GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE A PARTIR DE CASOS DE USO: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

Trabalho de Conclusão de Curso

Engenharia da Computação

Lamartine Veras Sampaio de Souza
Orientador: Prof. Gustavo Henrique Porto de Carvalho

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

LAMARTINE VERAS SAMPAIO DE SOUZA

GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE A PARTIR DE CASOS DE USO: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, maio de 2011.

De acordo

Recife

_____/_____/_____

_____
Orientador da Monografia

Dedico este trabalho aos meus avôs, Sebastião Morais e Antônio Jorge, que já não estão mais entre nós, mas tenho certeza que estão em paz e zelando por nossa família.

Agradecimentos

Gostaria de agradecer inicialmente a minha amada esposa, Louize Vanielle, pelo apoio e principalmente pela paciência e compreensão pelas horas dedicadas a este trabalho. Também quero agradecer aos meus pais, sempre o meu porto seguro nas horas difíceis e, por fim, ao meu orientador Gustavo Carvalho, cuja paciência, dedicação e organização me ajudaram imensamente neste trabalho.

Resumo

Testes de software é uma etapa importante dentro do processo desenvolvimento e nesta etapa, uma das atividades que demandam um esforço maior é a geração de casos de teste. A automação desta atividade pode melhorar o desempenho desta etapa tão importante no processo de desenvolvimento de software. A proposta desse trabalho é apresentar um mapeamento sistemático da literatura a respeito da geração automática de casos de teste a partir de casos de uso. Essa abordagem tem como objetivo mostrar quais trabalhos existem na área, quais as principais técnicas de automação que estão sendo estudadas ou mesmo implementadas, que ferramentas estão sendo utilizadas, quais as dificuldades encontradas. Portanto, fornecer uma visão geral a respeito da geração automática de casos de teste a partir de casos de uso.

Palavras-Chave: Testes de software, automação, casos de teste, casos de uso, mapeamento sistemático.

Abstract

Software testing is an important step in the development process and at this stage, one of the activities that requires a greater effort is the test cases generation. This activity automation can improve the performance of this very important step in the software development process. This paper purpose is to present a systematic literature mapping on the automatic generation of test cases from use cases. This approach aims to show which studies exist in the area, what are the main automation techniques being studied or implemented, which tools are being used, which are the difficulties. Thus providind an overview about the automatic generation of cases testing from use cases.

Keywords: Software testing, automation, test cases, use cases, systematic mapping.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

AOP – *Aspect Oriented Programming*

ATCGT – *Automated Test Case Generation Tool*

CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

CFSM – *Control Flow-Based State Machine*

DSL – *Domain Specific Language*

DST – *Diagram State Transition*

FSM – *Finite State Machine*

IFA – *Interaction Finite Automaton*

ISTQB – *International Software Testing Qualification Board*

LNC – Linguagem Natural Controlada

OWL – *Web Ontology Language*

QAI – *Quality Assurence Institute*

RUP – *Rational Unified Process*

SSD – *System Sequence Diagram*

UML – *Unified Modeling Language*

Capítulo 1
Introdução

Há bem pouco tempo, a atividade de testes era, em geral, colocada em segundo plano no processo de desenvolvimento. Se o cronograma estava sendo atendido, testes eram realizados; caso contrário, a atividade de testes era comprometida ou mesmo ignorada. Porém, com o tempo, percebeu-se que os erros que ocorriam em sistemas já em produção eram mais caros de se corrigir do que os erros encontrados ainda em desenvolvimento. Molinari (2005), afirma que um defeito descoberto no início do projeto custa cerca de 10 centavos de dólar. Quando é descoberto perto do *release*, custa cerca de 100 dólares ou mais. Na prática, o custo está na ordem de 10^n. Com essa ordem de grandeza, a atividade de testes passou a ser observada, então, sob outra perspectiva.

Hoje, a disciplina de testes de software é considerada uma das mais importantes no processo de desenvolvimento. Segundo o *Rational Unified Process-RUP* (2003), testes é uma atividade que deve acompanhar o processo de desenvolvimento desde a fase de concepção até a fase de transição. Pressman (1995) diz que a atividade de teste de software é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificação, projeto e codificação.

A automação na área de testes surge como uma prática que tende a se consolidar, dado os seus benefícios para os testes como, por exemplo, o ganho de tempo. Para Molinari (2010), a principal razão do uso e da disseminação da automação dos testes de software é justamente a urgência cada vez maior de realizar mais testes em menos tempo. As aplicações ficaram mais complexas, ao mesmo tempo, que se passou a exigir maior qualidade do produto entregue.

Uma das vertentes da automação de testes é a geração automática de testes a partir de requisitos. Chen (2010), afirma que a escrita manual de casos de teste é considerada ineficiente e tediosa e propõe o uso de tecnologias de automação para

gerar casos de teste. A elaboração manual de casos ou cenários de testes é, de fato, um tanto quanto tediosa e consome tempo considerável de um ou mais recursos da equipe de testes. Além do mais, como toda atividade manual, é passível de falhas. Somé (2008) argumenta que uma forma de reduzir o esforço de teste e, ao mesmo tempo, garantir a sua eficácia é gerar casos de teste automaticamente.

Portanto, o problema de pesquisa deste estudo é: quais são as técnicas já publicadas na literatura sobre geração automática de testes a partir de requisitos? Para responder esta pergunta, este trabalho pretende realizar um mapeamento sistemático da literatura. Com isso pretende-se avaliar as diferentes abordagens a respeito do tema, definir um conjunto das melhores práticas, descrever as principais dificuldades e limitações encontradas, bem como, citar e avaliar algumas das principais ferramentas aplicadas na geração automatizada de testes a partir de casos de uso.

1.1-Objetivos

O objetivo geral deste trabalho é realizar um mapeamento sistemático da literatura a respeito das diversas técnicas que tratam da geração automática de testes a partir de casos de uso escritos em linguagem natural.

1.1.1-Objetivos específicos

A partir do objetivo geral, têm-se os seguintes objetivos específicos:

- Avaliar as diferentes abordagens a respeito da geração automática de testes a partir de casos de uso escritos em linguagem natural;
- Definir um conjunto das melhores práticas;
- Descrever as principais dificuldades e limitações encontradas;
- Citar e avaliar algumas das principais ferramentas aplicadas na geração automatizada de testes.

1.2-Resultados e impactos esperados

O produto desse mapeamento sistemático será um documento de referência contendo um conjunto de boas práticas, dificuldades, ferramentas utilizadas e outras informações para quem tiver interesse em geração automática de testes a partir de casos de uso.

O trabalho pronto servirá como base ou fonte de pesquisa para outros estudantes ou profissionais de empresas que tenham interesse na área de testes, ou mais especificamente, na geração automática de casos de testes a partir de casos de uso. Portanto, estudantes podem utilizar o trabalho com fonte de pesquisa na área ou como insumo para desenvolver novos projetos e pesquisas.

Profissionais ou empresas interessados em automatizar a geração de casos de testes a partir de casos de uso, podem descobrir quais ferramentas são utilizadas, quais abordagens são sugeridas, quais são as dificuldades em implantar uma determinada abordagem, vantagens e desvantagens da automatização, etc. Enfim, podem ter um conjunto importante de informações, oriundas de diversas fontes, concentradas e organizadas em um único documento.

1.3-Estruturação do trabalho

O presente trabalho é estruturado da seguinte maneira: O Capítulo 2 aborda uma visão geral a respeito da disciplina de testes de software dentro do processo de desenvolvimento. O Capítulo 3 trata principalmente da metodologia e estratégia de ação para o desenvolvimento do trabalho, com foco voltado para a explicação do protocolo de pesquisa utilizado como base para a mesma. O Capítulo 4 traz os resultados da avaliação dos artigos selecionados para o estudo. Por fim, o Capítulo 5 apresenta as conclusões deste estudo.

Capítulo 2
Testes de software

Hoje em dia não há dúvidas que a importância da disciplina de teste de software dentro de um processo de desenvolvimento já está consolidada. Molinari (2010) deixa claro que a atividade de teste é fator fundamental para garantia da qualidade de software e ao investir em testes está se investindo diretamente em qualidade de software. E desta forma, certifica-se que aquilo que se deseja está sendo entregue de acordo com as especificações e necessidades. Com isso, salienta o autor, tem-se: maior satisfação do usuário, melhoria da imagem da empresa, maior redução das incertezas que cercam o software e redução do custo de manutenção em produção do produto entregue.

Delamaro, et. al (2007) argumentam que a construção de software não é uma tarefa simples e pode se tornar complexa dependendo do sistema a ser criado. Por isso, está sujeita a diversos tipos de problemas que acabam resultando na obtenção de um produto diferente daquele que se esperava. E a maioria dos fatores causadores de tais problemas tem uma origem única: erro humano.

De olho na qualidade dos produtos desenvolvidos, diversas empresas investem na qualidade de software, é o que afirma Molinari (2005). Isto pode ser comprovado com os dados de Jim Heumann (2001), que afirma que em muitas organizações os custos com testes de software podem chegar a 50% do custo do software.

A consolidação da atividade de testes pode ser verificada, também, no *Rational Unified Process* - RUP (2003), que estabelece que esta atividade deve acompanhar o processo de desenvolvimento desde as fases iniciais até o fim do processo.

2.1-Processo de teste

O *Rational Unified Process* – RUP é um processo de engenharia de software que tem uma abordagem baseada em disciplinas que visa organizar o desenvolvimento dentro de uma organização, atribuindo tarefas e responsabilidades para cada grupo que a compõe. A figura abaixo mostra a arquitetura geral do RUP. Pode-se verificar que existem duas dimensões principais:

- No eixo horizontal o tempo é representado e mostra os aspectos do ciclo de vida do processo à medida que se desenvolve;
- No eixo vertical são representadas as disciplinas, que agrupam as atividades de maneira lógica, por natureza.

É fácil observar que a disciplina de testes percorre todas as fases e, não por acaso, atinge seu auge de atividade na mesma fase que a disciplina de implementação, ou seja, na fase de construção.

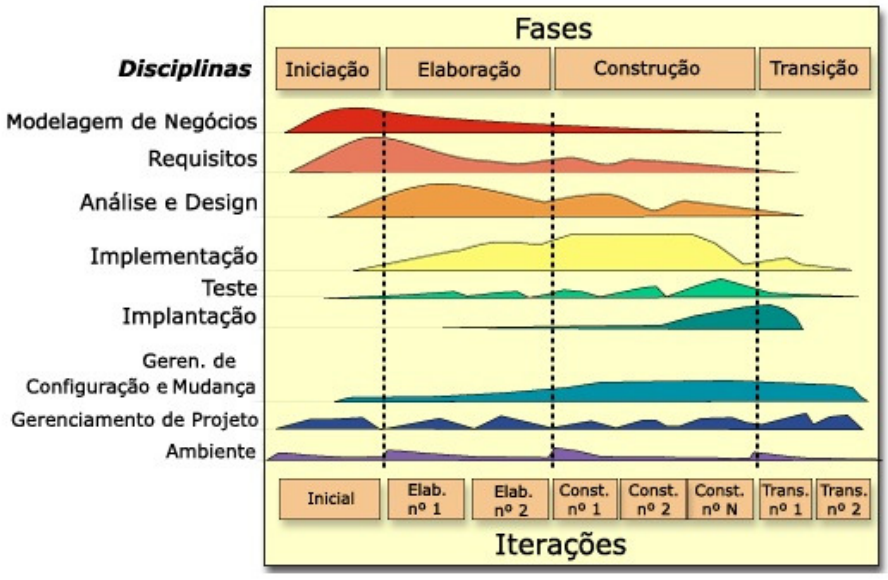


Figura 1: Arquitetura do RUP
Fonte: Rational Unified Process

A disciplina de testes também se relaciona com quase todas as demais disciplinas do processo, conforme se pode observar a seguir:

- A disciplina de **Requisitos**, responsável pela definição dos requisitos do sistema, é utilizada como fonte principal para identificar os testes que devem ser executados;

- A disciplina de **Análise e Design**, responsável por definir o design mais adequado para o software, também é fonte para identificar os testes que devem ser executados;

- A disciplina de **Implementação** é responsável pelo desenvolvimento dos builds do sistema que serão avaliados e validados pela disciplina de testes;

- A disciplina **Ambiente** é responsável por desenvolver e realizar a manutenção dos artefatos de suporte usados durante o teste, tais como o Guia de Teste e o Ambiente de Teste;

- A disciplina **Gerenciamento,** responsável por planejar o projeto e o esforço de trabalho necessário em cada iteração através de um Plano de Iteração, fornece uma base importante para definir o esforço de teste.

- A disciplina **Gerenciamento de Configuração e Mudança,** responsável por controlar a mudança dentro da equipe de projeto, também interage com a disciplina de testes para analisar se cada mudança foi efetuada de maneira correta.

De maneira geral o fluxo de atividades que compõe a disciplina de testes tem início com o planejamento dos testes, onde se tem como resultado um artefato denominado plano de teste. Numa segunda etapa tem-se a definição de cenários teste e posteriormente a escrita dos casos de teste com base nos cenários definidos. Paralelamente à especificação dos testes, pode-se ter a preparação do ambiente para testes. Na terceira etapa é feita a validação da estabilidade da versão do sistema a ser testado. A quarta etapa é onde tem-se a execução dos testes. Por fim, a quinta etapa é a avalição dos resultados da execução. No entanto, o próprio RUP admite que o fluxo de atividades pode requerer variações com base nas necessidades específicas de cada iteração e projeto.

2.2-Papéis dentro de equipe de testes

Ainda segundo o RUP, a disciplina de testes é gerenciada e mantida por uma série de recursos, distribuídos em diversos papéis, são eles:

- **Gerente de teste:** é o responsável por gerenciar as atividades da equipe de testes e a disciplina como um todo, planejando e acompanhando as atividades e o esforço para executá-las. Em linhas gerais, o gerente deve ter conhecimento profundo do processo, possuir habilidades para gerir e lidar com pessoas, experiência com técnicas e ferramentas de testes, entre outras características inerentes aos gestores.

- **Analista de teste:** é o responsável por identificar e definir os testes necessários, verificar e avaliar a cobertura dos testes e a qualidade alcançada ao executar os testes. Também é de responsabilidade do analista avaliar os resultados a cada rodada de testes e escrever os casos de teste que serão executados pelo testador. São características importantes para um bom desempenho deste papel: a capacidade de análise, curiosidade e atenção aos detalhes, conhecimento do sistema em teste, experiência em testes.

- **Designer de teste:** é responsável por determinar qual a abordagem de teste será utilizada e garantir a correta implementação da abordagem escolhida. Também é de responsabilidade do Designer apontar as técnicas e identificar as ferramentas apropriadas para implementar os testes necessários, bem como especificar e verificar as configurações do ambiente de teste. Como características, o designer deve ter experiência em testes, capacidade para identificar e resolver problemas, vasto conhecimento sobre instalação e configuração de hardware e software, experiência bem sucedida no uso de ferramentas de automatização de testes, entre outras.

- **Testador:** é o responsável pela execução dos testes em si, ou seja, é o testador quem realiza a execução dos casos de teste criados pelo

analista e é, também, responsável por reportar os resultados e os problemas, por ventura, encontrados durante a execução do ciclo de testes. As habilidades necessárias para um testador podem variar de acordo com o sistema a ser testado, mas, em geral, um testador deve, principalmente, ser um conhecedor das abordagens e das técnicas de teste e possuir a capacidade para diagnosticar e resolver problemas.

2.3-Principais conceitos de teste

É importante fazer uma abordagem sucinta a respeito dos principais conceitos de testes. Uma visão geral da disciplina será útil para melhor compreensão do trabalho desenvolvido. Molinari (2010) descreve com objetividade e clareza alguns destes conceitos.

- Plano de testes: é o resultado do processo de planejamento dos testes. É composto por requisitos de testes, casos de testes e cenários de testes a serem utilizados nos testes.

