“Um Modelo Sistêmico para Atividade de Avaliação e Testes de Software”

Eduardo de Vasconcelos Silva

Trabalho Final de Mestrado Profissional em Computação

DE BC
AMADA TIUNICAMP
Si38m
EX
BC/ 71485
16.145-07
D X
11,00
26-02-07
401202

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Miriam Cristina Alves – CRB8a / 859

Silva, Eduardo de Vasconcelos
Si38m Um modelo sistêmico para a atividade de avaliação e testes de software / Eduardo de Vasconcelos Silva -- Campinas, [S.P. :s.n.], 2006.

Orientadora: Ana Cervigni Guerra
Co-orientador: Rogério Drumond

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Software – Testes. 2. Software - Avaliação. 3. Teoria bayesiana de decisão estatística. 4. Engenharia de software. I. Guerra, Ana Cervigni. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: A systematic model for evaluation activity and software testing.

Palavras-chave em inglês (Keywords): 1.Software - Testing. 2. Software – Evaluation. 3. Bayesian theory of statistical decision. 4. Software engineering.

Área de concentração: Engenharia de Software.

Titulação: Mestre Profissional em Computação

Banca examinadora: Prof. Dr. Rogério Drumond (IC-UNICAMP)
Profa. Dra. Silvia Maria Fonseca Silveira Massruhá (CNPTIA-Embrapa)
Prof. Dr. Mário Lúcio Côrtes (IC-UNICAMP)

Data da defesa: 24/02/2006

Programa de Pós-Graduação: Mestrado Profissional em Computação

INSTITUTO DE COMPUTAÇÃO

UNIVERSIDADE ESTADUAL DE CAMPINAS

“Um Modelo Sistêmico para Atividade de Avaliação e Testes de Software”

Eduardo de Vasconcelos Silva

Fevereiro de 2006

Banca Examinadora:

- **Prof. Dra. Ana Cervigni Guerra (Orientadora)**
  Centro de Pesquisa Renato Archer – CenPRA/MCT
- **Prof. Dr. Rogério Drumond (Co-orientador)**
  Instituto de Computação - Unicamp
- **Profa. Dra . Silvia Maria Fonseca Silveira Massruhá**
  CNPTIA/Embrapa
- **Prof. Dr. Mário Lúcio Côrtes**
  Instituto de Computação - Unicamp

“Um Modelo Sistêmico para Atividade de Avaliação e Testes de Software”

Este exemplar corresponde à redação final do Trabalho Final devidamente corrigido e defendido por Eduardo de Vasconcelos Silva e aprovado pela Banca examinadora.

Campinas, fevereiro de 2006

Profa. Dra. Ana Cervigni Guerra
(Orientadora)

Prof. Dr. Rogério Drumond
(co-orientador)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito Parcial para a obtenção do título de Mestre em Computação na área de Engenharia de Computação.

TERMO DE APROVAÇÃO

Trabalho Final Escrito defendido e aprovado em 24 de fevereiro de 2006, pela Banca Examinadora composta pelos Professores Doutores:

Profa. Dra. Sílvia Maria Fonseca Silveira Massruha
CNPTIA/ Embrapa

Prof. Dr. Mário Lúcio Côrtes
IC - UNICAMP

Profa. Dra. Ana Cervigni Guerra
CenPRA/ MCT

© Eduardo de Vasconcelos Silva, 2006
Todos os direitos reservados

A minha esposa Lucia,
Guilherme meu filho,
e Alfredo, muito mais
que meu irmão.

Agradecimentos

A minha esposa Lucia pelo amor e compreensão das horas, noites e fins de semana que foram necessárias a esse trabalho.

A meu irmão Alfredo que mesmo distante fez-se muito presente com seu apoio e amor imensuráveis.

Ao pequeno Guilherme, meu filho, que mesmo sentido a redução do nosso tempo juntos foi um gigante com seu afeto.

A minha orientadora Ana Guerra, elemento fundamental e motivador, sua influência foi um diferencial.

A Motorola, em especial Alice, por todo o suporte que me foi dado.

Resumo

A atividade da escrita de casos de testes para software é em sua essência não determinística e por conseguinte envolve riscos. Por outro lado, o grau de densidade de testes dos requisitos do sistema sofre influência da forma que esses requisitos são interpretados.

Uma proposta para sistematizar o processo de criação de casos de teste encontra-se no uso de uma rede Bayesiana que modela a arquitetura de testes a ser implementada, aliada a uma ponderação estatística de riscos a cenários. A rede é muito adequada a problemas não determinísticos e que envolvam riscos.

Aliada ao uso da rede Bayesiana, essa sistematização engloba o critério de adequação, cujo objetivo é minimizar o impacto da cobertura de requisitos. Cada requisito do sistema é interpretado segundo critérios pré definidos.

Como produto final desta proposta sistêmica, além do ferramental gráfico que possibilita a descrição dos casos de testes segundo uma sequência lógica e simulação de cenários, têm-se uma matriz que reúne todos os casos de testes obtidos da rede e demais oriundos da análise dos requisitos, segundo o enfoque do critério de adequação.

O estudo experimental sinaliza um incremento de quatro vezes e meio na densidade de testes de requisitos comparativamente a técnica tradicional. Na fase de análise há uma tendência de redução de esforço em torno de um quarto.

Um interessante resultado dessa técnica sistematizada, está na identificação de cenários não previstos pelos requisitos o que vem agregar na atualização da documentação de *design*.

Abstract

The task of writing software use cases is in essence non deterministic and therefore involves risks. On the other hand, the coverage level of requirements depends the way of document requirements are interpreted.

In order to bring a systematic approach in developing software use cases, the Bayesian network technique helps supporting with this problem. The current software test architecture is modeled in a graphical way, adding scenario simulation and risk statistic. The network addresses very well risks and non deterministic scenarios.

Along with the Bayesian Network , the proposed systematic approach encompasses the adequacy criteria which main goal is to improve requirements coverage. Each system requirement is interpreted according to well defined criteria.

A matrix will be the final product, as a result from the current systematic approach. All use cases obtained from the network and those discovered thanks of adequacy criteria can be seen together. In addition this, the technique allows scenarios simulation.

The experimental results show an increase of four times and half in requirements coverage, as compared against the traditional technique. In the analysis phase , there is a trend of reducing the effort by a factor of 4.

One interesting result using this technique is the amount of new scenarios identified. Most of them are not properly described in the requirement document. This brings a powerful tool to add value in updating the design document.

Índice

Índice de Figuras

Índice de Tabela

Siglas

MGB	Modelo Gráfico Bayesiano
AS	Ação de Software
UML	*Unified Model Language*
TC	Caso de Teste
Req	Requisito
BVA	*Bondary Value Analysis*

Capítulo I - Introdução

1.1 Panorama Atual

Com a evolução da tecnologia, o software tem sido requisitado em quase todas as áreas da atividade humana. Esta crescente dependência contribui para aumentar a complexidade dos sistemas e para conscientizar tanto os usuários no sentido de, cada vez mais, exigirem software confiáveis, quanto a indústria de software no sentido de desenvolver, cada vez mais, produtos de qualidade.

Com objetivo de se tornarem mais competitivas, algumas empresas de software vêm implantando a gerência sistemática dos processos utilizados para o desenvolvimento e a manutenção de software. Por meio da avaliação de seus produtos e da melhoria dos seus processos, estas empresas têm obtido a necessária melhoria da qualidade de seus produtos e melhores resultados nos negócios.

O desenvolvimento de produtos de software com boa qualidade ainda é complexo e caro. No momento que a sociedade depende cada vez mais de software, os problemas históricos associados com seu desenvolvimento ainda não foram adequadamente solucionados.

Embora a tarefa de validação de programas de computador seja tão antiga quanto o primeiro programa produzido, estórias de erros nos sistemas em operação ainda são comuns. Ao longo destes anos, surgiram várias técnicas que ajudaram a melhorar o nível de qualidade dos sistemas desenvolvidos. Entretanto, tal melhoria não conseguiu acompanhar o aumento da complexidade dos sistemas de software.

Na medida em que o emprego de sistemas de software cresceu ao ponto em que boa parte de nossa vida depende cada vez mais de software e computadores, passa a ser de vital importância a existência de software confiável – software que fornece resultados corretos quando alimentado com dados válidos e que identifica corretamente dados inválidos. Principalmente quando se trata de aplicações onde um simples defeito pode causar um grande prejuízo ou simplesmente uma catástrofe, tais como software de aplicações espaciais, software de controle de processos na área médica, controle de processos em usinas nucleares.

Algumas empresas consideram os defeitos de software existentes nos sistemas desenvolvidos como um fato normal e inevitável. Mesmo que não sejam catastróficas, as conseqüências de defeitos são desagradáveis quando acontecem. Porém, as empresas desenvolvedoras de software contam com a parceria de seus clientes, dispostos até mesmo a pagar uma razoável taxa anual para terem o direito a receber novas versões dos programas que corrigem os defeitos que não deveriam existir[Cappovila99].

Entretanto, esta situação esta mudando. As empresas desenvolvedoras de software descobriram que produtos com defeitos custam caro – muitas vezes, muito caro. Por exemplo a American Arlines e a United Arlines estimam uma perda em torno de 20 mil dólares por minuto cada vez que os respectivos sistemas de reserva de passagem deixam de funcionar. Mesmo em casos menos drásticos, é comum que as empresas de software percam clientes insatisfeitos com a quantidade de defeitos existentes em seus produtos.

Além de problemas econômicos, existem problemas mais graves na medida em que vidas humanas passam a depender com mais freqüência de sistemas de software.

No entanto, se as empresas que desenvolvem software compreendem a importância de se produzir sistemas com baixo nível de defeitos bem como a importância do processo para atingir este objetivo. Mesmo assim os sistemas atuais ainda contêm um alto índice de defeitos. A validação de software não é uma atividade trivial. A atividade de teste exige conhecimentos, habilidades, e infra-estrutura específica. Um bom desenvolvedor ou projetista de software sem esta base dificilmente realizaria uma boa tarefa de testes.

Outro elemento que influencia a qualidade dos testes de software é o esforço que as empresas produtoras de software normalmente gastam em tarefas rotineiras (principalmente corrigindo defeitos!) suprimindo tempo e esforço necessários para identificar os procedimentos, técnicas e ferramentas de software para essas atividades, tão importantes.

No Brasil, a realidade das práticas de testes de software em uso pelas empresas foi retratada na última pesquisa “Qualidade no Setor de Software Brasileiro” realizada pelo MCT/SEPIN em 2001[MCT 2001]. Segundo os resultados, apurados somente 56% das 589 empresas que responderam o questionário realizavam testes funcionais dos produtos desenvolvidos. Os resultados não foram muito diferentes no caso de testes de aceitação – realizados somente por 48% das empresas. Apenas 24% das empresas entrevistadas realizavam o teste de unidade. A pesquisa não traz resultados sobre os procedimentos e nem sobre as técnicas de teste empregadas

nas empresas. Desta forma, das empresas que aplicam teste, não é possível avaliar o procedimento e nem o conhecimento sobre as técnicas utilizadas.

Esta dissertação será desenvolvida, buscando trazer uma técnica sistemática a atividade de testes de software.

1.2 Objetivo

Essa dissertação tem como objetivo sugerir uma sistemática alternativa aos testes de software , trazendo uma maior racionalidade na criação e execução dos casos de testes.

Será utilizado uma rede bayesiana para o modelamento do problema e uma matriz que reunirá todos os casos de testes criados., essas ferramentas tendem a buscar cenários afetados além dos limites do documento de requisitos. Isso assegura ao desenvolvedor de testes, segurança e domínio da corrente implementação, tendo como resultado imediato, possíveis cenários não detalhados no documento de requisitos e sua correta atualização.

