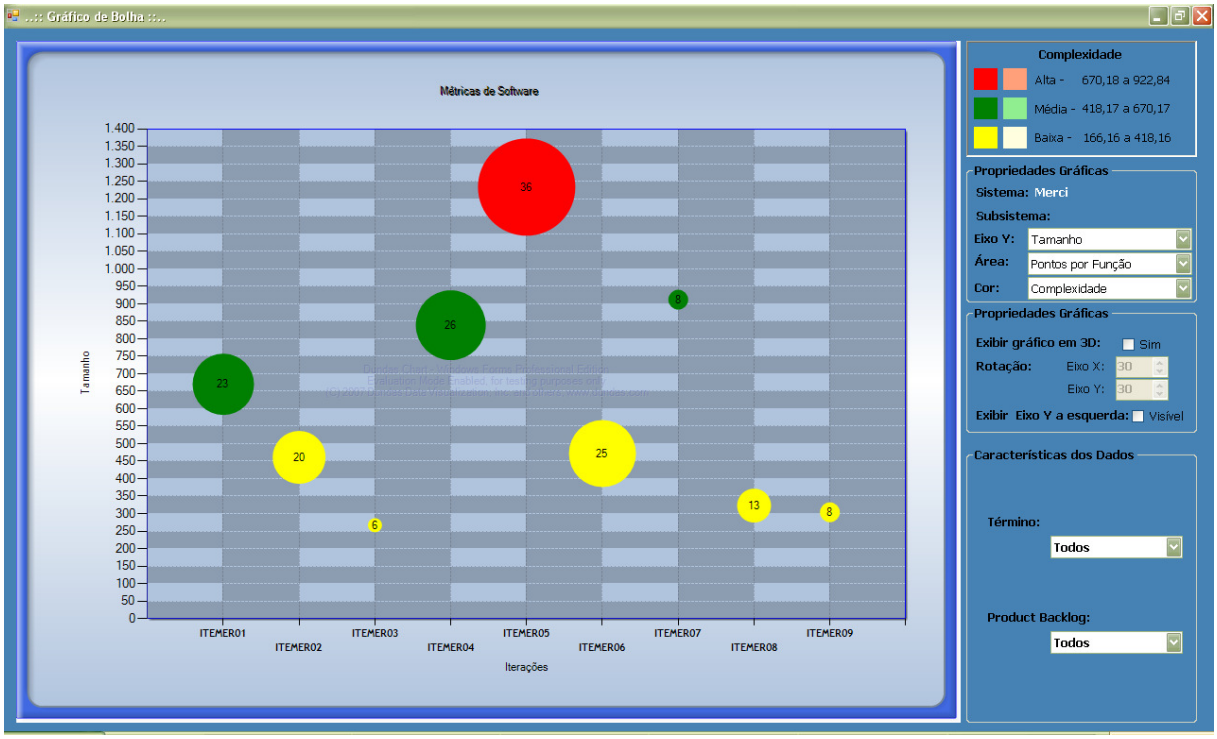


Figura 42: Visão: eixo y, área e cor representam
tamanho, pontos por função e complexidade de código respectivamente.

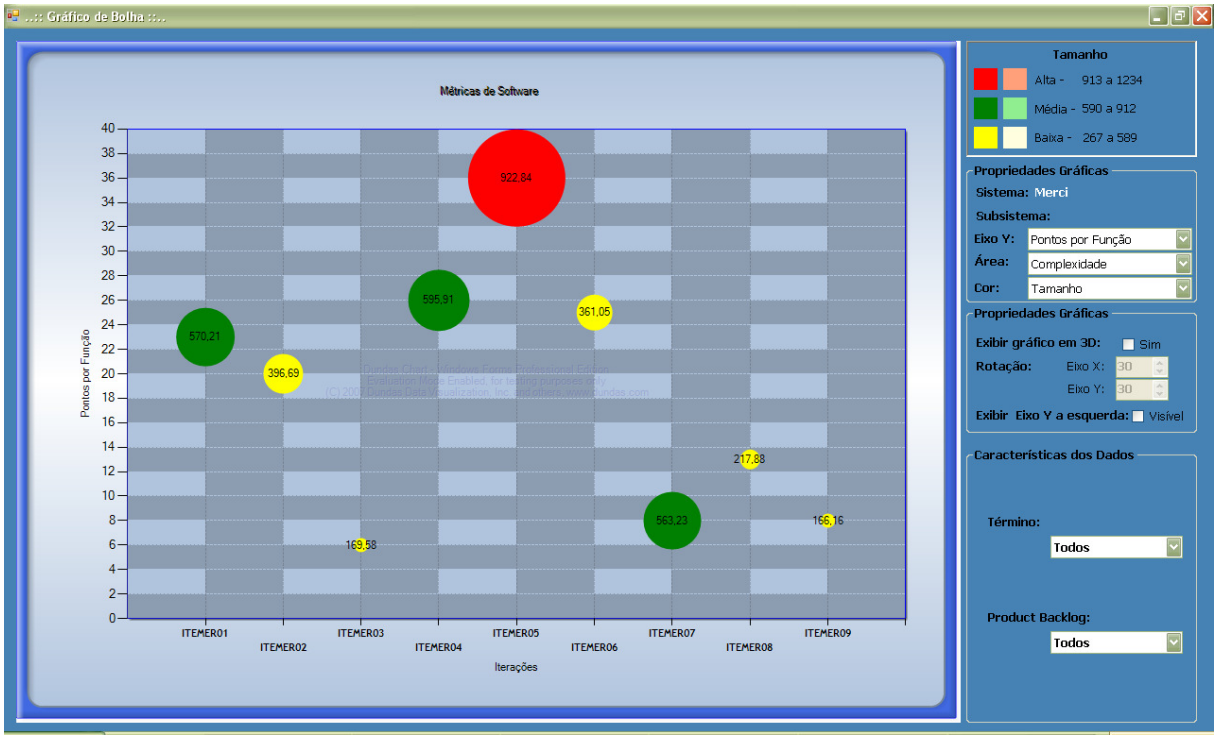


Figura 43: Visão: eixo y, área e cor representam
pontos por função, complexidade de código e tamanho respectivamente.

1) Analisando a Figura 41:

As iterações que possuem maior complexidade de código estão mais ao topo do gráfico, os trechos de código mais extensos são os que representam maior área e a complexidade funcional (ponto por função) é alta para as iterações representadas pela bolha vermelha, média para as bolhas verde e baixa para as bolhas amarelas.

2) Analisando a Figura 42:

As iterações que possuem maior tamanho (trechos mais extensos) estão mais ao topo do gráfico, os trechos de código com maior complexidade funcional (pontos por função) são os que representam maior área e a complexidade de código é alta para as iterações representadas pela bolha vermelha, média para as bolhas verde e baixa para as bolhas amarelas.

3) Analisando a Figura 43:

As iterações que possuem maior complexidade funcional (pontos por função) estão mais ao topo do gráfico, os que apresentam maior complexidade de código são os que representam maior área e os trechos de código são mais extensos para as iterações representadas pela bolha vermelha, médio para as bolhas verde e baixo para as bolhas amarelas.

4.2 SISTEMA DE CONTROLE DO CELULAR PRÉ-PAGO BABY

O sistema de controle do celular pré-pago Baby controla todo o sistema de telefonia envolvendo todas as funcionalidades oferecidas por ele, tais como: controle dos créditos disponíveis e chamadas realizadas e recebidas. Esse sistema possui 15 casos de uso, apresentado na Figura 44. O desenvolvimento desse caso de uso foi criado por profissionais da Embrapa Informática Agropecuária como material do curso “Modelagem de Requisitos Utilizando Casos de Uso” (Chaim, 2002). Em Carniello (2003) os casos de testes para o sistema foram gerados conforme a abordagem de Heumann (2008). E, finalmente em Rocha (2005) as métricas UUSP e UUSPF já estavam calculadas. Foi utilizado o valor não ajustado para as métricas, pois, esse independe dos fatores ambientais e técnicos.

