NATÁLIA COSTA BIFANO DE OLIVEIRA

ESTUDO ANALÍTICO DE FERRAMENTAS PARA ANÁLISE DE CONFIABILIDADE DE SOFTWARE

LAVRAS – MG

2012

NATÁLIA COSTA BIFANO DE OLIVEIRA

ESTUDO ANALÍTICO DE FERRAMENTAS PARA ANÁLISE DE CONFIABILIDADE DE SOFTWARE

Monografia apresentada ao Colegiado do Curso de Sistemas de Informação, para obtenção do título de Bacharel em Sistemas de Informação.

Orientadora
Profa. Juliana Galvani Greghi

LAVRAS – MG

2012

NATÁLIA COSTA BIFANO DE OLIVEIRA

ESTUDO ANALÍTICO DE FERRAMENTAS PARA ANÁLISE DE CONFIABILIDADE DE SOFTWARE

Monografia apresentada ao Colegiado do Curso de Sistemas de Informação, para obtenção do título de Bacharel em Sistemas de Informação.

APROVADA em 17 de outubro de 2012.

Prof. Hermes Pimenta de Moraes Júnior UFLA

Prof. André Pimenta Freire UFLA

Orientadora

Profa. Juliana Galvani Greghi

LAVRAS – MG

2012

AGRADECIMENTOS

Agradeço a minha família por todo amor e apoio incondicional. Por estarem sempre presentes quando eu precisei.

A professora e orientadora Juliana, pelo aprendizado e conselhos dados ao longo do desenvolvimento desse trabalho e por ter aceitado me orientar.

Aos meus amigos (as) Cristiane Coelho, Marcelo Rufato, Aline Antunes, Luciano Brasil, por estarem sempre presentes em todos os momentos durante minha graduação. E a todos meus amigos que de alguma forma estiveram presentes em minha vida.

A todos os professores da UFLA, que durante estes cinco anos conseguiram fazer com que eu aprendesse muito.

Enfim, agradeço a todos que me ajudaram a concluir esse trabalho.

RESUMO

A atividade de teste de software tem crescido nos últimos anos devido a fatores como a importância dos produtos com qualidade oferecidos aos clientes, e, apesar de poderem ser bastante onerosos, os testes de software auxiliam na obtenção de software de maior qualidade. Dentre os atributos verificados, a confiabilidade é bastante importante e existem várias ferramentas disponíveis para auxiliar o tratamento deste atributo. Entretanto, muitas ferramentas apresentam obstáculos para sua utilização, como a incompatibilidade com alguns sistemas operacionais ou a indisponibilidade de documentação. Este trabalho faz a análise de cinco ferramentas para análise de confiabilidade de software, apontando suas principais características e dificuldades encontradas para sua manipulação.

Palavras chaves: confiabilidade, estatísticas, modelos, software, testes.

ABSTRACT

Software Testing activities has grown in the last years, and one of the factors that lead to this was the importance of high quality products, offered to clients. This activities can be very onerous but they are very important to obtain high quality software. One of the attributes of quality is reliability and there are many tools to prediction, support and maintainability of reliability. This work aims to analyze five tools to software reliability analysis, pointing their main characteristics and difficulties.

Keywords: reliability, statistics, models, software, tests.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELA

SUMÁRIO

1. INTRODUÇÃO

A atividade de teste de software tem crescido nos últimos anos devido a diversos fatores, como a importância dos produtos com qualidade oferecidos aos clientes, o aumento na competitividade do mercado e os clientes estarem cada vez mais exigentes.

O processo de teste pode ser considerado oneroso e complexo, e estas características podem contribuir para que empresas desconheçam técnicas de testes adequadas ou negligenciem tais atividades. Entretanto, muitas delas já estão se conscientizando da importância e benefícios gerados no resultado final de um software.

A atividade de teste de software é bastante abrangente, podendo-se testar um software de diversas maneiras. E, por ser uma atividade utilizada para se obter maior qualidade dos softwares produzidos, é necessário entender a importância de processos que avaliam o quanto um software está em conformidade com os requisitos. Um sistema em conformidade com a especificação de requisitos pode ser dito confiável.

Verificar o grau de confiança de um software é uma das atividades relacionadas à dependabilidade. A dependabilidade é uma das características buscadas pela área de Qualidade de Software e pode ser definida pelos seguintes atributos: segurança a de funcionamento, disponibilidade, manutenabilidade, segurança, comprometimento do desempenho e confiabilidade.