O processo de testes é encarado dentro de um contexto não determinístico, como sendo um gerenciamento de risco [Woo02], logo um cuidado estatístico especial será utilizado para o modelamento do problema.

Outro ponto importante, dentro do modelamento proposto, é o uso intensivo do engenheiro de testes, cuja experiência é incorporada ao modelo através da ponderação adequada dos fatores que influenciam o resultado final.

O trabalho tem de maneira implícita a melhoria da qualidade do produto de software colaborando para a criação de produtos mais robustos e confiáveis, através da melhoria do processo, o que indubitavelmente representa vantagens financeiras interessantes.

Os dados experimentais coletados são iniciais, estando o projeto em implantação. Futuras coletas de métricas devem ser levantadas quando em regime, de forma a validar os números preliminares.

1.3 Justificativa

É indiscutível o impacto econômico oriundo da deficiência da qualidade de software. Estudos mostram que o custo despendido na manutenção em campo pode chegar a 90% do total investido no projeto [Pig97].

Outro ponto de grande importância é a necessidade de um método sistêmico que se agregue ao processo de desenvolvimento de testes, mas especificamente aqueles baseados exclusivamente em requisitos funcionais, onde ferramentas estatísticas de simulação de cenários bem como critérios de adequação [Zhu96] irão suprir uma deficiência efetiva.

1.4 Escopo e Limitações

O processo de desenvolvimento de testes de software baseados em requisitos funcionais é a linha mestra que norteia esse trabalho. Objetiva agregar uma sistematização de forma a permitir simulações estatísticas, melhorar a cobertura e reduzir duplicações de casos de testes sobre o processo corrente e dessa forma aumentar a contenção de defeitos e a redução no custo do produto.

O grau de detalhamento do caso de teste está fora do escopo desse trabalho, mas certamente sofre influência do grau de conhecimento da pessoa envolvida no trabalho de executar os testes, ou se a execução será manual ou automática.

Este trabalho não pretende esgotar o assunto, nem mesmo ditar regras de processos a serem seguidos, seu objetivo é fornecer uma opção prática a ser seguida.

A classe de testes de caixa branca bem como testes unitários e de regressão estão fora do escopo desse trabalho.

1.5 Estrutura da Dissertação

No Capítulo 2 são abordados conceitos sobre testes de software como testes caixa preta, caixa branca, testes unitários, de regressão, que são base para o desenvolvimento da dissertação.

No capítulo 3 é dedicado ao conceito de modelo gráfico Bayesiano aplicado no modelamento de casos de testes e simulação. Será sugerida uma ferramenta que possibilita a simulação de forma computacional. Também é abordado o conceito de critério de adequação,

outro elemento básico para a formulação do trabalho, que reúne elementos como caso de teste, especificação funcional, e software dentro de um equacionamento baseado em critérios.

O Capítulo 4 propõe a unificação da rede Bayesiana que modela o domínio de entrada com o critério de adequação , sugerindo uma sistematização no processo de criação de testes. O produto final desta sistematização é uma planilha que mapeia todos os casos de testes, visando melhorar a cobertura e eliminando duplicações.

O Capítulo 5, se concentra uma análise quantitativa de quatro projetos de testes em andamento, focando no esforço de análise e cobertura e mostra a implementação prática de um *design* de testes.

Nas considerações finas são levantadas as conclusões e oportunidades de melhoria.

Capítulo II – Conceitos em Testes de Software

Os conceitos aqui apresentados são largamente utilizados em testes de software e são importantes para a compreensão e desenvolvimento do trabalho. O objetivo é trazer as definições mais comumente encontradas em trabalhos acadêmicos e de pesquisa.

2.1 Definindo Teste

Teste é uma forma de verificação dinâmica que consiste em executar o programa com um conjunto de dados de entrada e determinar se ele se comporta conforme o esperado, isto é, de acordo com sua especificação. Em geral, é impossível testar um programa exaustivamente; o importante é selecionar um conjunto finito de casos de testes que permitam testá-lo adequadamente [Mar00].

Pode-se afirmar que um teste bem-sucedido é aquele que descobre um erro ainda não descoberto, logo um bom caso de testes é aquele que tem alta probabilidade de encontrar um novo erro [Mye79].

Conceitos de erro, de falha e de defeito, são utilizados segundo o IEEE Std. Glossary of Software Engineering Terminology, padrão 610.12/1990, definidos como:

- erro: engano cometido por um desenvolvedor (analista, projetista, programador);
- falha: manifestação do erro (uma especificação ou código incorretos);
- defeito: evento notável ao usuário, ativação da falha.

2.2 Resultados

Segundo Pressman, se erros graves que necessitam da alteração do projeto forem encontrados com regularidade, a qualidade do software e confiabilidade são suspeitos e testes adicionais são indicados. Se por outro lado, o software aparenta estar funcionando adequadamente e os erros encontrados são fáceis de serem corrigidos, pode se concluir uma das duas alternativas: (1) a qualidade e confiabilidade são aceitáveis ou (2) os testes são inadequados para descobrir erros graves! Finalmente, se os testes não descobrirem erros, existem poucas

dúvidas de que a configuração de testes não esteja adequada e de que os erros estão escondidos no software [Oli03].

2.3 Importância dos Testes

Segundo Som[92], a atividade de testes de um programa tem dois objetivos: o primeiro é comprovar que o programa está de acordo com sua especificação e o segundo é exercitar o programa de tal maneira que os erros fiquem expostos .

É errado afirmar que um software não contém nenhum erro, pois com certeza eles estarão lá. A execução de testes consegue verificar se o software está funcionando de acordo com sua especificação, porém se o projeto for especificado de forma errada e construído de acordo com esta especificação, o teste não conseguirá reverter esta situação, ou seja, o processo de testes não irá influenciar na qualidade do software [Oli03].

A análise e projeto do software podem ser vistas como tarefas construtivas, sob o ponto de vista psicológico. Porém, a atividade de testes, sob o ponto de vista do desenvolvedor, pode ser considerada destrutiva, pois os casos de teste visam encontrar erros, ou seja, "demolir" o software que está sendo desenvolvido. A fim de evitar o conflito de interesses recomenda-se que o desenvolvedor seja responsável por testar as unidades individuais (módulos) do programa, se certificando que cada uma execute a função para a qual foi projetada. Em muitos casos, o desenvolvedor pode também executar o teste de integração, que é a etapa de teste dedicada ao teste da estrutura completa do programa.

Somente depois que a arquitetura do programa foi totalmente testada é que uma equipe de testes independente deve ser envolvida. O desenvolvedor e a equipe de testes trabalham próximos, enquanto os testes são conduzidos pela equipe de testes, o desenvolvedor deve estar disponível para corrigir os erros que estão sendo descobertos [Oli03].

2.4 Fases do Teste

Existem diversas fases de teste, cada fase devendo ser preparada ao longo do desenvolvimento. Em geral, o que se sugere é um processo de desenvolvimento em "V", mostrado na Figura 2-1, englobando tanto as fases de desenvolvimento, quanto às fases de testes [Mar00].

O processo de testes inicia-se com os testes de unidade, 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 essa fase de testes de integração. O objetivo dos testes de integração é encontrar defeitos de interfaceamento entre os módulos e os subsistemas. Os testes de validação visam determinar se o software satisfaz aos requisitos especificados na fase de análise. E, finalmente, os testes de sistema visam exercitar o sistema como um todo, incorporando todos os componentes (hardware e software) para determinar se o sistema completo satisfaz à sua especificação [Mar00].

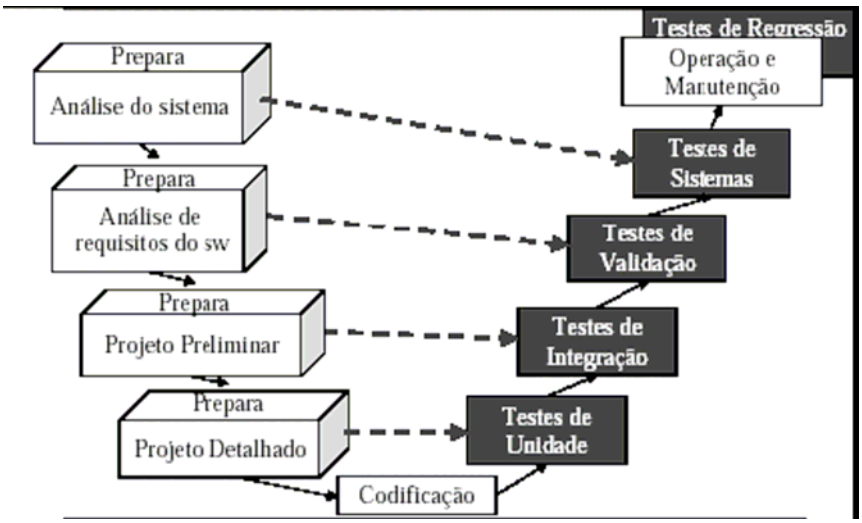


Figura 2-1- Processo de Desenvolvimento em "V" [Mar00]

2.5 Classes de Testes

Podemos classificar os testes baseados em duas distintas abordagens segundo [Pre02]:

- Teste caixa branca: teste baseado na implementação, a seleção do teste é baseada na informação obtida a partir do código do programa. Assim sendo, esses métodos são também chamados de teste caixa branca ou caixa de vidro ou ainda teste estrutural, significando que a estrutura da implementação é considerada para a escolha dos testes [Fid03].

- teste caixa preta: visam verificar a funcionalidade e a aderência aos requisitos em uma ótica externa ou do usuário, sem se basear em qualquer conhecimento do código e da lógica interna do componente testado [Rio03].

O foco dessa dissertação se restringe a essa última classe de testes, logo um maior detalhamento será dedicado na próxima seção.

2.6 Testes Caixa Preta

Como os testes caixa preta baseiam-se nas especificações, uma especificação de boa qualidade que seja testável faz-se necessária. Para isso, a especificação deve ser não ambígua (cada termo tem uma e somente uma definição), consistente (cada termo é usado somente de uma forma) e completa (todas as informações necessárias e suficientes para os testes foram definidas). Com isso, fica mais fácil identificar com precisão os dados de entrada e os resultados esperados [Mar00].

Os testes caixa preta tentam encontrar erros das seguintes categorias [Pre02]:

- funções incorretas ou omitidas;
- erros de interface;
- erros de estrutura de dados ou de acesso à base de dados externa;
- erros de comportamento ou desempenho;
- erros de iniciação e término.

2.6.1 Técnicas de Testes de Caixa Preta

2.6.1.1 Particionamento de equivalência

A partição de equivalência é uma técnica que determina quais classes de dados de entrada tem propriedades comuns. O programa deve se comportar da mesma maneira para todos os elementos de determinada partição de equivalência. Existem partições de equivalência para entradas e saídas.

A partição de equivalência pode ser identificada pela especificação do programa ou documentação do usuário e pela experiência do engenheiro de testes em prever quais classes de entradas de dados são possíveis de detectar erros. Como exemplo, se a especificação de uma entrada diz que algum valor de entrada deve ser um inteiro de 5 dígitos, isto é, entre 10.000 e

99.999, a partição de equivalência pode ser valores menores que 10.000, valores entre 10.000 e 99.999, e valores maiores que 99.999.

A técnica de partição de equivalência é útil para seleção de instâncias de cada possível entrada para teste. Entretanto, mesmo quando um programa funciona adequadamente para entradas individuais de testes, combinações destas entradas podem detectar erros no programa. A partição de equivalência não ajuda na seleção destas combinações. Entretanto, testadores experientes costumam desenvolver seu próprio conjunto de combinações que possam descobrir o erro [Som92].