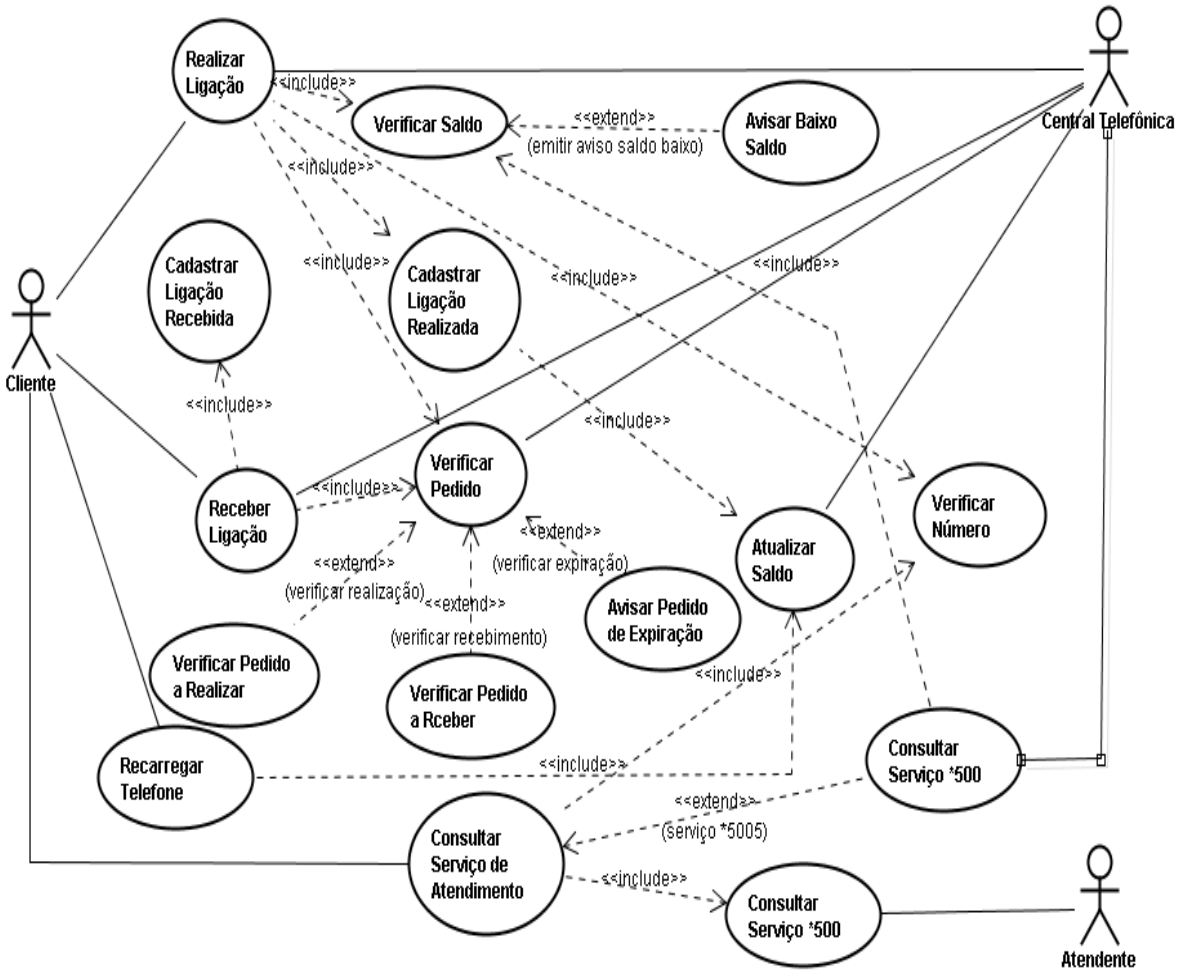
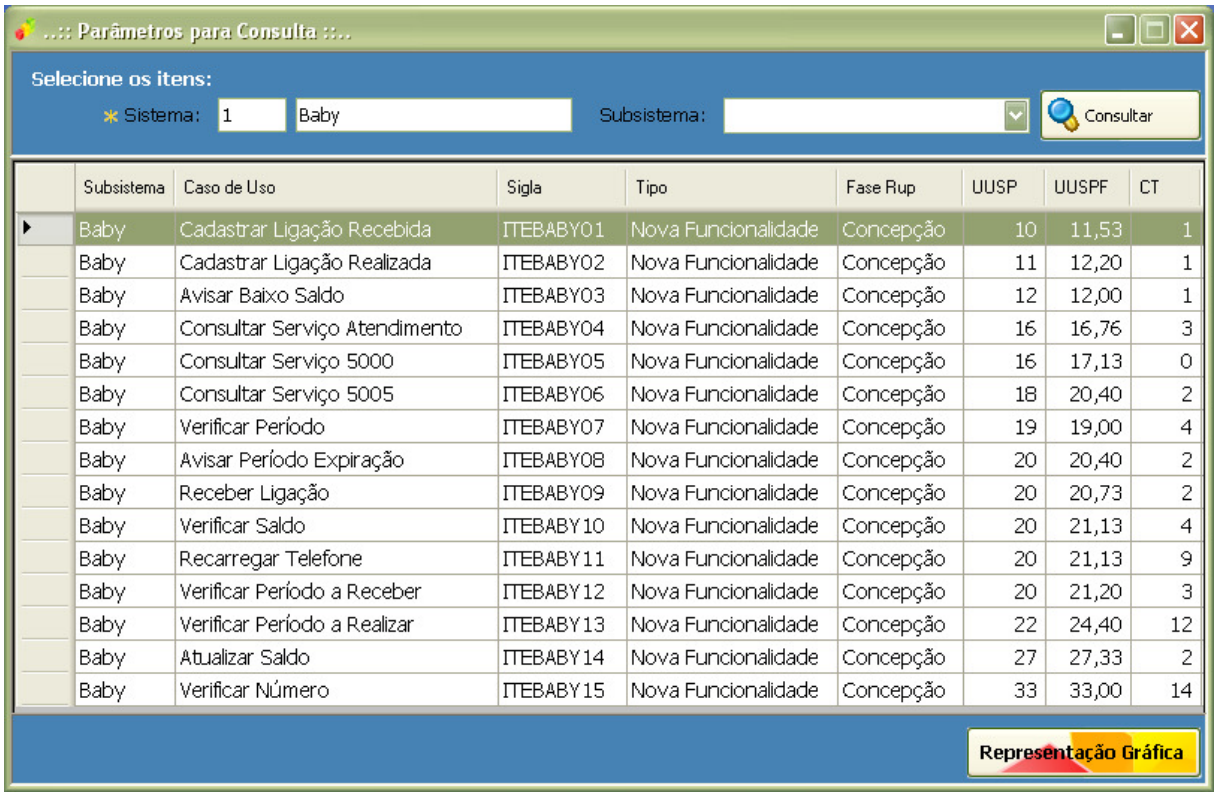


Figura 44: Diagrama de Caso de Uso do Sistema Baby.

Tabela 17: Dados Tabulados para o sistema *Baby*

Caso de Uso	Sigla	UUSP	UUSPF	Caso de Teste (CT)
Cadastrar Ligação Recebida	UCBABY01	10	11,53	1
Cadastrar Ligação Realizada	UCBABY02	11	12,20	1
Avisar Baixo Saldo	UCBABY03	12	12,00	1
Consultar Serviço Atendimento	UCBABY04	16	16,76	3
Consultar Serviço 5000	UCBABY05	16	17,13	0
Consultar Serviço 5005	UCBABY06	18	20,40	2
Verificar Período	UCBABY07	19	19,00	4
Avisar Período Expiração	UCBABY08	20	20,40	2
Receber Ligação	UCBABY09	20	20,73	2
Verificar Saldo	UCBABY10	20	21,13	4
Recarregar Telefone	UCBABY11	20	21,13	9
Verificar Período a Receber	UCBABY12	20	21,20	3
Verificar Período a Realizar	UCBABY13	22	24,40	12
Atualizar Saldo	UCBABY14	27	27,33	2
Verificar Número	UCBABY15	33	33,00	14

Os dados da Tabela 17 (caso de uso e as métricas: UUSP, UUSPF e CT) foram cadastrados no TestPlan possibilitam que uma consulta fosse realizada e todos os casos de uso fossem selecionados. Obtendo desta forma um conjunto de dados já tabulados. A Figura 45 exibe os dados obtidos pela consulta.