A confiabilidade é um dos atributos da dependabilidade e uma das características responsável pela qualidade do software, pois se o software não é confiável, pode-se dizer que a qualidade é baixa (JINO; CRESPO, 2007).

Existem diversos métodos e processos que visam aumentar a qualidade de um software e, neste trabalho, serão apresentados métodos relacionados à confiabilidade.

Devido à importância da área, várias ferramentas foram sendo desenvolvidas ao longo dos anos para Estimativas de Confiabilidade, como: CASRE, COQUALMO, SMERFs, RELIABILITY ANALYTICS TOOLKIT, R entre outras. Algumas delas utilizam métodos, estatísticas e modelos probabilísticos para realizar as análises desejadas.

Neste sentido, pode-se imaginar que qualquer ferramenta de análise estatística e probabilística poderia ser utilizada para análise de confiabilidade de software. Entretanto, a Teoria da Confiabilidade que será discutida na Seção 2.2.1.1, define métodos probabilísticos específicos para a análise de confiabilidade. E para que uma ferramenta possa ser utilizada, ela deve implementar tais métodos.

Neste trabalho serão analisadas características de algumas ferramentas para tratamento da Confiabilidade, seja para predição de falhas ou para análise de confiabilidade de um produto.

1.1 Objetivos do Trabalho

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é fazer um estudo analítico de ferramentas para análise de confiabilidade de software.

1.1.2 Objetivos Específicos

O objetivo específico é fazer a análise das ferramentas COQUALMO, R e Reliability Analytics Toolkit, identificar características positivas de cada uma e analisar a viabilidade de utilizar a ferramenta R para análise de confiabilidade de software.

1.2 Motivação

As atividades relacionadas à busca pela confiabilidade têm crescido no âmbito empresarial, visto que as empresas necessitam de práticas eficientes que maximizem a confiabilidade de seus produtos. Embora muitas empresas utilizem dessas atividades, muitas organizações não possuem conhecimento sobre sua importância, ou não investem na área devido essa atividade possuir um custo elevado. De forma geral, as empresas devem avaliar seu processo de produção, visando cumprir os prazos de entrega, reduções dos custos, entre outros aspectos que dependem das características de seus produtos e sistemas (SELLITO *et al*, 2002).

Para avaliar a confiabilidade de um produto não existe um procedimento padrão a ser seguido. Cada um possui sua particularidade e aplicação dos modelos disponíveis.

Ainda há empresas que não empregam processos de análise de confiabilidade, em parte por desconhecimento dos benefícios à serem alcançados, em parte pelos custos envolvidos.

Muitas das ferramentas específicas para análise de confiabilidade de software são comerciais e podem demandar um montante monetário substancial para a aquisição de licenças de uso. Ferramentas específicas para análise de confiabilidade de software gratuitas ou de códigos aberto, são mais difíceis de serem encontradas e, em muitas vezes, não são disponibilizadas atualizações ou documentações para suporte ao uso.

Diante dessa situação, pode-se levantar as seguintes perguntas: existem ferramentas específicas para análise de confiabilidade de software gratuitas? Se sim, existe documentação ou suporte para o uso destas ferramentas? A ferramenta gratuita para análise estatística R, que implementa vários dos

métodos definidos pela Teoria da Confiabilidade, pode ser utilizada para análise de confiabilidade de software? Quais as dificuldades em utilizá-la?

O restante do documento está organizado da seguinte forma: o capítulo 2 apresenta os conceitos estudados e usados para embasar este trabalho, o capítulo 3 apresenta a metodologia utilizada, o capítulo 4 a ferramenta estatística R, o capítulo 5 os resultados das análises realizadas e, finalmente, as conclusões são apresentadas no capítulo 6.

2 REFERENCIAL TEÓRICO

2.1 Teste de Software

Teste de software é o processo de execução de um produto para determinar se tal produto atingiu seus requisitos e funcionou corretamente no ambiente em que foi desenvolvido. Segundo Delamaro (2007) "o seu objetivo é revelar falhas em um produto, para que as causas dessas falhas sejam identificadas e possam ser corrigidas pela equipe de desenvolvimento antes da entrega final.".

Para entender melhor o que é a atividade de teste, é importante entender os conceitos de falhas, defeitos, erros e engano, comuns no cotidiano das pessoas envolvidas com atividades de teste.