- Requisito de testes: são requisitos específicos de testes. Representam o que deve-se testar em termos de uma meta.

- Tipo de teste: são os tipos de testes realizados em um ambiente qualquer, tais como teste funcional, de performance, de segurança, etc.

- Caso teste: é uma seqüência de passos de execução que geram resultados, definidos previamente. São criados a partir dos requisitos e documentação do sistema.

- Cenário de teste: é uma situação mais geral a respeito do sistema. O Cenário de teste pode resultar em um ou mais casos de testes.

- Ambiente de teste: é o conjunto de artefatos lógicos e físicos que compõe o ambiente onde os testes são planejados e executados. É importante que ambiente de testes seja o mais similar possível ao ambiente de produção.

- Relatório de teste: é o relatório que contém as informações dos testes.
- Defeito: não conformidade encontrada na aplicação. Por exemplo, um requisito que foi implementado e não funciona corretamente.
- *Bug* ou Falha: problema encontrado na aplicação e que não está ligado a algum defeito. É algo não previsto.

2.4-Abordagens de testes

De uma maneira geral, a literatura divide a disciplina de testes em duas abordagens principais: Testes Caixa-Preta (*Black-Box Test*) e Testes Caixa-Branca (*White-Box Test*).

2.4.1-Teste Caixa-Preta

Molinari (2005) afirma que nesta abordagem, o objetivo dos testes é garantir que todos os requisitos ou comportamentos do sistema ou de um componente estejam corretos, a base deste tipo de abordagem são os requisitos funcionais do software. Em geral essa abordagem é utilizada nas últimas etapas da atividade de testes.

Já Pressman (1995), argumenta que os testes caixa-preta referem-se aos testes que são realizados nas interfaces do software; são usados para demonstrar que as funções do software são operacionais; que a entrada é adequadamente aceita e a saída é corretamente produzida; que a integridade das informações externas é mantida. Um teste caixa-preta examina aspectos do sistema sem se preocupar muito com a estrutura lógica interna do software.

2.4.2-Testes Caixa-Branca

Nesta abordagem, Pressman (1995) conclui que o teste caixa-branca baseia-se num minucioso exame dos detalhes procedimentais. Os caminhos lógicos através do software são testados, fornecendo-se casos de teste que põem a prova conjuntos específicos de condições e/ou laços. O estado do programa pode ser examinado em

vários pontos para determinar se o estado esperado ou estabelecido corresponde ao estado real.

Já Molinari (2005), argumenta que o objetivo é garantir que todas as linhas de código e condições foram executadas pelo menos uma vez e, obviamente, estejam corretas. O responsável por esse tipo de teste deve garantir a execução de todas as decisões lógicas, todos os laços, incluindo suas fronteiras, devendo passar por todos os caminhos independentes dentro dos módulos e utilizar as estruturas de dados presentes no código para validar sua funcionalidade. Para essa abordagem o engenheiro de testes deve ter um bom conhecimento da linguagem de programação utilizada no sistema.

2.5-Tipos de teste

Dentro das abordagens descritas anteriormente, existem uma infinidade de tipos de testes, muitas vezes, por conta de uma mudança sutil na definição ou execução do teste, surge uma nova classificação quanto ao tipo de teste. Mas de uma maneira geral, podem-se definir os principais tipos abaixo, obtidos e adaptados a partir de Molinari (2005).

- Teste Unitário: é o teste que é executado em pequenos trechos ou fragmentos do código, em geral, a responsabilidade pela execução deste tipo de teste é do desenvolvedor.

- Teste de Integração: muitas aplicações são desenvolvidas em partes (componentes); os testes de integração avaliam se um ou mais componentes desenvolvidos, quando combinados, funcionam corretamente.

- Teste de sistema: nesse teste é avaliada a aplicação como um todo, uma entidade única que deve fazer aquilo que se documentou que faria.

- Teste Operacional: garante que a aplicação pode executar muito tempo sem falhar.

- Teste de Regressão: é o teste que verifica toda a aplicação cada vez que uma característica é acrescentada. É um dos mais importantes tipos de teste.

- Teste funcional: testa se as funcionalidades que constam nos requisitos do sistema, incluindo as regras de negócio, funcionam como o especificado.

- Teste de *Performance:* é o teste que verifica se o tempo de resposta da aplicação está dentro do intervalo desejado.

- Teste de Carga: é o teste que verifica o comportamento do sistema para uma carga maior de usuários, transações, etc. em relação à média prevista, como acontece, por exemplo, nos horários de pico.

- Teste de Stress: muitas vezes é confundido com o teste de carga, mas este tipo de teste é o que verifica o comportamento do sistema para cargas muito maiores que o previsto, que podem ocorrer devido a situações incomuns. É o responsável por verificar os limites do sistema.

- Teste de Integridade: é o responsável por testar a integridade dos dados armazenados pelo sistema.

- Teste de segurança: é o responsável por testar a segurança da aplicação como um todo. Deve levar em conta os princípios básicos de segurança da informação: confidencialidade (sigilo), integridade, disponibilidade, não-repúdio e autenticidade.

2.6-Projeto de caso de teste

Segundo Pressman (1995), o projeto de teste de software e de outros produtos trabalhados por engenharia pode ser tão desafiador quanto o projeto inicial do próprio produto. Contudo, os engenheiros de software muitas vezes tratam a atividade de teste como uma reflexão tardia, desenvolvendo casos de teste que

podem parecer certos, mas que apresentam pouca garantia de serem completos. Devem-se projetar testes que tenham a mais alta probabilidade de descobrir a maioria dos erros com uma quantidade mínima de tempo e esforço.

Segundo Heumann (2001), um caso de teste é um conjunto de entradas de teste, condições de execução e resultados esperados para um objetivo particular: exercer um caminho particular do programa ou verificar o cumprimento de um requisito específico, por exemplo.

A proposta de um caso de teste, argumenta o autor, é identificar e comunicar condições que seriam implementadas no teste. Casos de teste são necessários para verificar a implementação bem sucedida e aceitável dos requisitos do produto (casos de uso).

Ainda baseado em Heumann (2001), o mesmo estabelece um processo composto por três passos básicos para geração de casos de testes a partir de casos de uso. São eles: (1) para cada caso de uso gerar um conjunto de cenários de casos de uso; (2) para cada cenário, identificar ao menos um caso de teste e as condições que irão torná-lo executável; (3) para cada caso de teste, identificar os valores de dados com os quais o sistema deverá ser testado.

2.7-Testes baseado em modelos

Segundo Delamaro et. al (2007), uma especificação é um documento que representa o comportamento e as características que um sistema deve possuir. Porém, o autor alerta que um risco potencial com a descrição textual é a possibilidade de se incluírem inconsistências na especificação, uma vez que as linguagens naturais geralmente são ambíguas e imprecisas. O uso de outras formas de especificação torna-se importante em cenários nos quais tais imprecisões podem causar problemas.

É necessário perceber que o uso de linguagem natural não é, por si só, um problema. Afinal, o que é necessário é a precisão e o rigor da especificação. A especificação formal só é útil se for possível, a partir dela, aumentar a compreensão

do sistema, aliando-se à possibilidade de analisar de forma sistemática as propriedades da especificação.

A captura do conhecimento sobre o sistema e sua reutilização durante diversas fases do desenvolvimento é permitida com a modelagem. Grande parte da atividade de teste é gasta buscando-se identificar o que o sistema deveria fazer. Um modelo é muito importante nessa tarefa, pois, se bem desenvolvido, captura o que é essencial no sistema. No caso de modelos que oferecem a possibilidade de execução, o modelo pode ser utilizado como oráculo, gerando previsões dos resultados esperados, conforme descreve Sommerville (2003), e definindo a linha que separa o comportamento adequado do comportamento errôneo, conforme argumentam Delamaro et. al (2007).

No entanto, o autor alerta para a seguinte questão: como saber que o modelo está correto? Ou seja, o modelo torna-se um artefato que deve ser testado tanto quanto o próprio sistema. Uma vez verificado o problema de o modelo está ou não adequado ele é extremamente importante para o teste, servindo tanto quanto oráculo quanto como base para a geração de scripts e cenários de teste.

Durante a realização dos testes, um grande obstáculo é determinar exatamente quais são os objetivos do testes, ou seja, quais itens serão testados e como. Em geral especificações não rigorosas deixam margem a opiniões e especulações. A forma da especificação pode variar de um grafo de fluxo de chamadas inter-módulos a um guia de usuário. Uma especificação clara é importante para definir o escopo do trabalho de desenvolvimento e, em especial, o de teste. Se a única definição precisa de o que o sistema deve fazer é o próprio sistema, os testes podem ser improdutivos.

Delamaro et. al (2007) descrevem algumas técnicas de geração de modelos baseadas em Máquinas de Transição de Estados e afirma em suas considerações finais, que a utilização de modelos para auxiliar a geração de testes traz diversas vantagens. A possibilidade de se eliminar a ambigüidade, ou ao menos reduzi-la, auxilia na elaboração mais rigorosa das expectativas do que o sistema deve satisfazer. Pode-se, portanto, obter maior rigor. Outra importante vantagem é a

possibilidade de automação. Várias técnicas podem ser utilizadas para sistematicamente derivar casos de teste a partir do modelo.

2.8-Automação de teste

Automação de testes pode assumir principalmente dois níveis. No primeiro, a preocupação encontra-se na geração automática de casos de teste a partir de alguma notação prévia, como casos de uso. Os casos de testes gerados são, então, executados de forma automática ou não. Já o segundo nível considera justamente a execução automática de casos de teste. O ideal, é a integração destes níveis, contudo, esta não é uma tarefa fácil.

No segundo nível, segundo Molinari (2010), existe uma diferença clara entre testes e automação de testes. No primeiro é realizada a tarefa de testar e no segundo você usa um software que imita a interação com a aplicação no que se refere ao teste tal qual um ser humano faria, muito embora com algumas limitações. Ainda segundo os autores, existem quatro dimensões, das quais Molinari (2010) destaca três delas, que diferenciam um teste manual de um automatizado:

1-Dimensão efetiva: ambos os testes, manual e automatizado, devem ser efetivos, atingindo seus objetivos.

2-Dimensão evolutiva: um teste manual evolui mais rápido do que o automatizado, devido a flexibilidade do ser humano.

3-Dimensão econômica: um teste automatizado é mais econômico a partir da segunda execução do que um teste manual. Quanto mais for reutilizado mais barato sai o teste automatizado.

2.9-Desafios na automação de testes

A decisão de automatizar os testes deve ser avaliada com cautela, pois não existe solução definitiva e de fácil implementação. Molinari (2010) cita, o que ele considera, os principais desafios para automação de testes de software:

Expectativas irreais: pode parecer óbvio, mas não é, para a maioria dos gerentes, testadores e executivos, porque parcela significativa se deixa seduzir pelas promessas dos fabricantes de software de automação de testes. A necessidade de resolver problemas imediatos muitas vezes vai de encontro com a maior característica da automação: não existe uma única solução através da qual tudo é possível resolver. Quando se começa a utilizar, percebe-se que nem sempre automatizar é tão fácil assim, mas também não é algo inatingível.

Obter suporte de gerenciamento da automação de teste: quando se introduz o uso de uma ferramenta de automação de teste, é necessário obter ou estabelecer o suporte ao seu uso, seja pela definição de boas práticas no ambiente ou pela definição de padrões de uso. Isso é importante para que exista transparência no processo de automação de testes, seja em que nível for. E mais ainda, é importante contar com o apoio de sua gerência para que a ferramenta seja utilizada da forma mais objetiva e transparente possível.

Automação de testes é um novo projeto: o uso da automação é, na verdade, o estabelecimento de um novo projeto, derivado de um projeto de testes. Os gestores acham que tudo é a mesma coisa. Deve-se ter uma equipe na empresa voltada para automação: suporte, criação de scripts e pesquisa de novas estratégias de automação. Sempre será necessário deixar algum tempo para pesquisar novas estratégias e técnicas de automação.

Envolver especialistas de testes experientes no processo de avaliação e uso das ferramentas de automação: é importante envolver profissionais com vivência em testes, esta é uma questão técnica e assim deve ser tratada.

Ainda segundo Molinari (2010), o teste automatizado tem uma produtividade maior e consegue atingir em tempo menor aquilo que em geral é rotineiro no teste. O teste manual não pode ser eliminado, argumenta; deve, sim, ser reduzido ao máximo possível e focado naquilo que é muito caro automatizar.

2.10-Ferramentas de teste

Os testes constituem uma fase dispendiosa e trabalhosa do processo de software, argumenta Sommerville (2003). Como resultado, conclui o autor, as ferramentas de testes foram as primeiras ferramentas de software a serem desenvolvidas. Atualmente, essas ferramentas oferecem diversos recursos e seu uso reduz significativamente o custo do processo de testes. Diferentes ferramentas podem ser integradas ao que alguns autores chamam de *Workbenches* de testes ou bancada de testes.

Molinari (2010) classifica as ferramentas de automação de acordo com duas visões distintas, uma delas é a visão do *Quality Assurance Institute*-QAI e a outra de acordo com a visão do *International Software Testing Qualification Board* - ISTQB. Já Sommerville (2003), tem uma classificação mais sucinta e que, de maneira geral, não diverge diretamente das classificações estabelecida pelo QAI e ISTQB. São elas:

- **Gerenciador de testes:** gerencia a execução dos testes do programa. O gerenciador de testes mantém o acompanhamento dos dados do teste, dos resultados esperados e dos recursos de programa testados.

- **Gerador de dados de teste:** gera dados de teste para o programa a ser testado. Isso pode ser realizado selecionando-se dados a partir de um banco de dados ou utilizando-se padrões para gerar dados aleatórios de modo correto.

- **Oráculo:** gera previsões dos resultados esperados para o teste. Os oráculos podem ser versões prévias dos programas ou sistemas protótipos. Os testes envolvem executar o oráculo e o programa a ser testado em paralelo. As diferenças em suas saídas são evidenciadas.

- **Comparador de arquivos:** compara os resultados dos testes de programa com os resultados de testes precedentes e relata as diferenças entre eles. Os comparadores são essenciais nos testes de regressão, quando são comparados os resultados de execução de versões antigas e

novas. As diferenças nesses resultados indicam problemas em potencial com a nova versão do sistema.

- **Gerador de relatório:** fornece uma definição de relatório e recursos de geração para os resultados dos testes.

- **Analisador dinâmico:** adiciona código a um programa para contar o número de vezes que cada declaração foi executada. Depois que os testes foram executados, é gerado um perfil de execução que mostra a freqüência com que cada declaração de programa foi executada.

- **Simulador:** diferentes tipos de simuladores podem ser fornecidos. Os simuladores-alvo simulam a máquina em que o programa deve ser executado. Os simuladores de interface com o usuário são programas orientados por scripts, que simulam múltiplas interações simultâneas com o usuário. A utilização de simuladores de E/S significa que o *timing* de seqüências de transações pode ser repetido, ou seja, definidas as entradas as transações podem ser executadas diversas vezes e, a cada execução, as saídas podem ser avaliadas.

2.11-Geração de caso de teste

Quanto ao primeiro nível de automação antes mencionado, Delamaro et. al (2007) afirmam que diversos tipos de ferramentas de teste têm sido utilizadas para aumentar a produtividade na atividade de automatização de testes, que tende a ser extremamente dispendiosa. Em particular, argumentam os autores, no projeto de casos de teste torna-se essencial a existência de ferramenta de suporte que compute os requisitos de teste de um determinado critério e verifique a adequação de um conjunto de teste.

Ainda assim, muito trabalho manual deve ser empregado para criar casos de teste que sejam adequados ao critério utilizado. Nesse sentido, técnicas para geração automática de casos de teste são bastante desejadas para permitir a automatização também desse aspecto do teste. Porém, fica claro que não existe um

procedimento geral que permita a geração de conjuntos adequados a qualquer critério de teste. Na prática, técnicas específicas são projetadas para cada critério de teste.