Subsistema	Caso de Uso	Sigla	Tipo	Fase Rup	UUSP	UUSPF	CT
Baby	Cadastrar Ligação Recebida	ITEBABY01	Nova Funcionalidade	Concepção	10	11,53	1
Baby	Cadastrar Ligação Realizada	ITEBABY02	Nova Funcionalidade	Concepção	11	12,20	1
Baby	Avisar Baixo Saldo	ITEBABY03	Nova Funcionalidade	Concepção	12	12,00	1
Baby	Consultar Serviço Atendimento	ITEBABY04	Nova Funcionalidade	Concepção	16	16,76	3
Baby	Consultar Serviço 5000	ITEBABY05	Nova Funcionalidade	Concepção	16	17,13	0
Baby	Consultar Serviço 5005	ITEBABY06	Nova Funcionalidade	Concepção	18	20,40	2
Baby	Verificar Período	ITEBABY07	Nova Funcionalidade	Concepção	19	19,00	4
Baby	Avisar Período Expiração	ITEBABY08	Nova Funcionalidade	Concepção	20	20,40	2
Baby	Receber Ligação	ITEBABY09	Nova Funcionalidade	Concepção	20	20,73	2
Baby	Verificar Saldo	ITEBABY10	Nova Funcionalidade	Concepção	20	21,13	4
Baby	Recarregar Telefone	ITEBABY11	Nova Funcionalidade	Concepção	20	21,13	9
Baby	Verificar Período a Receber	ITEBABY12	Nova Funcionalidade	Concepção	20	21,20	3
Baby	Verificar Período a Realizar	ITEBABY13	Nova Funcionalidade	Concepção	22	24,40	12
Baby	Atualizar Saldo	ITEBABY14	Nova Funcionalidade	Concepção	27	27,33	2
Baby	Verificar Número	ITEBABY15	Nova Funcionalidade	Concepção	33	33,00	14

Figura 45: Tela de Consulta para o sistema Baby.

Através dos controles seletivos “Eixo Y”, “Área” e “Cor” os gerentes podem alterar as métricas selecionadas, criando novas visões. Há a possibilidade de gerar três formatos distintos alterando os controles seletivos das métricas.

1) Determinando as métricas:

   - Eixo Y = “UUSP”;
   - Área = “UUSPF” e
   - Cor = “CT”.

   O gráfico ficará como mostra a Figura 46.

2) Determinando as métricas:

   - Eixo Y = “UUSPF”;
   - Área = “CT” e
   - Cor = “UUSP”.

   O gráfico ficará como mostra a Figura 47.

3) Determinando as métricas:

   - Eixo Y = “CT”;
   - Área = “UUSP” e
   - Cor = “UUSPF”.

   O gráfico ficará como mostra a Figura 48.

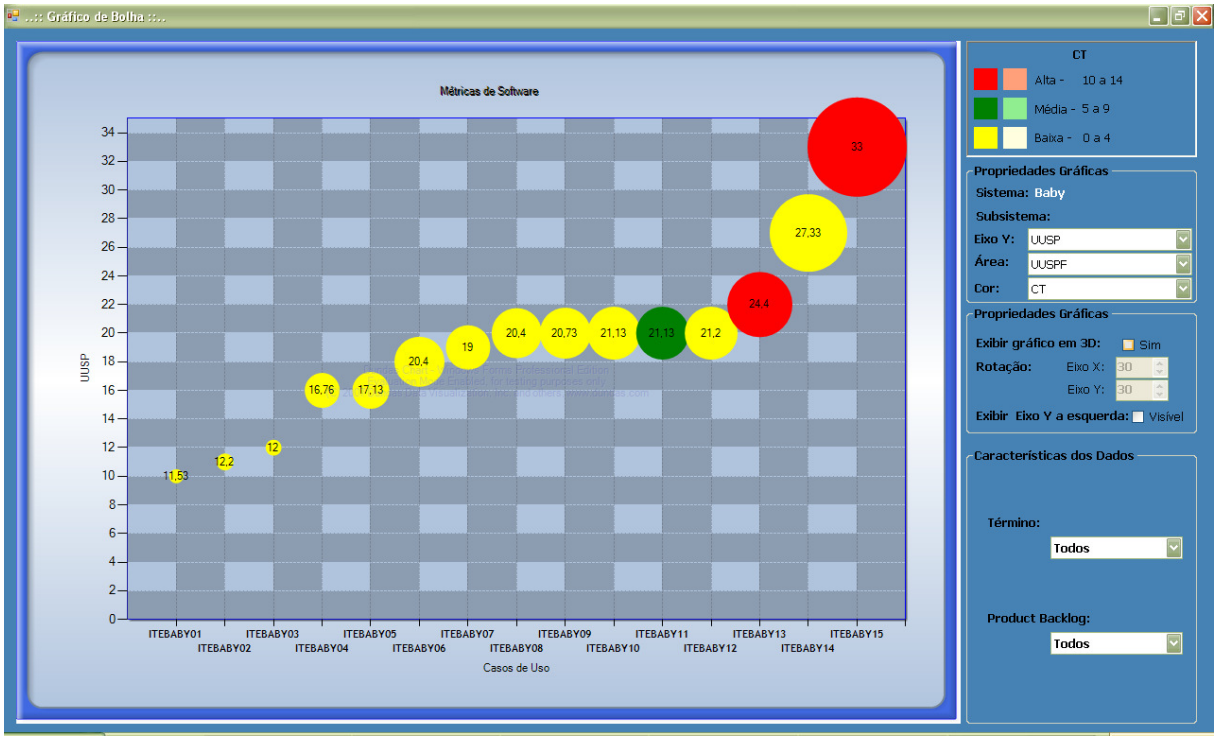


Figura 46: Visão: eixo y, área e cor representam UUSP, UUSPF e CT respectivamente.