Falha é a definição de dados incorretos, como instrução incorreta no código, já o engano é proveniente da ação humana que acarreta o resultado incorreto. Portanto a presença de uma falha pode ocasionar um erro durante a execução de um programa, que se caracteriza por uma situação inesperada. O defeito é o desvio entre o resultado obtido pela aplicação em relação aos requisitos exigidos (LAPRIE *et al*, 2007).

O teste é uma das técnicas de verificação de software mais utilizada na prática. Se usado de maneira adequada, fornecerá qualidade e confiabilidade a um produto (DELAMARO, 2007). Portanto, a atividade de teste é dividida em categorias com objetivos diferentes. Dentre elas estão, teste de unidade, de integração, de sistemas e de regressão.

O teste de unidade é realizado para garantir o funcionamento das menores unidades de um programa, que podem ser funções, procedimentos, métodos ou classes. Como cada unidade é testada separadamente o teste de unidade pode ser aplicado à medida que ocorre a implementação das unidades,

sem a necessidade de que o sistema esteja totalmente finalizado. Já o teste de integração deve ser realizado após serem testadas as unidades individualmente; a ênfase é dada na construção da estrutura do sistema (DELAMARO, 2007).

Depois que se tem o sistema completo, dá-se início ao teste de sistema, que tem o objetivo de verificar se as funcionalidades especificadas nos documentos de requisitos estão todas corretamente implementadas. E, por último, mas não menos importante, o teste de regressão que é realizado para verificar se mudanças promovidas no software introduziram novos erros na fase de manutenção (DELAMARO, 2007).

2.1.1 Teste Funcional

O teste funcional é uma técnica considerada caixa preta e, para executá-lo, são fornecidas entradas e avaliadas as saídas geradas, verificando-se com os objetivos especificados. Nessa técnica, os detalhes de implementação não são considerados e o software é avaliado segundo o ponto de vista do usuário (FABBRI *et al*, 2007).

Os critérios mais conhecidos da técnica de teste funcional são o Particionamento de Equivalência, Análise de Valor Limite e Grafo Causa- Efeito (FABBRI *et al*, 2007).

O Particionamento de Equivalência divide o domínio de entrada de um programa em classe de dados a partir das quais os casos de teste podem ser derivados. Este procura definir um caso de teste que descubra classes de erros, reduzindo o número total de casos de teste que devem ser resolvidos. Esse tipo de teste funcional tem o objetivo de tornar a quantidade de entrada de dados finita e mais viável para a atividade de teste (FABBRI *et al*, 2007).

A Análise do Valor Limite é usada em conjunto com o Particionamento de Equivalência, mas os dados não são escolhidos aleatoriamente. A maneira

pela qual eles são selecionados é pelo limitante de cada classe tanto do ponto de vista de entrada como de saída (FABBRI *et al*, 2007).

Segundo Fabbri (2007), " a experiência mostra que casos de teste que exploram condições limites têm uma maior probabilidade de encontrar defeitos."

O Grafo de Causa-Efeito verifica o efeito combinado de dados de entrada. As condições de entrada e ações são combinadas em um grafo a partir do qual é montada uma tabela de decisão, a partir desta, são derivados os casos de testes e as saídas (FABBRI *et al*, 2007). Este critério ganha a vantagem em relação aos anteriores, pois explora combinações dos dados de entrada.

A vantagem do Teste Funcional é que ele requer somente a análise em relação ao requisito de software, validando assim o produto testado, mas como baseia-se somente na especificação do produto, não se pode assegurar que partes críticas do código tenham sido cobertas. É fundamental que as técnicas de teste sejam vistas como complementares, e que sejam utilizadas em conjunto com outras técnicas, para que o produto possa ser explorado por diferentes pontos de vista (FABBRI *et al*, 2007).

2.1.2 Teste Estrutural

O teste estrutural, também conhecido como teste caixa branca, baseia-se no conhecimento da estrutura interna do programa. Para isso, é preciso determinar um conjunto de casos de teste, onde grupos de condições e laços, uso de variáveis e escopos são analisados através de casos de teste definidos por caminhos lógicos do software testados. Os casos de teste devem ser elaborados para detectar uma falha e corrigí-la. Eles são definidos para classes particulares exigindo serem executados todos os elementos dessa classe (PEZZÈ, 2008) (BARBOSA *et al*, 2007).

O teste funcional pode ser complementado pelo teste de fluxo de controle, que inclui casos que não são identificados a partir dos requisitos especificados. Na prática, critérios de fluxo de controle são usados para avaliar a profundidade de suítes de teste derivadas a partir de critérios de teste funcional, identificando elementos do programa que não foram exercitados adequadamente (PEZZÈ, 2008).