2.6.1.2 Análise de valor limite (BVA - *Bondary Value Analysis*)

Esta técnica leva a escolha de casos de teste que põem à prova os valores fronteiriços, onde costumam ocorrer o maior número de erros. A análise do valor limite completa a técnica de particionamento de equivalência. A BVA também deriva casos de testes do domínio de saída. Devem ser exercitados os valores máximos e mínimos, os valores logo abaixo deles e logo acima respectivamente, relatórios de saída que produzam o número máximo de entradas permitidas numa tabela, casos de teste que exercitem o número máximo de entradas de um *array*, entre outros [Oli03].

2.6.1.3 Teste de recuperação

São testes que provocam defeitos no software (defeitos no disco, defeitos de interface, memória insuficiente) a fim de avaliar se a recuperação do sistema é adequadamente executada.

Se a recuperação for automática (realizada pelo próprio sistema), a reinicialização, recuperação de dados e reinicio são avaliados. Se a recuperação exigir intervenção humana, o tempo médio até o reparo também é avaliado para determinar se ele se encontra dentro de limites aceitáveis. É desejável que o sistema seja capaz de se recuperar rapidamente e com um mínimo de intervenção humana [Int01].

2.6.1.4 Teste de estresse

O teste de estresse avalia como o software se comporta perante volumes anormais, desta forma, o sistema é carregado com volumes não usuais com intenção de pará-lo. Neste momento é monitorada a perda de desempenho do sistema e a sua suscetibilidade à falhas durante estas cargas. Se o sistema pára como resultado de uma alta carga, este deverá passar por mais alguns

testes de recuperação. O engenheiro de testes deve utilizar o sistema com recursos em quantidade, freqüência e volume anormais, tais como procuras excessivas de dados em disco, aumento excessivo de índice de dados, abertura de muitas janelas e utilização de arquivos com formato não compatível aos esperados pelo programa, entre outros [Int01].

2.6.1.5 Teste de desempenho

Para sistemas de tempo real, um software que ofereça as funções exigidas, mas não atinja os requisitos de desempenho, é inaceitável. O teste de desempenho é, às vezes, combinado com teste de estresse e freqüentemente exige instrumentação de hardware e de software, ou seja, muitas vezes é necessário medir a utilização dos recursos rigorosamente.

O teste de desempenho envolve o monitoramento e o registro dos níveis de desempenho durante cargas de estresse regular, baixas e altas. Ele testa o conjunto de recursos utilizados a cerca das condições descritas e serve como base para se efetuar uma previsão dos recursos adicionais necessários no futuro. [Int01]

2.7 Estratégias de Testes

2.7.1 Estratégias *Top Down*

A estratégia *top-down* testa o alto nível do sistema antes de testar seus componentes detalhadamente. O programa é representado por um componente abstrato com subcomponentes representados por *stubs*. Os programas *stubs* podem ser implementados como uma versão simplificada de um componente requerido. O *stub* simplesmente solicita que seja introduzido um valor adequado ou simula a ação de um componente. Depois que os componentes de alto nível são testados, seus sub-componentes são implementados e testados da mesma forma. Este processo continua recursivamente até que os componentes de baixo nível sejam implementados. A vantagem desta estratégia é que o sistema funcionando fica disponível num estágio inicial de desenvolvimento. O processo de validação pode começar antes. Os erros de projeto podem ser detectados num estágio inicial do processo de testes. A detecção dos erros mais cedo significa evitar que a fase de implementação, e eventualmente a de projeto, tenham que ser refeitas. Quanto às desvantagens, se o componente é muito complicado, se torna impraticável produzir um *stub* que o simule perfeitamente. Esta estratégia não é apropriada para sistemas orientados a objetos, pois coleções de objetos não são usualmente integrados numa severa forma hierárquica [Oli03].

2.7.2 Estratégias *Bottom Up*

A estratégia *bottom-up* é o contrário da *top-down*. O teste começa com os módulos de baixo nível na hierarquia e vai subindo na hierarquia até que o módulo final seja testado. As vantagens da estratégia *bottom-up* são as desvantagens da *top-down* e vice-versa.

Quando o teste *bottom-up* é utilizado, "*test drivers*" devem ser desenvolvidos a fim de exercitar os componentes de baixo nível. Estes "*test drivers*" simulam o ambiente do componente sendo testado.

Erros na arquitetura costumam ser descobertos depois que muito do sistema já foi testado. As correções destes defeitos podem envolver escrever e testar novamente os módulos de baixo nível do sistema. Devido a este problema, a estratégia de testes *bottom-up* foi criticada pelos proponentes do desenvolvimento *top-down*. Porém, o teste *bottom-up* dos componentes críticos de baixo nível do sistema, é sempre necessário [Oli03].

2.7.3 Manutenção casos de Testes

Depois de várias liberações de versões de software, um conjunto de casos de testes pode se tornar muito grande. Sendo assim, alguns casos de teste podem se tornar obsoletos e devem ser removidos. Muitos testadores são relutantes em descartar casos de testes com medo de estarem jogando fora casos que poderiam revelar alguma falha. Contudo, os seguintes tipos de casos de teste podem ser (ou devem ser) descartados [BIN99]:

- casos de testes "inadequados": casos de testes para um componente ou interface que foram removidos ou alterados; logo, os resultados esperados não serão iguais aos resultados obtidos, daí a necessidade de descartar esses casos de testes;
- casos de testes obsoletos: os casos de testes não se aplicam mais devido a mudanças nos requisitos (regra de negócio);
- casos de testes incontroláveis: os casos de testes são sensíveis a dados de entrada e estados que não estão sob controle. Esses tipos de testes devem ser relegados ao status de "teste não efetivo", no qual o teste "passa" se a entrada não provoca uma exceção ou término anormal do software sob teste.

2.7.4 Testes de Regressão

Testes de Regressão são necessários a cada vez que modificações são feitas no software, e consistem na reexecução de um subconjunto dos casos de testes já aplicados para determinar se as alterações não produziram nenhum efeito indesejado [MAR00].

Uma importante diferença entre testes de regressão e testes durante o desenvolvimento é o fato de que durante os testes de regressão um conjunto de testes pré-existente deve estar disponível para poder ser reutilizado.

Uma abordagem para testes de regressão consiste em retestar tudo, selecionando todos os casos de testes do conjunto já existente. Essa estratégia pode consumir tempo e recursos excessivos. Uma abordagem alternativa é o reteste seletivo, o qual se baseia na seleção de um subconjunto de testes a partir do já existente, usando esse subconjunto para testar o programa modificado [Fid03].

2.7.5 Aplicações para Testes de Regressão

Os testes de regressão devem ser aplicados nas seguintes situações [MAR01]:

- aplicações críticas que devem ser retestadas freqüentemente;
- produto de software que é alterado constantemente durante o desenvolvimento (por exemplo, processo incremental);
- averiguar se componentes reutilizáveis são adequados para o novo sistema;
- durante os testes de integração;
- durante os testes, após as correções;
- na fase de manutenção (corretiva, adaptativa, evolutiva ou preventiva);
- na identificação de diferenças no comportamento do sistema quando há mudanças de plataforma.

2.7.6 Limitações Testes de Regressão

Dentre as limitações dos testes de regressão temos [HAR94]:

- uma seqüência de testes que pode ser usada como seqüência de regressão deixa de ser útil como seqüência de testes primária;

- uma seqüência de regressão não tem as mesmas metas de cobertura de uma seqüência de testes primária;
- uso de seqüência de testes inadequada como seqüência de regressão não melhora sua qualidade.

A dependência de uma especificação de software de boa qualidade é extremamente sensível ao desenvolvimento de testes de caixa preta. A técnica bayesiana proposta tende a buscar cenários não vislumbrados na documentação, fornecendo um conjunto de casos de testes mais adequado a validação do sistema.

Capítulo III – Estudo Rede Bayesiana e Critérios de Adequação

Neste capítulo serão apresentados dois conceitos fundamentais para o desenvolvimento do trabalho. Critério de adequação, que em síntese, define um relacionamento entre especificação, casos de teste e programa e rede Bayesiana que proverá um ferramental para o modelamento do teste de software, possibilitando inclusive simulações estatísticas.

3.1 Critérios de Adequação

Um dos conceitos extremamente valiosos para a elaboração teórica do presente trabalho. Sua implementação prática será apresentado na forma de uma matriz, reunindo os casos de testes e um conjunto selecionado de critérios de adequação.

Um importante resultado no uso dos critérios de adequação será a identificação de cenários não documentados nos requisitos, contribuindo na cobertura dos requisitos..

3.1.1 Definição

Segundo [Zhu96], critérios de adequação são regras que definem se um trecho de software foi adequadamente testado. Um vasto número de critérios de adequação tem sido propostos e investigados, podendo ser citado os seguintes:

- Critério baseado em fluxo de controle;
- Critério baseado em fluxo de dado;
- Critério baseado em texto de programa;
- Critério baseado em falhas.

Todos esses critérios são baseados em código de software (caixa branca). O enfoque deste trabalho deve concentrar em critérios que estejam relacionados com testes caixa preta, ou seja, baseados em especificações.

Nesse sentido, o critério de adequação pode ser interpretado como uma função que associa um conjunto de testes, dedicados a testar um código de software, contra uma determinada especificação.

Expondo esse conceito num formato de equação:

Um critério de adequação C é uma função definida por:

C: P x S x T -> {verdadeiro, falso}.

Onde:
P: conjunto dos programas (software) a serem testados
S: conjunto das especificações
T: conjunto dos casos de testes

C(p,s,t) = verdadeiro, implica que segundo o critério de adequação C, o teste t é adequado para testar o programa p segundo a especificação s.

São pertinentes as seguintes definições relativas a critérios de adequação [Zhu96]:

Definição 1: sejam C1 e C2 dois critérios de adequação , tais que $C1 \supseteq C2$, ou seja o critério C1 contém C2, se para todo programa p sob teste , toda especificação s e todo conjunto de teste t , se t é adequado segundo C1 , implica que t também será adequado segundo C2.

Definição 2: sejam C1 e C2 dois critérios de adequação tais que $C1 \supseteq C2$, ou seja o critério C1 contém C2, define-se uma função f que mapeia (p,s,t) em um número real R que representa a quantidade de defeitos encontrados. Se para todos os conjuntos de testes x,y , programas p e especificação s, tal que x contém y , implica que $f(x,p,s) \geq f(y,p,s)$.

De forma concisa, temos que se o critério de adequação C garante a detecção de pelo menos um defeito no software, tem-se que qualquer critério de adequação que contém C também garantirá a retenção de pelo menos um defeito [Zhu96].

Um emprego prático das definições 1 e 2 estão numa sistematização de cobertura de testes, formalizando critérios que determinam quando os casos de testes desenvolvidos são suficientes e atendem a especificação disponível.

3.2 Redes Bayesianas (MGB)

O segundo conceito base para o presente trabalho garante sustentação na descrição gráfica do problema de software, possibilitando o mapeamento de áreas de riscos e consolidando uma boa compreensão da arquitetura de testes.

Definição

Na formação de uma rede bayesiana, cada nó representa uma variável em estudo ,contendo a distribuição probabilística dos valores que essa variável poderá assumir. . Note que a variável em questão poderá ser um escalar discreto, contínuo ou lógico.

Um arco direcional com uma seta conectando dois nós, sinaliza uma dependência do nó origem (pai) , para com o nó filho destino (ponta da seta). Essa dependência pode ser interpretada como sendo o nó pai, causador do nó filho. Com essa construção obtém-se uma hierarquia na rede.

A teoria de MGB (modelo gráfico Bayesiano) vem sendo utilizada em aplicações que necessitam de um modelamento para as incertezas envolvidadas, em problemas complexos onde uma grande quantidade de fatores contribui para o resultado final. Relações entre os fatores são capturadas mesmo sendo providos de alto grau de incertezas ou mesmo imprecisos [Woo02].