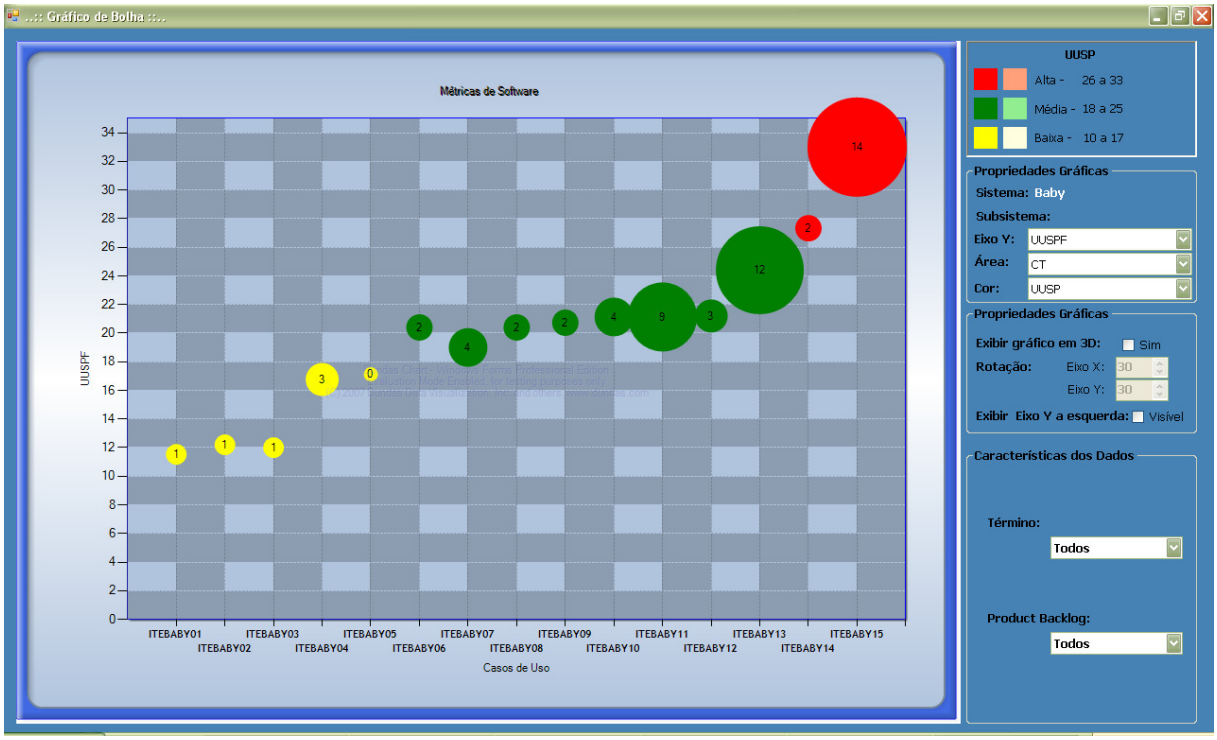


Figura 47: Visão: eixo y, área e cor representam UUSPF, CT e UUSP respectivamente.

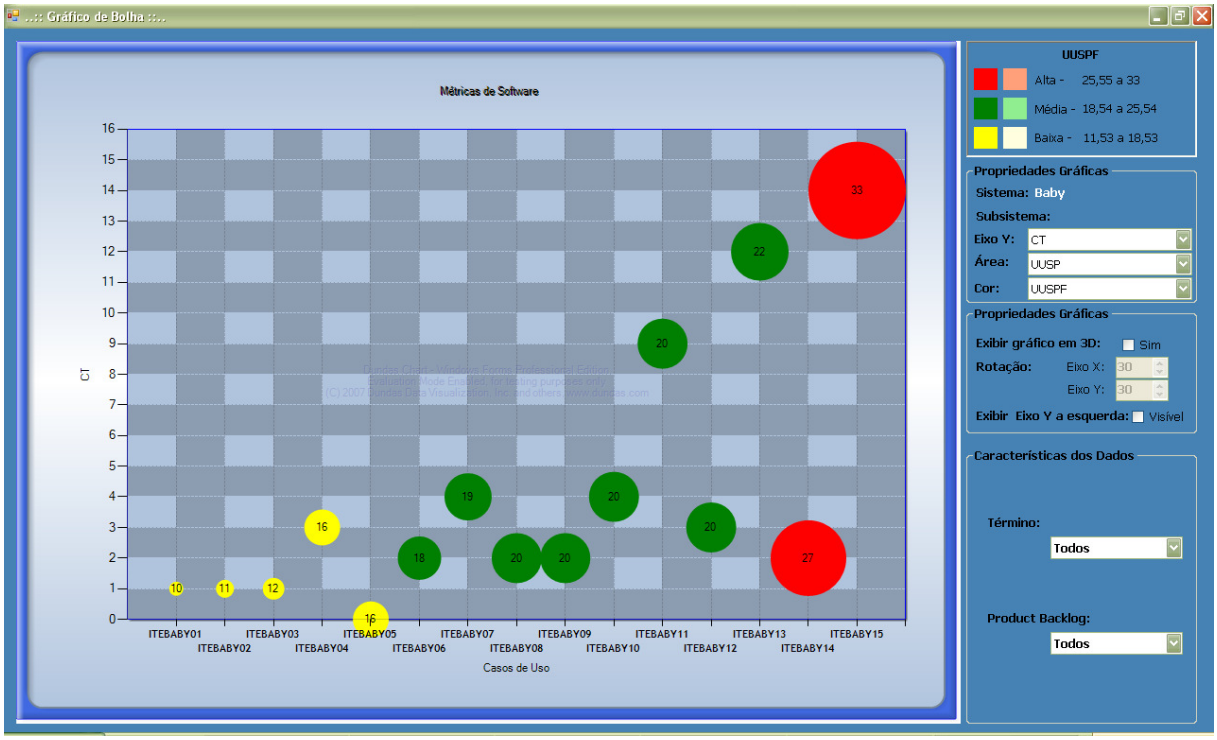


Figura 48: Visão: eixo y, área e cor representam CT, UUSP e UUSPF .

1) Analisando a Figura 46:

Os casos de uso que possuem maior USP estão mais ao topo do gráfico, os com valores maiores de UUPF são os que representam maior área e a quantidade de casos de teste é alta para os casos de uso representados pela bolha vermelha, média para as bolhas verde e baixa para as bolhas amarelas.

2) Analisando a Figura 47:

Os casos de uso que possuem maior UUSPF estão mais ao topo do gráfico, as quantidades maiores de casos de teste são os que representam maior área e a UUSP é alta para os casos de uso representadas pela bolha vermelha, média para as bolhas verde e baixa para as bolhas amarelas.

3) Analisando a Figura 48:

Os casos de uso que possuem quantidade maior de casos de teste estão mais ao topo do gráfico, os que apresentam maior UUSP são os que representam maior área e os valores maiores de UUSPF para os casos de uso representados pela bolha vermelha, médio para as bolhas verde e baixo para as bolhas amarelas.

5 CONCLUSÃO E TRABALHOS FUTUROS

Analisar o processo de planejamento de teste de software por meio de dados apresentados em interfaces textuais é um trabalho que tende a ser dispendioso, por pelo menos dois motivos. O primeiro deles é a coleta dos dados necessários à análise; o segundo, a dificuldade de se entender esses dados em níveis mais abstratos, nos quais se possa depreender relacionamentos, tendências e padrões existentes e relevantes para essa análise.

Tendo em vista o foco desta proposta na fase de planejamento de teste de software, considera-se que a solução apresentada seja inovadora para esta área por aplicar conjuntamente a ela conceitos de visualização de informação e Scrum, aplicação esta da qual não se tem indícios anteriores segundo referências consultadas.

Neste sentido, pesquisas em visualização de informação aplicada ao processo de teste estão mais preocupadas na atividade de execução de testes, como por exemplo, o registro de execuções e de resultados obtidos, diferença entre resultados obtidos e esperados, registros dos defeitos encontrados durante a execução dos testes, utilizando algumas representações gráficas. No contexto de planejamento do teste de software o que se encontra são ferramentas que produzem artefatos, como por exemplo, o Plano de Teste. E, as que fazem priorização das execuções de testes, são de maneira *ad hoc*, sem uma metodologia pré definida. No entanto, como esta contribuição focou-se em atender de análises, na fase de planejamento do teste de software que ainda não possuía, antes desta dissertação, uma solução que possibilitasse aos gerentes de teste especificar, de alguma forma, o foco das análises desejadas ou os dados que se pretendia comparar.

Focando-se em prover este tipo de solução, esta dissertação, utilizou conceitos de visualização de informação e metodologia ágil (Scrum) para propor uma solução que procura ser efetiva em termos de possibilidade de consulta de dados, que se adequa em termos de análise visual de dados e que fornece meios para a priorização dos trechos de códigos a serem testados.

Unindo todos esses conceitos de ambas as áreas, a dissertação apresentou um protótipo que permite aos gerentes de teste definir uma consulta por

meio da escolha de atributos da base de dados, e analisar visualmente os dados por meio de estruturas visuais interativas.

Também com relação a essa interligação de áreas, a escolha de estruturas visuais mais efetivas para a análise dos gerentes de teste, com relação ao um conjunto de dados selecionados, foi possibilitada pela definição de categorias (nominal, ordinal ou quantitativo) para cada atributo da base de dados selecionada. Conhecendo a categoria dos atributos foi possível aplicar conceitos de efetividade de propriedades de marcas de estruturas visuais escolhendo assim, a estrutura visual mais adequada.