Ferramentas automáticas servem para medir a adequação do fluxo de controle. O grau de cobertura de fluxo de controle é usado como um indicador de progresso. As primeiras ferramentas utilizadas para atividades de teste estrutural foram a RXVP80 e a TCAT (*Test- Coverage Analysis Tool*) e eram baseadas somente no fluxo de controle. Posteriormente foram desenvolvidas outras ferramentas como a Asset (*A System to Select and Evaluate Tests*), Protest, POKE-TOOL (*Potential Uses Criteria Tool for Program Testing*) e JaBUti *(Java Bytecode Understanding and testing*) impulsionando atividades de teste automatizados (BARBOSA *et al*, 2007).

O teste estrutural complementa os outros tipos de testes, pois verifica diferentes classes de defeitos, e também é ressaltado como importante para técnicas de manutenção, depuração e avaliação da confiabilidade dos sistemas. Inicialmente técnicas estruturais eram propostas para teste de unidade de programas procedimentais, limitando-se ao escopo da unidade. Com o tempo, o uso de técnicas estruturais ampliaram-se para testes de integração e posteriormente para testes de programas Orientado a Objetos, adataptando sua atividade em contextos diversos (BARBOSA *et al*, 2007).

São utilizados dentro do teste estrutural vários tipos de teste na implementação do programa como: teste de comando, teste de decisão, teste de condição, teste de caminho, teste de chamada de procedimento, teste de fluxo de dados e testes baseados na complexidade do programa (PEZZÈ, 2008).

2.1.3 Teste Baseado em Modelos

O conhecimento sobre o sistema é compreendido e reutilizado durante várias fases de desenvolvimento através do uso da modelagem: um modelo bem elaborado captura o que é essencial no sistema. Para isso, é muito importante determinar quais são os objetivos do teste, o que será testado e como (SIMÃO, 2007).

Segundo Pezzè (2008), "modelos podem ser representados de várias formas, modelos formais são exemplos de máquinas de estados finitos, que fornecem informação suficiente para permitir a geração automática de casos de teste. Modelos semi-formais, diagramas de classes e objetos, podem precisar de alguma avaliação humana para gerar casos de teste.".

Modelos são muito úteis para a geração de casos de teste, pois obtêm a efetividade custo-benefício de produzir requisitos formais ou semi-formais. Outro fator de extrema importância devido a utilidade de modelos é a possibilidade de automação do teste de software (PEZZÈ, 2008).

2.1.4 Teste Orientado a Objetos

O teste Orientado a Objetos utiliza as mesmas características abordadas no teste procedural, iniciando com testes funcionais através dos requisitos especificados pelos clientes, e posteriormente acresentam-se teste estruturais a partir dos testes unitários e de integração, baseados na estrutura do programa. Embora o teste orientado a objetos e o teste procedural apresentem características semelhantes, há diferenças significativas em suas abordagens (PEZZÈ, 2008).

O comportamento de um programa orientado a objetos depende do comportamento de seu objeto, visto que os métodos construídos não dependem apenas de seus parâmetros, mas também do estado do objeto instanciado.

Programas orientados a objetos incluem classes que herdam características de uma “super” classe através da herança (PEZZÈ, 2008). Portanto programas orientados a objetos apresentam suas próprias características como, comportamento dependente do estado, encapsulamento, herança, polimorfismo, classes abstratas, tratamento de exceções e concorrência.

O encapsulamento proporciona que uma alteração na implementação interna da classe não afete o resto do código. O polimorfismo possui uma única chamada ao método que pode ser ligada a diferentes métodos. As classes abstratas não permitem realizar qualquer tipo de instância. É necessário testá-las sem nenhum conhecimento sobre como podem ser instanciadas. O tratamento de exceções é necessário para verificar as exceções do ponto onde foi lançada até onde ela será tratada, restringindo o erro do usuário. A concorrência introduz novos tipos de possibilidades de falhas como condições de corrida e deadlocks, embora haja uma vantagem significativa com o uso de múltiplas threads (PEZZÈ, 2008).

Pode-se desmembrar o teste orientado a objetos em três fases, que são: intraclasse, interclasse, e sistema e aceitação, e testar classes individuais fazendo a interação e integração entre essas. Define-se como Intraclasse classes isoladas para o teste unitário, Interclasse a integração entre classes para o teste de integração e Sistema e Aceitação os teste funcionais verificando requistos do sistema (PEZZÈ, 2008).