Capítulo 3
Metodologia e estratégia de ação

Para elaborar este estudo, será realizado um *Mapeamento Sistemático da Literatura*. Pode-se afirmar que este *mapeamento* é um caso particular de uma *Revisão Sistemática da Literatura.*

Segundo Kitchenham (2007), revisão sistemática da literatura é um meio de identificar, avaliar e interpretar todas as pesquisas disponíveis relevantes para uma questão de pesquisa específica, ou área temática, ou fenômeno de interesse. Já um mapeamento sistemático, também conhecido como estudos preliminares, é projetado para fornecer uma visão ampla de uma área de investigação, para determinar se existe evidência de pesquisa sobre um tema e fornecer uma indicação da quantidade de evidências.

Este trabalho seguirá as diretrizes definidas no guia de Kitchenham (2007), que aborda como realizar uma revisão sistemática da literatura, bem como um mapeamento sistemático.

O trabalho começa a tomar forma a partir da necessidade de responder a questão: **Como gerar casos de teste de maneira automática a partir de casos de uso escritos em linguagem natural?** É com base nela que as pesquisas serão realizadas e os artigos selecionados e lidos para fundamentar o trabalho.

Seguindo o guia de Kitchenham (2007), a primeira etapa consiste na elaboração do protocolo do mapeamento sistemático. Nele constarão informações, tais como: a questão de pesquisa, o processo de busca, as fontes de pesquisa, os critérios de seleção dos artigos, bem como os critérios de exclusão.

Portanto, a questão de pesquisa será o norte para a realização do estudo. É a partir da questão que todo o trabalho se desenvolve.

O processo de busca estabelece a maneira com a qual serão realizadas as pesquisas e buscas de artigos relacionados com o tema; estabelece a definição de

palavras-chave sobre a área e, a partir delas, tem-se a criação de *strings* de busca que serão utilizadas na pesquisa por artigos relevantes para o desenvolvimento do tema.

As buscas serão realizadas de forma manual utilizando mecanismos de busca da Internet, com o auxílio da *string* de busca definida no protocolo e, devido à limitação de tempo da pesquisa, serão concentradas apenas nas bibliotecas digitais *IEEE Computer Society Digital Library* e *ACM Digital Library*.

Após realizado o processo de busca, os artigos que foram trazidos pela string montada serão selecionados para leitura, com base em critérios definidos também no protocolo e, obviamente, deverão estar diretamente relacionados com a pesquisa. Em seguida, os mesmos deverão ser catalogados e as informações relevantes de cada artigo devem ser registradas após a sua leitura. É importante e necessário que os artigos não selecionados também sejam registrados e, para cada um, seja definido o motivo da não seleção.

Após a etapa de seleção e leitura dos artigos, serão realizadas a avaliação e estruturação das informações que foram catalogadas a respeito de cada artigo, com isso será possível atingir os objetivos definidos neste projeto.

É importante lembrar que a idéia do mapeamento sistemático é que duas pessoas diferentes, com auxílio de um protocolo elaborado, possam chegar ao mesmo resultado sobre um determinado tema que, no caso deste trabalho, diz respeito à geração automática de casos de teste a partir de casos de uso escritos em linguagem natural.

3.1-Protocolo

Segundo Kitchenham (2007), a definição de um protocolo é importante, pois ele define os métodos para realizar o estudo e diminui a chance de um trabalho tendencioso. Sem um protocolo, argumenta ela, é possível que a seleção de estudos individuais ou a análise possa ser impulsionada pelas expectativas do pesquisador.

Para definição da estrutura desse protocolo, seguiram-se as orientações do guia de Kitchenham (2007). Para tanto, foram definidos a questão de pesquisa, palavras-chave, a string de busca montada a partir destas últimas, as fontes de busca, bem como os critérios de inclusão e exclusão de artigos no estudo.

3.1.1-Questão de pesquisa

A questão de pesquisa é a responsável por guiar a realização deste trabalho e ficou assim definida: **Q0 - Como gerar casos de teste de maneira automática a partir de casos de uso escritos em linguagem natural?**

3.1.2-Palavras-chave

As palavras-chave são elementos básicos que auxiliarão nas pesquisas, e estão obrigatoriamente relacionadas ao tema do trabalho. Para cada palavra devem ser encontrados sinônimos e todas devem ser traduzidas para inglês, que é o idioma utilizado nas fontes de busca escolhidas.

As palavras definidas são:

Teste de software: *software test*, test case, test scenario*;*

Casos de uso: *Use case*, UML, requirement*;*

Geração automática: *Automat* generation.*

O caractere ‘ * ’ indica que variações da palavra são permitidas. Por exemplo: test* => test, tests, testing, tested, etc. Apesar da ocorrência de “linguagem natural” na questão da pesquisa, optou-se por não incluir a mesma na string de busca para não torná-la muito restritiva. Contudo, durante a seleção dos estudos pertinentes, observou-se a partir de qual notação os casos de testes eram automaticamente gerados.

3.1.3-*String* de busca Q0

A *string* de busca é montada com base nas palavras-chave definidas no tópico anterior juntamente com operadores booleanos (ANDs e ORs), formando, assim, uma forma conjuntiva normal. Desta forma, a *string* ficou assim definida:

(((((("test case") OR "test scenario") OR "software test") AND ((("use case") OR "requirement") OR "UML")) AND (("generation") AND "automated"))

3.1.4-Fontes de busca

A *string* de busca, definida no tópico anterior, foi aplicada nas seguintes fontes de busca:

- IEEE Xplorer;
- ACM Digital Library.

As fontes de busca selecionadas são bibliotecas digitais reconhecidas e o acesso aos seus artigos e periódicos foi possível, em virtude de convênio entre a CAPES e as referidas fontes.

Existem outras fontes de pesquisa igualmente reconhecidas e respeitadas, tais como: Springer e Science Direct. Porém, devido a limitação do tempo de pesquisa, as referidas fontes não foram selecionadas.

3.1.5-Seleção dos estudos primários

Após a pesquisa nas fontes de busca com o uso da string de busca, a seleção dos estudos primários deve ser realizada com base em critérios de inclusão e exclusão de trabalhos bem definidos.

Um dos preceitos da revisão sistemática, bem como do mapeamento sistemático é que a inclusão ou exclusão dos trabalhos seja revisada por outro pesquisador. Neste caso, o orientador deste trabalho de conclusão de curso, Prof. Gustavo Henrique Porto de Carvalho, desempenhou este papel.

- Critérios de inclusão: A inclusão de um trabalho na pesquisa é definida pela relevância do mesmo em relação ao tema. Para tal, foram definidos os seguintes critérios:

1. Estudos que tratem primariamente técnicas, abordagens, modelos e ferramentas de apoio à geração automática de casos de teste a partir de casos de uso;

2. Estudos que tratem primariamente das dificuldades, problemas, limitações e desafios na geração automática de casos de teste a partir de casos de uso;

3. Estudos que tratem primariamente boas práticas, lições aprendidas e fatores de sucesso na geração automática de casos de teste a partir de casos de uso.

- Critérios de exclusão: Serão excluídos os artigos não relevantes para a pesquisa, após avaliação feita do título e resumo. Na primeira triagem será feita a leitura do título. Os artigos que passarem para segunda triagem terão seus resumos lidos para melhor avaliação e serão excluídos caso claramente não sejam relevantes para o desenvolvimento do trabalho, caso estejam incompletos, caso sejam repetidos, ou seja, trabalhos que já apareceram em diferentes fontes de pesquisa, neste caso apenas o primeiro será considerado e o segundo será excluído. E, por fim, caso sejam duplicados, ou seja, trabalhos que apresentem estudos semelhantes, neste caso apenas o mais completo será incluído.

Tabela 1: Seleção de estudos primários

Seleção de Estudos Primários						
		Primeira triagem (Título)	**Segunda triagem (Resumo)**			
			Excluídos			**Incluídos**
Fontes	**Estudos Retornados (String de busca)**	**Estudos relevantes**	**Não relevante**	**Repetido / Duplicado**	**Incompleto**	**Estudos Primários**
IEEE Xplore	21	4	3	0	0	1
ACM Digital Library	510	11	7	0	0	4

É possível observar na Tabela 1 acima que poucos trabalhos foram reportados na literatura sobre o problema em questão. É também interessante notar que a segunda triagem excluiu a maior parte dos artigos selecionados na primeira triagem. Isto é uma conseqüência do seguinte fato: a maioria dos estudos que falam de geração automática de casos de teste, pauta esta geração a partir não de casos de uso escritos em linguagem natural, mas sim a partir de diagramas UML ou notações parecidas. Por este motivo, os estudos foram preliminarmente incluídos, mas ao ler o resumo observou-se que a fonte de geração não era a desejada por este trabalho.

3.1.6-Segunda revisão

Após a seleção dos artigos trazidos pela string de busca, conforme a tabela acima, observou-se que poucos artigos eram realmente relevantes e foi necessária a realização de uma segunda revisão.

A segunda revisão tem como objetivo avaliar as referências dos trabalhos selecionados para tentar identificar possíveis trabalhos relevantes sobre o tema, mas que tenham ficado de fora daqueles trazidos pela string de busca.

Após a avaliação das referências de todos os 5 trabalhos selecionados, mais dois trabalhos foram selecionados, totalizando 7 (sete) trabalhos relevantes, conforme Tabela 2 abaixo.

Tabela 2: Total de estudos incluídos

Estudos Primários incluídos	Segunda Revisão	Total de estudos incluídos
5	2	7

3.1.7-Avaliação da qualidade dos estudos

Segundo Kitchenham (2007) é importante avaliar a qualidade dos estudos primários selecionados em complemento aos critérios gerais de inclusão e exclusão.

Uma dificuldade citada pela autora é que é que não existe uma definição consensual de "qualidade" de um estudo.

Para realizar a avaliação da qualidade dos estudos, foram definidas questões e estabelecidos pesos para cada tipo de resposta às questões, o intuito é verificar o quanto cada estudo atende aos objetivos da pesquisa e não avaliar diretamente o mérito do trabalho desenvolvido e reportado por cada estudo. As questões definidas são as seguintes:

Q1 – A técnica/ferramenta apresentada no trabalho está bem descrita o que permite a sua utilização?

Q2 – A técnica/ferramenta foi utilizada na prática?

Q3 – A utilização da técnica/ferramenta foi em um contexto real?

Q4 – Os objetivos ou questões do estudo são alcançados?

Q5 – Qual a classificação do estudo com base no critério CAPES/QUALIS?

Para responder as questões de Q1 à Q4 deve ser considerada a seguinte escala:

- ✓ **Sim (Pontos: 4)** – Deve ser concedido se o trabalho atende totalmente aos critérios da questão.
- ✓ **Parcialmente (Pontos: 2)** – Deve ser concedido se o trabalho atende apenas parcialmente aos critérios da questão.
- ✓ **Não (Pontos: 0)** – Deve ser concedido se o trabalho não atende aos critérios da questão.

Para responder a questão Q5, devemos considerar a classificação fornecida pela Coordenação de Aperfeiçoamento de Pessoal de Nível – CAPES, órgão do Governo Federal, que define um conjunto de procedimentos denominados Qualis. A classificação de periódicos desenvolvida pela CAPES é realizada pelas áreas de avaliação e passa por processo anual de atualização. Esses veículos são enquadrados em estratos indicativos da qualidade - A1, o mais elevado; A2; B1; B2; B3; B4; B5; C - com avaliação mais baixa.

Para tanto, foi definida para este trabalho uma seqüência de pesos para a classificação QUALIS que pode ser vista na Tabela 3.

Tabela 3: Pontuação para classificação Qualis

QUALIS	PONTOS
A1	4
A2	3
B1, B2	2
B3, B4, B5	1
C	0
Sem Classificação	0

Considerando o intervalo de pontuação obtido por cada artigo (maior ou igual a zero e menor ou igual a 20), este estudo definiu quatro estratos de qualidade assim denominados: artigos ruins, regulares, bons e ótimos (Tabela 4). O enquandramento de cada trabalho em um destes estratos deu-se da seguinte maneira:

- ✓ Artigos ruins: aqueles que obtiverem de 0 à 5 pontos, após somadas todas as pontuações obtidas em cada pergunta;
- ✓ Artigos regulares: aqueles que obtiverem de 6 à 10 pontos, após somadas todas as pontuações obtidas em cada pergunta;
- ✓ Artigos bons: aqueles que obtiverem de 11 à 15 pontos, após somadas todas as pontuações obtidas em cada pergunta;
- ✓ Artigos ótimos: aqueles que obtiverem de 16 à 20 pontos, após somadas todas as pontuações obtidas em cada pergunta.

Tabela 4: Escala de classificação dos artigos

Ruim	Regular	Bom	Ótimo
0 à 5 pontos	6 à 10 pontos	11 à 15 pontos	16 à 20 pontos

A avaliação e análise da qualidade média dos artigos lidos será feita no Capítulo 4. A pontuação individual obtida por cada artigo

pode ser observada nos fichamentos dos mesmos, presentes no apêndice B deste trabalho.

Capítulo 4
Geração automática de casos de teste a partir de casos de uso

4.1-Técnicas de geração automática

A literatura pesquisada a respeito do tema, fornece diversas técnicas para gerar casos de teste automaticamente, cada uma possui suas peculiaridades e características próprias, muito embora, também seja possível indentificar características comuns.

4.1.1-Técnica de Chen e Li

Chen e Li (2010) argumentam que automação requer modelos que forneçam a informação necessária. Isto é, para automatizar a geração de caso de teste a partir da especificação, um modelo apropriado precisa ser construído. Este modelo precisa representar a semântica da especificação formalmente. Só então, algoritmos relevantes podem ser desenvolvidos e implementados. Com base nessas afirmações, os autores propuseram, então, a formalização dos casos de uso a partir de um método batizado de Autômato de Interação Finito – IFA. A definição formal do IFA é a seguinte:

IFA = (S, s0, Sf, T, δ), onde:

S é um conjunto finito de estados

s0 ϵ S é o estado inicial único. Geralmente, a pré-condição de um caso de uso, pode ser considerada como um único estado.

Sf c S é um conjunto de todos os estados finais

T é um conjunto de disparadores de eventos

δ : S x T → S é a função total que representa as transições entre todos os estados.

É possível perceber que a descrição do IFA se assemelha bastante com a descrição de uma máquina de estados, o que facilita o entendimento.

A técnica dos autores consite em converter os casos de uso em IFA através de algoritmos específicos. E, após a converção, o autômato gerado representará fluxos de eventos de casos de uso como um modelo baseado em estados. O IFA contém apenas dois tipos de elementos: estados e transições entre estados. Na abordagem dos autores, este autômato é considerado como a entrada da geração de casos de teste. É importante salientar que entre o caso de uso e sua formalização com o IFA, existe um passo intermediário que é a especificação de cenários com o uso de diagramas de seqüência do sistema – SSD que é um tipo de diagrama de seqüência, que não mostra os objetos internos do sistema. No SSD, um sistema é considerado como uma entidade única que interage com os atores.

Após a formalização do caso de uso com o IFA, o resultado servirá como entrada para uma ferramenta desenvolvida pelos autores e denomidada 'Ferramenta Automatizada para Geração de Casos de Testes – ATCGT'. Está ferramenta é a responsável por gerar os casos de teste a partir das entradas que serão fornecidas. A ferramenta implementa algoritmos específicos e após sua configuração gera um conjunto de caminhos de teste. Com a semântica fornecida, cada caminho de teste é mapeado para um caso de teste real. Portanto, de maneira geral a proposta dos autores pode ser sumarizada na figura 2 abaixo.