Outra questão não menos importante é a atuação humana na modificação das estruturas de dados e das estruturas visuais do processo como parte da interação homem-computador. A disponibilização de controles dispostos na tela, como por exemplo, lista de seleção permitem que o gerente de teste modifique o conjunto de atributos e de dados a serem considerados na representação visual, e até mesmo a forma como o mapeamento visual representa esses dados. Neste trabalho, o gerente de teste pode escolher elementos de uma lista, a exemplo disso são as listas de seleções para modificar as propriedades gráficas: eixo y, área e cor; as listas de seleções de características dos dados: término e *backlog* do produto, podendo citar ainda a possibilidade de definição de intervalos de tempo, a seleção de subsistemas, a seleção dos dados para obter um maior detalhamento dos dados. Essas ações podem afetar quais dados serão exibidos na estrutura visual. Com isso, o gerente de teste pode modificar dinamicamente qual a fatia do conjunto de dados a ser exibida, reduzindo o excesso de dados e auxiliando seu processo de obtenção de informação.

Pela noção geral do processo apresentado nesta dissertação, pode-se considerar que tratar e recuperar informação são atividades que podem ser fortemente auxiliadas por conceitos e técnicas da área de visualização. Integrando os recursos visuais e cognitivos humanos aos recursos gráficos e interativos computacionais, visualização de informação pode ser usada no processo de planejamento do teste de software como uma forma de pessoas interagirem diretamente com um conjunto de dados representativo de uma situação, com o

objetivo de compreender melhor esses dados e de obter visões esclarecedoras sobre a situação estudada.

Contempla-se promissora a linha de pesquisa relacionada à aplicação de visualização de informação na fase de planejamento de teste de software. Do estado da arte dessa linha de pesquisa, depreende-se a existência de uma demanda significativa de pesquisa sobre como apresentar aos gerentes de teste as diferentes situações que acontecem no processo de teste, permitindo a identificação de possíveis problemas nesse ambiente e a tomada de decisão relativa a esses problemas. Usar visualização de informação é uma forma importante de tentar suprir essa demanda, provendo maneiras interativas de facilitar a análise visual dos dados.

Considera-se que a implementação do protótipo atingiu seus objetivos, não sendo ela em si própria a solução proposta pela dissertação, mas sim uma forma de viabilizar que esta solução fosse testada. Assim, pode-se concluir que, de fato, a solução proposta nesta dissertação é válida para auxiliar os gerentes de teste a obter respostas a consultas relevantes nesse domínio, permitindo tanto a elaboração de consultas por meio de atributos da base de dados, quanto a obtenção de respostas a essas consultas por análises visuais e interativas do gráfico gerado pelo protótipo sobre os dados desses atributos. É uma solução promissora, podendo ser adaptada para contemplar novas formas de visualização de dados.

Como complemento a este trabalho que trata especificamente da fase de planejamento de software apoiada pela visualização de informação houve o desenvolvimento em paralelo de outra dissertação (Abreu, 2009) que trata da fase de execução teste também apoiada pela visualização de software para a tomada de decisão por partes dos gerentes de teste.

Enfim, teste de software é uma atividade crítica para a garantia a qualidade do produto gerado pelo desenvolvimento de software. Pesquisas em testes de software têm um importante papel a ser desempenhado no futuro prático da Engenharia de Software. Nesta dissertação, foi apresentada uma técnica que apóia muitas das dificuldades enfrentadas pelos gerentes de teste na fase de

planejamento de teste. A técnica proposta é uma motivação para a falta de resultados que observamos nesta fase do teste de software.

Como trabalho futuro indica-se a validação do TestPlan envolvendo um software empresarial com subsistemas integrados e com o acompanhamento de um gerente de teste experiente. Podendo levantar outras demandas de consultas e de funcionalidades que não tenham sido previstas na implementação do TestPlan nem indicadas durante o estudo de caso realizado.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, Carlos B. *Sistema de Visualização Gráfica para Apoiar a Tomada de Decisão Durante a Fase de Teste*. Dissertação (Mestrado em Ciência da Comupatação) Universidade Metodista de Piracicaba, São Paulo, 2009.

ADAIME, L. M. *Aplicação do Visualization Toolkit para pós-processamento de análises pelo método dos elementos.* Dissertação (Mestrado em Métodos Numéricos em Engenharia) Universidade Federal do Paraná, Curitiba, 2005.

AGILE. *Agile Manifesto* disponível em <http:agilemanifesto.org> acessado em 14 de outubro de 2008.

ALBRECHT, Allan. *Measuring application development preductivity*. In Proc. Of the IBM Applications / development Symposuim. Outubro, 1979.

BARTIÉ, Alexandre. *Garantia da Qualidade de Software: adquirindo maturidade organizacional*. Editora Campus, Rio de Janeiro, 2002.

BASTOS, Anderson et al. *Base de conhecimento em teste de software*. Niterói: Traços e Photo, 2006.

BECKER, Márcio. *Uma Aplicação para Teste de Software Funcional sem Necessidade de Criação de Scripts*. Dissertação, Bacharelado em Ciência da Computação. Santa Cruz do Sul: UNISC, 2003

BOEHM, Barry W. V*erifying and Validating Software Requirements and design Specifications*. IEE Software, vol. 1, nº 1, janeiro de 1984.

BRAZ, Márcio R. Métricas de Software baseadas em caso de uso e teoria Fuzzy. Dissertação de Mestrado em Informática – Setor de Ciências Exatas, Universidade Federal do Paraná, Curitiba, 2004.

BRAZ, Márcio R. e VERGILIO Silva R. *Using Fuzzy Theory for Effort Estimation of Object-Oriented Software*. Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, 2004.

BRAZ, Márcio R. e VERGILIO Silva R. *Software Effort Estimation Based on Use Case*s. Proceedings of the 30th Annual International Computer Software and Applications Conference, 2006.

CHAIM, M. L.; FERREIRA D.; FRANCESCHINI, L. Modelagem de Requisitos Utilizando Casos de Uso. Centro de Pesquisa Embrapa Agropecuária. Campinas - SP, 2002.

CARD, Stuart. K.; MACKINLAY, Jock.; SHNEIDERNAB, Ben. *Readings In Information Visualization: Using Vision To Think.* San Francisco, CA: Morgan Kaufmann Plublishers, 1999.

CARNIELLO, Adriana**.** Teste baseado na estrutura de casos de uso. Dissertação de Mestrado em Engenharia Elétrica - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 2003.

DUNDAS. Dundas Chart versão shareware. Disponível: <http://www.dundas.com/downloads/index.aspx>, 2008.

FAYYAD, Usama M., PIATETSKY-SHAPIRO Gregory, SMYTH Padhraic, UTHURUSAMY Ramasamy. *Advances in Knowledge Discovery and Data Mining*. The MIT Press, 1996.

FREIRE, E. O método de pontos de caso de uso e o cálculo de estimativas. Developer's Magazine. Fevereiro, 2003.

HALSTEAD, M.H.; *Elements of Software Science*, New York, USA, 1977.

HETZEL, William. *Guia Completo ao teste de Software.* Editora Campus, Rio de Janeiro, 1987.

IEEE - *Standard Glossary of Software Engineering Terminology*. Standard 610.12, IEEE press, 1990.

KAN, Stephen. H.; Metrics and Models in Software Quality Engineering. 2nd ed. Addison-Wesley, 2003.

KARNER, Gustav. *Metrics of Objectory.* University of Linkoping, Sweden, 1993.

SCHWABER, K., *Agile Project Management with Scrum*. Microsoft Press, 2004.

KOSARA, Robert; SAHLING, Gerald; HAUSER, Helwing. *Linking scientific and information visualization with interactive 3D scatterplots*. In: International Conference In Central Europe On Computer Graphics, Visualization And Computer Vision Short Communication. 12., 2004, Proceedings... p. 133–140, 2004. *Apud* Rabelo, E; Avaliação De Técnicas De Visualização Para Mineração De Dados. Dissertação de Mestrado. Universidade Estadual de Maringá, 2007.