Como a programação orientada a objetos depende diretamente do estado do objeto, seu teste pode ser guiado por uma máquina de estados. Os estados se alteram de acordo com o comportamento dos métodos, retornando resultados diversos. Não é interessante obter grande quantidade de estados abstratos a de estados concretos; os estados devem ser refinados apenas o suficiente para capturar interações. A máquina de estados pode ser extraída da especificação do comportamento pretendido (língua informal, natural), mesmo que o requisito

não demonstre claramente os estados do objeto. Portanto muitas vezes a máquina de estados representa uma parte do projeto ou requisito, e pode ser apresentada na forma de um diagrama de transição de estados a partir da compreensão do código do programa (PEZZÈ, 2008).

As técnicas de teste apresentadas nesta seção são algumas das técnicas disponíveis que podem ser utilizadas para aumentar a qualidade dos softwares desenvolvidos. O processo de qualidade de software, apresentado na próxima seção tem, como uma de suas atividades, a aplicação de atividades de teste.

2.2 Processo de qualidade de software

O principal objetivo da qualidade de software é garantir que um produto final satisfaça às expectativas do cliente. Para se obter qualidade no software ou em qualquer produto, é necessário que sejam aplicadas técnicas de avaliação, e métodos de testes durante todo o processo de desenvolvimento do produto.

As atividades cotidianas de diversas áreas necessitam, cada vez mais, de sistemas automatizados e informatizados e a dependência por estes está cada vez maior, seja em empresas de finanças, telecomunicação, transações comerciais, controle de tráfego e muitos sistemas críticos, nos quais o surgimento de defeitos pode causar grandes danos. Nesse sentido, empresas que desenvolvem sistemas de computação precisam estar cada vez mais preocupadas com a dependência de seus sistemas (WEBER, 2002).

Embora deva-se evitar que um sistema apresente defeitos, é inevitável a ocorrência destes, uma vez que as atividades desenvolvidas estão diretamente ligadas às pessoas.

Os defeitos são considerados desvios da especificação do sistema, processo posteiror a um estado errôneo, que é gerado por uma falha, que

representa dados incorretos, como a causa física ou algorítima do erro. (WEBER, 2002).

A Figura 1 retrata o universo de ocorrência de falhas, erros e defeitos: a falha como o universo físico, o erro como universo da informação e o defeito, posterior ao processamento, representa um desvio da especificação do cliente.

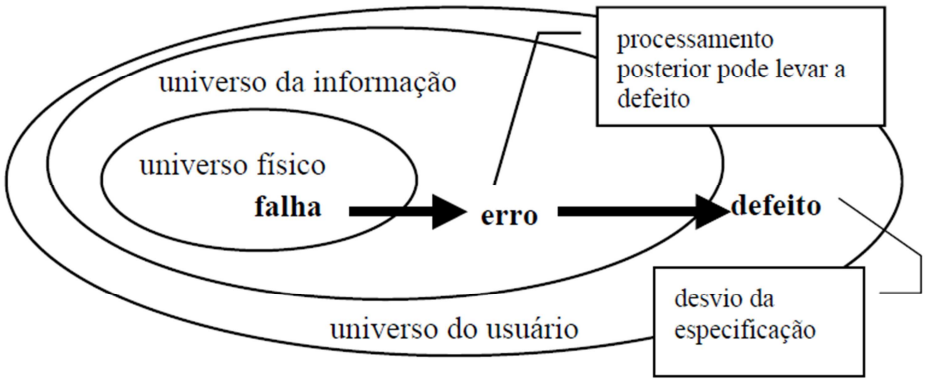

Figura 1: Modelo de 3 universos Fonte: (WEBER, 2002).

O objetivo de se minimizar as falhas é alcançar dependabilidade. Dado um sistema, diz-se que um serviço possui qualidade, analisando todos atributos da dependabilidade, apresentados a seguir, que correspondem às medidas numéricas. Assim será verificado se o sistema fornecido possui confiança (LAPRIE, 2007).

Para se obter uma melhor compreensão sobre dependabilidade é importante conhecer seus principais atributos, que são: segurança de funcionamento, disponibilidade, manutenabilidade, segurança, comprometimento do desempenho e confiabilidade. Segurança de funcionamento indica que um sistema deva estar operando corretamente, e que não prejudique nenhum outro sistema que dependa de seu funcionamento. A

ERROR: ioerror
OFFENDING COMMAND: image

STACK: