PAULO ANDRÉS VILLEGAS VIERA

AVALIAÇÃO DE IMPACTO DE DEFEITOS DE SOFTWARE SOBRE O PROCESSO DE PREVISÃO DE CONFIABILIDADE

São Paulo
2016

PAULO ANDRÉS VILLEGAS VIERA

AVALIAÇÃO DE IMPACTO DE DEFEITOS DE SOFTWARE SOBRE O PROCESSO DE PREVISÃO DE CONFIABILIDADE

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de Software.

São Paulo
2016

PAULO ANDRÉS VILLEGAS VIERA

AVALIAÇÃO DE IMPACTO DE DEFEITOS DE SOFTWARE SOBRE O PROCESSO DE PREVISÃO DE CONFIABILIDADE

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientador: Prof. Dr. Kechi Hirama

São Paulo
2016

Catalogação-na-publicação

VIERA, PAULO
AVALIAÇÃO DE IMPACTO DE DEFEITOS DE SOFTWARE SOBRE O PROCESSO DE PREVISÃO DE CONFIABILIDADE / P. VIERA -- São Paulo, 2016.
71 p.

Monografia (MBA em Tecnologia de Software) - Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia.

1.QUALIDADE DE SOFTWARE 2.CONFIABILIDADE DE SOFTWARE 3.MODELOS MATEMÁTICOS 4.ESTUDO DE CASO I.Universidade de São Paulo. Escola Politécnica. PECE – Programa de Educação Continuada em Engenharia II.t.

DEDICATÓRIA

Dedico este trabalho à Deus, à minha mãe e aos colegas de curso que, apesar de não conseguirem completar a jornada, contribuíram para a elaboração deste trabalho.

E à Raquel, SPS2R2!

AGRADECIMENTOS

À minha mãe pelo amor e apoio durante a jornada que chega à conclusão com este trabalho.

Ao meu orientador Kechi Hirama pela orientação e apoio na elaboração deste trabalho.

Aos colegas de curso, que apesar de alguns não chegarem até o fim do curso, contribuíram para a elaboração deste trabalho.

RESUMO

O objetivo é investigar a relevância em se utilizar a taxa da correção de defeitos junto com a de cobertura de código como métrica dentro de um modelo de previsão de crescimento de confiabilidade de software. No levantamento bibliográfico, constatou-se nas fontes estudadas a ausência da utilização da taxa de correção de defeitos como uma métrica junto com a taxa de cobertura do código, embora tais métricas tenham sido propostas em 2004 e 2007, respectivamente. Considerando-se esse cenário, foram aplicadas as duas métricas em um estudo de caso de software elaborado por uma instituição pública brasileira, a fim de avaliar o desempenho da previsão de confiabilidade das métricas citadas num cenário real. Portanto, este trabalho apresenta a análise dos resultados e perspectivas para pesquisas nesta área de trabalho.

Palavras-chave: Software. Qualidade. Confiabilidade. Modelo de crescimento de confiabilidade de software.

ABSTRACT

The aim of this study is to investigate the relevance of using the fault removal rate combined with code coverage rate as metrics in a software reliability growth model. While researching the specialized literature, it was noticed the absence of the use of fault removal rate as a metric along with the code coverage rate, even though these metrics have been proposed in 2004 and 2007, respectively. This study applied the two metrics using a case study of a software developed by a Brazilian public company in order to evaluate the performance of reliability estimation in a real-world scenario. This paper presents the results of the analysis and suggests directions for future research.

Keywords: Software. Quality. Reliability. Software reliability growth model.

LISTA DE FIGURAS

Pág.

SUMÁRIO

Pág.

1. INTRODUÇÃO

Este capítulo apresenta as motivações, o objetivo, as justificativas e a estrutura do trabalho.

1.1 Motivações

A utilização de software é cada vez mais intensa nas mais diversas funções empresariais e pessoais. Com isso, existe demanda para o desenvolvimento de software de alta qualidade. No entanto, a qualidade destes é um fator importante.

Neste sentido, ISO/IEC 25010 (ISO, 2011) apresenta um modelo com essa finalidade, de forma que um conjunto de características prevê uma estrutura de trabalho que permite especificar requisitos de qualidade e avaliar a qualidade de um produto. Para tanto, o modelo é composto por oito características: adequação funcional, eficiência de desempenho, compatibilidade, usabilidade, confiabilidade, proteção, manutenibilidade e portabilidade.

Segundo a ISO/IEC 25010 (ISO, 2011), confiabilidade é o grau no qual um sistema, software ou componente funciona sob determinadas condições por um determinado período de tempo. Por isso, a confiabilidde é uma das características mais importantes referentes à qualidade de software (YAMADA , 2014, p. 1).

Assim, para mensurar e gerenciar a confiabilidade do software, modelos matemáticos, tipicamente NHPP (do inglês, *Non-Homogeneous Poisson Process*), podem ser usados para fazer previsões sobre a quantidade de defeitos presentes no software em um dado momento e a possibilidade de encontrar novos defeitos durante a execução de um software por um determinado período (JAFFAL; TIAN, 2014, p. 246). Tais modelos

matemáticos são chamados de SRGM (do inglês, Software *Reliability Growth Model*). Um SRGM é uma ferramenta crítica para controlar a qualidade do software. No entanto, os modelos enfrentam dificuldades para representar exatamente, por meio de modelos matemáticos, o crescimento da confiabilidade de software no mundo real (OKAMURA; DOHI, 2008, p. 19). Além disso, nenhum SRGM proposto até o momento conseguiu atuar em todos os cenários possíveis de desenvolvimento de software (ALMERING *et al.*, 2007, p. 87; IMMONEN;NIEMELA, 2007, p. 49), sendo necessário um cuidado extra para se determinar o modelo mais adequado a ser aplicado, já que um modelo inadequado poderia fornecer dados que resultassem em decisões equivocadas dentro de um projeto de desenvolvimento de software (ULLAH; MORISO;VETRO, 2012, p. 188).

1.2 Objetivo

O objetivo do trabalho é avaliar a influência que o impacto dos defeitos encontrados exerce sobre o software no processo de previsão de confiabilidade do software. Também, o SRGM proposto por Cai e Lyu (CAI; LYU, 2007, p. 20), juntamente com o método proposto por Gokhale, Lyu e Trivedi (2004, p. 218) para calcular a taxa de remoção de defeitos foram combinados. O modelo resultante foi chamado de modelo Cai-Gokhale e este foi aplicado em um estudo de caso de software de *workflow* desenvolvido por uma instituição pública brasileira. A influência da taxa de remoção de defeitos sobre a previsão de confiabilidade foi discutida.

1.3 Justificativa

Nos últimos anos, diversos SRGM foram propostos utilizando-se diversas abordagens e métricas. Os primeiros modelos utilizaram uma única métrica. Adicionando-se outras

métricas, o desempenho da previsão de confiabilidade apresentou melhoras. Huang, Kuo e Lyu(2007, p. 198) apresentaram dois SRGSM, usando o esforço de teste como uma métrica sob o cenário de depuração perfeita e imperfeita, isto é, onde um ou mais defeitos podem ser inseridos no processo de remoção de um defeito. Já Wang, Moriso e Vetro (2007, p. 411) incorporaram a estrutura do software como um fator de influência, mensurando a confiabilidade de cada componente do software na previsão de confiabilidade do software como um todo. Por fim, Singh *et al.* (2007, p. 360) consideraram a dependência entre defeitos, isto é, não encarando cada um como independente de outros, mas sim, eventualmente, relacionando defeitos que provocam outros defeitos, resultando em um efeito em cascata.

No entanto, o aumento de métricas utilizadas aumenta a incerteza ao se obter as próprias medidas. Isto afeta a previsão de confiabilidade (LI; XIE; HUING, 2010, p. 3560), mesmo que existam maneiras de determinar o grau de incerteza dentro de uma previsão de confiabilidade (CHANDRAN; DIMOV; PUNNEKKAT, 2010, p. 227). Assim, mesmo a maneira como o modelo lida com as falhas, em si, afeta a previsão, quer seja considerando-se uma quantidade limitada ou ilimitada de possíveis defeitos no software (YADAV; KHAN, 2009, p. 116).

Okamura, Etani e Dohi (2010, p. 32) abordaram a questão da incerteza no momento de obtenção das medidas, utilizando técnicas estatísticas para gerar um SRGM mais simples e, consequentemente, com um nível menor de incerteza na previsão de confiabilidade. Quadri, Ahmad e Faroog (2011, p. 7), por sua vez, apresentaram um modelo onde o esforço de teste seguiu uma distribuição exponencial generalizada, assumindo que a taxa de surgimento de defeitos e o esforço de teste são proporcionais à quantidade restante de defeitos no software. Já Kapur, Pham e Anand (2011, p. 333) observaram que, a remoção de apenas um defeito pode provocar o surgimento de outros defeitos e a quantidade de defeitos removidos não necessariamente corresponde à mesma quantidade de defeitos observados. Por fim, Peng *et al.* (2014, p.

38) consideraram o esforço de detecção de defeitos e o esforço de correção dos mesmos como dois fatores distintos que podem ser incorporados em outros SRGM.

No levantamento bibliográfico deste trabalho, constatou-se que dois fatores que repetidamente aparecem nos SRGM são: o tempo gasto com testes, como no modelo apresentado por Singh *et al.* (2007, p. 361), e o processo de depuração, ou correção de defeito (ULLAH; MORISIO; VETRO, 2012, p. 187). Segundo Gokhale, Lyu e Trivedi (2004, p. 213), um fator que deve ser levado em consideração é o tempo necessário para corrigir um defeito. Também, Gokhale, Lyu e Trivedi (2004, p. 213) forneceram um método para correlacionar a taxa de detecção de defeitos com a taxa de remoção de defeitos. Durante os estudos para este trabalho, não foram encontrados modelos que utilizassem tanto a taxa de cobertura de código quanto a taxa de correção de defeitos como fatores para previsão de confiabilidade.

Este trabalho parte do pressuposto que o tempo de remoção de um defeito impacta na taxa de detecção de novos defeitos, seja porque a equipe de testes precisa aguardar que os defeitos sejam removidos (sob demanda ou por lote), seja porque defeitos não-detectados foram removidos juntos com os defeitos já detectados.

Portanto, o presente trabalho verifica a influência que a taxa de remoção de defeitos exerceu na previsão de confiabilidade, junto com a taxa de cobertura de código. Isso se dá por meio da aplicação de um SRGM utilizando esses dois fatores, fazendo uma análise do impacto da taxa de remoção de defeitos sobre a previsibilidade de confiabilidade de software.

1.4 Estrutura do Trabalho

Este trabalho está organizado na seguinte estrutura:

Capitulo 1 – Introdução – Este capítulo descreve as motivações, o objetivo, as justificativas e estrutura do trabalho.

Capítulo 2 – Qualidade de Software – Este capítulo trata de definições de qualidade de software, sua influência no desenvolvimento de software, especialmente na fase de testes. Também, define os termos usados como erro, defeito e falha, métrica e medida e confiabilidade de software.

Capítulo 3 – Modelos de Crescimento de Confiabilidade de Software – Este capítulo trata de modelos de crescimento de confiabilidade de software, modelos matemáticos envolvidos e a história da evolução destes, a aplicação de um ou dois fatores e o limite de fatores impostos pelas incertezas inerentes ao processo de previsão de confiabilidade de software.

Capítulo 4 – Modelo de Cai-Lyu e Método de Gokhale – Este capítulo apresenta o modelo proposto por Cai e Lyu (2007), suas características e flexibilidade que o tornam uma proposta interessante para análise proposta pelo presente trabalho. Também é apresentado o método de Gokhale, Lyu e Trivedi (2004), para estimar a taxa de remoção de defeitos e a influência que esta exerce sobre a previsão de confiabilidade de software.

Capítulo 5 – Modelo Proposto – Este capítulo apresenta uma proposta para avaliar a influência do impacto de defeitos sobre a previsão de confiabilidade de software. De forma que o SRGM proposto por Cai e Lyu (2007) é combinado com o método de Gokhale, Lyu e Trivedi (2004), sendo o modelo resultante chamado de modelo Cai-Gokhale.

Capítulo 6 – Aplicação e Análise – Este capítulo apresenta a aplicação do modelo Cai-Gokhale em dados coletados durante o desenvolvimento de um software por uma instituição pública brasileira, comparando-se os resultados do modelo Cai-Gokhale com o modelo Cai-Lyu.

Capitulo 7 – Considerações Finais – Este capítulo apresenta conclusões do trabalho, contribuições e aponta possíveis trabalhos futuros sobre o tema.

REFERÊNCIAS – Apresenta a lista de fontes usadas para a elaboração deste trabalho.

ANEXOS – Apresenta as tabelas com os dados utilizados no estudo de caso utilizado no Capítulo 6.

2. QUALIDADE DE SOFTWARE

A definição de qualidade é difícil, assim com é ainda mais difícil garantir a qualidade de algum produto (HIRAMA, 2011). O software é um produto resultante da necessidade do cliente e, como produto, precisa ter qualidade. Considerando-se que a utilização de software é cada vez mais intensa nas diversas funções empresariais e pessoais, há demanda para software de qualidade.

A qualidade de software é uma combinação complexa de características que variam de acordo com as aplicações e clientes que o solicitam. Também, as necessidades de software solicitadas pelos clientes estão tornando-se cada vez mais robustas, e assim, como resposta para atender a essa gama de necessidades surgem diversas tecnologias (PRESSMAN, 1995).

Além disso, tem havido uma conscientização maior da importância do gerenciamento de qualidade de software e da adoção de técnicas de gerenciamento de qualidade envolvidas no desenvolvimento de software. Alcançar um alto grau de qualidade nos produtos ou serviços é o principal objetivo da maioria das organizações. Assim, desenvolver e entregar produtos com baixa qualidade e reparar os problemas e as deficiências existentes depois que os produtos foram entregues ao usuário, não é mais aceitável atualmente (SOMMERVILLE, 2007).

Para se chegar em um produto de software ou para a manutenção de um já existente são executadas diversas atividades, gerando-se diferentes subprodutos que são necessários para a concepção do software. Essas atividades podem ser agrupadas em processos, os quais definem, em geral, o conjunto de atividades, ferramentas e métodos utilizados no desenvolvimento de um determinado produto (HUMPHREY, 1989, p. 284).

Pressman (1995) reforçou a tese que diversos esforços foram feitos para desenvolver medições precisas da qualidade de software, sendo que esses, às vezes, frustraram-se pela natureza subjetiva da atividade. Neste contexto, visando avaliar a qualidade de produto de software, foram criadas e atualizadas periodicamente normas internacionais e nacionais para tal finalidade.

Segundo a norma de qualidade ISO/IEC 25010 (ISO, 2011), a qualidade de software é definida como “o grau em que o sistema satisfaz as necessidades implícitas e explícitas de seus vários *stakeholders*”. Isto significa que as necessidades explícitas são expressas na definição de requisitos propostos pelo cliente, ao passo que as necessidades implícitas são aquelas que podem não estar expressas nos documentos, mas que são necessárias aos clientes.

O modelo apresentado na norma ISO/IEC 25010 (ISO, 2011) consiste em um modelo de qualidade composto por um conjunto de características que resultam em uma estrutura de trabalho que permite especificar requerimentos de qualidade e avaliar a qualidade de um produto.

A qualidade pode ser medida ao longo do processo de engenharia de software e depois que este foi entregue ao cliente e aos usuários. Na maioria dos empreendimentos técnicos, as medições de qualidade ajudam os profissionais envolvidos a entender o processo técnico usado para desenvolver um produto, como também o próprio produto. O processo é medido com a intenção de aprimorá-lo. Assim, o produto também é medido com a finalidade de aumentar a sua qualidade (PRESSMAN, 1995).

2.1 Modelo de Qualidade da Norma ISO/IEC 25010

O modelo de qualidade da norma ISO/IEC 25010 é composto por oito características: 1) adequação funcional; 2) eficiência de desempenho; 3) compatibilidade; 4) usabilidade; 5) confiabilidade; 6) proteção; 7) manutenibilidade; e 8) portabilidade (ISO, 2011).

2.1.1 Adequação funcional

A adequação funcional é o grau em que um produto ou sistema fornece funções que correspondam às necessidades explícitas e implícitas quando usado sob condições especificadas. Ela é composto por três subcaracterísticas: 1) **completude funcional**, que é o grau em que o conjunto de funções abrange todas as tarefas e objetivos específicos dos usuários; 2) **correção funcional**, que é o grau ao qual um produto ou sistema fornece os resultados corretos com a precisão necessária; e 3) **adequabilidade funcional**, que é o grau em que as funções facilitam a realização das tarefas e objetivos especificados.

2.1.2 Eficiência de desempenho

A eficiência de desempenho é a eficiência do produto ou sistema em relação à quantidade de recursos utilizados sob condições estabelecidas.

É composta por três subcaracterísticas: 1) **comportamento temporal**, que é grau em que os tempos de resposta e de processamento e taxas de transferência de um produto ou sistema, no desempenho das suas funções, atendem aos requisitos; 2) **utilização de recursos**, que é o grau em que as quantidades e tipos de recursos usados em um produto ou sistema, no desempenho das suas funções, para atender aos requisitos; e

3) **capacidade**, que é o grau em que os limites máximos de um parâmetro, de produto ou sistema, atendem os requisitos.

2.1.3 Compatibilidade

A compatibilidade é o grau em que um produto, sistema ou componente pode trocar informações com outros produtos, sistemas ou componentes, e/ou realizar suas funções necessárias, enquanto compartilha o mesmo ambiente de hardware ou software. Ela é composto por dois subcaracterísticas: 1) **co-existência**, que é o grau em que um produto pode desempenhar as suas funções de forma eficiente enquanto compartilha um ambiente e/ou recursos comums com outros produtos, sem impacto negativo em qualquer outro produto; e 2) **interoperabilidade**, que é o grau em que dois ou mais sistemas, produtos ou componentes podem trocar informações e utilizar as informações trocadas.

2.1.4 Usabilidade

A usabilidade é o grau em que um produto ou sistema pode ser usado por usuários específicos para alcançar objetivos específicos com efetividade, eficiência e satisfação em um contexto de uso específico.

Ela é composta por seis subcaracterísticas: 1) **reconhecimento de adequação**, que é o grau em que os usuários podem reconhecer se um produto ou sistema é adequado para as suas necessidades; 2) **capacidade de aprendizado**, que é o grau em que um produto ou sistema pode ser usado por usuários específicos para alcançar objetivos específicos de aprendizagem para usar o produto ou sistema com eficácia, a eficiência, a inexistência de risco e satisfação dentro de um contexto de uso específico; 3)

operabilidade, que é o grau em que um produto ou sistema tem atributos que o tornam fácil de operar e controlar; 4) **proteção de erros do usuário**, que é o grau em que um sistema protege os usuários de cometerem erros; 5) **estética da interface do usuário**, que é o grau em que uma interface do produto ou sistema permite a interação agradável e satisfatória para o usuário; e 6) **acessibilidade**, que é o grau cujo produto ou sistema pode ser usado por pessoas com a gama mais ampla de características e capacidades para atingir um objetivo específico em um contexto de uso.

2.1.5 Confiabilidade

A confiabilidade é o grau em que um sistema, produto ou componente executa funções especificadas sob condições específicas por um período de tempo estabelecido. Ela é composta por quatro subcaracterísticas: 1) **maturidade**, que é o grau em que um sistema, produto ou componente satisfaz as necessidades de confiabilidade em condições normais de operação; 2) **disponibilidade**, que é o grau em que um sistema, produto ou componente está operacional e acessível quando for necessária sua utilização; 3) **tolerância a defeito**, que é o grau em que um sistema, produto ou componente opera como pretendido, apesar da presença de defeitos de hardware ou *software*; e 4) **recuperabilidade**, que é o grau em que, no caso de uma interrupção ou uma falha, o produto ou sistema pode recuperar os dados diretamente afetados e re-estabelecer o estado desejado do sistema.

2.1.6 Proteção

A proteção é o grau no qual um produto ou sistema protege informações e dados, de modo que as pessoas, outros produtos ou sistemas possuam o apropriado grau de acesso aos dados, conforme seus respectivos tipos e níveis de autorização. Ela é

composto por cinco subcaracterísticas: 1) **confidencialidade**, que é o grau em que um produto ou sistema garante que os dados estarão acessíveis somente por pessoas autorizadas a ter acesso; 2) **integridade**, que é o grau em que um sistema, produto ou componente impede o acesso não autorizado, ou alteração de, programas de computador ou dados; 3) **não-repúdio**, que é o grau em que as ações ou eventos ocorridos dentro do produto ou sistema podem ter suas ocorrências provadas, de modo que os eventos ou ações não podem ser repudiados mais tarde; 4) **atribuição**, que é o grau em que as ações de uma entidade pode ser atribuídas exclusivamente a ela; e 5) **autenticidade**, que é o grau em que a identidade de um assunto ou recurso podem ser provado como único reivindicados.

2.1.7 Manutenibilidade

A manutenibilidade é grau de eficácia e eficiência com que um produto ou sistema pode ser modificado pela designada equipe de manutenção.

Ela é composta por cinco subcaracterísticas: 1) **modularidade**, que é o grau em que um programa de computador ou sistema é composto por componentes discretos, de tal forma que uma mudança em um determinado componente tem impacto mínimo sobre os demais; 2) **reusabilidade**, que é grau em que um componente pode ser utilizado em mais de um sistema, ou na construção de outros componentes; 3) **analisabilidade**, que é o grau de eficácia e a eficiência com a qual é possível avaliar o impacto de uma mudança sobre uma ou mais partes de um produto ou de um sistema, quer seja para diagnosticar deficiências ou causas de falhas de um produto, bem como para identificar componentes a serem modificados; 4) **modificabilidade**, que é o grau cujo produto ou sistema pode ser modificado de forma eficaz e eficiente sem a introdução de defeitos ou degradação da qualidade do produto existente; e 5) **testabilidade**, que é o grau de eficácia e eficiência com a qual é possível estabelecer critérios de teste de um sistema,

produto ou componente de modo que testes podem ser realizados para determinar se esses critérios foram ou não cumpridos.

2.1.8 Portabilidade

A portabilidade é o grau de eficácia e eficiência com que um sistema, produto ou componente pode ser transferida de um hardware, software ou outro ambiente operacional, assim como para outro uso. Ela é composta por três subcaracterísticas: 1) **adaptabilidade**, que é o grau cujo produto ou sistema pode eficaz e eficientemente ser adaptado para outro *hardware*, software ou outros ambientes operacionais ou de uso; 2) **facilidade de instalação**, que é o grau de eficácia e eficiência com que um produto ou sistema pode ser instalado e/ou removido de um ambiente especifico com sucesso; e 3) **facilidade de substituição**, que é o grau ao qual um produto pode substituir outro produto de software especificado para o mesmo fim no mesmo ambiente.

No presente trabalho utilizou-se a definição da ISO/IEC 25010 (ISO, 2011) para confiabilidade como parâmetro de qualidade de *software*.

2.2 Métrica e Medida

Métrica é definida por Bohem, Brown e Lipow (1976) como a extensão ou grau em que um sistema ou produto possui ou exibe uma certa característica. Métricas surgem da necessidade de avaliar um artefato objetivamente e são utilizadas no desenvolvimento de software para assegurar características como a confiabilidade. Medida é a indicação quantitativa de uma certa característica de um sistema ou produto.

2.3 Erro, Defeito e Falha

Erro é a ação humana que produz um resultado incorreto, por exemplo, quando o desenvolvedor, ao interpretar equivocadamente os requisitos do software, comete erros na codificação (HIRAMA, 2011).

Já o defeito é uma implementação incorreta dentro de um artefato, seja um defeito no software ou dentro de um manual de instruções. Eles são inseridos em artefatos por meio de erros (HIRAMA, 2011).

Por fim, falha é a incapacidade de um sistema ou componente em executar as funções requeridas dentro de um nível de desempenho definido. Assim, a falha pode ser entendida como a manifestação de um defeito dentro de um artefato (HIRAMA, 2011).

Com bases nestas definições, é possível afirmar que erros causam defeitos, mas defeitos não necessariamente produzem falhas. No entanto, erros podem ocorrer em qualquer fase de desenvolvimento de software.

2.4 Considerações do Capítulo

O desenvolvimento de software envolve a execução de diversas atividades gerando-se diferentes subprodutos que são necessários para a concepção do software (HUMPHREY, 1989, p. 284). O software desenvolvido visa satisfazer as necessidades implícitas e explícitas dos *stakeholders*, e o grau em que o software satisfaz tais necessidades é a definição da ISO/IEC 25010 (ISO, 2011) para qualidade.

Durante o desenvolvimento, erros introduzidos por ação humana afetam diretamente a qualidade do software, produzindo resultados incorretos. Nesse contexto, a confiabilidade do software é o grau em que um sistema, produto ou componente executa funções especificadas sob condições específicas por um período de tempo estabelecido (ISO, 2011). Ou seja, por quanto tempo um software consegue executar as funções esperadas pelos *stakeholders* sem a ocorrência de uma falha.

3. MODELOS DE CRESCIMENTO DE CONFIABILIDADE DE SOFTWARE

Uma ferramenta importante para ajudar um gestor de projeto de software a aferir a sua confiabilidade, isto é, por quanto tempo o software pode funcionar sem que falhas ocorram, é o modelo de crescimento de confiabilidade de software, ou como é conhecido: SRGM.

Assim, para utilizar tal modelo, é preciso coletar informações sobre o projeto, sendo que cada SRGM possui um ou mais tipos de informação como entrada, também chamados de fatores.

3.1 Confiabilidade de Software

Segundo Sommerville (2011, p. 656), os interessados em um software possuem diversas necessidades que vão além das suas funcionalidades. Portanto, o nível no qual o software atende a todas essas necessidades, além de atender as funcionalidades necessárias, é o nível de qualidade do sistema.

Nesse sentido, uma dessas necessidades é a confiabilidade do software. Segundo a ISO/IEC 25010 (ISO, 2011), confiabilidade é o grau no qual um sistema, software ou componente funciona sob determinadas condições por um determinado período de tempo. Desta maneira, a confiabilidade pode ser descrita como a probabilidade de um software apresentar o resultado esperado durante um certo tempo. Porém, isto é diferente de disponibilidade, onde o software apresenta o resultado esperado no momento que o usuário solicita o resultado. Logo, a confiabilidade implica em uso contínuo do software (SOMMERVILLE, 2011, p. 295).

Segundo Yamada (2014, p. 1), a confiabilidade de software é uma das características mais importantes referentes à qualidade. Assim, é importante ressaltar que a confiabilidade está ligada ao cenário de uso do software, sendo que o mesmo pode ter uma confiabilidade diferente para um usuário iniciante, do que para um usuário avançado, pois este pode explorar mais recursos e encontrar defeitos que o usuário iniciante não encontraria.

De acordo com Sommerville (2011, p. 300), não é possível ter plena certeza da completa ausência de defeitos em um software. Sendo assim, a confiabilidade pode ser estimada calculando-se a provável quantidade de defeitos persistentes em um software. Por isso, são efetuados testes levando-se em consideração os dados obtidos durante a revisão de especificações, *design* e codificação.

Para tanto, um modelo adequado de dados de teste ajuda a identificar a taxa de defeitos detectados em um software (CHANDRAN; DIMOV; PUNNEKKAT, 2010, p. 229). Assim, é possível estimar a quantidade de defeitos ainda presentes no software e, portanto, estimar durante quanto tempo o software apresentará o resultado esperado.

Por fim, a confiabilidade também pode ser usada para tomar decisões estratégicas sobre o software como, por exemplo, indicar a quantidade de tempo e recursos que devem ser aplicados para eliminar mais defeitos. Sendo que Okamura, Etani e Dohi (2010, p. 31) defenderam a tese de que a confiabilidade pode ser utilizada para determinar o momento do lançamento de um software no mercado.

3.2 Métodos para a Avaliação da Confiabilidade de Software

Segundo Chandran, Dimov e Punnekkat (2010, p. 229), há três grupos de métodos para avaliar a confiabilidade de um software: testes, simulação e *feedback* de usuários.

De acordo com Sommerville (2011, p. 206), o processo de testes possui dois objetivos distintos: demonstrar que o software atende às especificações e descobrir falhas para as devidas correções. Para isso, a avaliação de confiabilidade de software utiliza os dados coletados durante a busca de falhas. Portanto, o teste de software é a abordagem mais comum, sendo um elemento crítico na garantia de qualidade de software , representando a revisão das suas especificações, *design* e codificação.

O segundo método, isto é, a simulação, ao contrário do método de teste para avaliar a confiabilidade, parte do pressuposto de que a confiabilidade de software não depende somente de sua estrutura, mas também do tempo de execução, frequência de reutilização de componentes, tempo gasto, interações entre os componentes, etc (CHANDRAN; DIMOV; PUNNEKKAT, 2010, p. 230). Assim, o projeto dos requisitos e código da estrutura do software são revistos e a sua execução fornece o desempenho.

Por fim, outra abordagem para se obter dados que avaliem a confiabilidade de um software é por meio do *feedback* dos usuários. De modo diferente, os dados são obtidos após o lançamento do software no mercado e refletem a sua real utilização pelos usuários. Por isso, diferente da abordagem de testes, os dados provém do comportamento imprevisível do usuário, ao invés de usar um modelo de dados de teste ou testes planejados. Tipicamente, os dados sobre falhas do sistema são coletados por relatórios enviados pelo usuário e, posteriormente, cruzados com a frequência dos relatórios, a fim de se estimar a quantidade restante de defeitos, avaliando-se a confiabilidade do software.

3.2.1 Modelo de Crescimento de Confiabilidade de Software

Para avaliar a confiabilidade do software, a abordagem mais comum é a utilização de testes e coleta de dados sobre os resultados de falhas nos testes. Porém, quando os

dados de falha não estão disponíveis ou quando as falhas não são observadas durante os testes, pode-se adotar uma abordagem de estimação bayesiana para estimar a probabilidade de uma falha surgir (CHANDRAN; DIMOV; PUNNEKKAT, 2010, p. 229). Segundo Jaffal e Tian (2014, p. 247), para fazer a previsão da quantidade de defeitos restantes no software e o tempo de execução necessário para as correspondentes falhas manifestarem-se, modelos matemáticos podem ser usados. Tais modelos, segundo Ullah, Morisio e Vetro (2012, p. 187), assumem que a confiabilidade do software cresce após um defeito ser detectado e consertado. Com essa previsão, é possível decidir se é válido prosseguir com os testes ou interrompê-los.

Esses modelos matemáticos são chamados de SRGM (Software *Reliability Growth Model*), sendo que Yamada (2014, p. 40) definiu o SRGM como um modelo de análise matemática com a finalidade de medir e avaliar quantitativamente a confiabilidade do software. Desta maneira, o SRGM pode ser aplicado durante o processo de projeto, codificação, integração, testes e após o lançamento do software. No entanto, aplicar um SRGM durante o projeto e codificação não gera resultados aceitáveis, já que o software ainda é instável e não possui todas as funcionalidades necessárias, dependendo do estágio de desenvolvimento.

Embora seja possível executar um SRGM após o lançamento, isto pode ser tardio para evitar impactos negativos com os usuários. Por isso, Almering *et al.* (2007, p. 83) argumentaram que o momento ideal para executar um SRGM é durante o processo de testes.

Os modelos de confiabilidade, segundo Ullah, Morisio e Vetro (2012, p. 187), podem ser classificados de acordo com o tipo de teste de software que é realizado:

– quando o teste considera apenas os requisitos, sem atentar-se à estrutura interna do software, o teste é chamado de caixa-preta, sendo que este considera apenas o

resultado da interação do software com os dados de entrada. Por isso, os modelos de confiabilidade que utilizam dados provenientes deste tipo de teste são chamados de modelos de confiabilidade do tipo caixa-preta;

– quando o teste considera, além dos requisitos, a estrutura interna e os componentes do software, o teste é chamado de caixa-branca e os modelos de confiabilidade que consideram a estrutura interna do software para estimar a confiabilidade são chamados de modelos de confiabilidade do tipo caixa-branca.

3.2.2 Modelos de Confiabilidade do Tipo Caixa-Preta

Os modelos de confiabilidade do tipo caixa-preta são classificados em diferentes tipos: modelos de previsão antecipada, SRGM, modelos baseados no domínio de entrada e modelos híbridos de caixa-preta (ULLAH; MORISIO; VETRO, 2012, p. 187).

Geralmente, os modelos de caixa-preta são usados quando os dados dos resultados de teste ou informações sobre falhas, tipicamente provenientes de um *feedback* de usuários, estão disponíveis. Sabe-se que os modelos do tipo caixa-preta usam as informações de falha do software observadas e fazem previsões sobre falhas futuras, refletindo, assim, o crescimento da confiabilidade do software.

Segundo Yamada (2014), os SRGMs são classificados em três grandes grupos: modelos finitos, modelos infinitos com base no número total de falhas expressas com tempo infinito e modelos bayesianos. A sua notoriedade dá-se pelo fato de os SRGMs estão em uso desde início dos anos 1970.

3.2.3 Modelos de Confiabilidade do Tipo Caixa Branca

De acordo com a forma de apresentação da estrutura interna de sistemas de software, Chandran, Dimov e Punnekkat (2010, p. 230) dividiram os modelos de confiabilidade do tipo caixa-branca em três subgrupos principais: modelos baseados em estados finitos, modelos baseados em caminhos e modelos aditivos.

Também, utilizando-se a abordagem da avaliação de confiabilidade do tipo caixa-branca, há quatro passos básicos: identificação de módulos/componentes que constituem o sistema de software, construção de modelo de arquitetura, definição da falha de comportamento dos componentes e combinação do modelo de arquitetura com a definição da falha de comportamento dos componentes.

3.2.4 Modelos Matemáticos

O SRGM é um modelo matemático que especifica a forma geral do processo de falha de software em função de fatores como a introdução de defeitos, a remoção de defeitos e o ambiente operacional. Assim, um SRGM é composto por diferentes parâmetros. E, nesse sentido, parâmetro é determinado como uma constante ou variável arbitrária que aparece em uma expressão matemática, cada valor do que restringe ou determina a forma específica de uma expressão.

Sabe-se que a taxa de defeitos, isto é, defeitos por unidade de tempo, de um sistema de software é, geralmente, decrescente ao longo do processo de desenvolvimento, devido ao processo de detecção e remoção de defeitos. Desta maneira, a modelagem de confiabilidade é feita para estimar a forma da curva da taxa de defeitos por meio de estimativas estatísticas dos parâmetros associados ao modelo selecionado.

Segundo Ullah, Morisio e Vetro (2012, p. 187), a modelagem da curva de taxa de defeitos tem dois objetivos: estimar o tempo extra de execução necessária de teste para atender a um objetivo específico de confiabilidade e para identificar a confiabilidade esperada do software quando o produto é liberado.

3.2.4.1 Modelo de Processo Poisson Discreto Não-Homogêneo

Um processo Poisson é um conceito matemático usado, comumente, em estatística e que representa uma série de eventos aleatórios distribuídos ao longo do tempo. Por isso, o intervalo de tempo entre dois eventos próximos segue uma distribuição exponencial. Como consequência, o processo Poisson não-homogêneo permite que a taxa de aparição de eventos varie ao longo do tempo.

Segundo Kapur *et al.* (2011, p.332), os modelos NHPP (*Non-Homogeneous Poison Process*) descrevem a observação de falhas de software por meio de uma curva exponencial. De modo que os modelos utilizam dados observados durante o processo de testes, por exemplo, o tempo entre falhas, para estimar o número residual de defeitos no software e o tempo de teste necessário para os detectar. Assim, estes modelos podem lidar com dados de intervalo e ponto e assumem que a intensidade da taxa de falhas diminuiu conforme os defeitos foram sendo detectados e removidos (ALMERING et al., 2007, p. 83).

A modelagem da curva da taxa de defeitos assume uma forma côncava. Por isto, Almering *et al.* (2007, p. 83) argumentaram que a forma côncava geral da função está ligada ao fato de que os defeitos restantes no software são mais sutis e muitas vezes mais difíceis de se detectar e corrigir e, portanto, demandam mais tempo de testes.

Os modelos NHPP permitem distinguir duas abordagens para a modelagem da confiabilidade. Modelos finitos de defeitos partem do pressuposto de que o software possui um número finito de defeitos e, eventualmente, pode ficar livre de defeitos. Em tais modelos, a curva de taxa de defeitos aproxima-se de um valor finito. Mas, nos modelos infinitos, assume-se que o número de defeitos observados é infinito (ALMERING *et al.*, 2007, p. 83).

3.3 Utilização de Um Fator

No contexto de utilização de SRGM, as métricas empregadas pelo modelo são chamadas de fatores. E, dentre os diversos SRGM que foram propostos, os primeiros modelos usavam uma única métrica como parâmetro para a avaliação de confiabilidade.

A maioria dos SRGMs baseados em NHPP concentra-se apenas nos eventos de detecção de defeitos e remoção durante a fase de testes. Quando se estimam os parâmetros do modelo de SRGMs baseados em NHPP é necessário adquirir somente o tempo de detecção de falha ou o número de falhas detectadas durante o período de testes.

Embora tais modelos sejam fáceis de se manusear, Okamura, Etani e Dohi (2010, p. 31) afirmaram que esses parâmetros nem sempre são precisos devido à falta de informação estatística.

3.3.1 Utilização de Dois Fatores

Modelos que utilizam dois fatores apresentam melhor desempenho ao fazer previsões de confiabilidade. Okamura, Etani e Dohi (2010, p. 31) argumentaram que isso se deve ao fato que, dada as possíveis dificuldades em se obter os dados, o modelo que se utiliza somente de um fator não consegue fazer uma previsão de boa qualidade. Huang, Kuo e Lyu (2007, p. 198) confirmaram isso, demonstrando o desempenho superior de modelos que se utilizaram de dois fatores para a modelagem de confiabilidade, empregando-se a combinação do esforço de teste como uma métrica e o processo de depuração imperfeita, onde o processo de remoção de defeito introduz novos defeitos.

Por fim, Wang, Hemminger e Tang (2007, p. 411) buscaram incorporar a estrutura do software como um fator de influência, mensurando a confiabilidade de cada componente na previsão de confiabilidade do software como um todo. Singh *et al.* (2007, p. 360) consideraram a dependência entre defeitos, isto é, não encarando cada defeito como independente de outros defeitos, mas sim, relações entre defeitos, que provocam um efeito em cascata.

3.3.2 Agregação de Mais de Dois Fatores

O aumento de métricas utilizadas, no entanto, aumenta a incerteza na hora de se obter as próprias medidas. Isto acaba afetando a previsão de confiabilidade (LI; XIE; HUING, 2010, p. 3560), mesmo que existam maneiras de se determinar o grau de incerteza dentro de uma previsão de confiabilidade (CHANDRAN; DIMOV; PUNNEKKAT, 2010, p. 227).

Okamura, Etani e Dohi (2010, p. 32) abordaram a questão da incerteza no momento de obtenção das medidas, utilizando técnicas estatísticas para gerar um SRGM mais simples e, consequentemente, com um nível menor de incerteza na previsão de confiabilidade. Por sua vez, Quadri, Ahmad e Faroog (2011, p. 27) apresentaram um modelo onde o esforço de teste segue uma distribuição exponencial generalizada, assumindo-se que a taxa de surgimento de defeitos e o esforço de teste são proporcionais à quantidade restante de defeitos no software. Já Kapur *et al.* (2011, p. 333) observaram que a remoção de um defeito, por si só, pode provocar o surgimento de outros defeitos, notando-se que a quantidade de defeitos removidos, não necessariamente, corresponde a mesma de defeitos observados. Por fim, Peng *et al.* (2014, p. 38) consideraram o esforço de detecção de defeitos e o esforço de correção de defeitos como dois fatores distintos que podem ser incorporados em outros SRGM.

3.4 Incerteza Limite

Um SRGM é uma ferramenta crítica para controlar a qualidade do software. No entanto, segundo Okamura e Dohi (2008, p. 19), tais modelos enfrentam dificuldades em representar exatamente, por meio de modelos matemáticos, o crescimento da confiabilidade de software no mundo real. Além disso, segundo levantamento feito por Almering *et al.* (2007), nenhum SRGM proposto consegue cobrir todos os cenários possíveis de desenvolvimento de software. Por isso, tal característica vai além da relação intrínseca entre a confiabilidade e o cenário de execução, de modo que diferentes domínios de software apresentam dinâmicas distintas durante a fase de testes.

Sabe-se que a confiabilidade é fortemente dependente da maneira como o sistema será utilizado. Assim, uma vez que a confiabilidade e disponibilidade são características de execução, o impacto de defeitos na confiabilidade pode variar dependendo da forma

como é utilizado o sistema, isto é, a frequência com que a parte do sistema que apresenta defeito será executada. Portanto, a análise de diferentes formas e frequências para executar o sistema é um desafio para a previsão de confiabilidade do software, especialmente quando o perfil de seu uso não é conhecido de antemão (IMMONEN; NIEMELA, 2007, p. 50).

Dentre alguns fatores, os que influenciam a incerteza incluem as características de software como: a complexidade do programa, a cobertura de teste, ambiente de desenvolvimento e muitos outros, que aparecem durante o ciclo de desenvolvimento (CHANDRAN; DIMOV; PUNNEKKAT, 2010, p. 228).

Vale destacar que a maioria desses problemas aparecem, principalmente, devido à incerteza envolvida em parâmetros de confiabilidade. Já os fatores, que contribuem para a estimativa de confiabilidade de software, devem ser identificados (CHANDRAN; DIMOV; PUNNEKKAT, 2010. p. 227).
Portanto, é necessário um cuidado extra para se determinar o modelo mais adequado a ser aplicado, pois escolhendo-se um modelo inadequado pode-se fornecer dados que resultem em decisões equivocadas dentro de um projeto de desenvolvimento de software (ULLAH; MORISIO; VETRO, 2012, p. 189).

3.5 Depuração Imperfeita

A maioria dos modelos de crescimento de confiabilidade software propostas são baseadas na suposição de depuração perfeita, ou seja, que todos os defeitos detectados durante as fases de teste e operação são corrigidos e removidos perfeitamente.

Peng *et al.* (2014, p. 38) argumentaram que as ações de depuração no ambiente de teste e ambiente de operação não são sempre realizadas perfeitamente. Por exemplo, erros de digitação invalidam a atividade de correção de defeito ou a remoção do defeito não é realizada corretamente devido à análise incorreta dos resultados dos testes. Além disso, o processo de depuração é, geralmente, longe de ser perfeito. Também, muitos defeitos detectados pelos clientes são introduzidos durante a depuração.

Além da depuração imperfeita em atividades de correção de defeitos, é preciso considerar a possibilidade de introduzir novos defeitos no processo de depuração. Por isso, dois tipos de falhas de software existem nas fases de teste ou operação: falhas de software causados por defeitos originalmente latentes no sistema de software antes do teste (que são chamados defeitos inerentes) e falhas de software causados por defeitos introduzidos durante a operação de software devido à depuração imperfeita.

Geralmente, recursos de teste não são constantemente alocados durante a fase de teste de software que, em grande parte, pode influenciar a taxa de detecção de falhas e o tempo necessário para corrigi-las. Por exemplo, o depurador pode passar uma semana sem fazer qualquer trabalho de teste e trabalhar intensamente nos dias seguintes. Além disso, é natural que os depuradores cometam erros e introduzam novos defeitos durante os testes. Pois, eles tendem a introduzir mais defeitos quanto mais esforço é despendido nos teste, visto que o código sofreu mais mudanças (PENG *et al.* 2014, p. 42).

3.6 Taxa de Remoção de Defeitos

Sob a hipótese ideal de remoção instantânea e perfeita de defeitos, o número esperado de defeitos removidos é o mesmo que o número esperado de defeitos detectados. Gokhale, Lyu e Trivendi (2004, p. 222) argumentaram que, se leva-se ao se levar em

consideração o tempo necessário para a remoção, o número esperado de defeitos removidos, em qualquer momento, dado é inferior ao número esperado de defeitos detectados.

Por isso, a detecção de um defeito gera um impacto no processo de software, já que é preciso dispor de um tempo extra para a correção do mesmo. Assim, quanto mais complexo for o defeito localizado, maior é será o esforço e maior a probabilidade de introdução de novos defeitos.

Essa relação é denominada taxa de remoção de defeitos. A real taxa de defeitos no software, (levando em conta a remoção explícita de defeitos), é maior do que a taxa aparente de defeitos, quando não se é levado em consideração os efeitos do processo de remoção do defeito. Com base nisto, Gokhale, Lyu e Trivedi (2004) argumentam que, com base nisso a confiabilidade estimada, sem levar em consideração a taxa de remoção de defeitos, leva a uma estimativa que tende a ser otimista, ou seja, uma confiabilidade prevista maior do que a observada na prática.

3.7 Considerações do Capítulo

Um SRGM é um modelo matemático usado para prever quantos defeitos ainda existem e permitir a tomada de decisão de parar os testes e lançar o produto. Para tanto, é preciso escolher o modelo mais adequado para o software em questão, bem como as métricas a serem utilizadas. Mesmo tendo em mãos as métricas necessárias para utilizar um determinado SRGM, é preciso também levar em consideração o tipo de cenário para o qual o modelo foi pensado.

Ao se propor a utilização de um SRGM, é preciso considerar a necessidade de se encontrar um equilíbrio entre o esforço para realizar as medições para alimentar o SRGM e a qualidade da previsão que o modelo consegue gerar.

Por conta dessas características, muitas vezes recorre-se a um especialista da área, seja para escolher o modelo ideal ou para se obter uma “segunda opinião” sobre a previsão de confiabilidade (ALMERING *et al.* 2007, p. 87).

4. MODELO CAI-LYU E MÉTODO DE GOKHALE

Cai e Lyu (2007, p. 17) consideraram a cobertura de código durante a execução dos testes como uma métrica relevante para melhorar o desempenho de previsão de confiabilidade. Também, Cai e Lyu (2007, p. 18) relacionaram a cobertura de código com a passagem de tempo.

Por sua vez, Gokhale, Lyu e Trivedi (1994, p. 218) argumentaram que a taxa de remoção de defeitos de software afeta taxa de defeitos detectados e, portanto, deve ser considerada ao estimar a confiabilidade de um software.

4.1 Modelo Cai-Lyu

Visando-se melhorar a capacidade de previsão de confiabilidade, Cai e Lyu (2007) propuseram a utilização de dois fatores para calcular a confiabilidade do software: o índice de cobertura de código aplicado sobre um modelo de curva e a quantidade de defeitos detectados ao longo do tempo aplicado sobre um modelo de curva. Cai e Lyu (2007) afirmaram que o modelo proposto permite usar quaisquer modelos padrões de curva, dando grande flexibilidade para experimentar curvas diferentes. No presente trabalho, o modelo proposto por Cai é chamado de modelo Cai-Lyu.

No modelo de confiabilidade de software de dois fatores, relaciona-se o tempo de execução de testes entre a detecção de falhas e a quantidade de casos de testes executados em relação ao total de casos de teste.

Cai e Lyu (2007) afirmaram que a confiabilidade depende do conjunto de testes executados, de modo que a ordem com que os testes são executados influencia a geração da curva de predição de confiabilidade.

Vale destacar que a confiabilidade é composta por duas estimativas: uma da cobertura de teste e do tempo de execução. Por isso, o modelo Cai-Lyu foca nos modelos para estimativa da cobertura de teste. Já a estimativa do tempo de execução pode vir de outros SRGM. Assim, Cai e Lyu (2007) usaram dois modelos simplificados para obter a estimativa da cobertura de teste: modelos hiper-exponencial e modelo beta.

A relação entre a cobertura de teste e detecção de falhas é modelada usando-se NHPP. Desta maneira, Cai e Lyu (2007) propuseram utilizar a relação entre a cobertura de testes e tempo como parâmetros de correção para outros modelos. Esses dois parâmetros foram modelados usando-se NHPP, onde Os autores usaram NHPP e exponencial para demonstrar a utilização do modelo proposto.

$$\alpha_1(1-e^{-\gamma 1c})\, F_1(t) + \alpha_2(1-e^{-\gamma 2t})\, F_2(c)$$

Onde, $F_1(t)$ e $F_2(c)$ podem ser modelos tradicionais.

A cobertura de código é mensurada conforme os testes são realizados. A variável c vai de 0 a 1, 1 indicando cobertura total do código.

O modelo Cai-Lyu parte do pressuposto que a remoção de defeitos é perfeita e instantânea, ou seja, o processo de remoção de defeitos remove completamente o defeito e não introduz novos defeitos, bem como o processo é instantâneo e não é contabilizado dentro do tempo de testes.

4.2 Estudo de Caso do Modelo Cai-Lyu

Para demonstrar a eficácia do modelo, Cai e Lyu (2007) utilizaram os dados obtidos em um projeto realizado na Universidade de Hong-Kong em 2002, chamado de projeto CUHK-RSDIMU. Este consistiu em formar 34 equipes independentes de desenvolvimento com os alunos de graduação, que foram responsáveis por analisar, codificar, testar, avaliar e documentar um software de sistema crítico da área de aeronáutica.

Durante os quatro meses de duração do projeto, os dados acerca dos testes e da cobertura de código foram armazenados, sendo que cada equipe de desenvolvimento tinha que usar um software de controle de versão para registrar todas as alterações feitas no código-fonte.

Utilizando-se o controlador de versão, Cai e Lyu (2007) geraram variações dos softwares desenvolvidos, chamados de mutantes, emulando assim os possíveis defeitos encontrados. Desta forma, os defeitos presentes em todos os projetos dos estudantes foram analisados como um único projeto. Na sequência, um programa foi desenvolvido para testar todos os mutantes contra os testes do enunciado original e os dados foram coletados. Por fim, com a massa de dados, Cai e Lyu (2007) compararam a estimativa de confiabilidade gerada pelo modelo Cai-Lyu com outros modelos, atestando o melhor desempenho do modelo Cai-Lyu por contemplar a cobertura de código.

Dentro da mesma comparação, Cai e Lyu (2007) utilizaram modelos diversos para $F_1(t)$ e $F_2(c)$. Eles também constataram uma variação considerável no desempenho de estimativa de confiabilidade.

4.3 Limites do Modelo Cai-Lyu

O modelo de Cai (CAI; LYU, 2007) foi testado contra uma base de dados elaborada em um ambiente acadêmico, como resultado de um programa empregando alunos de graduação. Como tal, é importante ressaltar a necessidade de comparar o modelo proposto com dados de projetos elaborados em ambientes corporativos, testando o modelo contra outras dinâmicas de desenvolvimento e testes.

Cai e Lyu (2007) também salientaram que a escolha dos modelos para $F_1(t)$ e $F_2(c)$ afeta o desempenho da estimativa final. Outro fator limitante, mencionado pelos próprios autores, foi a necessidade de coletar uma quantidade maior de dados durante a fase de testes, no caso a cobertura de código. Porém, esta informação nem sempre está disponível em outros projetos.

4.4 Método de Gokhale

Gokhale, Lyu e Trivedi (2004, p. 218) propuseram considerar o impacto que a correção de defeitos detectados causa no processo de testes e na confiabilidade do software. O argumento é que, enquanto um modelo de confiabilidade pode utilizar a taxa de detecção de defeitos para aferir a quantidade de defeitos remanescentes no software, essa métrica assume a correção instantânea e perfeita de defeitos.

Assim, segundo esses pesquisadores (2004), a taxa de defeitos detectados acaba por ser uma métrica otimista, por não levar em consideração o tempo necessário para corrigir os defeitos detectados. Eles também defenderam duas linhas para modelar a relação entre a taxa de falhas detectadas e os defeitos corrigidos.

A primeira linha assume que a taxa de defeitos removidos é constante, isto é, não dependem da taxa de detecção de defeitos:

$$\alpha(1 - e^{-\beta t})$$

A segunda linha trata de defeitos latentes, isto é, ao se remover um defeito, o desenvolvedor acaba corrigindo também outras partes relacionadas com o defeito detectado, mas que ainda não se traduziram em defeitos detectados. Por isso, Os defeitos latentes são, inerentemente, mais difíceis de se remover:

$$\alpha e^{-\beta t}$$

Cabe salientar que a equação para taxa de remoção de defeitos, considerando-se os defeitos latentes, foi apresentada por Gokhale, Lyu e Trivedi (2004, p.218) como uma hipótese.

Para efeitos de avaliação, o presente trabalho utilizou a equação para taxa constante de remoção de defeitos.

4.5 Considerações do Capítulo

Cai e Lyu (2007) apresentaram um SRGM flexível, baseado na cobertura de código e o tempo de execução de testes. Utilizando um estudo de caso, Cai e Lyu demonstraram que o SRGM apresentado produzia uma estimativa de crescimento de confiabilidade melhor do que outros modelos que não utilizam a cobertura de código. Em contrapartida, o modelo de Cai e Lyu parte do pressuposto que defeitos são detectados e corrigidos instantâneamente, sem a inserção de novos defeitos.

Visando endereçar este tipo de pressuposto, Gokhale, Lyu e Trivedi (2004, p. 218) propôem um método para levar em consideração o impacto que o processo de correção de defeitos produz na taxa de detecção de defeitos. Gokhale, Lyu e Trivedi argumentam que, ao considerar a taxa de correção de defeitos, é possível produzir estimativas de crescimento de confiabilidade mais realistas.

5. MODELO PROPOSTO

Conforme discutido o capítulo 3.7, a escolha de um SRGM para utilização num determinado projeto de software deve levar em consideração o tipo de cenário para o qual o modelo foi pensado. Por conta disso, a escolha do SRGM mais adequado é mais trabalhosa e pode demandar a opinião de um especialista para a escolha do modelo a ser usado (ALMERING *et al.* 2007, p. 87).

Neste contexto, o modelo Cai-Lyu mostra-se bastante promissor por conta de sua flexibilidade e da extensa comparação com outros modelos (CAI; LYU, 2007, p. 24). De acordo com os autores, a relação da cobertura de testes com o tempo de execução dos testes é o diferencial do modelo e a razão para o desempenho superior em comparação com outros modelos.

A flexibilidade do modelo Cai-Lyu pode mitigar o problema de escolher um modelo adequado a um projeto de software. Porém, no levantamento bibliográfico do presente trabalho, não foram encontrados novos desdobramentos do modelo Cai-Lyu, especificamente no pressuposto assumido pelos autores, que os defeitos são corrigidos instantaneamente e, portanto, o processo de remoção de defeitos não afeta a estimativa de confiabilidade do software (CAI; LYU, 2007, p. 18).

Além deste fato, observa-se que o modelo Cai-Lyu foi utilizado com uma massa de dados de um projeto executado em ambiente universitário. Cabe, então, averiguar qual seria o desempenho do modelo Cai-Lyu utilizando-se os dados de um projeto executado fora do ambiente controlado por uma universidade. Mais do que isso, é interessante verificar que o pressuposto sobre o processo de remoção de defeitos mantém-se verdadeiro durante a aplicação em um estudo de caso.

O estudo de caso apresentado por Cai e Lyu não apresenta a remoção de defeitos, por tratar-se da análise de projetos previamente finalizados, portanto tornando a taxa de remoção de defeitos irrelevante para a análise apresentada pelos autores.

Por conta da flexibilidade do modelo Cay-Lyu e por conta do foco na cobertura de código, é possível utilizar um modelo matemático que leve em conta a taxa de remoção de defeitos para a estimativa do tempo de execução dos testes.

Durante o levantamento bibliográfico do presente trabalho, não foram encontrados artigos apoiando ou refutando, por meio de um estudo de caso, a utilização da taxa de remoção de defeitos como um fator relevante para a previsão da confiabilidade de software, conforme proposto por Gokhale, Lyu e Trivedi (2004). É plausível assumir que o tempo necessário para corrigir os defeitos afeta o processo de detecção de novos defeitos, além da contagem de detecção de defeitos não levar em consideração os defeitos que ainda estão na fila para serem corrigidos (GOKHALE; LYU; TRIVEDI, 2004, p. 228).

Este trabalho propôs-se a combinar o modelo de Cai-Lyu com o método de Gokhale para constatar a influência que a taxa de remoção de defeitos tem sobre um modelo de previsão de confiabilidade.

Gokhale, Lyu e Trivedi (2004) apresentaram dois métodos para relacionar os defeitos detectados e os defeitos corrigidos em um dado instante t. O primeiro método modela a taxa de defeitos corrigidos usando uma curva NHPP. O segundo método busca modelar defeitos latentes, ou seja, defeitos que ainda não foram detectados, porém são corrigidos durante a correção de um determinado defeito.

Como o método proposto por Gokhale, Lyu e Trivedi (2004) para taxa de remoção de defeitos latentes é hipotética, o presente trabalho não a utiliza, a fim de facilitar a análise dos resultados.

Sendo assim, o modelo de Cai-Lyu,

$$\alpha_1(1-e^{-\gamma 1c})\, F_1(t) + \alpha_2(1-e^{-\gamma 2t})\, F_2(c)$$

foi utilizado na sua forma original, empregando-se o método de Gokhale para taxa constante de remoção de defeitos como $F_1(t)$. Para $F_2(c)$, foi utilizada uma distribuição NHPP em relação à cobertura de código em função do tempo.

O modelo proposto utiliza NHPP para a cobertura de código, de modo a manter uniformidade com o modelo NHPP utilizado no método de Gokhale.

O modelo resultante foi chamado de Cai-Gokhale.

6. APLICAÇÃO E ANÁLISE

Para avaliar o modelo proposto, foram usados os dados de um sistema desenvolvido pela Secretaria de Tecnologia de Informação – SETI, Subsecretaria de Desenvolvimento e Manutenção de Sistemas – UDEM, do Tribunal Regional Federal da 3ª Região.

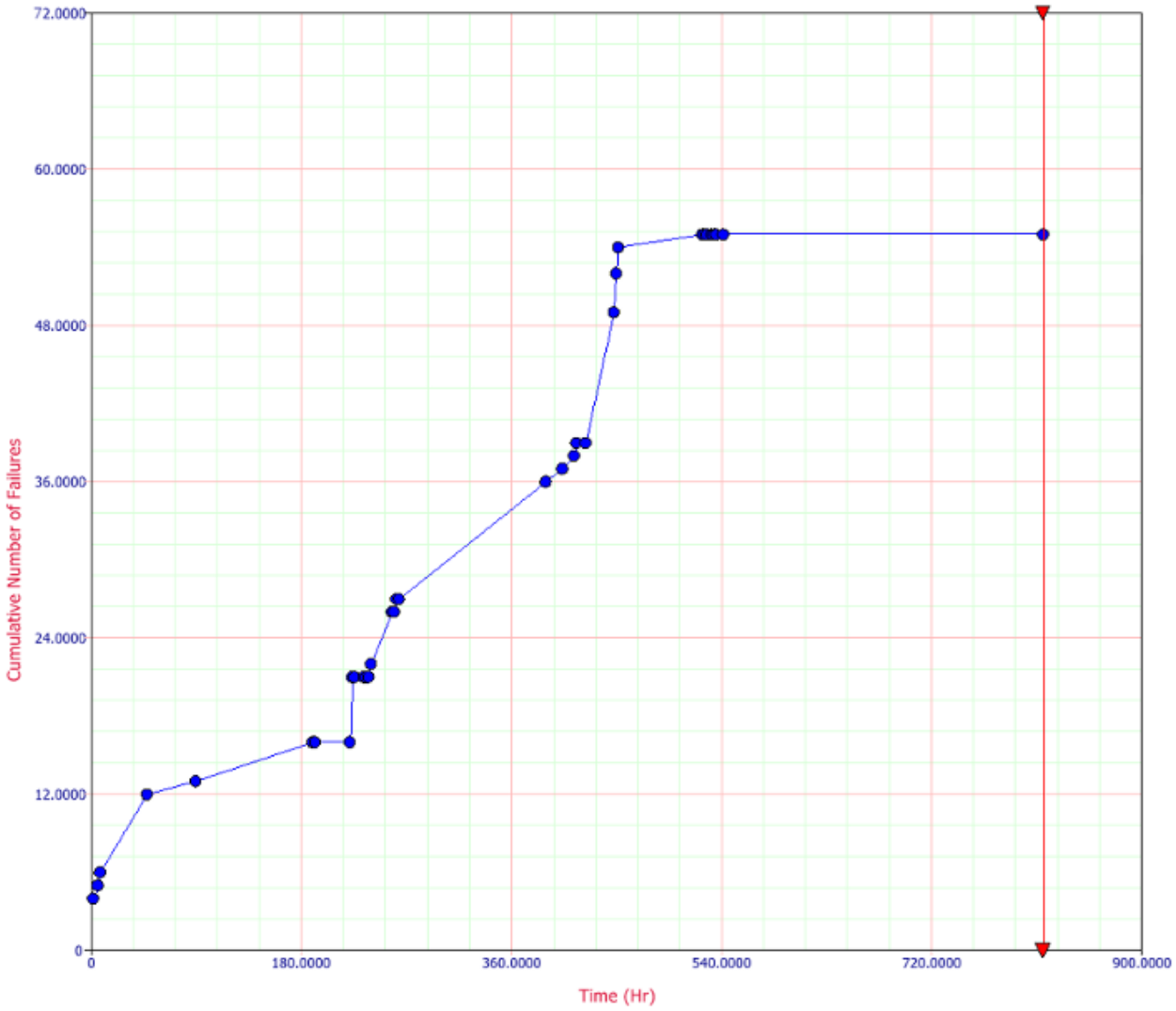
O sistema desenvolvido tem como objetivo integrar dois sistemas legados, implementando um sistema de *workflow*.

A relação dos módulos e correspondentes testes está listada no Anexo 1. No entanto, as partes sensíveis dos dados foram omitidas.

Para a fase de testes do sistema, foram usadas 812 horas, aplicando-se 146 testes. Como resultado, foram detectados 55 defeitos. E, para efetuar os testes e análises dos modelos propostos, foi utilizado o programa RGA versão 10 da suíte de software Reliasoft.

Na Figura 1 é possível observar que a curva formada possui um pico de falhas acumuladas x *vs* tempo, após 400 h de testes. Isto ocorreu porque neste período foram iniciados os testes do módulo M12 e, consequentemente, a correção dos primeiros defeitos desencadeou alterações profundas em outras partes do código que já tinham sido testadas previamente, o que gerou retrabalho nos testes. Como foi explanado anteriormente, este é um comportamento não previsto no modelo Cai-Lyu, o que torna este estudo de caso particularmente interessante para análise e discussão.

Figura 1 – Curva de falhas acumuladas durante a fase de testes



Fonte: Autor

Utilizando-se a relação de módulos e funcionalidades implementadas, foi possível mapear a cobertura de código em cada teste. Nota-se que um determinado trecho de código pode ser executado por mais de um teste. Segundo o desenvolvedor responsável pelo projeto, foi do entendimento do setor de desenvolvimento que os trechos comuns de código somente seriam considerados cobertos depois que todas as funcionalidades que fizessem referência ao dito trecho fossem executados.

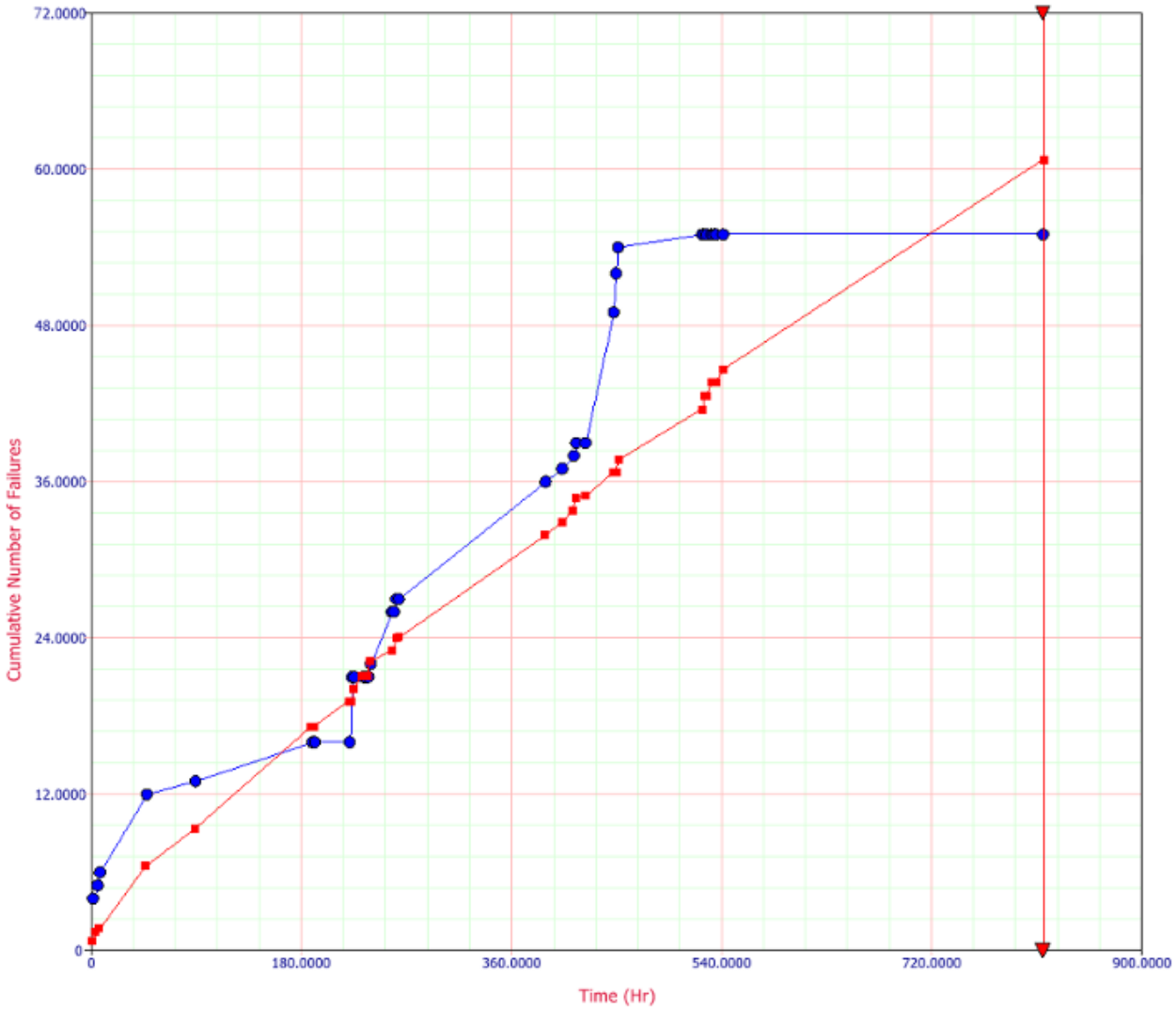
Assim, para adequar os dados obtidos de modo a utilizar os modelos propostos, foram calculadas as horas acumuladas de teste e as falhas acumuladas detectadas. Por isso, foi usado o Método dos Quadrados Mínimos (YAMADA, 2014, p. 31) para determinar os parâmetros necessários para a execução dos modelos.

beta: 0.523160
alfa: 0.504302

Para os modelos de taxa de defeitos encontrados, $F_1(t)$, e cobertura de código, $F_2(c)$, necessários para utilizar o modelo Cai-Lyu, foi usado o modelo NHPP.

Na Figura 2, é possível verificar a comparação entre a curva de falhas acumuladas durante a fase de testes e a curva de estimativa gerada pelo modelo Cai-Lyu.