3.2.1 Exemplo Clássico

O clássico exemplo do uso de MGB está no domínio médico. Cada novo paciente tipicamente corresponde a um novo caso e o problema é o diagnóstico (encontrar certezas para as não mensuráveis variáveis da doença) e predizer o que deverá acontecer baseado nas variáveis observáveis (sintomas). O médico é a pessoa experiente que define a rede e fornece os iniciais relacionamentos entre as variáveis (possivelmente na forma de probabilidades), baseado na experiência e treinamento com casos semelhantes. Desta forma a rede poderá ser calibrada utilizando dados estatísticos anteriores e futuros.

. Exemplificando, supondo que a variável de interesse seja temperatura, podendo assumir os valores fria, morna e quente. Logo associando probabilidades para cada estado segundo algum critério, temos:

Fria – 10%

Morna – 60%

Quente – 30%

MGB tem seu fundamento da metodologia estatística Bayesiana, o que é caracterizado por fornecer um modelo formal que combinam dados e avaliações de peritos, especificamente no caso de software, os engenheiros de software.

3.2.2 Conceituação de Transação

Pode-se definir transação como sendo o cenário principal que será modelado através de uma rede bayesiana... Existe uma estreita analogia entre o conceito de transação e casos de usos empregado no modelo UML. Desta forma, antes de iniciar o modelamento, faz-se necessário definir de forma clara qual transação será explorada. Uma rede bayesiana detalha as funcionalidades inclusas em cada transação selecionada.

3.2.3 Definição de Ação de Software (AS)

Ação de software é definida como o código responsável por um particular processamento e possível de ser especificamente testado. Dentro de uma transação poderá existir uma grande quantidade de AS. São identificadas três relações entre os AS, tomemos dois AS, A e B (se A e B formam partes de duas transações ou ocorrendo em pontos diferentes dentro da mesma transação) [Woo02]:

- Comum: A e B são comuns se estão contidos dentro de um mesmo pedaço de código, desta forma um conjunto de testes objetivando verificar A, necessariamente também testará B, pois como premissa ambos estão contidos no mesmo pedaço de software;
- Relacionados: A e B são relacionados se há a expectativas que os testes de A podem trazer informações sobre o grau de estabilidade de B. AS podem ser relacionadas por vários motivos entre eles: variantes de um mesmo pedaço de código, códigos distintos, mas utilizando o mesmo algorítmo ou mesmo o código ter sido desenvolvido pelo mesmo fornecedor.
- Independentes: A e B são independentes se os testes aplicados em A não trazem informações sobre o grau de estabilidade de B.

3.2.4 MGB em Testes de Software

O emprego do MGB possibilita uma avaliação estatística da estabilidade do software antes e durante o processo de testes. Essa avaliação possibilita uma forma natural para o desenvolvimento de testes. O modelo poderá ser utilizado tanto para a geração do conjunto de testes quanto auxiliar na decisão de como seqüênciar um conjunto de testes e verificar se há falta de testes para possíveis cenários esquecidos [Woo02].

3.2.5 Estruturando o Modelo

Baseado nas considerações de [Woo02] que considera que o sistema de software deve ser estruturado antes do modelamento. Muitas vezes essa estruturação pode ser obtida da própria especificação funcional. É necessário ter uma clara idéia de como as diferentes funcionalidades se interagem, quais são as ações de software (AS), quais são as possíveis entradas.

Como ponto de partida todas as transações devem ser listadas, como transação entende-se as funcionalidades fundamentais do software, como adicionar um cartão de crédito a uma base de dados, como modificar informações de um cliente, entre outros.

Uma ordem cronológica dos AS deve ser estabelecida. Isso é fundamental para o desenvolvimento e interpretação dos testes e levantamento das probabilidades de falhas. Para alcançar isso é necessário determinar para cada transação, quais AS observados devem estar completos antes que o próximo AS seja alcançado.

Uma vez determinado os AS é necessário determinar os espaço de entradas para cada um. Para alguns AS o espaço de entrada consiste de apenas uma única entrada, outros consistem em uma variedade de entradas. Tipicamente espaços de entrada são particionados de acordo com certas características (como num sistema de caixa automático a quantidade de dígitos do cartão ou se o número inicia-se com zero). Esse particionamento das entradas conjuntamente com o AS associado forma a unidade básica que será chamada de nó e representará uma entidade distinta no nosso modelo.

3.2.6 Ponderação Estatística dos nós

Especificar individualmente a probabilidade estatistica de cada nó com um apurado grau de precisão não é uma tarefa imediata. Nesse processo, fazem-se necessárias calibrações, análise a serem executadas sobre o levantamento inicial de forma a atingir uma melhor precisão. Informações relevantes para a ponderação estatística incluem informações históricas

como resultado de testes anteriores voltados para áreas correlatas. Outros pontos a serem considerados incluem:

- A complexidade do código associado ao nó. Quanto maior a complexidade, maior a chance de falhas;
- Comparativamente, ponderar partes do código quanto ao grau de estabilidade em relação a outras partes do código;
- O grau de maturidade do código:
    - Código já testado frente a um conjunto de testes;
    - Uma nova implementação;
    - Uma situação intermediária entre uma nova implementação e um código maduro.
- Historicamente avaliar a confiabilidade e estabilidade dos responsáveis pelo código. Uma respeitada *software house* ou uma nova provedora de código;
- Avaliar historicamente resultados de códigos similares que já tenham sido testados.

O levantamanto dos teste são realizados em duas etapas distintas. Na primeira baseia-se na própria rede bayesiana onde é realizado para um dado nó, a varredura de todas as partições de entrada combinadas com as partições de entrada do nó filho. Esta operação quando executada para todos os nós da rede determina os casos de testes imediatos. Num segundo momento, através da utilização da matriz de adequação são levantados mais testes buscando assim melhorar a cobertura. No próximo capítulo será dado um tratamento detalhado sobre essa técnica

3.2.7 Simulações

Todos os cenários podem ser simulados utilizando ferramentas computacionais capazes de realizar o algorítmo Bayesiana. Dois pacotes comerciais que realizam essa simulação são o Netica (Norsys Software Corporation, Vancouver , Canada - http://www.norsys.com) e Hugin (Hugin Expert A/S , Aalborg - Dinamarca – http://www.hugin.com). Para simulações dos problemas de teste neste trabalho, será utilizado o Netica.

Capítulo IV – Uma proposta Sistêmica

Neste capítulo será proposto um ferramental para o design de casos de testes de caixa preta a partir de um documento de requisitos. Para tanto será utilizado a conceituação de rede Bayesiana, descrita no item 3.2, para o entendimento adequado da problemática do software alvo dos testes, conjuntamente com o uso do critério de adequação (descrito em 3.1) necessário a garantir a devida cobertura dos requisitos.

4.1 Um novo enfoque

A abordagem proposta visa buscar um novo paradigma para o design de testes de software incorporando elementos gráficos e uma ponderação de riscos envolvidos. Nesta proposta faz-se necessário um estudo preliminar profundo sobre os novos requisitos a serem implementados e realizar um levantamento de quais casos de usos são afetados. Uma documentação de apoio como diagramas UML de casos de uso ou mesmo um diagrama de seqüência serão materiais de inestimável valia para o correto levantamento dos cenários.

Através do levantamento consistente dos cenários e suas inter-relações, tem-se os fundamentos necessários para o modelamento Bayesiana.

Nesta etapa, a preocupação está voltada para o modelamento do problema de software, buscando levantar todas as transações, ações de software e partições de entrada relacionadas. Deste conjunto de elementos constitui a arquitetura de testes, um documento estritamente direto quanto ao levantamento dos casos de testes necessários, possibilitando inclusive, simulação de cenários e conseqüentes estimativa dos testes mais indicados a identificar falha.

4.2 Construindo a Rede Bayesiana

Há um relacionamento unívoco entre casos de uso e transações, logo, para cada transação, uma rede específica deve ser desenvolvida contendo todas as ações de software envolvidas. O levantamento das ações de software é uma abstração intermediária que busca identificar partes do código dedicadas a certas operações ou funcionalidades. O próprio diagrama UML é uma fonte confiável para o levantamento das ações de software.

Por outro lado, o próprio entendimento da implementação consistentemente e de forma natural, leva ao levantamento de responsabilidades implementadas pelo código de software e constituem também, ações de software válidas para o modelo.

No levantamento das ações de software, é importante salientar que não há a preocupação de determinar a seqüência hierárquica entre os nós da rede, nesse momento a enumeração das ações de software constitui o foco da atenção.

A etapa seguinte preocupa-se para cada ação de software, o levantamento da partição de entrada relacionada. Enfatizando que partição de entrada é o conjunto de variáveis que guardam entre si uma característica determinante. As partições de entrada são efetivamente os dados de entrada para as ações de software, ou melhor, serão elementos onde os nós da Rede Bayesiana terão alguma ação e poderão resultar em saídas previstas ou não (falhas). Pode-se fazer uma analogia com ações de software e partições de entrada como sendo respectivamente uma função de software e seus respectivos parâmetros de entrada.

4.3 Ponderação Estatística

Nesta fase, para cada partição de entrada, deve-se estimar baseado na complexidade da própria classe de entrada e do algoritmo necessário ao seu processamento, como também o grau de estabilidade esperado, uma probabilidade de risco. Essa probabilidade de risco esta associado as chances de ocorrer uma falha, quando a ação de software processar a referida partição de entrada.

Dado um nó, o total de risco envolvido deve estar dentro dos 100%, logo a distribuição de risco entre as várias partições de entrada deve atender essa premissa, salientando que cada nó corresponde a uma ação de software.

4.4 Sequenciamento das Ações de Software (AS)

A etapa final para a montagem da rede consiste na relação hierárquica de cada ação de software que é modelada graficamente por uma seta originado da AS predecessora e finalizando na sucessora. Para essa determinação, a linha de raciocínio abordada consiste em identificar que devem ser finalizadas por completo antes que uma nova AS possa ser processada.

4.5 Uso de uma ferramenta para a modelagem

Para o propósito desse trabalho, será utilizado o Netica como citado em [3.2.7]. Com caráter meramente ilustrativo uma rede Bayesiana de relacionamento de animais é mostrada abaixo:

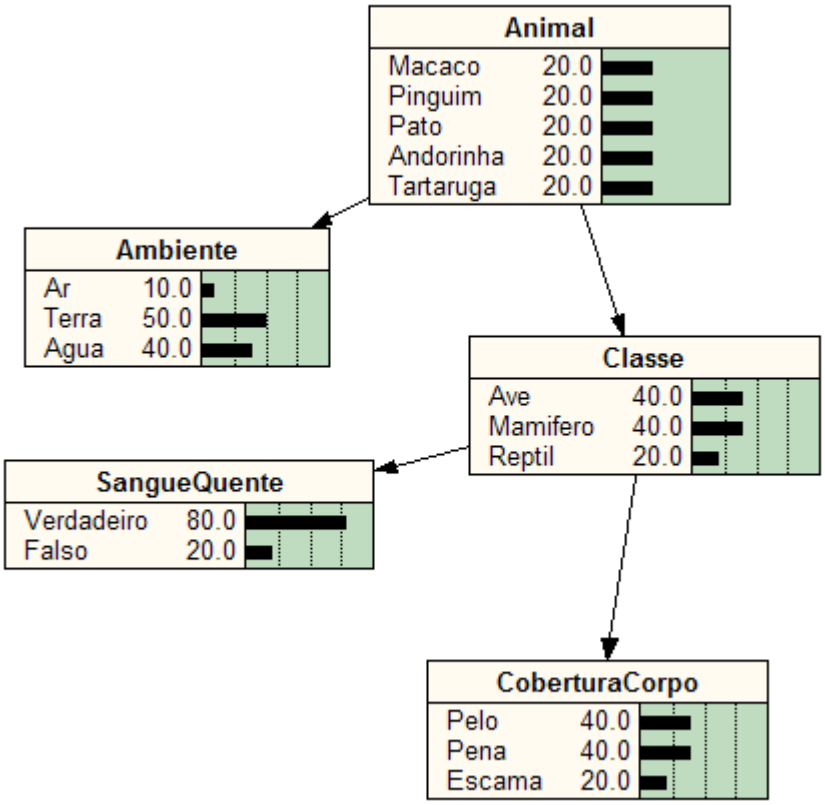
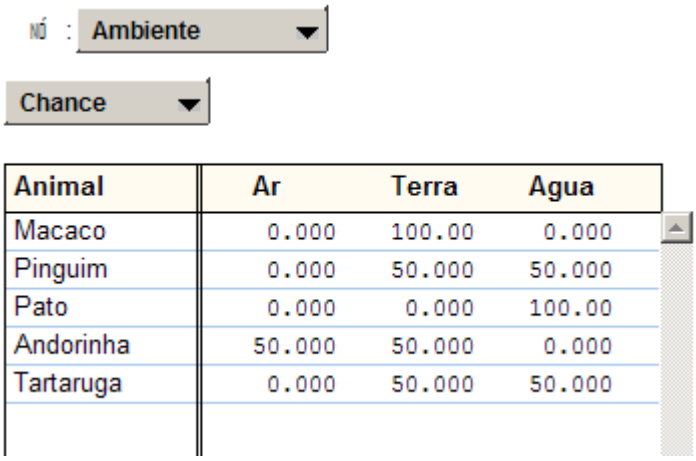