KRUCHTEN, Philippe. Introdução ao RUP: Rational Unified Process. Ed. Ciência Moderna, Rio de Janeiro – RJ, 2003.

LARMAN, Craig. *Agile and Interactive Development:: A Manager's Guide*. Addison Wesley, 2003.

LARMAN, Craig, *Utilizando UML e Padrões: uma introdução à análise e ao projeto orientados a objetos e ao Processo Unificado*. 2.ed. Porto Alegre: Bookman, 2004.

LUCCA, W. L., CORSO, A. L. ; JINO, M. ; VILELA, P. R. S. ; . Comparison of Size and Complexity Metrics as Predictors of the Number of Faults. In: 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, 2004, Madrid. 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, 2004. p. 1-12.

LUCCA, Waldo Luis de ; C. Fontana ; VILELA, P. ; JINO, M. . Análise da Correlação entre Métricas de Tamanho, Complexidade de Código, Complexidade Funcional e Defeitos de Software. In: VIII Simpósio Internacional de Melhoria de Processos de Software, 2007, São Paulo. VIII Simpósio Internacional de Melhoria de Processos de Software, 2007.

LUZZARDI, Paulo. R. G. *Critérios de avaliação de técnicas de visualização de Informações hierárquicas*. 2003. (Doutorado em Ciências da Computação) – UFRGS, Porto Alegre. 2003.

HEUMANN, Jim. *Generating Test Cases From Use Cases.* Disponível em: <http://therationaledge.com/content/jun_01/m_cases_jh.html>. Acesso em: agosto de 2008.

HOLANDA FERREIRA, Aurélio. B. *Dicionário Aurélio Básico da Lingua Portuguesa*. Editora Nova Fronteira S/A, Rio de Janeiro – RJ, 1988.

MACKINLAY, Jock. D. *Automating the Design of Graphical Presentations of Relational Information*. ACM Transactions on Graphics, 5(2), pp. 110-141, 1986.

MALDONADO, J. C. et. Al. Critérios Pontenciais Usos: Análise da Aplicação de um Benchmark. VI Simpósio Brasileiro de Engenharia de Software. Gramado – Novembro, 1992.

MCGREGOR, J. D; SYKES, D. A., *A pratical guide to testing object – oriented software*. Addison Wesley Longman. Boston, 2001.

MCCONNELL, Steve. *Code Complete*, 2ª Edição, Microsoft Press, Washington, Estados Unidos, 2004.

MICROSOFT. Visual Studio 2005 versão Express. Disponível em <http://www.microsoft.com/downloads>, 2008.

MOLINARI, Leonardo. *Teste de software: Produzindo Sistemas Melhores e Mais Confiáveis*. Editora Érica, São Paulo, 2006.

MYERS, Glenford J. *The Art of Software Testing.* Ed: John Wiley & Sons, John Wiley Professio, 2004.

NASCIMENTO, Hugo. Alexandre. D. e FERREIRA, Cristiane B. R.; *Visualização de Informação – Uma abordagem Prática.* Anais do XXV Congresso da Sociedade Brasileira de Computação. A Universalidade da Computação: Um Agente de Inovação e Conhecimento. UNISIMOS – São Leopoldo – RS. De 22 a 29 de julho de 2005.

OLIVEIRA, Maria Cristina F.; MINGHIM Rosane. *Uma Introdução à Visualização Computacional* – Texto do Curso Jornada de Atualização em Informática (JAI-03), XVII Congresso da SBC, Brasília, Cap.3, pp. 85-131, agosto de 1997.

OLIVEIRA, Maria Cristina F. e LEVKOWITZ, Haim. “From Visual Data Exploration to Visual Data Mininig: A Survey.” IEEE Transactions on Visualization and Computer

Graphics, 9(3): 378-394, 2003.

OMG; *OMG Specifications*. <http://www.omg.org> Acessado em: junlho de 2008.

PÁDUA, Wilson.; *Engenharia de software – Fundamentos, Métodos e Padrões*. 2ª edição. LTC Livros Técnicos e Científicos Editora S.A. Rio de Janeiro, R.J., 2001.

PAULA FILHO, Wilson de Pádua. Engenharia de Software – Fundamentos, Métodos e Padrões". Editora LTC, 2003.

PAULA FILHO, Wilson de Pádua. "Processo Praxis", Editora LTC. Disponível em <http:WWW.wppf.uaivip.com.br/práxis>. Acessado em 23 de setembro de 2008.

PETERS, James F.; PEDRYCZ, Witold. Engenharia de Software – Rio de Janeiro: Campos, 2001.

PFLEEGER, Shari L. *Engenharia de Software – Teoria e Prática*, 2ª edição, Prentice Hall, São Paulo, Brasil, 2004.

PLAISANT, C.; Milash, B.; Rose, A.; Widoff, S.; Shneiderman, B. (1996). LifeLines: Visualizing Personal Histories. *Proceedings of CHI'96*, pp. 221-227.

PRESSMAN, Roger S. *Engenharia de Software*, São Paulo: Makron Books, 2005.

PRITCHARD, Carl L. *Risk management: concepts and guidance*. Arlington: ESI International, 1997.

RYSER, J.; GLINZ, M. *A Pratical Approach to Validating and Test Software System Using Scenarios.* Internatiol Software Quality Week Europe. Sam Francisco, 1999.

RATIONAL; Rational Unified Process; Visão Geral; Disponível em: <http://www.wthreex.com/rup/process/ovu_proc.htm> Acessado em junho, 2008.

ROCHA, Ana Regina C; MALDONADO, José Carlos; WEBER, Kival C.; *Qualidade de Software: Teoria e Prática*. São Paulo: Prentice Hall, 2001.

ROCHA, Cláudio M. Explorando o Relacionamento entre Métricas Baseadas em Caso de Uso e o Número de Casos de Teste. Dissertação de Mestrado – Setor de Ciências Exatas da Universidade Federal do Paraná, Curitiba, 2005.

ROMANI, Luciana. A. S.; *InterMap: Ferramenta para Visualização da Interação em Ambientes de Educação a Distância na Web.* Instituto de Computação Universidade Estadual de Campinas. Dissertação de Mestrado, 2000.

SCRUMNET – Scrumnet.org - grupos de membros de discussão sobre a metodologia Scrum – Disponível em <http://groups.google.co.in/group/scrumnet>. Acessado em 07 de novembro de 2008.

SILVA, Celmar. G. *Exploração de bases de dados de ambientes de Educação a Distância por meio de ferramentas de consulta apoiadas por Visualização de Informação*. Tese de Doutorado. Instituto de Computação, Universidade Estadual de Campinas, 2006. Disponível em: <http://www.las.ic.unicamp.br/~celmar/tese/versao-final/tese-Celmar-versao-final-2007-03-08.pdf>. Acesso em 08 de janeiro de 2007.

SILVA, Celmar. G.; *Considerações sobre o uso de Visualização de Informação no auxílio à gestão de informação*. Anais do XXVII Congresso da SBC. SEMISH XXXIV Seminário de Software e Hardware. Rio de Janeiro - RJ. De 30 de junho a 06 de julho de 2007.

SOMMERVILLE, Ian. *Engenharia de Software*, Addison-Wesley, 2003.

SPENCE, Robert.; *Information Visualization*. Editora Addison – Wesley, 2001.

SIT; Software Testing Institute. <http:// www.softwaretestingintitute.com>. Acessado em 02/05/2007.

VERSIONONE. 3rd Annual Survey: The State of Agile Development" disponível em http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf> acessado em 14 de outubro de 2008.

VILELA, P. R. S.; LUCCA, W. L.; Corso A. Corso, e JINO, M. Comparison of Size and Complexity Metrics as Predictors of the Number of Faults, *Anais da IV Jornadas Iberoamericanas en Ingeniería del Software*, v. I, Madrid - Espanha, 2004.

VOLERE. *Free Requirement Templates*. Disponível em <http://www.volere.co.uk>. Acessado em 23 de maio de 2008.

WARE, Colin. *Information Visualization: Perception for Design*. Morgan-Kaufmann Publishers, 2004.

ANEXO A

1 REQUISITOS FUNCIONAIS E DE DADOS DO PROTÓTIPO

Nesta seção do Documento de Requisitos, será apresentado os requisitos funcionais e de dados do Produto, para tal, utilizaremos o uso de Fichas de Requisitos, segundo Template Volere (Volere, 2008).

1.1 EFETUAR LOGIN

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF001 (Efetuar Login)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Efetuar Login é necessário e primordial para que o protótipo tenha restrição de acesso tanto do usuário-adminstrador (gerente de teste) quanto ao usuário-operador (testadores), cada um terá um nível de acesso ao protótipo, o qual começará a contar com a identificação e senha de entrada. O gerente de teste terá acesso total e irrestrito a todas as áreas do protótipo, quanto aos testadores, terá acesso as áreas as áreas operacionais do protótipo.
Justificativa: Restrição de acesso.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Usu
Nome_Usu
Senha_Usu
ScrumMaster_Usu

1.2 CADASTRAR SISTEMA

Ficha de Especificação de Requisitos
(baseado no *Template Volere*)

Identificação do Requisito: RF002 (Cadastrar Sistema)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Cadastrar Sistema é necessário e primordial para que o protótipo possa dar continuidade a outros cadastros dependentes deste. Para que o cadastro seja feito corretamente, o gerente de teste deverá fornecer a descrição do sistema. O protótipo irá gerar automaticamente um código para o sistema.

Justificativa: Registrar dados dos sistemas para identificar qual o sistema estará em análise.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Sis
Descricao_Sis

1.3 CADASTRAR SUBSISTEMA

Ficha de Especificação de Requisitos
(baseado no *Template Volere***)**

Identificação do Requisito: RF003 (Cadastrar Subsistemas)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito cadastrar Subsistemas tem como objetivo cadastrar as subdivisões do sistema para que na visualização de informações o gerente de teste possa visualizar dados apenas de um determinado subsistema. Não é um requisito no qual é necessário seu preenchimento porque pode haver sistemas que não contam com subdivisões, neste caso, o subsistema deverá ser cadastrado com o mesmo nome do sistema. Para o preenchimento o gerente de teste deverá informar qual o sistema o subsistema pertence e a sua descrição.

Justificativa: Registrar dados dos subsistemas para possibilitar na analise visual dos dados qual subsistema estará em análise.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Sub
CodSitema_Sub
Descricao_Sub

1.4 CADASTRAR CASO DE USO

Ficha de Especificação de Requisitos
(baseado no *Template Volere***)**

Identificação do Requisito: RF004 (Cadastrar Caso de Uso)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Cadastrar Caso de Uso tem como objetivo registrar os casos de uso utilizados no modelo RUP. Cada caso de uso receberá um código que será gerado automaticamente pelo protótipo, cabendo ao gerente de teste fornecer dados do sistema e subsistema que o caso de uso pertence, bem como a sigla de identificação e a descrição do caso de uso.

Justificativa: É necessário cadastrar os casos de uso para que seja possível cadastrar as suas devidas iteração.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Uso
Sigla_Uso
CodSistema_Uso
CodSubsistema_Uso
Descricao_Uso

1.5 CADASTRAR ITERAÇÃO

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF005 (Cadastrar Iteração)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Cadastrar Iteração tem como objetivo registrar as *n* iterações que um caso de uso possa possuir e é importante para a análise sobre o planejamento de dados que serão sobre elas. O protótipo irá gerar o código da iteração automaticamente, cabendo ao gerente de teste o preenchimento da sigla, caso de uso que pertence, o número da iteração que vai de 1 até *n*, a data de liberação para o teste, o programador que a codificou, qual o tipo da iteração (nova funcionalidade, alteração de funcionalidade ou nova tecnologia), qual a fase do RUP que ela está inserida (Concepção, Elaboração, Construção ou Transição), as métricas de tamanho, complexidade, o impacto de um risco vir a ocorrer e o impacto deste risco para o usuário, esses dois últimos dados servem para que seja calculado a métrica de exposição ao risco. O protótipo inicialmente cadastrará a iteração como não concluída.

Justificativa: É necessário para que se possa fazer as visualização das iterações para a tomada de decisão do gerente de teste.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Ite	FaseRup_Ite
Sigla_Ite	Tamanho_Ite
CodCasoUso_Ite	Complexidade_Ite
Numero_Ite	Impacto_Ite
Data_Ite	Propabiidade_Ite
CodProgramador_Ite	Concluido_Ite
Tipo_Ite	

1.6 CADASTRAR PROGRAMADORES

Ficha de Especificação de Requisitos
(baseado no *Template Volere***)**

Identificação do Requisito: RF006 (Cadastrar Programadores)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Cadastrar Programadores é necessário para que possa registrar que codificou um caso de uso. Para tal, o gerente de teste deverá informar o nome do programador e selecionar a o seu nível de senioridade: sênior, intermediário e iniciante. O protótipo irá gerar automaticamente um código para cada registro de um programador.

Justificativa: Verificar a experiência do programador na codificação em uma iteração.

Solicitante:
Prioridade:
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Pro
Nome_Pro
Experiencia_Pro

1.7 CADASTRAR TESTADORES

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF007 (Cadastrar Testadores)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Cadastrar Testadores é necessário para registrar os testadores que farão parte do processo de teste. O protótipo irá gerar um código automático para cada registro de testadores, cabendo apenas ao gerente de teste preencher o nome do testador.

Justificativa: É necessário para identificar quem será responsável para testar uma iteração.

Solicitante:
Prioridade:
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Tes
Nome_Tes

1.8 CADASTRAR USUÁRIO

Ficha de Especificação de Requisitos
(baseado no *Template Volere***)**

Identificação do Requisito: RF008 (Cadastrar Sistema)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Cadastrar Sistema é necessário e primordial para que o protótipo possa dar continuidade a outros cadastros dependentes deste. Para que o cadastro seja feito corretamente, o gerente de teste deverá fornecer a descrição do sistema. O protótipo irá gerar automaticamente um código para o sistema.

Justificativa: Registrar dados dos sistemas para identificar a qual o sistema estará em análise.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Sis
Descricao_Sis

1.9 CADASTRAR SCRUM MASTER

Ficha de Especificação de Requisitos
(baseado no *Template Volere*)

Identificação do Requisito: RF009 (Cadastrar Scrum Master)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Cadastrar Scrum Master é necessário e primordial para que o protótipo possa dar continuidade a outros cadastros dependentes deste. Para que o cadastro seja feito corretamente, o gerente de teste deverá fornecer o nome do Scrum Master. O protótipo irá gerar automaticamente um código para ele.

Justificativa: Registrar Scrum Master para a definição do Scrum Team.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Mas
Nome_Mas

1.10 CADASTRAR SCRUM TEAM

Ficha de Especificação de Requisitos
(baseado no *Template Volere***)**

Identificação do Requisito: RF010 (Cadastrar Scrum Team)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Cadastrar Scrum Team é necessário e primordial para que o protótipo possa dar continuidade a outros cadastros dependentes deste. Para que o cadastro seja feito corretamente, o gerente de teste deverá fornecer o nome do Scrum Máster, uma descrição para a equipe, e registrar os membros pertencentes ao time. O protótipo irá gerar automaticamente um código para a equipe.

Justificativa: Registrar Scrum Team para a definição do Sprint.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Mas
Nome_Mas
Código_Mem
Nome_Mem

1.11 PESQUISAR DADOS PARA ANÁLISE VISUAL

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF011 (Pesquisar dados para análise visual)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito pesquisar dados para análise visual tem como objetivo buscar dados no banco de dados mediante parâmetros digitados pelo gerente de teste que são: qual o sistema que será apresentado, se desejar o gerente de teste poderá selecionar o subsistema a ser apresentado e o período desejado.

Justificativa: Buscar dados para a apresentação visual.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Ite	FaseRup_Ite
Sigla_Ite	Tamanho_Ite
CodCasoUso_Ite	Complexidade_Ite
Numero_Ite	Impacto_Ite
Data_Ite	Propabiidade_Ite
CodProgramador_Ite	Concluido_Ite
Tipo_Ite	

1.12 GERAR REPRESENTAÇÃO GRÁFICA

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF012 (Gerar Representação Gráfica)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s): Pesquisar dados para análise visual

Descrição: O requisito gerar representação gráfica tem como objetivo gerar um gráfico de bolhas a partir dos dados pesquisados anteriormente pelo gerente de teste para que com sua representação gráfica o gerente de teste possa analisá-lo e tomar decisões para o planejamento do teste de software. Deve possibilitar a alteração das métricas associadas ao eixo y, a cor e a área da bolha. Dando ao usuário a possibilidade de visualizar os dados das iterações de formas distintas. Como por exemplo, o eixo y pode assumir o valor quantitativo da métrica de complexidade, de tamanho ou exposição ao risco e, assim, para a cor e área também. Outra característica que o gráfico deve possuir é de filtrar dados, dando assim, a possibilidade do usuário visualizar as iterações que estão concluídas e as não concluídas e também as que já foram ou não adicionadas a lista de backlog do produto.

Justificativa: Gerar gráfico de bolha para a tomada de decisão do gerente de teste no planejamento do teste de software.
Solicitante:
Prioridade:
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Codigo_Ite	Propabiidade_Ite
Sigla_Ite	Concluido_Ite
CodCasoUso_Ite	Experiencia_Pro
Numero_Ite	Codigo_Uso
Data_Ite	Sigla_Uso
CodProgramador_Ite	Descricao_Uso
Tipo_Ite	Descricao_Sub
FaseRup_Ite	Descricao_Sis
Tamanho_Ite	Codigo_Age
Complexidade_Ite	Impacto_Ite

1.13 VISUALIZAR DADOS DA ITERAÇÃO

Ficha de Especificação de Requisitos
(baseado no *Template Volere*)

Identificação do Requisito: RF013 (Visualizar dados da Iteração)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito visualizar dados da iteração tem como objetivo exibir todos os dados sobre a iteração selecionada pelo gerente de teste no gráfico de bolhas. Os dados são: descrição do caso de uso, sigla do caso de uso, código do caso de uso, código da iteração, sigla da iteração, número da iteração, data de liberação para teste da iteração, fase do RUP que a iteração se apresenta, tipo da iteração, métrica da complexidade da iteração, métrica de tamanho da iteração, probabilidade de um risco vir a ocorrer com a iteração, o impacto de um risco vir a ocorrer com a iteração, exposição ao risco da iteração, experiência do programador que codificou a iteração e se a iteração está concluída ou não.

Justificativa: Exibir dados importantes sobre a iteração que está sendo analisada.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)
Dados:

Sigla_Ite	Impacto_Ite
CodCasoUso_Ite	Propabiidade_Ite
Numero_Ite	Concluido_Ite
Data_Ite	Experiencia_Pro
CodProgramador_Ite	Codigo_Uso
Tipo_Ite	Sigla_Uso
FaseRup_Ite	Descricao_Uso
Tamanho_Ite	Descricao_Sub
Complexidade_Ite	Descricao_Sis

1.14 CADASTRAR BACKLOG DO PRODUTO

Ficha de Especificação de Requisitos
(baseado no *Template Volere***)**

Identificação do Requisito: RF014 (Cadastrar o Backlog do Produto)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito cadastrar o backlog do produto tem como objetivo adicionar a iteração a uma lista priorizada na ordem que o teste deverá ser executado. Para isso, é preciso informar o tempo estimado para a realização do teste. O sistema irá gerar um número seqüencial para a prioridade e irá automaticamente cadastrar o estado correne da atividade como não iniciada. Uma atividade pode estar nos seguintes estados: não iniciada, em andamento, paralisada ou concluída. O protótipo irá gerar automaticamente um código para o backlog do produto.

Justificativa: Gerar uma lista priorizada de atividades para o teste de software.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)
Dados:
Código_Bac
CodIteracao_Bac
Prioridade_Bac
Estimativa_Bac
Estado_Bac
Adicionado_Bac
Executado_Bac

1.15 REORDENAR BACKLOG DO PRODUTO

Ficha de Especificação de Requisitos
(baseado no *Template Volere*)

Identificação do Requisito: RF015 (Reordenar o Backlog do Produto)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito reordenar o backlog do produto tem como objetivo dar a possibilidade ao usuário de alterar a prioridade das iterações adicionadas na lista que serão posteriormente testadas. Os dados que deverão ser exibidos são: prioridade, estimativa, estado, sigla da iteração, número da iteração, sigla do caso de uso e descrição do caso de uso.

Justificativa: Alterar a ordem da lista priorizada de atividades para o teste de software.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:
Código_Bac
CodIteracao_Bac
Prioridade_Bac
Estimativa_Bac
Estado_Bac
Sigla_Ite
Numero_Ite
Sigla_Uso
Descrição_Uso

1.16 CADASTRO DE SPRINT

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF016 (Cadastro de Sprint)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito cadastrar sprint tem como objetivo a edição de informações iniciais do sprint, essas informações são: descrição do sprint, a equipe que faz parte do sprint, a duração em horas e a data de início.

Justificativa: Configurar informações iniciais do projeto.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Código_Spr
Descricao_Spr
DataInicio_Spr
Horas_Spr
CodScrumTeam_Spr
CodSistema_Spr

1.17 ADICIONAR ATIVIDADES DO BACKLOG DO PRODUTO AO SPRINT

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF017 (Adicionar atividades do Backlog do Produto ao Sprint)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito Adicionar atividades do Backlog do Produto ao Sprint tem como objetivo definir as atividades a serem executadas em um determinado sprint e para quem será atribuída a responsabilidade por sua realização.

Justificativa: Configurar informações iniciais do projeto.

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:
Código_Ati
CodSprint_Ati
CodBacklog_Ati
CodTestador_Ati

1.18 MANUTENÇÃO DO BACKLOG DO SPRINT

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF018 (Manutenção do Backlog do Sprint)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito manutenção do backlog do sprint tem como objetivo atualizar o andamento das atividades inseridas em um sprint. Será permitido alterar o estado da atividade e o tempo de execução da atividade, a qual será acumulativa conforme seu andamento. Deverá ser permitido filtrar por sprint, descrição de caso de uso e responsável.

Justificativa: Atualizar as informações das atividades do backlog do sprint

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:

Código_Ati
Sigla_Ite
Descricao_Ite
Sigla_Uso
Descricao_Uso
Numero_Ite
Tipo_Ite
Estimativa_Bac
Estado_Bac
Executado_Bac
Nome_Tes

1.19 GERAR GRÁFICO BURNDOWN

Ficha de Especificação de Requisitos

(baseado no *Template Volere***)**

Identificação do Requisito: RF019 (Gerar Gráfico BurnDown)
Tipo de Requisito: Funcional
Caso(s) de Uso(s) vinculado(s):

Descrição: O requisito gerar gráfico de burndown tem como objetivo exibir para o usuário para que o mesmo acompanhe o andamento das horas restantes do sprint.

Justificativa: Visualizar o gráfico de horas restantes do sprint

Solicitante:
Prioridade: 3 (alta)
Material de Apoio: Diagrama de Caso de Uso (DCU-Base)
Diagrama de Classes (DC-Base)

Histórico: Solicitação Inicial (/ /2008)
Solicitação de Mudança (/ /2008)

Dados:
Código_Ati
Estimativa_Bac
Estado_Bac
Executado_Bac