Figura 2 – Curva do modelo Cai-Lyu (vermelho) em relação à curva de falhas acumuladas (azul)



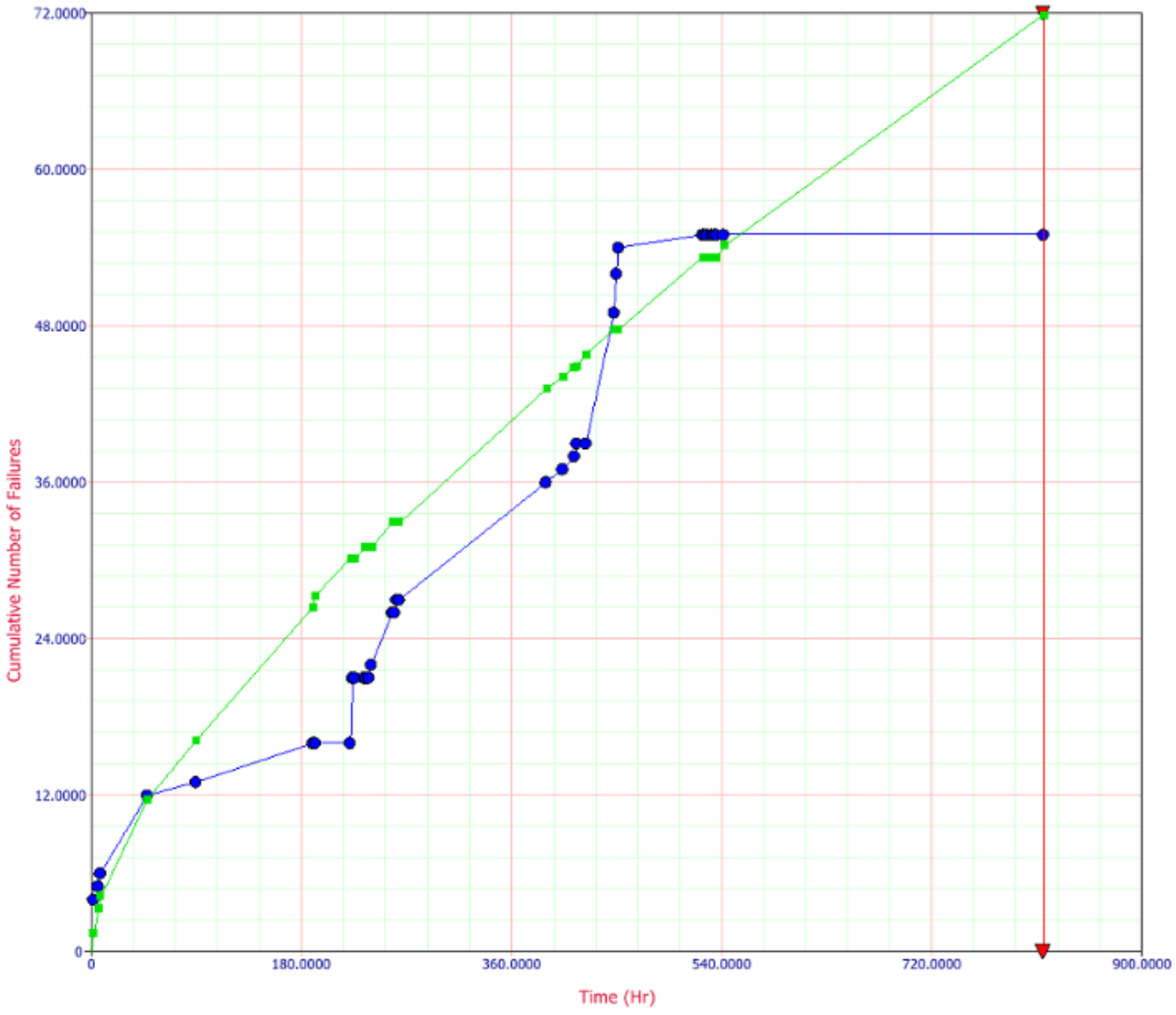
Fonte: Autor

Calculando-se a área formada pelas curvas de falhas acumuladas e a curva formada pelo modelo Cai-Lyu, é possível quantificar quão próximo a previsão do modelo chegou da realidade (YAMADA, 2014, p. 58). Neste caso, a área formada pela curva de falhas e o modelo Cai-Lyu é de 7,02 horas por falha detectada, isto é, o modelo Cai-Lyu prevê

um esforço de horas de teste por falha detectada 7,02 horas menor do que o observado na curva de falhas acumuladas.

Na Figura 3, é possível observar a comparação entre a curva de falhas acumuladas durante a fase de testes e a curva de estimativa gerada pelo modelo Cai-Gokhale.

Figura 3 – Curva do modelo Cai-Gokhale (verde) em relação à curva de falhas acumuladas (azul)



Fonte: Autor

A área formada pela curva de falhas e o modelo Cai-Gokhale é 9,26 horas por falha detectada, isto é, o modelo Cai-Gokhale prevê um esforço de testes de 9,26 horas a mais do que o observado na curva de falhas acumuladas.
Utilizando-se somente a área formada pelos modelos, seria possível concluir que o modelo Cai-Lyu possui um desempenho melhor do que o modelo Cai-Gokhale: a diferença entre o esforço previsto pelo modelo Cai-Lyu em relação ao observado na prática é menor do que a diferença entre o esforço previsto pelo modelo Cai-Gokhale em relação ao observado na prática.

Além disso, é possível observar o efeito do método de Gokhale em ação, ao gerar uma previsão mais pessimista, prevendo um esforço maior de horas de teste, onde o modelo Cai-Lyu prevê um esforço menor do que o observado na prática.

Como a diferença entre os dois é a utilização do método de Gokhale, à primeira vista, tem-se a impressão que a utilização da taxa de remoção de defeitos como fator em um SRGM não melhora o desempenho da estimativa de confiabilidade. No entanto, é possível tirar mais conclusões após analisar o gráfico dividido em regiões. Para facilitar a análise, o gráfico foi dividido em quatro regiões, conforme está apresentado na Figura 4.

Figura 4 – Regiões de interesse para análise da curva de falhas acumuladas

Fonte: Autor

O gráfico apresentado na Figura 4 foi dividido em quatro regiões para ressaltar a diferença de desempenho entre as previsões dos modelo Cai-Lyu e Cai-Gokhale.

Em cada região, foi considerada a área formada entre a curva de falhas acumuladas e os modelos aqui analisados.

A Região 1, que comprime os dados das primeiras 180 h de teste, apresenta uma área de 4,36 horas de diferença por falha detectada entre a curva de falhas e o modelo Cai-Lyu. Já contra uma área de 5,62 horas de diferença por falha detectada entre a curva de falhas e o modelo Cai-Gokhale.

A Região 2 corresponde aos dados coletados entre 180 h e 360 h de teste. Nesta região, a área entre a curva de falhas e o modelo Cai-Lyu é 3,54 horas de diferença por falha detectada, contra uma área de 11,67 horas de diferença por falha detectada entre a curva de falhas e o modelo Cai-Gokhale.

A Região 3 é a de maior interesse para esta análise, pois representa um pico de falhas acumuladas, decorrentes do início dos testes do módulo denominado M12 (vide Anexo 1), sendo que ela comprime os dados coletados entre as marcas de 360 h e 540 h de testes.

Observa-se que nesta região o modelo Cai-Gokhale possui um desempenho melhor, justamente por conta da abordagem “pessimista” do método de Gokhale, de modo que a área entre a curva de falhas e o modelo Cai-Lyu é 14,22 horas de diferença por falha detectada, contra uma área de 6,77 horas de diferença por falha detectada do modelo Cai-Gokhale em relação à curva de falhas.

A Região 4 compreende os dados coletados entre as marcas de 540 h e 816 h de teste. Nesta região, o modelo Cai-Lyu volta a ter um desempenho melhor, apresentando uma área de 6,37 horas de diferença por falha detectada em relação à curva de falhas, contra uma área de 11,41 horas de diferença por falha detectada do modelo Cai-Gokhale em relação à curva de falhas.

Conclui-se que o modelo de Cai-Lyu possui melhor aproximação com a realidade no começo, porém este modelo ajusta a curva conforme a cobertura de testes e o tempo de execução, não levando em conta o repentino aumento da taxa de detecção de falhas no começo dos testes de M12.

Comparando-se as curvas geradas pelo modelo Cai-Lyu e Cai-Gokhale, é possível observar que o primeiro de fato gerou uma estimativa mais otimista, possivelmente por não levar em consideração o processo de remoção de defeitos. Sendo que esse efeito confirma a observação de Gokhale, Lyu e Trivedi (2004, p. 228).

A abordagem mais “pessimista” do método de Gokhale pode ser observada na curva do modelo Cai-Gokhale, onde somente o pico de detecções de falhas na Região 3 supera a previsão do modelo.

7. CONSIDERAÇÕES FINAIS

O objetivo do trabalho foi avaliar a influência que o impacto dos defeitos encontrados exerce sobre o software no processo de previsão de confiabilidade do software. Também, o SRGM proposto por Cai e Lyu, juntamente com o método proposto por Gokhale, Lyu e Trivedi para calcular a taxa de remoção de defeitos foram combinados. O modelo resultante foi chamado de modelo Cai-Gokhale e este foi aplicado em um estudo de caso de software de *workflow* desenvolvido por uma instituição pública brasileira. A influência da taxa de remoção de defeitos sobre a previsão de confiabilidade foi discutida.

Este trabalho verificou a relevância do tempo de correção de defeitos como métrica junto da métrica de cobertura de código para avaliar a confiabilidade de software. Para tal, utilizou-se um estudo de caso recente no Brasil, fora do ambiente controlado por trabalho acadêmico.

7.1 Conclusões

Ao comparar os resultados obtidos pelo modelo Cai-Lyu e o modelo proposto por este trabalho, chamado Cai-Gokhale, averiguou-se que o modelo Cai-Lyu representa melhor a curva de dados. Porém, a exceção ocorre na Região 3, onde o modelo Cai-Lyu não acompanha o repentino aumento de detecção de falhas.

Deste modo, concluiu-se que a utilização da métrica de tempo de correção de defeitos detectados no SRGM não resultou em uma melhoria na previsão de crescimento da confiabilidade do software. Por outro lado, o fato da curva gerada pelo modelo Cai-Gokhale acompanhar o repentino aumento na taxa de falhas, na Região 3, indica um potencial para futuros trabalhos.

7.2 Contribuições do Trabalho

Este trabalho constatou que o tempo de remoção de defeitos, de fato, exerce impacto na taxa de detecção de defeitos. Para tal, criou-se um modelo híbrido, utilizando-se a flexibilidade do modelo Cai-Lyu com a possibilidade de levar em consideração o tempo de remoção de defeitos.

Durante o estudo de caso, foi possível observar que a previsão de crescimento de confiabilidade do software mostrou-se pessimista, isto é, superestimando os possíveis defeitos ainda não detectados. Esta característica é exatamente descrita por Gokhale, Lyu e Trivedi (2004, p. 228). Sendo assim, o modelo proposto Cai-Gokhale une a flexibilidade do modelo Cai-Lyu, que permite utilizar outros modelos de acordo com a necessidade do projeto em questão, com a “prudência” do método de Gokhale, Lyu e Trivedi.

Já a flexibilidade do modelo Cai-Lyu é, particularmente, interessante porque mitiga a limitação dos SRGM de não serem capazes de cobrir todos os cenários de projeto de *software*, sendo necessária a seleção de um modelo adequado (ALMERING *et al.*, 2007, p. 87; IMMONEN; NIEMELA, 2007, p. 49). Em contrapartida, o método de Gokhale, Lyu e Trivedi (2007) mitiga possíveis previsões “otimistas”, que correm o risco de subestimar a quantidade real de defeitos restantes no software.

7.3 Trabalhos Futuros

Para trabalhos futuros, existe a possibilidade de se empregar outros modelos ou curvas a fim de averiguar se a diferença entre falhas acumuladas e falhas corrigidas pode ser expressa por outras curvas, além da NHPP, proposta por Gokhale. Como se sabe, o modelo de Cai-Lyu abre possibilidade para várias explorações, por conta de sua

flexibilidade para escolha de modelos para $F_1(t)$ e $F_2(c)$. Enquanto o método de Gokhale, para modelagem de correção de defeitos latentes, também abre espaço para maiores explorações.

Outro aspecto a ser considerado, é a maneira como a cobertura de código é mensurada. No estudo de caso aqui empregado, foi feito um mapeamento prévio dos testes, onde uma função foi considerada coberta pelos testes depois que fossem executados todos os testes que poderiam executar aquela função.
Portanto, uma consequência da utilização da cobertura de código como fator é que se parte do pressuposto que a cobertura de código é sempre crescente. Isto poderia ser questionado neste estudo de caso, pois o pico de detecção de defeitos na Região 3, possivelmente, implicou em alteração de códigos que já tinham sido testados previamente, e precisaram ser testados novamente. Segundo a sistemática adotada por Cai e Lyu (2007), a cobertura de testes não retrocede de acordo com as correções feitas.

Desta maneira, este é um potencial tema para futuros trabalhos, já que o retrocesso na cobertura de códigos foi aplicado neste estudo de caso. Assim, de modo a facilitar e incentivar futuras explorações, os dados do estudo de caso utilizados na análise deste trabalho encontram-se nos anexos para consulta.

REFERÊNCIAS

ALMERING, V.; GENUCHTEN, M.; CLOUDT, G.; SONNEMANS, P. J. M. **Using Software Reliability Growth Models in Practice**, IEEE Computer Society , 2007.

BOEHM, B. W.; BROWN, J. R.; LIPOW, M. **Quantitative evaluation of software quality. 2Nd International Conference on Software Engineering**, IEEE Computer Society Press, Los Alamitos, 1976.

CAI, X.; LYU, M. R. **Software Reliability Modeling with Test Coverage: Experimentation and Measurement with A Fault-Tolerant Software Project**, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, 2007.

CHANDRAN, S. K.; DIMOV, A.; PUNNEKKAT, S. **Modeling uncertainties in the estimation of software reliability – a pragmatic approach**. Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement, 2010.

GOKHALE , S. S.; LYU, M. R.; TRIVEDI, K. S. **Analysis of Software Fault Removal Policies Using a Non-Homogeneous Continuous Time Markov Chain**. Software Quality Journal, 12, Kluwer Academic Publishers , 2004.

HIRAMA, K. **Engenharia de Software: qualidade e produtividade com tecnologia**. Elsevier. Rio de Janeiro , 2011.

HUANG, C.; KUO, S.; LYU, M. R. **An Assessment of Testing-Effort Dependent Software Reliability Growth Models**, IEEE TRANSACTIONS ON RELIABILITY, Vol. 56, No. 2, 2007.

HUMPHREY, W. S. **Managing the Software Process**. Addison-Wesley. USA. 1989.

IMMONEN, A.; NIEMELA, E. **Survey of reliability and availability prediction methods from the viewpoint of software architecture**, Springer-Verlag, 2007.

ISO. ISO/IEC 25010. **Systems and software engineering -systems and software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models**. INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Geneva, 2011.

JAFFAL, W.; TIAN, J. **Defect Analysis and Reliability Assessment for Transactional Web Applications**. IEEE International Symposium on Software Reliability Engineering Workshops, 2014.

KAPUR, P. K.; PHAM, H.; ANAND, S.; YADAV, K. **A Unified Approach for Developing Software Reliability Growth Models in the Presence of Imperfect Debugging and Error Generation**. IEEE Transactions on Reliability, Vol. 60, No. 1, 2011.

LI, X.; XIE, M.; HUI NG, S. **Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points**. Applied Mathematical Modelling, No. 34, 2010.

OKAMURA, H.; DOHI, T. **Software Reliability Modeling Based on Mixed Poisson Distributions**. International Journal of Reliability, Quality and Safety Engineering, Vol. 15, No. 1, 2008.

OKAMURA, H.; ETANI, Y.; DOHI, T. **A Multi-Factor Software Reliability Model Based on Logistic Regression**. IEEE 21st International Symposium on Software Reliability Engineering, 2010.

PENG, R.; LI, Y. F.; ZHANG, W. J.; HU, Q. P. **Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction**. Reliability Engineering and System Safety 126, 2014.

PRESSMAN, R. S. **Engenharia de software**. São Paulo: Pearson Education do Brasil, 1995.

QUADRI, S. M. K.; AHMAD, N.; FAROOG, S. U. **Software Reliability Growth modeling with Generalized Exponential testing–effort and optimal SOFTWARE RELEASE policy**. Global Journal of Computer Science and Technology Volume 11 Issue 2, 2011.

SINGH, V. B.; YADAV, K.; KAPUR, R.; YADAVALLI, V. S. S. **Considering the Fault Dependency Concept with Debugging Time Lag in Software Reliability Growth Modeling Using a Power Function of Testing Time**. International Journal of Automation and Computing, 2007.

SOMMERVILLE, I. **Engenharia de Software**. 9a. Edição. Addison-Wesley. São Paulo, 2011.

ULLAH, N.; MORISIO, M.; VETRO, A. **A Comparative Analysis of Software Reliability Growth Models using defects data of Closed and Open Source Software**. IEEE 35th Software Engineering Workshop, 2012.

WANG, W.; HEMMINGER, T. L.; TANG, M. **A Moving Average Non-Homogeneous Poisson Process Reliability Growth Model to Account for Software with Repair and System Structures**. IEEE TRANSACTIONS ON RELIABILITY, Vol. 56, No. 3, 2007.

YADAV, A.; KHAN, R. A. **Critical Review on Software Reliability Models**. International Journal of Recent Trends in Engineering and Technology, Vol. 2, No. 3, 2009.

YAMADA, S. **Software Reliability Modeling - Fundamentals and Applications**. Tottori, Springer , 2014.

ANEXOS

Anexo 1 – Relação de módulos e casos de uso do sistema usado no estudo de caso

Módulo	Caso de uso
M1	UC-1
	UC-2
	UC-3
	UC-4
	UC-5
	UC-6
	UC-7
	UC-8
	UC-9
	UC-10
	UC-11
	UC-12
M2	UC-13
	UC-14
	UC-15
	UC-16
	UC-17
	UC-18
	UC-19
	UC-20
	UC-21
	UC-22
	UC-23
	UC-24
	UC-25
	UC-26
	UC-27
	UC-28
M3	UC-29
	UC-30
	UC-31
	UC-32
	UC-33
	UC-34
	UC-35
	UC-36
	UC-37
	UC-38
	UC-39

Anexo 1 (continuação) – Relação de módulos e casos de uso do sistema usado no estudo de caso

Módulo	Caso de uso
M4	UC-40
	UC-41
	UC-42
	UC-43
	UC-44
	UC-45
	UC-46
	UC-47
M5	UC-48
	UC-49
	UC-50
	UC-51
	UC-52
	UC-53
	UC-54
	UC-55
	UC-56
	UC-57
	UC-58
	UC-59
	UC-60
	UC-61
	UC-62
	UC-63
	UC-64
	UC-65
	UC-66
M6	UC-67
	UC-68
	UC-69
	UC-70
	UC-71
	UC-72
	UC-73
	UC-74
	UC-75
M7	UC-76
	UC-77
	UC-78
	UC-79

Anexo 1 (continuação) – Relação de módulos e casos de uso do sistema usado no estudo de caso

Módulo	Caso de uso
	UC-80
	UC-81
	UC-82
	UC-83
	UC-84
M8	UC-85
	UC-86
	UC-87
	UC-88
	UC-89
	UC-90
	UC-91
	UC-92
	UC-93
	UC-94
	UC-95
M9	UC-96
	UC-97
	UC-98
	UC-99
M10	UC-100
	UC-101
	UC-102
	UC-103
	UC-104
	UC-105
	UC-106
	UC-107
	UC-108
	UC-109
	UC-110
	UC-111
	UC-112
	UC-113
M11	UC-114
	UC-115
	UC-116
	UC-117
	UC-118
	UC-119

Anexo 1 (conclusão) – Relação de módulos e casos de uso do sistema usado no estudo de caso

Módulo	Caso de uso
	UC-120
	UC-121
	UC-122
	UC-123
	UC-124
	UC-125
	UC-126
	UC-127
	UC-128
	UC-129
	UC-130
	UC-131
	UC-132
	UC-133
M12	UC-134
	UC-135
	UC-136
	UC-137
	UC-138
	UC-139
	UC-140
	UC-141
	UC-142
	UC-143
	UC-144
	UC-145
	UC-146

Anexo 2 – Relação de horas acumuladas de teste e defeitos acumulados detectados

Horas	Defeitos
2	0
6	4
8	5
48	6
90	12
190	13
192	16
222	16
224	16
226	21
234	21
236	21
238	21
240	21
258	22
260	26
262	26
264	27
390	27
404	36
414	37
416	38
424	39
448	39
450	49
452	52
524	54
526	55
528	55
532	55
534	55
536	55
542	55
816	55

Anexo 3 – Relação de horas acumuladas de teste e cobertura de código

Horas	Cobertura
2	0
6	0.0294117647
8	0.0588235294
48	0.0882352941
90	0.1176470588
190	0.1470588235
192	0.1764705882
222	0.2058823529
224	0.2352941176
226	0.2647058824
234	0.2941176471
236	0.3235294118
238	0.3529411765
240	0.3823529412
258	0.4117647059
260	0.4411764706
262	0.4705882353
264	0.5
390	0.5294117647
404	0.5588235294
414	0.5882352941
416	0.6176470588
424	0.6470588235
448	0.6764705882
450	0.7058823529
452	0.7352941176
524	0.7647058824
526	0.7941176471
528	0.8235294118
532	0.8529411765
534	0.8823529412
536	0.9117647059
542	0.9411764706
816	0.9705882353

Anexo 4 – Relação de horas acumuladas de teste e previsão de falhas acumuladas detectadas pelo modelo Cai-Lyu

Horas	Falhas
2	0
6	1
8	1
48	6
90	9
190	17
192	17
222	19
224	19
226	20
234	21
236	21
238	21
240	22
258	23
260	23
262	24
264	24
390	32
404	33
414	34
416	35
424	35
448	37
450	37
452	38
524	42
526	43
528	43
532	44
534	44
536	44
542	45
816	61

Anexo 5 – Relação de horas acumuladas de teste e previsão de falhas acumuladas detectadas pelo modelo Cai-Gokhale

Horas	Falhas
2	1
6	3
8	4
48	12
90	17
190	28
192	29
222	32
224	32
226	32
234	33
236	33
238	33
240	33
258	35
260	35
262	35
264	35
390	46
404	47
414	48
416	48
424	49
448	51
450	51
452	51
524	57
526	57
528	57
532	57
534	57
536	57
542	58
816	77