Figura 4-1 - Rede Bayesiana Animais

A Rede Bayesiana é uma rede fortemente hirerárquica. Cada nó está sob a influência do nó pai (no diagrama a ponta da seta sinaliza o nó filho). A partição de entrada do nó pai será obrigatoriamente elemento na tabela de partição de entrada do nó filho. Essa dependência é que sustenta a simulação de cenários, com consequente re-arranjo das probabilidades de ocorrer uma falha, em função de uma dada partição de entrada escolhida.

A Figura 4-2, mostra- a tabela da partição de entrada para o nó *Ambiente*. O nó *Animal* é pai do nó *Ambiente*, logo toda a partição de entrada de *Animal* deve ser mapeada com probabilidade estatística na partição do nó filho (*Ambiente*).



Animal	Ar	Terra	Agua
Macaco	0.000	100.00	0.000
Pinguim	0.000	50.000	50.000
Pato	0.000	0.000	100.00
Andorinha	50.000	50.000	0.000
Tartaruga	0.000	50.000	50.000

Figura 4-2 – Tabela Distribuição Estatística

A rede da Figura 4-1 - Rede Bayesiana Animais**Erro! A origem da referência não foi encontrada.** esta em repouso, ou seja, não há simulações de cenários, a probabilidade de cada animal é mesma. logo se tem uma distribuição para os animais enumerados neste exemplo, com relação ao ambiente em que vivem, da seguinte forma:

No ar vivem 10% dos animais colocados no exemplo.
Na terra vivem 50% dos animais colocados no exemplo.
Na água vivem 40% dos animais colocados no exemplo.

Essa distribuição estatística é explicada utilizando a Figura 4-2 – Tabela Distribuição Estatística. A primeira coluna, são os elementos de entrada obtidos da partição de entrada do nó pai:

Partição Entrada pai (Animal)
Macaco
Pingüim
Pato
Andorinha
Tartaruga

Tabela 4-1 – Tabela Partição de Entrada nó pai (Animal)

Para o nó *Ambiente*, foi levantado como relevante a seguinte partição de Entrada:

Partição Entrada nó filho (Ambiente)
Ar
Terra
Água

Tabela 4-2 – Tabela Partição de Entrada nó filho (Ambiente)

A probabilidade total dessa partição deverá ser de 100%. Logo a Tabela 4-1 – Tabela Partição de Entrada nó pai (Animal) (que contém cinco elementos) deverá ser totalmente distribuída de acordo com a Tabela 4-2 – Tabela Partição de Entrada nó filho (Ambiente), como mostrado na Figura 4-2 – Tabela Distribuição Estatística.

De acordo com a Figura 4-2, exemplificando para Terra, temos a seguinte distribuição:

Terra
100.00
50.000
0.000
50.000
50.000

Figura 4-3 – Distribuição Estatística para Terra

Analisando a Tabela 4-3 – Matriz dos Critérios de Adequação, temos que a somatória desses valores resulta em 250 Como há 5 partições de entrada, temos que cada partição contribui com 100, totalizando 500, o que corresponde a probabilidade de 100%. Caso obtivéssemos para a partição Terra ao invés de 250 o valor de 500, teríamos 100% (500/500) de certeza que todos os animais tem como ambiente a terra, não sendo relevante para essa análise a inclusão das partições Ar e Água, que devem ser excluídas. Desta forma para o nó ambiente em específico para a partição de entrada Terra, estando a rede em repouso, uma distribuição estatística de 50% (250/500).

Facilmente pode-se realizar uma simulação e verificar o impacto entre os nós ao longo da rede. Nessa simulação, parte-se do pressuposto de que o animal em questão é o macaco com 100% de certeza como mostrado na Figura 4-4 - Simulação de Cenário, logo a rede refletirá essa nova realidade e para o nó *Ambiente* tem-se agora a seguinte distribuição:

Ar - 0%

Terra – 100% (total certeza de somente encontrar macacos no ambiente terrestre)

Água - 0%

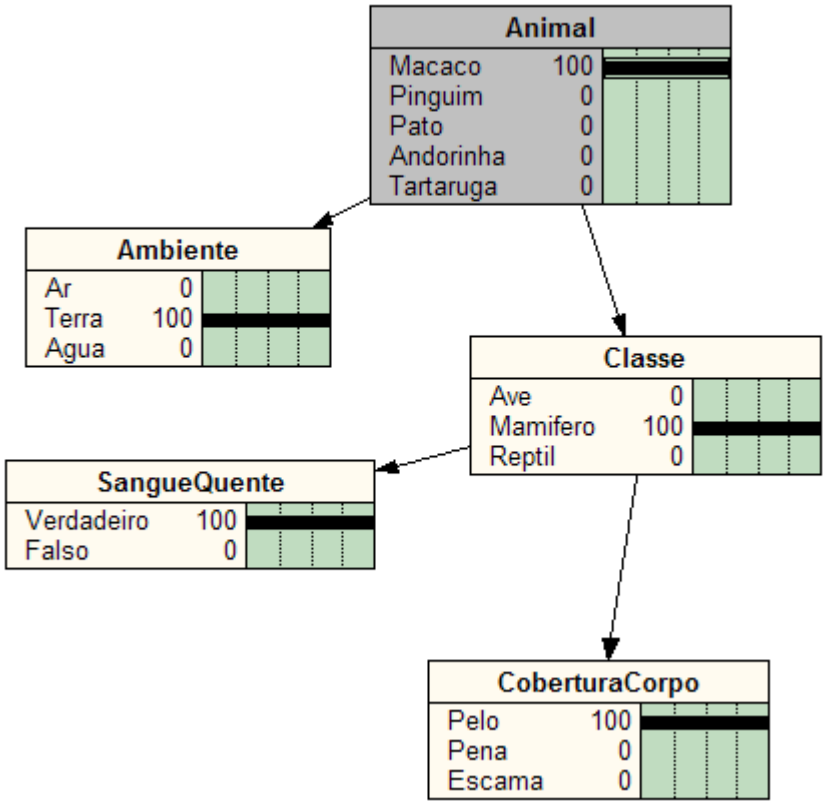


Figura 4-4 - Simulação de Cenário

4.6 Casos de Testes e Rede Bayesiana

Quando o modelamento de uma transação finaliza, obtém-se a arquitetura de testes e os casos de testes a serem aplicados, de forma muito direta e objetiva.

A rede deve ser simulada partindo dos nós de mais alta hierarquia e para cada partição de entrada todas as partições de entrada do nó sucessor devem ser verificadas. Um novo caso de teste deve ser criado quando a probabilidade for diferente de zero e a interação corrente fizer sentido lógico. O grau de detalhamento do caso de teste está fora do escopo desse trabalho, como foi dito anteriormente, mas vale aqui dizer que essa tarefa sofre influência do grau de conhecimento da pessoa encarregada de executar a tarefa de teste, ou se a execução será manual ou automática.

4.6 Cenários não previsíveis

A construção da Rede Bayesiana, constitui um ferramental muito eficiente no entendimento adequado do problema e busca de situações que não estão contempladas nos requisitos. Nestes

casos, o comportamento esperado não é documentado, o que acaba por exigir uma revisão dos requisitos. Desta forma a documentação do projeto , por exigência da técnica, torna-se mais robusta, completa e adequada, reduzindo assim, os problemas com manutenção no futuro.

4.7 Garantindo a cobertura dos Requisitos

Somente o uso da rede não garante a cobertura adequada dos requisitos, sendo necessária uma nova ferramenta que faça a associação dos testes criados com os requisitos segundo critérios de cobertura. A ferramenta proposta para essa tarefa é o uso do critério de adequação focado em testes. Logo, segundo a seção 3.1, temos uma relação da seguinte forma:

C(p,s,t) = verdadeiro, implica que segundo o critério de adequação C, o teste t é adequado para testar o programa p segundo a especificação s.

O grau de complexidade e a quantidade de testes a serem criados é uma função direta da quantidade de critérios que serão verificados. A proposta desse trabalho é usar como critérios de adequação os casos mais triviais encontrados na prática de testes de software, sendo eles:

- Testes Positivos: verificam requisitos baseados na especificação;
- Testes Negativos: verificam como o sistema responde com um estímulo contrário à especificação;
- Testes de Limite: verificam condições de contorno (mínimo -1, mínimo, mínimo + 1 máximo, máximo – 1,máximo + 1);
- Testes de Volume: verificam como o sistema se comporta frente a grandes volumes de entradas;
- Testes de Interação: verificam como o sistema se comporta na interação com outros componentes do sistema;

De forma a consolidar essas informações, tem-se a seguinte matriz:

Critério	Requisito	Teste	Comentário
Positivo	REQ-0001	T1	
	REQ-0002	T2	
	REQ-0003	T3	
	REQ-0004	T4	

Critério	Requisito	Teste	Comentário
Negativo	REQ-0001	t6	
	REQ-0002	N/A	
	REQ-0003	N/A	
	REQ-0004	T7	

Critério	Requisito	Teste	Comentário
Limite	REQ-0001	T1	
	REQ-0002	N/A	
	REQ-0003	N/A	
	REQ-0004	T8	

Critério	Requisito	Teste	Comentário
Volume	REQ-0001	N/A	
	REQ-0002	N/A	
	REQ-0003	T9	
	REQ-0004	T10	

Critério	Requisito	Teste	Comentário
Interação	REQ-0001	N/A	
	REQ-0002	N/A	
	REQ-0003	N/A	
	REQ-0004	N/A	

Tabela 4-3 – Matriz dos Critérios de Adequação

Sistematicamente, o preenchimento dos testes obedece a certas condições:

- A matriz contém todos os requisitos do sistema listados na coluna Requisito;
- Existem cinco grupos de critérios que possuem a mesma seqüência de requisitos;
- Há a possibilidade de um teste satisfazer mais que um critério ao mesmo tempo;
- Os testes são extraídos da rede Bayesiana (detalhado em 4.2 Construindo a Rede Bayesiana)e classificados segundo essa matriz;
- Verifica-se que ao extrair todos os testes da rede Bayesiana, podem existir entradas com nenhum teste classificado na matriz. Isto é uma condição perfeitamente normal e esperada. Neste momento procede-se a análise caso seja aplicável à criação de um novo teste, de forma a cobrir o requisito segundo o respectivo critério;
- As entradas N/A (não aplicável), referem-se a testes não aplicáveis para o requisito segundo o critério em questão.

Espera-se que o uso da matriz induza o desenvolvedor dos testes a se concentrar em todos os requisitos de entrada e assegurar que estão cobertos segundo os critérios pré-estabelecidos.

Capítulo V – Analisando as primeiras Medidas

De forma a tornar mais elucidativo o uso dessa nova técnica, foram levantadas métricas relativas a esforço de análise, cobertura de testes e razão entre o esforço entre as técnicas bem como a razão de cobertura. Esses são dados reais obtidos na criação de testes de software na área de telecomunicação.

Será também mostrada uma simulação da técnica proposta na criação de testes sobre um sistema fictício de cadastro de cartões de crédito.

As métricas levantadas são iniciais, estando os experimentos ainda em andamento. Futuras métricas devem ser levantadas estando a técnica já em regime, de forma a validar os números preliminares.

5.1 A descrição do Experimento

Os dados aqui mostrados são referentes a quatro experimentos que ainda estão em andamento baseados exclusivamente na utilização da nova técnica proposta. Esses quatro experimentos são quatro novas implementações de software para telecomunicação que estão agora submetidos ao grupo de testes para validação dos requisitos e aprovação. São números iniciais e estão sujeitos a influência da curva de aprendizagem da própria técnica e ferramental associado (software de modelamento, matriz de adequação), mas sinalizam uma tendência comparativa. Um novo levantamento deve ser executado a médio prazo, de forma a validar além dos números iniciais já obtidos, a métrica de retenção de defeitos comparativamente com a antiga técnica. Nesta fase inicial da experimentação (análise) não foi possível o levantamento da métrica de retenção de defeitos, que comumente é obtido após a execução dos ciclos de testes.

5.2 A Técnica Tradicional

Como técnica Tradicional, entende-se o conjunto de métodos utilizados na corporação, para análise dos requisitos e criação dos casos de testes. Basicamente engloba no estudo da

documentação de requisitos e associação de casos de testes para assegurar cobertura e correto funcionamento da implementação de software. Neste método, há uma forte parcela subjetiva envolvida.

5.3 As medidas comparativas

Na comparação entre a técnica tradicional e a técnica proposta serão utilizados medidas simples de esforço (tempo gasto) e a densidade de testes por requisito. As implementações selecionados quandam entre si uma quantidade de requisitos muito próximas, detalhes serão dados na seção 05.4 Os dados práticos.

5.4 Os dados práticos

A Tabela 5-1 consolida quatro grupos de teste de software Cada grupo é formado por duas implementações distintas de software para quais casos de testes devem ser criados. Em cada grupo, uma análise de requisitos foi executada pela técnica tradicional já consolidada na organização e uma utilizando a técnica aqui proposta. O objetivo final é a criação de novos casos de testes que validem a implementação de acordo com o documento de requisitos. Outro ponto importante a ressaltar, é que são novas funcionalidades a serem agregadas sobre um sistema já desenvolvido, ou seja são incorporações de novas funcionalidades a um sistema de software já em funcionamento. O critério utilizado para esse agrupamento baseou-se na semelhança do número de requisitos a serem verificados e no grau de identidade entre as implementações (componente de software alterado, funcionalidade, classe de protocolo). Vale enfatizar, que o grupo 1 é uma exceção, na qual as duas *features* são a mesma, analisada com as duas técnicas.

A seguir encontra-se uma descrição dos campos da Tabela 5-1.

	Feature	# Req	# TC	Tempo Análise [horas]	Técnica	Comentário	Tempo/TC [min]	Tempo/TC [min/tc] (Tradicional)	Tempo/TC [min/tc] BAYESIANO	TC/Req (BAYESIANO)	TC/Req (Tradicional)	Razão Esforço Análise (Tradicional/Bayesiano)	Razão Densidade de Testes
Grupo 1	A	11	100	8	BAYESIANA	13111	4.8		4.8	9.1		3.9	1.6
	A	11	64	20	TRADICIONAL	13111	18.8	18.8			5.8		
Grupo 2	C	7	180	41	BAYESIANA	15508	13.7		13.7	25.7		4.2	7.3
	D	6	21	20	TRADICIONAL	13062	57.1	57.1			3.5		
Grupo 3	E	17	43	11	BAYESIANA	16744	15.3		15.3	2.5		3.9	3.9
	F	23	15	15	TRADICIONAL	14669	60.0	60.0			0.7		
Grupo 4	G	47	263	33	BAYESIANA	14705	7.5		7.5	5.6		1.4	6.2
	H	50	45	8	TRADICIONAL	14293	10.7	10.7			0.9		
	Valor Médio							36.6	9.1	10.7	2.7	3.4	4.8

Tabela 5-1 – Dados Experimentais para Análise de Requisitos

- Feature: Implementação de software para o qual serão criados casos de teste;
- # Req: Número de Requisitos;
- #TC: Número de casos de Teste;
- Tempo Análise [horas]: Esforço total em horas para análise (criação da rede e matriz de adequação);
- Técnica: Descrição da técnica utilizada (Tradicional – atualmente em uso na organização e Bayesiana – baseada na proposta deste trabalho);
- Tempo/TC [min/tc] (Tradicional): Essa relação será chamada de **ESFORÇO**, e quantifica o tempo gasto para criação de um caso de teste, segundo a técnica Tradicional;
- Tempo/TC [min/tc] (Bayesiana): Essa relação será chamada de **ESFORÇO**, e quantifica o tempo gasto para criação de um caso de teste, segundo a técnica Bayesiana;
- TC/Req (Tradicional): Essa relação será chamada de **DENSIDADE DE TESTES**, e quantifica o número de casos de testes criados por requisito, segundo a técnica Tradicional;
- TC/Req (Tradicional): Essa relação será chamada de **DENSIDADE DE TESTES**, e quantifica o número de casos de testes criados por requisito, segundo a técnica Bayesiana;
- Razão Esforço Análise (Tradicional/Bayesiana): Quantifica a magnitude do esforço de análise Tradicional em comparação ao esforço de análise Bayesiana;
- Razão Densidade de Testes (Bayesiana/Tradicional): Quantifica o grau de densidsade de testes Bayesiana frente à densidade de testes utilizando a técnica Tradicional.

Com os dados da Tabela 5-1, foram definidas duas medidas utilizadas para quantificar o grau de cobertura dos requisitos e esforço de análise. A primeira, DENSIDADE DE TESTES, sinaliza a quantidade de casos de testes por requisito. A segunda, ESFORÇO, indica o tempo despendido para o levantamento dos casos de teste necessários, medida em tempo por casos de teste.

5.5 Uma Análise Quantitativa

Utilizando a Tabela 5-1, foram criados os seguintes gráficos analíticos:

- Esforço de Análise: Mostra a métrica comparativamente para cada grupo com relação ao esforço em minutos consumidos;
- Densidade de Testes: Mostra a métrica comparativamente entre cada grupo com relação à quantidade de testes criados por requisito;
- Razão Esforço de Análise: Mostra a magnitude entre o Esforço da técnica Tradicional frente a técnica Bayesiana;
- Razão de Densidade de Testes: Mostra a magnitude entre o Densidade de testes da técnica Tradicional frente a técnica Bayesiana;

5.6 Gráfico Esforço de Análise

A Figura 5-1 ilustra que o esforço demandado para a análise da técnica Tradicional é sempre comparativamente maior do que ao da técnica Bayesiana. Uma provável justificativa para o maior esforço da técnica Tradicional, seria a falta de uma sistematização do processo de análise, o que acaba por criar uma dependência da experiência e interpretação dos requisitos do engenheiro de testes, que subjetivamente cria casos de teste.

Outro fator importante consiste no limitado entendimento da corrente implementação e suas interações, já que não existem ferramentais gráficos que consolidem a arquitetura de teste e possibilitem simulações de cenários.

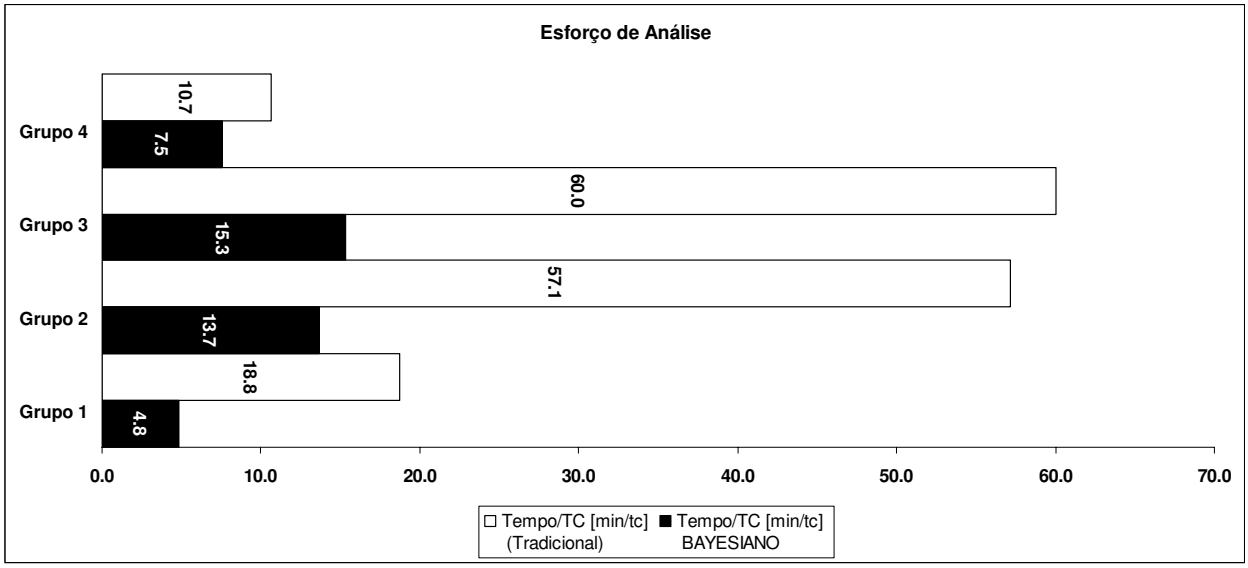


Figura 5-1 – Gráfico Esforço de Análise (min/tc)

A razão entre o esforço de análise utilizando a técnica tradicional pelo esforço de análise pela técnica Bayesiana é mostrada na Figura 5-2. Verifica-se uma razão em torno de quatro vezes, com exceção do grupo 4 que mostra uma relação em torno de uma vez e meia. O grupo 4 difere dos demais grupos por serem implementações de software de um protocolo diferente, o qual está relacionada com uma equipe técnica de desenvolvimento de testes em separado, distinta dos demais grupos. Existe aqui um indicativo que a experiência dos engenheiros de testes

influenciou numa variação menor do esforço comparativo. Como essa equipe é dedicada a criação de testes de um componente específico com um escopo reduzido em comparação aos outros componentes acabou por trazer um diferencial importante na redução do esforço de análise. Esse grupo possui um domínio maior do componente em questão, logo a criação da rede Bayesiana foi apenas uma rápida interpretação do documento de requisitos para capturar os novas ações de software a serem adicionadas. Outro ponto relevante, diz respeito a curva de aprendizagem, já que este grupo foi o primeiro a ser treinado , tendo um tempo maior de contato com a nova tecnologia.

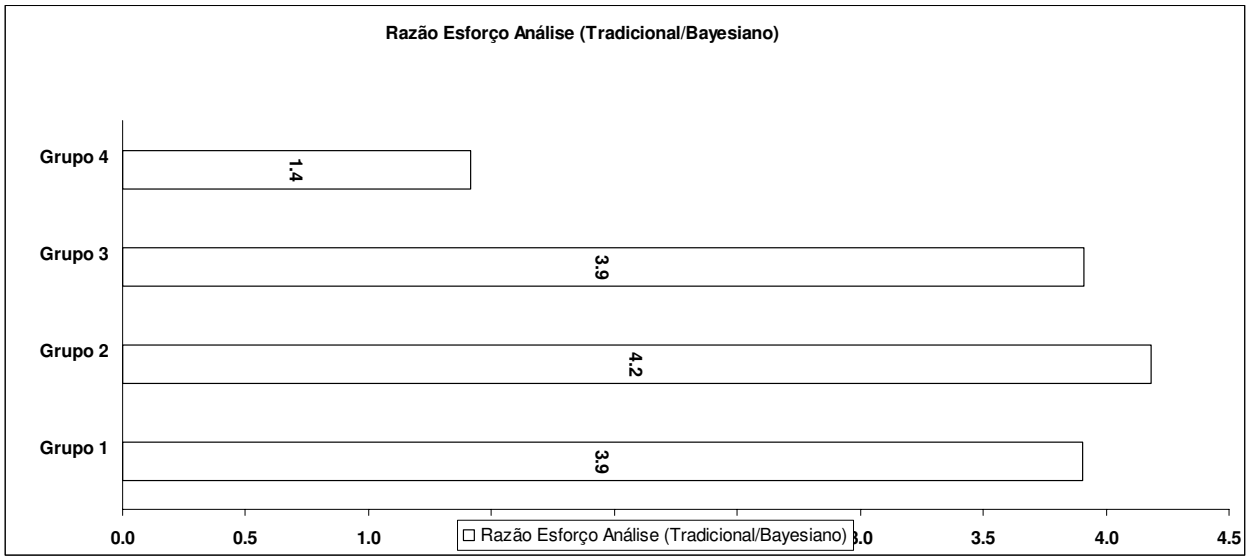


Figura 5-2 – Gráfico Razão Esforço Análise

5.7 Gráfico de Densidade de Testes

A Figura 5-3 mostra que a quantidade de testes por requisito utilizando a técnica Bayesiana é superior comparativamente a técnica Tradicional. Uma hipótese que pode estar diretamente relacionado a essa maior densidade de testes é o emprego de um modelamento gráfico . Esse modelamento mostra as dependências e relações entre os componentes de software, possibilitando identificar cenários não cobertos pelos requisitos, ou seja, novos testes.

O uso da matriz de adequação é outro relevante fator a ser considerado. Os cinco critérios sugeridos (Positivo, Negativo, Limite, Volume e Interação), direcionam o desenvolvedor de testes a olhar cada requisito sobre perspectivas diferentes, contribuindo para o aumento da quantidade de testes.

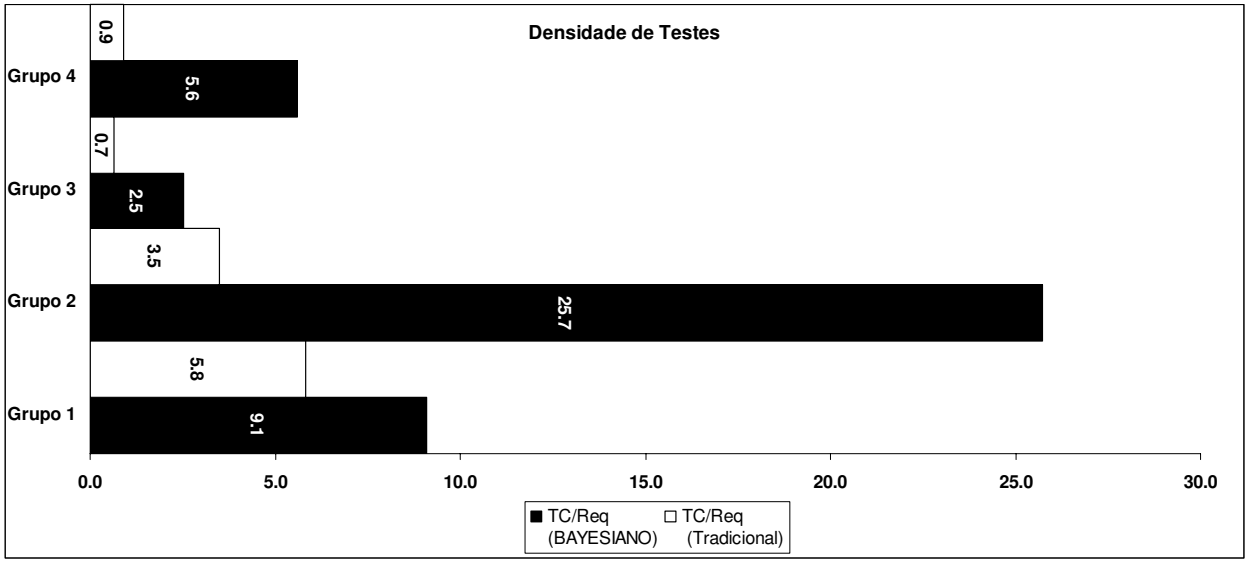


Figura 5-3 - Gráfico de Densidade de Testes (TC/Req)

A sistematização do processo é um diferencial a ser considerado reduzido a subjetividade e incorporando maturidade ao processo de desenvolvimento de teste.

A Figura 5-4 mostra que há uma vantagem da técnica da densidade de testes Bayesiana frente a técnica Tradicional, evidenciado pela quantidade de testes criados comparativamente (Tabela 5-1). A densidade de testes mostrada possui um valor médio em torno de quatro vezes e meio . A utilização da rede Bayesiana para o adequado entendimento da arquitetura de testes a ser implementada tende a trazer cenários novos, já que mostra graficamente o conjunto de casos de teste possíveis. Aliada ao uso da matriz de adequação, acaba por resultar em uma maior quantidade de casos de teste. Esses números iniciais devem ser confirmados em novos ensaios, quando a curva de aprendizagem da técnica estará em regime.

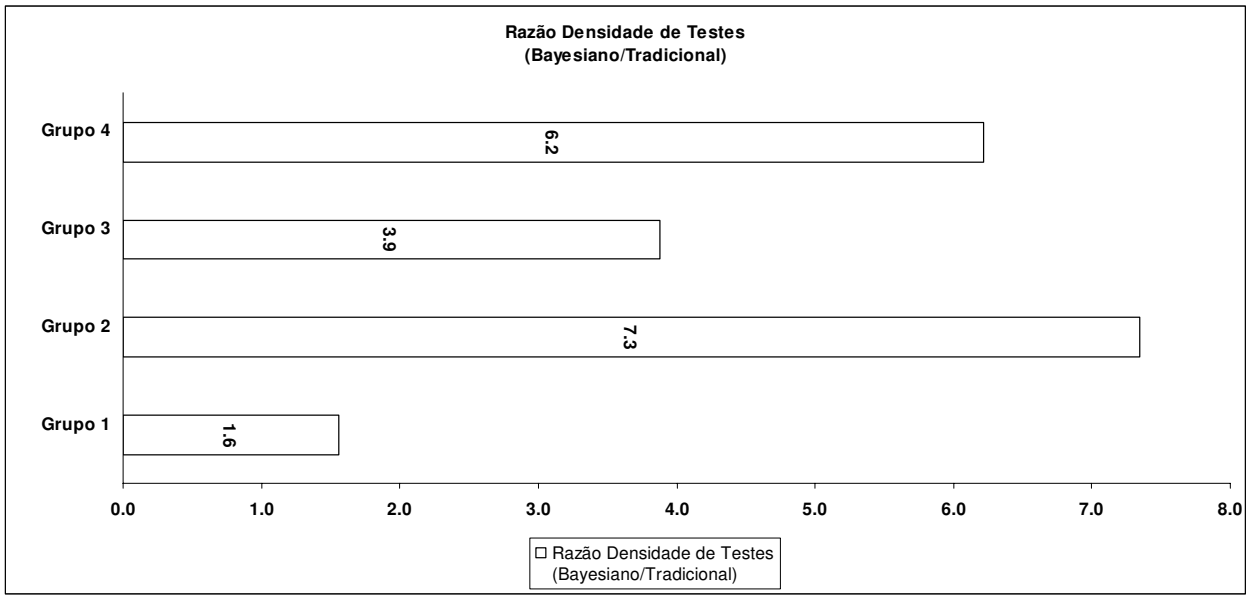


Figura 5-4 – Gráfico Razão Densidade de Testes (Bayesiana x Tradicional)

5.8 Uma rede Simples

A seguir é exemplificado o uso da técnica na criação de casos de testes para um sistema de banco de dados, onde será criada uma rede e populado parcialmente a matriz de adequação.

O sistema fictício de validação de cartões de crédito em questão possui os seguintes requisitos:

Identificação	Descrição
Req_1	O sistema realizará a validação de cartões de crédito das operadoras A, B e C somente, recusando as demais operadoras;
Req_2	A identificação da operadora será baseada no número do cartão, a saber: Operadora A: número do cartão inicia-se com zero; Operadora B: número do cartão não se inicia com zero e possui no total dez dígitos; Operadora C: número do cartão não se inicia com zero e possui no total doze dígitos. O sistema devera suportar até 100 validações simultâneas;
Req_3	Para as operadoras B e C deve-se validar o código de segurança do cartão (baseado num algorítmo que envolve o próprio número do cartão);
Req_4	Deve-se validar também se o cartão possui a data de validade não vencida somente para os cartões das operadoras B e C.

Tabela 5-2 – Requisitos Sistema de Cartão

Baseado nestes requisitos é mostrado na Figura 5-5 uma proposta para o modelamento do problema numa rede Bayesiana. A distribuição estatística do nó VALIDA_OPERADORA, foi levantando baseado na complexidade do algoritmo necessário para cada operadora, neste exemplo a operadora C é que possui a maior complexidade e por consequente maior probabilidade de falhas. Os demais nós foram construídos baseados nas considerações apresentadas em 04.5 Uso de uma ferramenta para a modelagem:

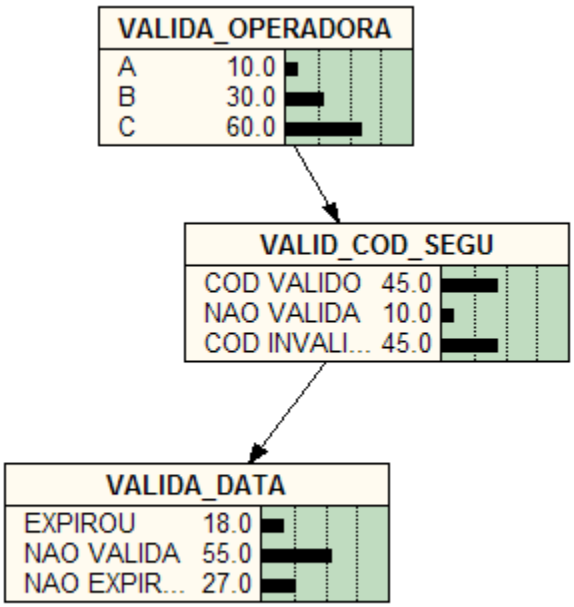


Figura 5-5 – Rede Bayesiana modelando o Sistema de Cartão

Nesta proposta, foram criadas três ações de software (nós, AS):

- Valida_Operadora: responsável por validar segundo o número do cartão a operadora que pertence;
- Valida_Cod_Segu: responsável por validar o código de segurança;
- Valida_Data: responsável por validar se o cartão em questão está válido.

A partição de entrada para cada nó é mostrada na Tabela 5-3:

Ação de Software (AS)	Partições de Entrada
Valida_Operadora	Operadora A Operadora B Operadora C
Valida_Cod_Segu	Cod_Valido Nao_Valida Cod_Invali
Valida_Data	Expirou Nao_Expirou Nao_Valida

Tabela 5-3 – Tabela das Partições de Entrada

Em uma simulação pode-se como exemplo adotar que o cartão corrente é da operadora A (100% de certeza) e verificar qual o impacto sobre as demais ações de software, indicando quais cenários serão mais propensos a apresentar falha. Na ferramenta basta apenas selecionar a operadora A com probabilidade de 100% que o sistema automaticamente se re-arranja de forma a mostrar o impacto da simulação.A Figura 5-6 mostra o resultado:

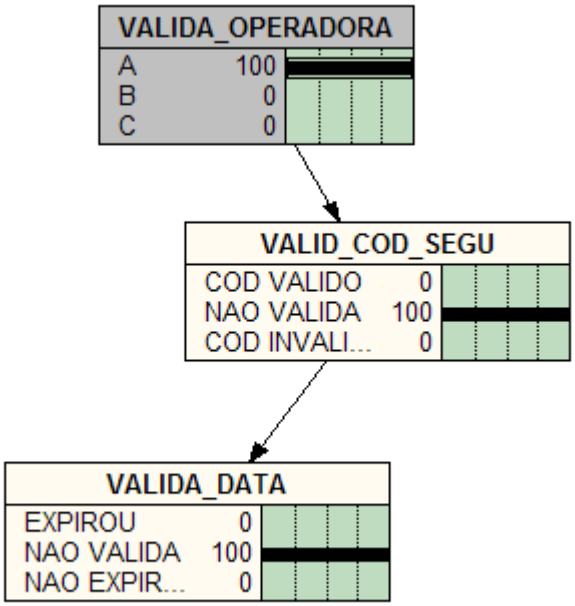


Figura 5-6 – Rede Bayesiana Simulando a Operadora A

5.10 A Matriz de Adequação

Para finalizar a análise, deverá ser criada a matriz de adequação com os casos de testes obtidos da rede. Novos casos de testes deverão ser criados, se aplicável. Segundo os critérios de adequação escolhidos. As Tabelas que se seguem exemplificam essa abordagem, onde foram levantados para os quatro requisitos trezes casos de testes. Vale ressaltar que obtivemos 3 testes não claramente presentes(T9, T10,T11) nos requisitos e que foram identificados no critério do Limite.

Critério	Req	Test	Comentário
Positivo	REQ_1	T1	Testar cartões das operadoas A,B e C e verificar se são reconhecidos
	REQ_2	N/A	
	REQ_3	T2	Utilizar cartões da operadora B e C com código de segurança válidos

Critério	Req	Test	Comentário
Negativo	REQ_1	T4	Testar cartões das demais operadoras
	REQ_2	N/A	
	REQ_3	T5 T6	T5: Utilizar cartoes da operadora A e verificar que código não é verificado T6: Utilizar cartões da operadora B e C com código segurança inválido

	REQ_4	T3	Utilizar cartões da operadora B e C não vencidos e verificar se são aceitos		REQ_4	T7 T8	T7:Utilizar cartões da operadora A e verificar que data não é validada T8: Utilizar cartões operadoras B e C com data vencida. Verificar se são recusados

Tabela 5-4 – Matriz de Adequação (Positivo,Negativo)

Critério	Req	Test	Comentário	Critério	Req	Test	Comentário
Limite	REQ_1	N/A		Volume	REQ_1	N/A	
	REQ_2	T9	Utilzar cartões com número entre os limiares inferior e superior e além desses limiares		REQ_2	T12	Validar lotes de 100 cartões simultaneamente
	REQ_3	T10	Utilizar código de segurança nos limites númericos possíveis a acima deles		REQ_3	N/A	
	REQ_4	T11	Utilizar data de expiração no ano bisexto, último dia do mês e virada de anos. Utilizar datas 50 anos a frente e 100 atrasados		REQ_4	N/A	

Tabela 5-5 – Matriz de Adequação (Limite, Volume)

Critério	Req	Test	Comentário
Interação	REQ_1	N/A	
	REQ_2	T13	Verificar como o sistema se comporta na validação simultânea de dois cartões , um estando validando a operadora e o segundo validando código de segurança
	REQ_3	N/A	
	REQ_4	N/A	

Tabela 5-6 – Matriz de Adequação (Interação)

Neste experimento , toda documentação de entrada foi um documento extremamente simples (Tabela 5-2), contento quatro requisitos. Com a aplicação da técnica sistêmica proposta, foram criados trezes casos de teste, resultando num valor médio de quatro testes por requisito. Este valor esta muito próximo aos quatro e meio (11% de desvio) que a análise da Figura 5-3 - Gráfico de Densidade de Testes (TC/Req)demostra.

Os testes baseados nos critérios positivo e negativo são os mais imediatos e obtidos da rede Bayesiana pelo simples varredura sequencial dos nós (e suas partições de entrada), como demonstrado na seção 4.2 Construindo a Rede Bayesiana . Nas simulações de cenários, partições com 0% de probabilidade são casos de testes que se enquadram no critério negativo. Com relação aos critérios Limite, Volume e Interação, a matriz de adequação exige o foco em cada requisito segundo prismas diferentes buscando identificar casos de teste que sejam aplicáveis.

As métricas de ESFORÇO e DENSIDADE DE TESTES, foram sinalizadores de uma tendência inicial da técnica com vantagens de menor tempo de análise e casos de teste que tendem trazer uma melhor validação dos requisitos.

5.10 Trabalhos semelhantes

Não foram encontrados trabalhos acadêmicos no meio científico que reunisse rede bayesiana aplicada a testes conjuntamente com critério de adequação. Um experimento com a utilização de apenas a rede bayesiana, pode se encontrado em [Woo02], onde são relatados uma melhora de cobertura com a utilização de 30% dos testes,utilizados com a técnia não bayesiana. No entanto a falta da utilização da matriz de adequação com os critérios de adequação pré selecionados, pode colocar em dúvida a melhora da cobertura conseguida, já que alguns cenários não presentes no próprio documento de requisitos, não foram identificados utilizando a matriz.

Considerações Finais

No capítulo III, foram abordadas a rede Bayesiana e o Critério de Adequação, fundamentos para a proposta sistêmica aqui apresentada. A busca de uma sistematização do processo de criação de casos de teste teve em seus primórdios dois limitantes básicos: deveria agregar técnicamente proporcionando um ferramental leve para o modelamento da arquitetura de testes e segundo, não poderia embutir uma sobrecarga relevante ao esforço de criação.

Os dados experimentais obtidos sinalizam que o modelamento gráfico da arquitetura de testes acaba por vislumbrar cenários não cobertos no documento de requisito e que devem ser incluídos na documentação, de forma a evitar comportamento dúbio.

Essa contribuição torna-se ainda mais acentuada quando o desenvolvimento de testes inicia-se paralelamente ao *design* da arquitetura de software. Assim, a arquitetura do sistema acaba por sofrer influência direta da atualização dos requisitos, resultando numa implementação mais robusta.

O uso sistemático de uma metodologia para criação de casos de teste é também uma maneira de eliminar a subjetividade do processo e permitir um melhor controle na fase de criação, um ganho gerencial interessante no tocante a estimativas, com melhores previsões, quanto a disponibilidade dos casos de testes.

A necessidade do uso de uma ferramenta comercial para o modelamento da rede é uma desvantagem imediata. Isto pode inviabilizar a implantação do modelo em alguns casos. A criação da rede com as consequentes simulações estatísticas é inviável sem a utilização da ferramenta adequada.

Neste trabalho, não foi abordado técnicas e vantagens do uso de testes exploratórios (técnica onde os testes não são formamente detalhados, apenas cenários e guias gerais são definidos). Uma proposta para um trabalho futuro seria a união da corrente proposta sistêmica com o uso extensivo de testes exploratórios.

O uso de testes exploratórios tornaria menor o esforço de escrita de testes que compartilham casos de usos comuns ,com pequenas alterações do estado inicial, bem como testes de alta complexibilidade e não usuais. No estágio do *design* da matriz de adequação podem-se detalhar os testes que seriam os prováveis candidatos para exploratórios.

No capítulo V, há uma análise quantitativa da técnica com dados reais. Os dados e gráficos comparativos, sinalizam uma tendência positiva referente a uma melhor densidade de testes (quatro vezes e meio) com menor esforço de análise (um quarto), aliado a um sólido conhecimento da arquitetura de teste. Esses dados mostram-se muito atraentes frente ao desafio atual, da busca da qualidade ao menor custo, com certeza um diferencial competitivo.

Referências Bibliográficas

[Bin99] Binder, Robert; Testing OO Systems: Models, Patterns and Tools;
Addison-Wesley, 1999, cap. 08, 14, 15.

[Cappovila 99] CAPOVILLA, I. G. G. Elementos Intrínsecos do Software e sua Influência na Qualidade do Processo de Desenvolvimento. 1999. 108 f. Dissertação (Mestrado em Qualidade) –IMECC - Instituto de Matemática, Estatística e Computação Científica, UNICAMP - Universidade Estadual de Campinas, Campinas

[Fid03] Fidelis,Wennder Indalécio Oliveira. FireWeb: Uma Ferramenta de Suport aos Testes de Regressão de Aplicações Web. Universidade de Campinas. Dezembro, 2003.

[Fra99] Frankl, P.G. Hamlet,R.G. Littlewood,B. Strigini,L. Evaluating Testing Methods by Delivered Reliability. IEEE Trans. Software Eng. Vol 24 , pp586-601,1998. Erratum Vol 25, pp 286,1999.

[Har94] M.J Harrold and Gregg Hothermel; A Framework for Evaluating
Regresion Test Selection Techiniques; Proc. of the 16th Int'l Conf. On Softw. Eng.;
Sorrento, Italy, May 1994, pages 201-210.

[Int01] Cândida Inthurn. Qualidade e Teste de Software. Visual Books, 2001.

[Mar00] Martins, Eliane.; Manutenção e Ferramentas CASE. IC-UNICAMP, 2000.

[Mcc76] McCabe, T.J. "A complexity Measure," IEEE Trans. Software Engineering,
vol. 2, 2n,o . 4, pp. 308-320, 1976. 1976.

[Mcc83] McCabe, T.J. "Structered Testing". IEEE CS Press, 1983.

[Mcc85] McCabe, T.J. and Schulmeyer ,G.C. ”System Testing Aided by
Structured Analysis: A Practical Experience,” IEEE trans software Engineering, V.1, n° 9, pp. 917-921, Sept. 1985.

[Mye79] Myers, G.J.. The Art of Software Testing. John-Wiley & Sons, 1979.

[Oli03] Oliveira, Ângela Cristina de. ETIERP – Estratégia de Testes para Implantação de Sistemas ERP. Universidade de Campinas. Dezembro, 2003

[Pig97] Pigoski, Thomas M; Practical Software Maintenance: Best Practices
for Managing Your Software Investiment. Wiley Computer Publishing, 1997.

[Pre02] Pressman, Roger S.; Engenharia de Software , 5. Edição. McGraw-Hill,
Inc., 2002.

[Red99] Redmil, F. Why System go up in Smoke. The Computer Bulletim. Sep 1999.

[Rio03] Rios, Emerson e Moreira Filho, Trayahu; Projeto & Engenharia de
Software – Teste de Software, ed. Alta Books LTDA, 2003.

[Som92] Sommerville, Ian. Software Engineering – 4th. Edition. Addison-Wesley Inc., 2000.

[Woo02] Wooff, David A. Goldstein, Michael. Coolean,Frank P. A. Bayesian Graphical Models for Software Testing. IEEE Transactions on Software Engineering Vol 28, N° 5, May 2002.

[Zhu96] Zhu,Hong. A formal Analysis of the Subsume
Relation Between Software Test Adequacy Criteria. IEEE TRANSACTIONS SOFTWARE ENGINEERING, VOL. 22, NO. 4, APRIL 1996.