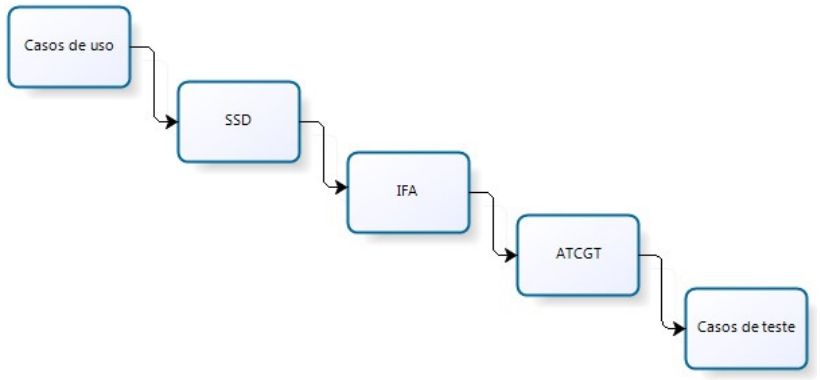


Figura 2: Visão geral da técnica de Chen e Li
Fonte: Adaptado de Chen e Li (2010)

4.1.2-Técnica de Im

Im et. al (2008) afirmam que teste é uma busca por falhas e testar um pedaço de software envolve três fases essenciais. Primeiro, os testes são planejados. Em segundo lugar, os testes e a infra-estrutura de teste são construídos. Finalmente, os testes são executados e os resultados são avaliados.

O foco do trabalho desenvolvido pelos autores está na primeira e segunda fases, quando os testes são planejados e construídos fisicamente. O planejamento do teste e fases da construção são repetidos com menos frequência do que a execução e a fase de avaliação, mas particularmente no desenvolvimento iterativo essas fases serão repetidas muitas vezes. O objetivo é reduzir o esforço destas duas fases, através da extração automática de casos de teste a partir de casos de uso especificados a partir de uma linguagem de domínio específico – DSL.

Uma linguagem de domínio específico - DSL é uma notação que os designers de produtos podem utilizar para definir o produto e os testadores podem usar para criar os casos de teste. Linguagens de domínio específico são linguagens adaptadas a um determinado domínio. Elas fornecem notações e constroem um domínio de aplicação o que resulta em significativa facilidade de uso.

A técnica de automação de teste dos autores é dividida em dois segmentos de atividades: o segmento de linha de produto e o segmento de um produto específico. Em uma linha de produto de software de uma organização, o time da linha de produto cria os ativos fundamentais, os ativos que são utilizados em vários produtos da linha de produtos. O núcleo da equipe de desenvolvimento da linha de produtos cria uma infra-estrutura, incluindo a linguagem de domínio específico, o teste de modelos, e cria a parte do modelo de casos de uso e suíte de testes que são aplicáveis aos vários produtos. Times de desenvolvimento de produto específico desenvolvem ativos únicos para seus produtos. Na figura abaixo, é possível observar os dois segmentos de atividades e os desdobramentos que compõem a técnica descrita pelos autores.

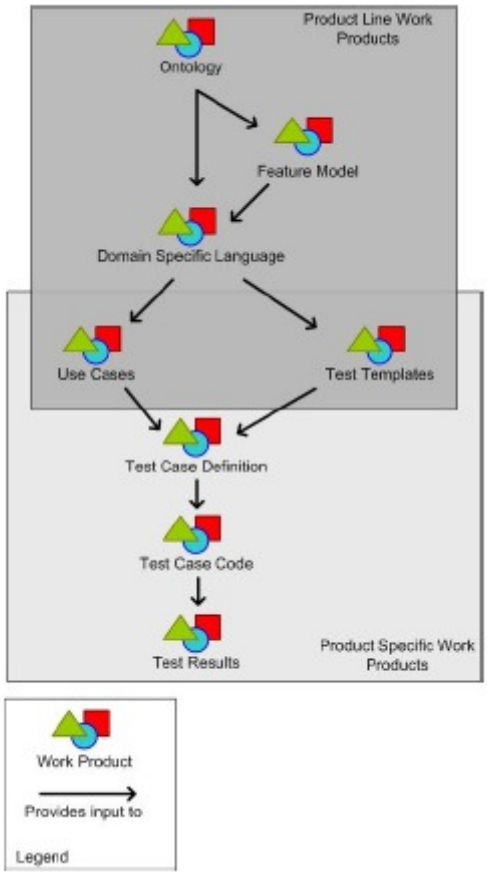


Figura 3: Segmentos de atividades
Fonte: Im et. al (2008)

Para criar a DSL, os autores combinaram os conceitos de domíno e as relações capturadas na análise do domínio com as propriedades visíveis ao usuário com relação ao produto, identificadas durante a análise das características.

A análise de domínio identifica os conceitos e relações entre os conceitos no domínio da aplicação e captura-os em uma ontologia. É processado uma série de documentos escritos sobre o domínio, usando uma ferramenta que identifica os substantivos e os verbos nas frases. A ferramenta captura um vocabulário de partida para o domínio. Especialistas de domínio auxiliam na identificação das relações entre os substantivos e as ações sobre os conceitos, os verbos. A ontologia é representada na Web Ontology Language - OWL para que ela possa ser manipulada automaticamente.

A linguagem de domínio específico evolui naturalmente como os substantivos e verbos encontrados na ontologia que são combinados em frases e usados para

descrever características dos produtos. A linguagem é capturada em forma de frases que são úteis para a próxima etapa - a utilização na definição de casos de uso. Desenvolveu-se, então, frases padrão que são adequadas para estímulos e porções de respostas de descrições de casos de uso.

A especificação de cada produto é construída como um conjunto de casos de uso. Os estímulos e as respostas em cada caso de uso são escritos usando as frases na linguagem de domínio específico. Um editor personalisado é gerado a partir da DSL para o uso na construção de casos de uso. O editor guia a definição do caso de uso apresentando o conjunto de frases definidas para estimulos e respostas e permitindo que a pessoa que define o caso de uso selecione as frases apropriadas.

Um conjunto de modelos de teste é criado a partir de padrões de teste. Os padrões de teste são associados com frases da DSL e são escolhidos com base na descrição DSL e no tipo de teste que está sendo planejado. O teste do sistema irá utilizar os padrões que são baseados em como o produto será utilizado, conforme descrito nos casos de uso, em vez de como o produto é construído. Os modelos de teste derivados dos padrões estão associados com os requisitos.

Os casos de teste são gerados a partir dos casos de uso e dos modelos de teste e as frases da DSL estão associadas com os modelos de teste. Este mapeamento é incorporado em um editor de caso de teste que é gerado automaticamente. Por fim, o testador usa o editor de teste para criar os casos de teste necessários para satisfazer os objetivos do teste.

4.1.3-Técnica de Xu e He

A técnica proposta por Xu e He (2007) consiste em gerar requisitos de teste de sistema a partir de casos de uso orientados a aspectos. O foco desta abordagem é a formalização de um modelo de sistema testável a partir de casos de uso orientados a aspectos. Foram capturadas várias restrições entre casos de uso base e aspectual. Especificamente, foram transformados diagramas e descrições de casos de uso orientado a aspectos em redes de petri orientada a aspectos.

A formalização de casos de uso orientada a aspectos permite gerar significativas seqüências de caso de uso com relação a vários critérios de cobertura (como a cobertura de casos de uso, cobertura de transição e cobertura de estado). Quando cenários de teste para casos de uso individuais estão disponíveis, eles podem ser compostos em testes do sistema de acordo com a sequência de casos de uso gerado.

Na abordagem caso de uso tradicional ou caso de uso base, o comportamento requerido de um sistema é definido como uma coleção de casos de uso. Um diagrama de caso de uso retrata atores, casos de uso e relações entre eles. As relações primárias entre casos de uso são inclusão, extensão e generalização. Cada caso de uso implica em uma unidade funcional útil que o sistema fornece aos seus atores. Esta especificação tipicamente inclui o fluxo básico (cenário de sucesso), fluxo alternativo (variações do fluxo básico), sub-fluxos (para repetições no fluxo básico), pontos de extensão (onde comportamentos adicionais podem ser inseridos), precondições, pós-condições, etc.

Já na abordagem adotada de análise de requisitos orientada a aspectos com caso de uso, mantém-se o uso tradicional de extensões (comportamentos opcionais) e adota-se os casos de uso aspectual para descrever requisitos transversais. Um caso de uso aspectual consiste de *pointcuts* e fluxos (básico e alternativo) de conselhos, cada *pointcut* é uma coleção de pontos de junção (*joinpoints*), que se referem aos passos dos fluxos básicos, fluxos alternativos, etc, em casos de uso tradicionais. Casos de uso tradicionais não precisam saber onde os aspectos serão aplicados. Isso é útil para muitas características transversais (segurança, por exemplo) que são adições aos casos de uso tradicionais.

Redes de petri são um método formal bem definido, com notação gráfica e matemática para especificação e análise de sistemas distribuídos. Redes de petri orientada a aspectos são motivadas a capturar as características essenciais (*join points, pointcuts, advice, etc.*) da orientação a aspectos para modelagem de sistemas com redes de petri. Similar às noções de Programação Orientada a Aspectos (POA ou AOP), um modelo de rede de petri orientada a aspectos, consiste de uma rede de petri básica, rede de petri baseada em aspectos e precedência em

aspectos. Cada aspecto é uma entidade encapsulada de *pointcuts, advice,* declarações inter-modelo e introduções. Um *pointcut* define um grupo de *join points* (ou seja, transições, lugares e arcos) em redes base ou tradicionais.

A técnica descrita pelos autores consite em formalizar os caso de uso tradicionais bem como os casos de uso orientado a aspectos para redes de petri orientadas a aspectos. A ideia dos autores é que a formalização pode detectar possíveis erros de requisitos e modelagem. E, uma vez que o caso de uso é convertido em rede de petri, pode-se utilizar uma grande variedade de técnicas de análise formal para avaliar o resultado da especificação, descobrir possíveis problemas na descrição original do caso de uso e, ainda, definir cenários de teste para diferentes tipos de cobertura.

4.1.4-Técnica de Somé e Cheng

Somé e Cheng (2008) afirmam que um caminho para reduzir o esforço de teste, assegurando a sua eficácia, é gerar casos de teste automaticamente a partir de artefatos usados nas fases iniciais do desenvolvimento de software. O trabalho dos autores propõe a geração de casos de teste a partir dos requisitos capturados como casos de uso. Nesta técnica os autores apresentam uma abordagem para geração automática de representações abstratas de propostas de testes denominadas cenários de testes. Os autores afirmam que superaram a natureza informal dos casos de uso utilizando uma linguagem natural restrita para casos de uso.

A fim de garantir uma cobertura adequada de seqüências de eventos, foi gerado uma máquina de estados de controle de fluxo a partir de caso de uso e utilizou-se técnicas de cobertura de código tradicionais para derivar seqüências de testes. Finalmente, foram inferidas relações seqüenciais entre casos de uso, baseado numa comparação de pré-condições e pós-condições de casos de uso. Isto permitiu ao autores combinar comportamentos de casos de uso em um controle de fluxo global baseado em modelo de estados.

Uma máquina de estados finito – FSM (*Finite State Machine*) é uma tupla (Σ, S, F, S0), onde Σ é o alfabeto, S é o conjunto de estados, F é uma função de

transição e S0 é o estado inicial. Os autores utilizaram o termo Controle de Fluxo Baseado em Máquina de Estados – CFSM para FSMs que descrevem fluxos de eventos dos casos de uso. Os cenários de teste são gerados a partir de uma seleção de sequências de caminho de acordo com um objetivo de cobertura.

Os autores concluem afirmando que a automatização total da geração de casos de teste dos requisitos é atualmente inviável. Eles argumentam que existe uma lacuna de abstração entre os cenários de teste e casos de teste concretos que precisa ser preenchida com intervenção manual. Além disso, os requisitos incluem aspectos como performance e segurança que não são adequadamente capturados nos casos de uso. As dificudades e limitações das diversas técnicas aqui descritas serão abordadas mais detalhadamente em um tópico mais adiante.

4.1.5-Técnica de Chatterjee e Johari

Outra técnica pesquisada foi a proposta de Chatterjee e Johari (2010). Os autores propõem uma técnica de geração automática de casos de teste a partir de requisitos informais. Segundo os autores a formalização é um esforço que implica a transformação dos requisitos informais numa notação formal em conformidade com alguma racionalidade para atingir um objetivo desejado. O objetivo dos autores é formalizar requisitos descritos de maneira informal em um formato que seja possível a automação de casos de teste. Segundo eles a geração de casos de teste, diretamente da especificação de requisitos (testes baseado em especificação), melhora a especificação de requisitos em si, provocando a localização e erradicação de ambigüidades e inconsistências na especificação de requisitos.

A técnica proposta pelos autores consiste em formalizar os requisitos do sistema utilizando um diagrama de transição de estados - DST e através da ferramenta desenvolvida pelos autores, denominada "STATEST 1.1.0", gerar os casos de teste. O funcionamento da ferramenta será descrito mais adiante.

4.1.6-Técnica de Nebut

Nebut et. al (2006), descrevem mais uma técnica para geração automática de testes a partir de casos de uso. Os autores propõem uma nova abordagem para

automatização da geração de cenários de testes a partir de casos de uso. A idéia é que essa proposta seja aplicada no contexto de software embarcado orientado a objeto e leve em conta os problemas de rastreabilidade entre as visões de alto nível e a execução do caso concreto de teste.

O método desenvolvido é baseado em um modelo de caso de uso que desvenda as muitas ambigüidades dos requisitos escritos em linguagem natural. Foram construídos casos de uso UML reforçados com contratos (baseado nas pré e pós-condições dos casos de uso). Os autores propuseram tornar esses contratos executáveis, escrevendo-os na forma de expressões lógicas no nível de requisitos. Baseado nestes requisitos mais formais, mas ainda em alto nível, definiu-se um modelo de simulação de casos de uso. Desta forma, uma vez que os requisitos são escritos em termos de casos de uso e contratos, eles podem ser simulados para checar sua consistência e corretude. O modelo de simulação é também usado para construir explicitamente um modelo de todas as seqüências válidas de casos de uso e, a partir daí, extrair caminhos relevantes utilizando critérios de cobertura. Esses caminhos são chamados objetivos de teste. A geração dos objetivos de teste a partir dos casos de uso constitui a primeira fase da abordagem. A segunda fase tem como objetivo gerar cenários de teste a partir desses objetivos de teste.

A contribuição desta técnica é gerar testes a partir da formalização dos requisitos do sistema, no contexto de software embarcado orientado a objeto. O objetivo maior da abordagem é diminuir o trabalho na geração de teste por meio da automação e transferir o esforço para a atividade de especificação. Outro intuito, não menos importante, é gerar eficientemente casos de teste detectando falhas em softwares embarcados.

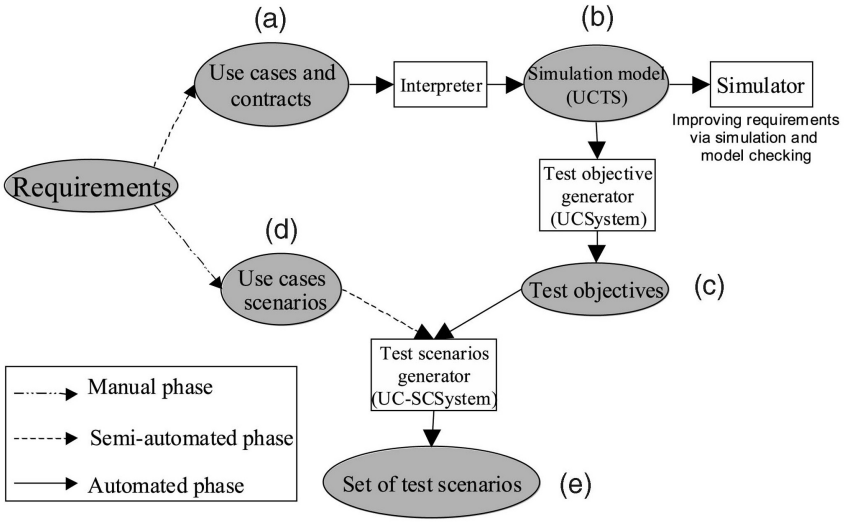


Figura 4: Visão global da técnica de Nebut
Fonte: Nebut et. al (2006)

Observando a figura 4, tem-se uma visão geral da proposta dos autores. A primeira fase do método (passos 'a' até 'c' na figura 1) tem como objetivo gerar os objetivos de teste a partir da visão de caso de uso do sistema. Casos de uso não correspondem à forma como as funções principais serão implementadas no sistema, casos de uso são visões globais do sistema. É por isso que casos de testes não podem ser gerados apenas a partir dos casos de uso. Os autores, então, propõem uma abordagem de requisitos por meio de contratos. Estes contratos não são executados antes ou depois da execução de um caso de uso, mas são usados para inferir a ordenação parcial correta de funcionalidades que o sistema deve oferecer.

Quando parâmetros são especificados para um caso de uso, contratos são especificados na forma de pré e pós-condições envolvendo esses parâmetros. Os contratos de caso de uso são expressões lógicas de primeira ordem combinando predicados com operadores lógicos. Um predicado tem um nome, uma variedade e um conjunto (potencialmente vazio) de parâmetros formais digitados. Os predicados são usados para descrever fatos no sistema. É importante argumentar que, uma vez que a lógica booleana é usada, um predicado ou é verdadeiro ou é falso, mas nunca indefinido.

A partir dos casos de uso e seus contratos, um protótipo de ferramenta (UC-System), constrói um modelo de simulação (passo 'b' na figura 1) e gera seqüências corretas de casos de uso (passo 'c' na figura 1). Em seguida, tal seqüência correta

de casos de uso é chamada de objetivo de teste. Como mostrado na figura 1, o modelo de caso de uso pode ser simulado. Esta fase de simulação permite ao engenheiro de requisitos checar e possivelmente corrigir os requisitos antes dos testes serem gerados.

A segunda fase (passos 'c' até 'e' da figura 1) tem como objetivo gerar os cenários de testes. Um cenário de teste pode ser diretamente utilizado como um caso de teste executável, ou pode precisar que o testador faça algumas alterações adicionais, caso algumas mensagens ou parâmetros estejam faltando. Para ir do objetivo de teste para os cenários de teste, são necessárias informações adicionais, especificando as trocas de mensagens envolvidas entre o ambiente e o sistema. Tais informações podem ser anexadas a um dado caso de uso na forma de diversos artefatos: diagramas de seqüência, máquina de estados ou diagrama de atividades. Todos esses artefatos descrevem os cenários que correspondem ao caso de uso. Por questões de simplicidade, os autores lidaram apenas com diagramas de seqüência que são chamados cenários de casos de uso. O princípio da transformação de objetivos de teste para cenários de teste, consiste de substituir cada caso de uso do objetivo de teste por um dos seus cenários de caso de uso, utilizando a ferramenta protótipo UC-SCSystem. Os artefatos necessários para aplicar nessa abordagem são casos de uso com o reforço de contratos e cenários anexados a estes casos de uso. A partir destas entradas a geração de cenários de teste é automática.

4.1.7-Técnica de Ibrahima

A proposta de Ibrahima et. al (2007) é de uma ferramenta para geração automática de casos de teste a partir dos requisitos do sistema. Com base no trabalho feito por Heumann (2001), que trata da geração manual de casos de teste usando casos de uso e fluxos de eventos, a proposta dos autores é automatizar o processo de geração de casos de teste para reduzir os custos de teste e economizar o tempo de gerar os casos de teste manualmente. Por isso, eles utilizaram diagramas de caso de uso com base nos requisitos do sistema para gerar os casos de testes automaticamente.

Os autores argumentam que os requisitos do sistema são normalmente capturados durante a fase de análise. Os requisitos são classificados em funcionais e não-funcionais. Os requisitos funcionais se preocupam com as tarefas que o novo sistema irá executar enquanto que os requisitos não funcionais se preocupam com todas as restrições do novo sistema. Os requisitos funcionais descrevem o que o sistema deve fazer. Os casos de uso são usados para definir os requisitos do sistema e representam as funcionalidades do sistema. Na maioria das vezes, cada caso de uso é convertido em uma função representando as tarefas do sistema. Embora, possa se converter um ou mais casos de teste de cada caso de uso.

A proposta dos autores foi construir uma ferramenta batizada de GenTCase (Gerador de Caso de Teste), que pode ser usada para o layout do diagrama de caso de uso de qualquer sistema. Este diagrama de caso de uso representa um requisito funcional do sistema. A ferramenta também é capaz de gerar automaticamente casos de teste de sistemas de acordo com o diagrama de caso de uso que tenha sido formado anteriormente. Um descrição mais detalhada da ferramenta será vista mais adiante.

4.2-Principais dificuldades e limitações

Embora cada técnica descrita na seção anterior apresente dificuldades particulares a cada uma delas, a idéia nesta seção é tratar das dificuldades comuns a maioria das técnicas apresentadas.

Talvez a maior dificuldade encontrada para geração de casos de teste automaticamente a partir de casos de uso seja a falta de formalização dos casos de uso. Segundo Chen e Li (2010), em geral a especificação de caso de uso em si é informal e muito difícil de processar diretamente pelo computador. Assim, um transformação formal é necessária. Xu e He (2007) argumentam que especificações de requisitos são freqüentemente cheias de nuances e ambigüidades, além disso, acrescentam os autores, contradições, omissões e imprecisões são freqüentemente encontrados na forma textual de descrições de caso de uso. Nebut et. al (2006), também afirma que algumas das dificuldades encontradas são subespecificação ou erros nos requisitos.

Algumas das técnicas apresentadas, em algum momento dos seus processos, fazem uso do diagrama de casos de uso para conseguir gerar casos de teste automaticamente. É o caso, por exemplo, da abordagem descrita por Ibrahima et. al (2007) e Xu e He (2007). Esses útimos apontam como dificuldades na utilização dos diagramas de casos de uso a não captura de restrições sequenciais. Além disso, argumentam eles, outros tipos de dependências podem existir, como, por exemplo, concorrência e alternância. As dependências são resultado direto da lógica de negócio que o sistema propõe suportar. A abordagem de caso de uso tradicional exclui concorrência e dependência dos diagramas de caso de uso. Embora isto possa manter diagramas de caso de uso mais simples, isto levanta um problema de testabilidade. Quando se planeja os testes do sistema, precisa-se identificar possíveis sequências de execução de casos de uso. É provável que eles disparem diferentes falhas durante os testes. Este problema também é citado por Somé e Cheng (2008) que afirmam que restrições seqüenciais importantes entre casos de uso, são geralmente deixadas fora dos casos de uso e são assumidas apenas implicitamente. Para resolver o problema de testabilidade, as dependências entre casos de uso devem ser explicitamente modeladas.

Enfim, existem diversos problemas que podem limitar ou restringir a geração automática de casos de testes a partir de casos de uso. Somé e Cheng (2008) chegam a afirmar em suas conclusões que a automatização total da geração de casos de teste dos requisitos é inviável, pois existe uma lacuna de abstração entre os cenários de teste e casos de teste concretos que precisa ser preenchida com intervenção manual. Além disso, argumentam os autores, os requisitos incluem aspectos como performance e segurança que não são adequadamente capturados nos casos de uso.

4.3-Melhores práticas

De maneira análoga a seção anterior, a idéia nesta seção é definir um conjunto de melhores práticas comuns a todas as técnicas ou pelo menos comuns a maioria.

Depois da análise das técnicas apresentadas, é possível afirmar que uma prática comum a todas foi a formalização dos casos de uso para só então prosseguir com o processo de geração de casos de teste automaticamente. Chatterjee e Johari (2010) afirmam categoricamente que a formalização da especificação dos requisitos é o passo intermediário e imprecindível para geração automática de casos de teste a partir de requisitos. Xu e He (2007) argumentam que a formalização pode, também, auxiliar na detecção de possíveis erros de requisitos e modelagem.

Porém, a formalização seria um segundo passo, o primeiro seria a escrita de casos de uso precisos, completos e objetivos sem deixar margem para ambiguidades. Xu e He (2007) defendem que uma visão bem definida de caso de uso não é só necessário para a corretude do sistema e implementação, mas também serve como base para evolução futura do sistema.

4.4-Ferramentas utilizadas

As técnicas de geração automática de casos de testes descritas anteriormente, em geral, fizeram uso de ferramentas desenvolvidas pelos próprios autores para proceder com a automatização.

Chen e Li (2010) desenvolveram um protótipo de uma ferramenta denominada ‘Ferramenta Automatizada para Geração de Casos de Testes – ATCGT’ que pode ser utilizada para gerar casos de teste a partir de casos de uso de acordo com a abordagem proposta pelos autores. A ATCGT é composta de 3 subsistemas: Editor de SSD, modelador de IFA e o gerador de casos de teste, conforme é possível obervar na figura 5 abaixo.

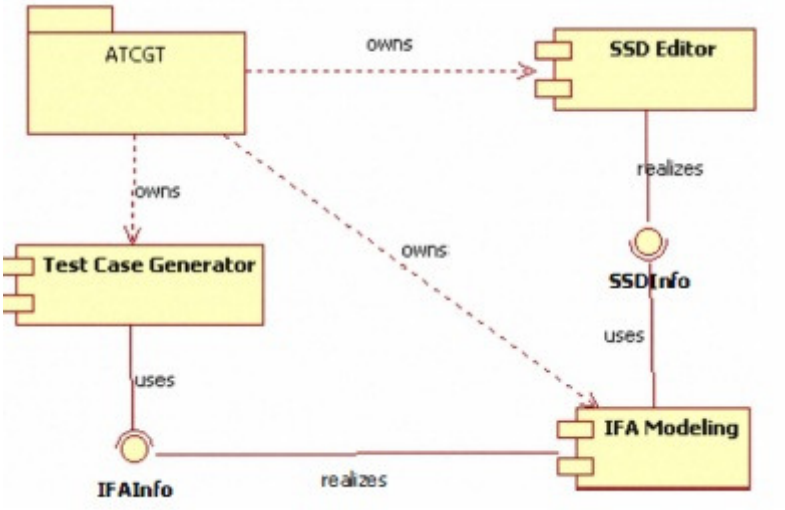


Figura 5: Arquitetura da ferramenta ATCGT
Fonte: Chen e Li (2010)

O Editor SSD é usado para especificar casos de uso com SSD e prepara a entrada para o Modelador IFA com a realização chamada SSDInfo. Quando o SSD é submetido, o IFA é modelado automaticamente no Modelador IFA que gera uma representação textual para os usuários possam navegar e verificar. A interface chamada IFAInfo é criada para fornecer o insumo para geração de casos de teste.

O gerador de casos de teste implementa algoritmos específicos e após sua configuração gera um conjunto de caminhos de teste. Com a semântica fornecida, cada caminho de teste é mapeado para um caso de teste real.

Im et. al (2008) utilizaram a ferramenta *open source* de desenvolvimento, Eclipse, que com a instalação de *plugins* específicos assimiu, principalmente, as funções de: editor de ontologia, editor de caso de uso, editor de caso de teste e editor de template. E com isso auxilia os autores na geração de casos de teste.

Chatterjee e Johari (2010) desenvolveram uma ferramenta chamada "STATEST 1.1.0" que foi projetada e desenvolvida para atingir o objetivo de gerar automaticamente uma suíte de testes a partir da especificação de requisitos, que utiliza diagrama de transição de estados como a representação formalizada. A referida ferramenta assimila casos de uso, cenários e diagramas de transição de estados como elementos-chave para a geração de pacote de teste, uma vez que estes modelos são pró-ativos e eficientes para atingir o objetivo. A ferramenta possui, ainda, as seguintes características:

- A ferramenta tem um banco de dados embutido (mantido pelo engenheiro de requisitos), de onde se pega a entrada. A presença do banco de dados facilita o uso da ferramenta pelo testador, pois elimina a necessidade de introduzir manualmente as entradas para um novo cenário.

- A ferramenta atua em duas etapas principais: primeiro na formalização dos requisitos e segundo na geração automática dos casos de teste.

A ferramenta automatiza a descrição de cenário dada, declarado como etapas, categorizando em "Passos básicos" e/ou "Passos alternativos" em um digrama de transição de estados.

Uma base de dados mantém a descrição dos cenários incluindo os passos que representam as ações do usuário e um arquivo de referência ao diagrama de transição de estados junto com pré e pós-condições do diagrama.

Outra base de dados mantém gravados os casos de teste gerados pela ferramenta. Duas tabelas são utilizadas, uma grava os casos de teste gerados a partir do fluxo principal e outra grava os casos de teste gerados a partir do fluxo alternativo

Nebut et. al (2006) desenvolveram duas ferramentas para auxiliar na geração automática dos casos de testes, são elas:

- UC-System: constrói um modelo de simulação a partir dos casos de uso e seus contratos e gera seqüências corretas de casos de uso.

- UC-SCSystem: esta ferramenta trabalha com o princípio da transformação dos objetivos de teste em cenários de teste que consite em substituir cada caso de uso do objetivo de teste por um dos seus cenários de caso de uso.

Ibrahima et. all (2007) desenvolveram uma ferramenta batizada de GenTCase, esta ferramenta pode ser usada para o layout do diagrama de caso de uso de qualquer sistema. Este diagrama representa um requisito funcional do sistema. A ferramenta também é capaz de gerar automaticamente casos de teste de

sistemas de acordo com o diagrama de caso de uso que tenha sido formado anteriormente.

A ferramenta possui três componentes principais, definidos como: Workspace, Engine e Test Cases.

O workspace é usado pelo usuário para fornecer os requisitos do sistema por meio de um diagrama de caso de uso, fluxo de eventos e diagrama de seqüência. No workspace é possível criar, editar e exibir os diagramas de casos de uso, fluxo de eventos e diagramas de seqüência. Uma vez que os diagramas de caso de uso tenham sido finalizados, o usuário pode desenvolver o fluxo de eventos e o diagrama de seqüências.

A Engine irá pegar todos os casos de uso, checar a consistência do fluxo de eventos com os diagramas de seqüência e procurar as palavras-chave usadas na base de dados fornecida. A base de dados consiste da maioria das palavras-chave padrão de um caso de uso. Uma vez que o caso de uso utilizado encontre a palavra-chave dentro da base de dados, a engine gerará seus respectivos casos de teste de acordo com seu caso de uso. Os casos de testes gerados automaticamente pela ferramenta são exibidos na tela e podem ser armazenados em arquivos com formato texto.

As demais técnicas estudadas não utilizaram nenhuma outra ferramenta que os auxiliasse diretamente na geração automática de casos de teste.

Uma outra ferramenta pesquisada foi a TaRGeT, que não foi referenciada por nenhum dos trabalhos selecionados, mas que tem como objetivo gerar casos de teste automaticamente a partir de casos de uso escritos em linguagem natural. A ferramenta não foi encontrada com o uso da *string* de busca, porque ela é apenas citada superficialmente por Aranha e Borba (2007) em um trabalho que trata da estimativa de esforço de teste o que não é o foco da pesquisa. Apesar disto, considerou-se esta ferramenta em função do fato da mesma ter sido desenvolvida no escopo local (Pernambuco) e utilizada no contexto real de uma empesa.

Embora a idéia da ferramente seja gerar testes a partir de casos de uso escritos em linguagem natural, os casos de uso são escritos seguindo um esquema

XML, desenvolvido para conter a informação necessária para geração do procedimento de teste, descrição e requisitos relacionados. Na Figura 6 abaixo, pode-se observar a interface da ferramenta.

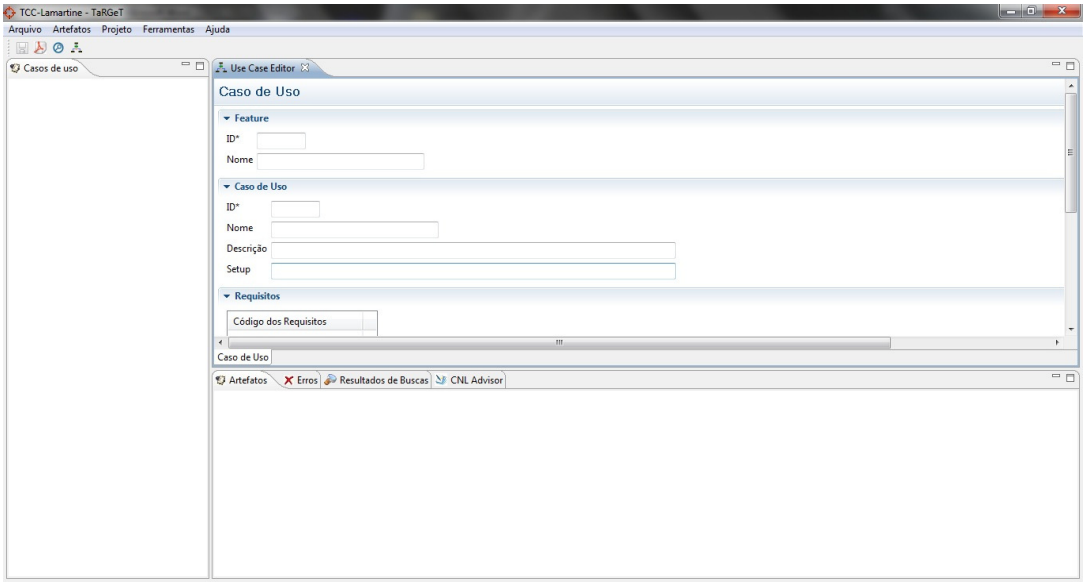
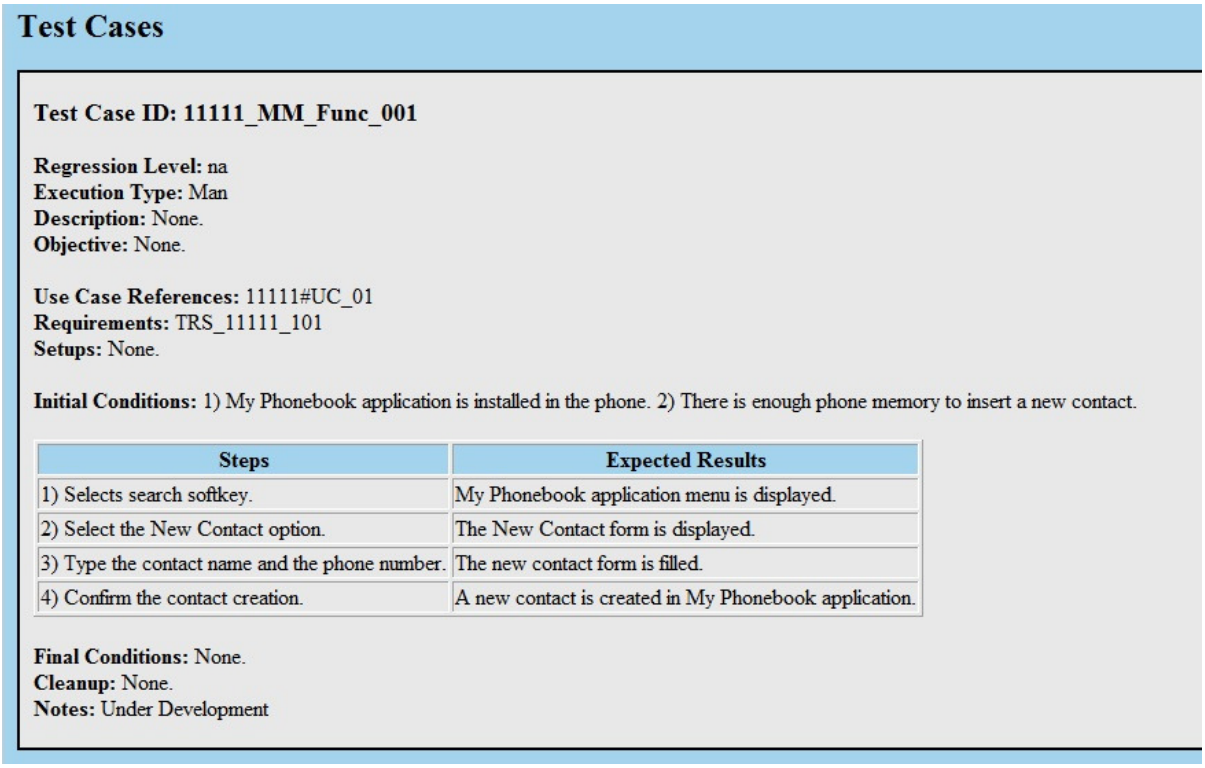


Figura 6: Interface da ferramenta TaRGeT

A ferramenta possui as seguintes características:

- Filtros que suportam a seleção de casos de teste;
- Gerenciamento que permite comparar suites de teste e manter a integridade dos id´s de casos de teste;
- Linguagem Natural Controlada – LNC que define algumas regras de escrita e um vocabulário restrito a fim de evitar que autores introduzam ambigüidades em seus passos de caso de teste;
- Editor de caso de uso que permite a criação e edição de casos de uso dentro da ferramenta;
- Permite a exportação de suites de casos de teste em vários formatos, incluindo xls, html e xml.

A Figura 7, abaixo, mostra um exemplo de caso de teste gerado pela ferramenta TaRGet. O mesmo foi gerado com o formato html.



Test Cases

Test Case ID: 11111_MM_Func_001

Regression Level: na
Execution Type: Man
Description: None.
Objective: None.

Use Case References: 11111#UC_01
Requirements: TRS_11111_101
Setups: None.

Initial Conditions: 1) My Phonebook application is installed in the phone. 2) There is enough phone memory to insert a new contact.

Steps	Expected Results
1) Selects search softkey.	My Phonebook application menu is displayed.
2) Select the New Contact option.	The New Contact form is displayed.
3) Type the contact name and the phone number.	The new contact form is filled.
4) Confirm the contact creation.	A new contact is created in My Phonebook application.

Final Conditions: None.
Cleanup: None.
Notes: Under Development

Figura 7: Exemplo de caso de teste gerado pela TaRGeT

4.5-Resumo das abordagens

A tabela a seguir faz uma síntese de maneira objetiva dos estudos analisados. Com ela pode-se observar que essa área de estudo é relativamente recente, com o estudo mais antigo datando de 2006. Outro ponto importante é que o principal problema apontado pela maior parte dos autores está relacionado com a característica informal de escrita dos casos de uso e a solução, quase sempre, apontanda para este problema é a formalização do mesmo, com o uso de técnicas diversas.

Tabela 5: Resumo das abordagens

Artigo	Principais Problemas	Soluções apontadas	Ferramentas
Chen e Li (2010)	➢ Dificuldade de processar especificação informal	Formalizar caso de uso com IFA	➢ ATCGT
Im et. al (2008)	N/A	N/A	➢ Modelador de ontologia ➢ Editor de Caso de uso ➢ Editor do caso de teste ➢ Editor de template
Xu e He (2007)	➢ Casos de uso imprecisos ➢ Casos de uso excluem concorrência e dependência	Formalização com auxílio de redes de petri e redes de petri orientada a aspectos	N/A
Somé e Cheng (2008)	➢ Dificuldade de processar especificação informal ➢ Casos de uso não implementam restrições seqüenciais ➢ Casos de uso não implementam requisistos de performance e segurança	Formalização dos casos de uso com auxílio de FSM	N/A
Chatterjee e Johari (2010)	➢ Falta de formalização dos requisitos	Formalização dos requisitos com auxílio de DST	➢ STATEST 1.1.0
Nebut et. al (2006)	➢ Possível subespecificação e erros nos requisitos ➢ Ambigüidades dos requisitos escritos em linguagem natural	Formalização dos requisitos com o auxílio de contratos (expressões lógicas= predicados + operadores lógicos)	➢ UC-System ➢ UC-SCSystem
Ibrahima et. al (2007)	N/A	N/A	➢ GenTCase
Aranha e Borba (2007)	N/A	N/A	➢ TaRGeT

4.6-Qualidade dos estudos

A Tabela 6 mostra a pontuação obtida por cada artigo, após avaliação dos mesmos, feita com base nos critérios de avaliação da qualidade definidos no protocolo.

Tabela 6: Pontuação geral de cada artigo

Id	Título	Autor(es)	Pontuação
01	Automated test case generation from use case: A model based approach	Lizhe Chen; Qiang Li	8
02	Automating test case definition using a domain specific language	Kyungsoo Im, Tacksoo Im, John D. McGregor	10
03	Generation of test requirements from aspectual use cases	Dianxiang Xu, Xudong He	6
04	An approach for supporting system-level test scenarios generation from textual use cases	Stéphane S. Somé, Xu Cheng	5
05	A prolific approach for automated generation of test cases from informal requirements	Ram Chatterjee, Kalpana Johari	12
06	Automatic test generation: a use case driven approach	Nebut, C. ; Fleurey, F. ; Le Traon, Y. ; Jezequel, J. M.	20
07	An Automatic Tool for Generating Test Cases from the System's Requirements	Rosziati Ibrahima; Mohd Zainuri Saringat; Noraini Ibrahim; Noraida Ismailb.	9

A partir dos dados da tabela acima foi gerada uma estratificação dos estudos para uma melhor análise, conforme ilustra o gráfico abaixo.

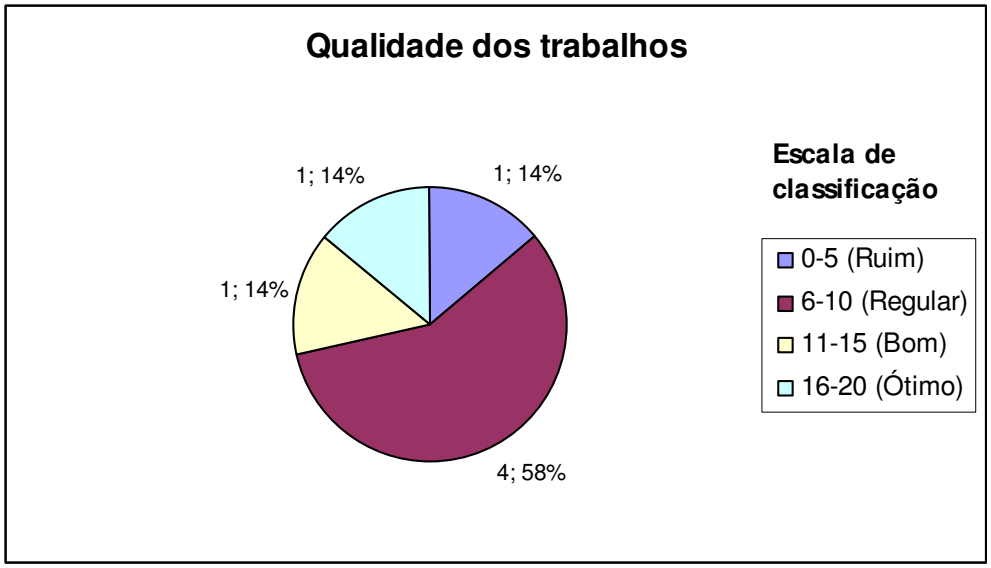


Figura 8: Qualidade dos estudos

Com base no gráfico acima, pode-se perceber que 86% dos estudos avaliados estão classificados como regular, bom ou ótimo; apenas 14%, o que equivale a um estudo, foi classificado como ruim. Portanto, pode-se afirmar que as conclusões obtidas por este estudo baseiam-se em estudos, na sua maioria, de média e boa qualidade.

Capítulo 5
Conclusão e trabalhos futuros

A idéia da geração automática de casos de teste a partir de casos de uso surge para diminuir o esforço empregado na atividade manual de geração dos testes que é, de fato, tediosa e consome recursos importantes da atividade de testes. A geração automática de casos de teste a partir de casos de uso, envolve várias etapas. Embora cada técnica estudada siga um caminho diferente para chegar ao mesmo objetivo, há um consenso entre elas de que uma das primeiras etapas a ser executada é a formalização dos casos de uso. Isso se deve principalmente as imprecisões obervadas na linguagem informal com a qual são escritos os casos de uso. Ambigüidades, erros de especificação, subespecificação, entre outros problemas originados da atividade humana são passíveis de serem observados na descrição dos casos de uso. A formalização também ajudar a resolver outro problema observado na automação que é a dificuldade do computador de lidar com a linguagem natural ou informal com a qual, normalmente, os casos de uso são escritos.

A formalização dos casos de uso com o objetivo de gerar casos de teste automaticamete leva, naturalmente, a uma transferência de esforço para a atividade de especificação.

Durante o trabalho realizado seguindo o guia de Kitchenham (2007) para mapeamento sistemático, a pesquisa realizada nas bibliotecas digitais com o auxílio da *string* de busca definida no protocolo, trouxe diversos artigos a respeito da geração automática de casos de teste a partir de linguagem UML.

Como proposta para trabalhos futuros, seria interessante realizar os seguintes estudos:

- Realizar um novo mapeamento sistemático para a geração automática de casos de teste a partir de linguagem UML, uma vez que a string de busca retornou diversos artigos que abordavam esta temática.

- Realizar um novo mapeamento sistemático com o mesmo tema proposto neste trabalho, só que utilizando outras fontes de pesquisa, uma vez que com as bases utilizadas poucos trabalhos foram retornados e além disso, conforme foi explicado no texto, bases reconhecidas de pesquisa não foram utilizadas.

- Pesquisar em empresas e fábricas de software se elas implementam alguma técnica de geração automática de casos de teste, se desenvolveram alguma ferramenta própria para tal. Isso seria interessante, pois outras empresas talvez tenham interesse em implementar as técnicas e/ou as ferramentas.

Bibliografia

ARANHA, E.; BORBA, P. **Test Effort Estimation Models Based on Test Specifications.** Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007), 2007.

COORDENAÇÃO DE APERFEIÇOAMENTO DE PESSOAL DE NÍVEL SUPERIOR – CAPES. **Qualis Periódicos**. Disponível em: <http://www.capes.gov.br/avaliacao/qualis>. Acesso em: 22/06/2011.

CHATTERJEE, R.; JOHARI, K. **A prolific approach for automated generation of test cases from informal requirements.** SIGSOFT Software Engineering Notes. 2010.

CHEN, L.; LI, Q. **Automated test case generation from use case: A model based approach.** Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on. 2010.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software.** 1ª Edição. Rio de Janeiro – RJ. 2007.

HEUMANN, J. **Generating Test Cases From Use Cases.** The Rational Edge e-zine for the rational community. 2001.

IBRAHIMA, R.; SARINGAT, M. Z.; IBRAHIMA N.; ISMAILB, N. **An Automatic Tool for Generating Test Cases from the System's Requirements.** Seventh International Conference on Computer and Information Technology. 2007.

IM, K.; IM, T.; MACGREGOR, J. D. **Automating test case definition using a domain specific language.** 46th Annual Southeast Regional Conference. 2008.

KITCHENHAM, B. **Guidelines for performing Systematic Literature Reviews in Software Engineering.** Version 2.3 EBSE Technical Report, EBSE-2007-01.

MOLINARI, L. **Inovação e Automação de Testes de Software.** 1ª Edição. São Paulo-SP. 2010.

MOLINARI, L. **Testes de Software Produzindo Sistemas Melhores e Mais Confiáveis.** 2ª Edição. São Paulo-SP. 2005.

NEBUT, C.; FLEUREY, F.; LE TRAON, Y.; JEZEQUEL, **J. M. Automatic test generation: a use case driven approach.** IEEE Transactions on software engineering. 2006.

PRESSMAN, R. S. **Engenharia de Software.** 3ª Edição. São Paulo-SP. 1995.

RATIONAL UNIFIED PROCESS **Best Practices for Software Development Teams**. 2003. Disponível em: <http://www.ibm.com/developerworks/rational/library>. Acesso em: 05/04/2011.

SOMÉ, S. S.; CHENG, X. **An approach for supporting system-level test scenarios generation from textual use cases.** 2008 ACM symposium on Applied computing. 2008.

SOMMERVILLE, I. **Engenharia de Software.** 6ª Edição. São Paulo – SP. 2003.

XU, D.; HE, X. **Generation of test requirements from aspectual use cases.** 3rd workshop on Testing aspect-oriented programs. 2007.

Apêndice A

Trabalhos Incluídos

Trabalhos incluídos					
Id	**Fonte**	**Título**	**Autor(es)**	**Local de Publicação**	**Ano**
01	IEEE Xplorer	Automated test case generation from use case: A model based approach	Lizhe Chen; Qiang Li	Computer Science and Information Tecnology (ICCSIT), 2010 3rd IEEE International Conference	2010
02	ACM Digital Library	Automating test case definition using a domain specific language	Kyungsoo Im, Tacksoo Im, John D. McGregor	46th Annual Southeast Regional Conference	2008
03	ACM Digital Library	Generation of test requirements from aspectual use cases	Dianxiang Xu, Xudong He	3rd workshop on Testing aspect-oriented programs	2007
04	ACM Digital Library	An approach for supporting system-level test scenarios generation from textual use cases	Stéphane S. Somé, Xu Cheng	2008 ACM symposium on Applied computing	2008
05	ACM Digital Library	A prolific approach for automated generation of test cases from informal requirements	Ram Chatterjee, Kalpana Johari	SIGSOFT Software Engineering Notes	2010
06	IEEE Xplorer	Automatic test generation: a use case driven approach	Nebut, C. ; Fleurey, F. ; Le Traon, Y. ; Jezequel, J. M.	IEEE Transactions on software engineering	2006
07	IEEE Xplorer	An Automatic Tool for Generating Test Cases from the System's Requirements	Rosziati Ibrahima; Mohd Zainuri Saringat; Noraini Ibrahim; Noraida Ismailb.	Seventh International Conference on Computer and Information Technology	2007

Apêndice B

Fichamento dos artigos

B.1 - Automated test case generation from use case: A model based approach

Formulário de Coleta de Dados					
Id:	01	**Título:**	Automated test case generation from use case: A model based approach	**Ano da Publicação:**	2010
Autores:	Lizhe Chen; Qiang Li				

FUNDAMENTAÇÃO E TÉCNICA:

Caso de uso é composto de muitos eventos, que descrevem comportamentos de um sistema sem revelar a estrutura interna do mesmo. Casos de testes são geralmente derivados a partir de fluxos de eventos dos casos de uso, incluindo o fluxo básico e os fluxos alternativos.

Enquanto o método manual é considerado ineficiente e tedioso, tecnologias de automação são propostas para gerar casos de teste. Na teoria, automação requer modelos que forneçam a informação necessária. Isto é, para automatizar a geração de caso de teste a partir da especificação, um modelo apropriado precisa ser construído. Este modelo precisa representar a semântica da especificação formalmente. Só então, algoritmos relevantes podem ser desenvolvidos e implementados.

Os autores propõem a formalização dos casos de uso a partir de um método que os autores batizaram de Autômato de Interação Finito – IFA. Baseado no IFA, um grupo de critérios de teste é representado para gerar casos de teste e algoritmos relevantes são também representados. Para adotar a abordagem acima na prática,

foi implementado um protótipo de uma ferramenta automática para geração de casos de teste (Automated Test Case Generation Tool - ATCGT)

Uma definição formal do IFA é a seguinte:

IFA = (S, s0, Sf, T, δ), onde:

S é um conjunto finito de estados

s0 ϵ S é o estado inicial único. Geralmente, a pré-condição de um caso de uso, pode ser considerada como um único estado.

Sf c S é um conjunto de todos os estados finais

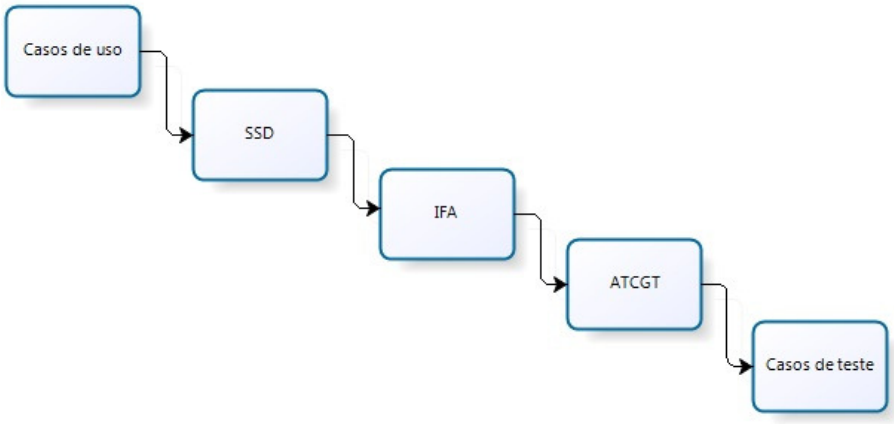
T é um conjunto de disparadores de eventos

δ : S x T → S é a função total que representa as transições entre todos os estados.

Os autores descrevem algoritmos que transformam casos de uso em IFA.

O IFA gerado representa fluxos de eventos de casos de uso como um modelo baseado em estados. O IFA contém apenas dois tipos de elementos: estados e transições entre estados. Na abordagem dos autores o IFA é considerado como a entrada da geração de casos de teste.

De maneira geral a proposta dos autores pode ser sumarizada na figura abaixo.



DIFICULDADES:

Em geral, a especificação de caso de uso em si é informal e muito difícil de processar diretamente pelo computador. Assim, um transformação formal é necessária.

Com a cobertura de transição, cada fluxo de evento do caso de uso pode ser testado. Mas falhas podem estar presentes quando alguns estados ou transições executam repetidamente. Embora a cobertura enumere todos os caminhos possíveis de teste do IFA, um looping pode estar presente e gerar um número inaceitável de caminhos de teste. Deste modo, restringir a cobertura é recomendado nesta abordagem.

MELHORES PRÁTICAS:

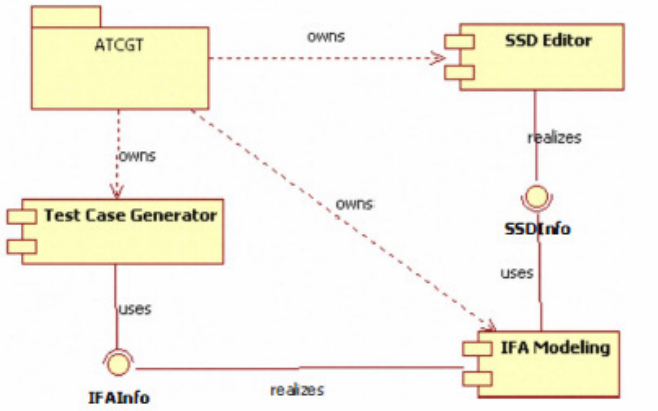
Os autores argumentam que muitas das teorias consideram máquina de estado ou diagramas de estados como o modelo para formalizar caso de uso.

A transformação formal do caso de uso é necessária.

FERRAMENTAS UTILIZADAS:

A ferramenta desenvolvida pelos autores para a geração automática de casos de teste foi denominada ‘Ferramenta Automatizada para Geração de Casos de Testes – ATCGT’ e pode ser utilizada para gerar casos de teste a partir de casos de uso de acordo com a abordagem corrente.

A ATCGT é composta de 3 subsistemas: Editor de SSD, modelador de IFA e o gerador de casos de teste.



O Editor SSD é usado para especificar casos de uso com SSD e prepara a entrada para o Modelador IFA com a realização chamada SSDInfo.

Quando o SSD é submetido o IFA é modelado automaticamente no Modelador IFA que gera uma representação textual para os usuários possam navegar e verificar. A interface chamada IFAInfo é criada para fornecer o insumo para geração de casos de teste.

O gerador de casos de teste implementa algoritmos específicos e após sua configuração gera um conjunto de caminhos de teste. Com a semântica fornecida, cada caminho de teste é mapeado para um caso de teste real.

OUTRAS INFORMAÇÕES RELEVANTES:

Cenários podem ser especificados com linguagem natural, tabelas ou diagramas definidos em UML. Nesta abordagem, diagrama de seqüência do sistema – SSD é fortemente recomendada para representar cenários.

SSD é um tipo de diagrama de seqüência, que não mostra os objetos internos do sistema. No SSD, um sistema é considerado como uma entidade única que interage com os atores.

CONCLUSÕES E TRABALHOS FUTUROS:

A fim de gerar casos de teste a partir de casos de uso, os autores propuseram a criação de um modelo baseado em estados chamado IFA. A Semântica modelo do IFA é derivada da classificação das interações entre os atores e um sistema. Assim,

o modelo formal de casos de uso só depende das interações exteriores sem conhecer as estruturas internas dos sistemas.

O IFA é considerado a entrada principal para gerar os casos de teste. Os algoritmos necessários para a geração automática bem como os subsistemas necessários para tal, encontram-se na ferramenta desenvolvida pelos autores e denominada ATCGT – Ferramenta Automatizada para Geração de Casos de Teste.

AVALIAÇÃO DA QUALIDADE:

Questão	Pontuação
Q1	4
Q2	0
Q3	0
Q4	4
Q5	0
TOTAL	8

B.2 - Automating test case definition using a domain specific language

Formulário de Coleta de Dados					
Id:	02	**Título:**	Automating test case definition using a domain specific language	**Ano da Publicação:**	2008
Autores:	Kyungsoo Im, Tacksoo Im, John D. McGregor				

FUNDAMENTAÇÃO E TÉCNICA:

Segundo os autores o teste é uma busca por falhas e testar um pedaço de software envolve três fases essenciais. Primeiro, os testes são planejados. Em segundo lugar, os testes e a infra-estrutura de teste são construídos. Um ambiente em que o software sob teste será executado é montado ou construído. Finalmente, os testes são executados e os resultados são avaliados. Muitos trabalhos focam na etapa final e ambientes, tais como Junit que tem sido amplamente utilizado na automatização da fase de execução e avaliação. Esta fase é repetida com freqüência para verificar as alterações na aplicação.

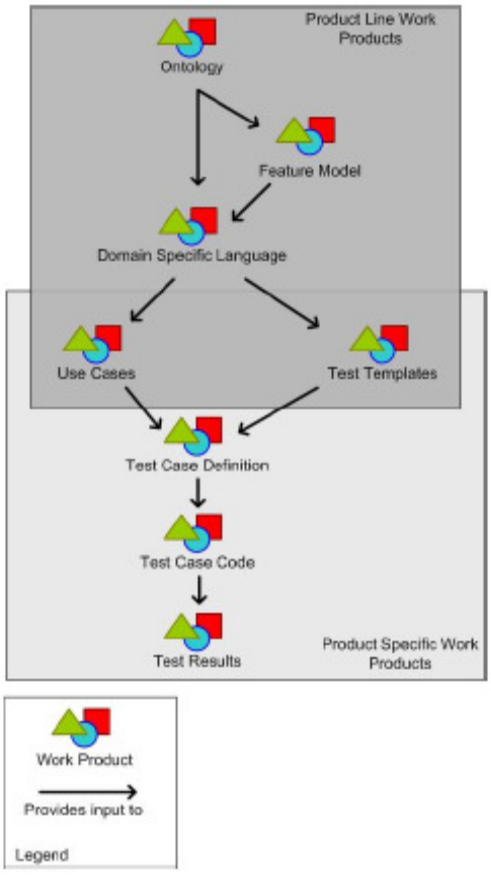
O foco do trabalho está na primeira e segunda fases, quando os testes são planejados e construídos fisicamente. O planejamento do teste e fases da construção são repetidos com menos frequência do que a execução e a fase de avaliação, mas particularmente no desenvolvimento iterativo essas fases serão repetidas muitas vezes. O objetivo é reduzir o esforço destas duas fases.

Uma linguagem de domínio específico - DSL é uma notação que os designers de produtos podem definir o produto e os testadores podem usar para criar os casos de teste. Linguagens de domínio específico são linguagens adaptadas a um determinado domínio. Elas fornecem notações e constroem um domínio de aplicação o que resulta em significativa facilidade de uso.

Os casos de uso mostram a utilização de um sistema a partir da perspectiva do usuário. Por esta razão, os casos de uso servem como especificações valiosas

para a concepção de casos de teste ao nível do sistema. Existem uma série de abordagens na geração de casos de teste baseado em casos de uso. Algumas destas abordagens transformam o caso de uso em um gráfico de estados ou outras notações gráficas manuais para, então, derivar casos de teste.

A técnica de automação de teste dos autores é dividida em dois segmentos de atividades: o segmento de linha de produto e o segmento de um produto específico. Em uma linha de produto de software de uma organização, o time da linha de produto cria os ativos fundamentais, os ativos que são utilizados em vários produtos da linha de produtos. O núcleo da equipe de desenvolvimento da linha de produtos cria uma infra-estrutura, incluindo a linguagem de domínio específico, o teste de modelos, e cria a parte do modelo de casos de uso e suíte de testes que são aplicáveis aos vários produtos. Times de desenvolvimento de produto específico desenvolvem ativos únicos para seus produtos. As atividades do time de ativos principais da linha de produtos resulta nos artefatos ilustrados na figura abaixo.



Uma linguagem de domínio específico, que deve ser suficientemente expressiva para apoiar a descrição de todos os produtos em uma linha de produtos, é criada em primeiro lugar. Casos de uso e modelos de teste são criados usando DSL e podem ser desenvolvidos em paralelo.

A técnica usada pelos autores para criar a DSL combina os conceitos de domíno e as relações capturadas na análise do domínio com as propriedades visíveis ao usuário com relação ao produto, identificadas durante a análise das características.

A análise de domínio identifica os conceitos e relações entre os conceitos no domínio da aplicação e captura-os em uma ontologia. É processado uma série de documentos escritos sobre o domínio, usando uma ferramenta que identifica os substantivos e os verbos nas frases. A ferramenta captura um vocabulário de partida para o domínio. Especialistas de domínio auxiliam na identificação das relações entre os substantivos e as ações sobre os conceitos, os verbos. A ontologia é representada na Web Ontology Language - OWL para que ela possa ser manipulada automaticamente.

A linguagem de domínio específico evolui naturalmente como os substantivos e verbos encontrados na ontologia que são combinados em frases e usados para descrever características dos produtos como eles são adicionados às características dos modelos. A linguagem é capturada em forma de frases que são úteis para a próxima etapa - a utilização na definição de casos de uso. Neste trabalho, desenvolveu-se frases padrão que são adequadas para estímulos e porções de respostas de descrições de casos de uso.

A especificação de cada produto é construída como um conjunto de casos de uso. Os estímulos e as respostas em cada caso de uso são escritos usando as frases na linguagem de domínio específico. Um editor personalisado é gerado a partir da DSL para o uso na construção de casos de uso. O editor guia a definição do caso de uso apresentando o conjunto de frases definidas para estimulos e respostas e permitindo que a pessoa que define o caso de uso selecione as frases apropriadas.

Um conjunto de modelos de teste é criado a partir de padrões de teste. Os padrões de teste são associados com frases da DSL e são escolhidos com base na descrição DSL e no tipo de teste que está sendo planejado. O teste do sistema irá utilizar os padrões que são baseados em como o produto será utilizado, conforme descrito nos casos de uso, em vez de como o produto é construído. Os modelos de teste derivados dos padrões estão associados com os requisitos.

Os casos de teste são gerados a partir dos casos de uso e dos modelos de teste e as frases da DSL estão associadas com os modelos de teste. Este mapeamento é incorporado em um editor de caso de teste que é gerado automaticamente. O testador usa o editor de teste para criar os casos de teste necessários para satisfazer os objetivos do teste.

DIFICULDADES:

Os autores afirmam que muito progresso tem sido feito na execução automática de casos de teste, mas pouco progresso tem sido feito na definição automática dos testes.

Os autores afirmam que a utilização da técnica descrita se adapta melhor a uma linha de produção de software, onde é possível se aproveitar componentes para softwares diversos.

MELHORES PRÁTICAS:

N/A

FERRAMENTAS UTILIZADAS:

Modelador de ontologia: Um plugin para Eclipse prevê um modelador OWL (Web Ontology Language). Especialistas de domínio irão usar o editor para definir as entidades que comporão a DSL.

Editor de Caso de uso: é gerado a partir de uma linguagem de domínio específico. As entradas para a definição do caso de uso vem da seleção dos menus dos elementos da DSL, que é feita pelo usuário do editor. Restrições sobre a

linguagem são incorporadas no editor que impede o usuário de entrar com expressões de linguagem inválidas.

O editor do caso de teste: A entrada para o editor de caso de teste vem de uma transformação de cada caso de uso para uma novo meta-modelo.

Editor de template: Sempre que um novo modelo é necessário, um editor de texto padrão é usado para produzir o código para o modelo. A linguagem JET de definição de modelo é semelhante a linguagem Java Server Pages.

OUTRAS INFORMAÇÕES RELEVANTES:

N/A.

CONCLUSÕES E TRABALHOS FUTUROS:

A investigação dos autores centrou-se sobre o estabelecimento de um conjunto de ferramentas para apoiar um processo de desenvolvimento orientado por modelo para a criação de casos de teste. Esta investigação foi no contexto do esforço de uma linha de produto de software. A técnica usa uma linguagem de domínio específico – DSL como a linguagem em que casos de uso e casos de teste são escritos. Aproveitando-se dos padrões de teste, são criados modelos de apoio à geração automática de casos de teste executáveis. Além disso, os experimentos mostraram que a maioria do esforço é gasto no nível de linha de produtos e este esforço é então amortizado em todos os produtos de software que são construídos na linha de produtos.

AVALIAÇÃO DA QUALIDADE:

Questão	Pontuação
Q1	4
Q2	2
Q3	0
Q4	4
Q5	0
TOTAL	10

B.3 - Generation of test requirements from aspectual use cases

Formulário de Coleta de Dados					
Id:	03	**Título:**	Generation of test requirements from aspectual use cases	**Ano da Publicação:**	2007
Autores:	Dianxiang Xu, Xudong He				

FUNDAMENTAÇÃO E TÉCNICA:

Este trabalho apresenta uma abordagem para geração de requisitos de teste de sistema a partir de casos de uso orientado a aspectos. O foco desta abordagem é a formalização de um modelo de sistema testável a partir de casos de uso orientado a aspectos. Foram capturadas várias restrições entre casos de uso base e aspectual. Especificamente, foram transformados diagramas e descrições de casos de uso orientado a aspectos em redes de petri orientada a aspectos. Redes de petri são um método formal bem definido, com notação gráfica e matemática para especificação e análise de sistemas distribuídos. A formalização de casos de uso orientada a aspectos permite gerar significativas seqüências de caso de uso com relação a vários critérios de cobertura (como a cobertura de casos de uso, cobertura de transição e cobertura de estado). Quando cenários de teste para casos de uso individuais estão disponíveis, eles podem ser compostos em testes do sistema de acordo com a sequência de casos de uso gerado.

Na abordagem caso de uso, o comportamento requerido de um sistema é definido com uma coleção de casos de uso. Um diagrama de caso de uso retrata atores, casos de uso e relações entre eles. As relações primarias entre casos de uso são inclusão, extensão e generalização. Cada caso de uso implica em uma unidade funcional útil que o sistema fornece aos seus atores. Esta especificação tipicamente inclui o fluxo básico (cenário de sucesso), fluxo alternativo (variações do fluxo básico), sub-fluxos (para repetições no fluxo básico), pontos de extensão (onde comportamentos adicionais podem ser inseridos), precondições, pós-condições, etc.

Na abordagem para análise de requisitos orientada a aspectos com caso de uso, mantém o uso tradicional de extensões (comportamentos opcionais) e adota os casos de uso aspectual para descrever requisitos transversais. Um caso de uso aspectual consiste de pointcuts e fluxos (básico e alternativo) de conselhos, cada pointcut é uma coleção de pontos de junção, que se referem aos passos dos fluxos básicos, fluxos alternativos, etc, em casos de uso base. Casos de uso base não precisam saber onde os aspectos serão aplicados. Isso é útil para muitas características transversais (segurança, por exemplo) que são adições aos casos de uso base.

Redes de petri orientada a aspectos são motivadas a capturar as características essenciais (join points, pointcuts, advice, etc.) da orientação a aspectos para modelagem de sistemas com redes de petri. Este artigo é baseado em redes predicado/transição (PrT). Similar as noções de Programação Orientada a Aspectos (POA ou AOP), um modelo de rede de petri orientada a aspectos, consiste de uma rede de petri básica, rede de petri baseada em aspectos, e precedencia em aspectos. Cada aspecto é uma entidade encapsulada de pointcuts, advice,declarações inter-modelo e introduções. Especifica uma separação preocupada com respeito as redes base. Um pointcut define um grupo de join points (ou seja, transições, lugares e arcos) em redes base.

Em suma, os caso de uso tradicionais bem como os casos de uso orientado a aspectos são formalizados para redes de petri orientadas a aspectos.

DIFICULDADES:

Especificações de requisitos são freqüentemente cheias de nuances e ambigüidades.

A abordagem de caso de uso tradicional também tem suas limitações. Casos de uso de extensão (ou simples extensões), que são adicionados aos casos de uso base, requerem designação a princípio de todos os pontos de extensão; preocupação igual que são casos de uso distintos com realizações sobrepostas, pode resultar em efeitos de espalhamento e emaranhamento.

Cada caso de uso descreve um sistema de funcionalidade a partir da perspectiva do usuário. Ambigüidades, contradições, omissões e imprecisões são freqüentemente encontrados na forma textual de descrições de caso de uso.

O diagrama de caso de uso, em geral, não captura restrições sequenciais. Outros tipos de dependências podem existir. Exemplos incluem concorrência e alternância. As dependências são resultado direto da lógica de negócio que o sistema propõe suportar. A abordagem de caso de uso tradicional exclui concorrência e dependência dos diagrmas de caso de uso. Embora isto possa manter diagramas de caso de uso simples, isto levanta um problema de testabilidade. Quando planejamos os testes do sistema, precisamos identificar possíveis sequências de execução de casos de uso. É provável que eles disparem diferentes falhas durante os testes. Para resolver o problema de testabilidade, as dependências entre casos de uso devem ser explicitamente modeladas. Os métodos existentes para capturar as dependências, tais como diagramas de atividade UML, e gráficos de casos de uso não suportam a noção casos de uso orientados a aspectos ou validação rigorosa.

MELHORES PRÁTICAS:

Uma visão bem definida de caso de uso não é só necessário para a corretude do sistema e implementação, mas também serve como base para evolução futura do sistema.

A formalização pode detectar possíveis erros de requisitos e modelagem.

Uma vez que o caso de uso é convertido em rede de petri, pode-se utilizar uma grande variedade de técnicas de análise formal para avaliar o resultado da especificação e descobrir possíveis problemas na descrição original do caso de uso e, ainda, definir cenários de teste para diferentes tipos de cobertura.

FERRAMENTAS UTILIZADAS:

N/A.

OUTRAS INFORMAÇÕES RELEVANTES:

N/A.

CONCLUSÕES E TRABALHOS FUTUROS:

Foi apresentada uma abordagem para a geração de requisitos de teste do sistema a partir de casos de uso orientado a aspectos. Esta abordagem baseia-se na identificação de restrições entre casos de uso base e aspectual, formalização dos dois tipos como uso de rede de petri orientada a aspectos. Essa abordagem pode reduzir a distância entre a análise de requisitos orientada a aspectos e a atividade de desenvolvimento subseqüente.

AVALIAÇÃO DA QUALIDADE:

Questão	Pontuação
Q1	2
Q2	0
Q3	0
Q4	4
Q5	0
TOTAL	6

B.4 - An approach for supporting system-level test scenarios generation from textual use cases

Formulário de Coleta de Dados					
Id:	04	**Título:**	An approach for supporting system-level test scenarios generation from textual use cases	**Ano da Publicação:**	2008
Autores:	Stéphane S. Somé, Xu Cheng				

FUNDAMENTAÇÃO E TÉCNICA:

Segundo os autores teste é conhecida como uma fase chave no processo de desenvolvimento de software que conta com grande parte do esforço de desenvolvimento. Um caminho para reduzir o esforço de teste, assegurando a sua eficácia, é gerar casos de teste automaticamente a partir de artefatos usados nas fases iniciais do desenvolvimento de software. O trabalho dos autores propõe a geração de casos de teste a partir dos requisitos capturados como casos de uso.

No trabalho os autores apresentam uma abordagem para geração automática de representações abstratas de propostas de testes denominadas cenários de testes. Os autores afirmam que superaram a natureza informal dos casos de uso utilizando uma linguagem natural restrita para casos de uso. A fim de garantir uma cobertura adequada de seqüências de eventos, foi gerado uma máquina de estados de controle de fluxo a partir de caso de uso e usando técnicas de cobertura de código tradicionais para derivar seqüências de testes. Finalmente, foram inferidas relações seqüenciais entre casos de uso, baseado numa comparação de pré-condições e pós-condições de casos de uso. Isto permitiu ao autores combinar comportamentos de casos de uso em um controle de fluxo global baseado em modelo de estados.

Uma máquina de estados finito – FSM (Finite State Machine) é uma tupla (Σ, S, F, S0), onde Σ é o alfabeto, S é o conjunto de estados, F c S × Σ × S é uma função de transição e S0 é o estado inicial.

Os autores utilizaram o termo Controle de Fluxo Baseado em Máquina de Estados – CFSM para FSMs que descrevem fluxos de eventos dos casos de uso.

Os cenários são gerados a partir de uma seleção de sequências de caminho de acordo com um objetivo de cobertura. Diferentes medidas de cobertura têm sido propostas para a geração de testes baseados em FSMs.

Neste artigo, os autores focaram em três tipos básicos de medidas de cobertura que corresponde a medida de cobertura de código “clássica”.

1-Todos os estados – cobertura alcançada com um conjunto de sequências de caminhos tais que cada estado na CFSM apareça ao menos uma vez.

2-Todas as arestas – cobertura alcançada com um conjunto de sequências de caminho completas tais que cada transição na CFSM apareça ao menos uma vez.

3- Todos os caminhos com a repetição de cobertura com limite n.

DIFICULDADES:

Os autores citam três dificuldades principais para a geração automática de casos de testes a partir de casos de uso:

A primeira é que os casos de uso textuais são, em geral, escritos em linguagem natural informal.

O segundo desafio é garantir uma cobertura adequada de todas as seqüências de ações definidas em cada caso de uso.

Por fim, um terceiro problema é que restrições seqüenciais importantes entre casos de uso, são geralmente deixadas de fora dos casos de uso e são assumidas apenas implicitamente.

O número de cenários de teste gerados de acordo com o critério de cobertura de todos os caminhos é em geral grande demais para serem executados na prática.

No estágio atual da abordagem dos autores, caso de teste concreto (executável) precisam ser manualmente derivados de cenários de teste por conta de uma “lacuna de abstração” entre os cenários de teste e casos de teste concretos. Cenários de teste são representações de casos de teste no mesmo nível de abstração, como os casos de uso dos quais são derivados.

MELHORES PRÁTICAS:

N/A.

FERRAMENTAS UTILIZADAS:

N/A.

OUTRAS INFORMAÇÕES RELEVANTES:

N/A.

CONCLUSÕES E TRABALHOS FUTUROS:

Os autores concluem afirmando que a automatização total da geração de casos de teste dos requisitos é atualmente inviável. Existe uma lacuna de abstração entre os cenários de teste e casos de teste concretos que precisa ser preenchido com intervenção manual. Além disso, os requisitos incluem aspectos como performance e segurança que não são adequadamente capturados nos casos de uso. No entanto, os autores argumentam que a abordagem apresentada é útil para validar a funcionalidade necessária dos sistemas de software do ponto de vista de seus usuários. Os trabalhos futuros almejados pelos autores incluem a derivação de dados de teste usando informação capturada em um modelo de domínio.

AVALIAÇÃO DA QUALIDADE:

Questão	Pontuação
Q1	4
Q2	0
Q3	0
Q4	0
Q5	1
TOTAL	5

B.5 - A prolific approach for automated generation of test cases from informal requirements

Formulário de Coleta de Dados					
Id:	05	**Título:**	A prolific approach for automated generation of test cases from informal requirements	**Ano da Publicação:**	2010
Autores:	Ram Chatterjee, Kalpana Johari				

FUNDAMENTAÇÃO E TÉCNICA:

Segundo os autores a formalização é um esforço que implica a transformação dos requisitos informais numa forma formal em conformidade com alguma racionalidade para atingir algum objetivo desejado. O objetivo do trabalho é formalizar requisitos descritos de maneira informal em um formato que seja possível a automação de casos de teste. A geração de casos de teste, diretamente da especificação de requisitos (testes baseado em especificação), melhora a especificação de requisitos em si, provocando a localização e erradicação de ambigüidades e inconsistências na especificação de requisitos.

Um fator importante que dá ênfase em testes baseados em especificação é a característica de vulnerabilidade da especificação que é fonte principal de erros.

Teste baseados em especificação são mais vantajosos do que testes baseados em código, com ela é possível criar testes iniciais que facilitam o planejamento eficaz e a utilização dos recursos.

A técnica porposta pelos autores consiste em formalizar os requisitos do sistema utilizando um diagrama de transição de estados e através da ferramenta desenvolvida pelos autores, denominada "STATEST 1.1.0", gerar os casos de teste.

DIFICULDADES:

Falta de formalização dos requisitos.

MELHORES PRÁTICAS:

A formalização da especificação dos requisitos é o passo intermediário e imprecindível para geração automática de casos de teste a partir de requisitos.

FERRAMENTAS UTILIZADAS:

Um instrumento chamado "STATEST 1.1.0" foi projetado e desenvolvido para atingir o objetivo de gerar automaticamente uma suíte de testes a partir da especificação de requisitos, que utiliza Diagrama de Transição de Estados como a representação formalizada, realizado pela transformação da especificação de requisitos, como a etapa intermediária da atividade de formalização.

É importante saber que a "característica tediosa" das linguagens formais e as deficiências associadas a elas são as razões principais para a adoção de uma abordagem semiformal para formalização durante a concepção e o desenvolvimento da STATEST 1.1.0. O referido instrumento assimila Casos de Uso, Cenários e Diagramas de Transição de Estados como elementos-chave para a geração de pacote de teste, uma vez que estes modelos são pró-ativos e eficientes para atingir o objetivo em relação às línguagens de formalização como "Z","B", "SOFL", e semelhantes.

Características da Ferramenta:

- A ferramenta tem um banco de dados embutido (mantido pelo engenheiro de requisitos), de onde se pega a entrada. A presença do banco de dados facilita o testador em usar a ferramenta sem ter de introduzir manualmente as entradas para um novo cenário.

- A Ferramenta atua em duas etapas principais: Primeiro na formalização dos requisitos e segundo na geração automática dos casos de teste

A ferramenta automatiza a descrição de cenário dada, declarado como etapas, categorizando em "Passos básicos" e/ou "Passos alternativos" em um digrama de transição de estados.

Uma base de dados mantém a descrição dos cenários incluindo os passos que representam as ações do usuário e um arquivo de referência ao diagrama de transição de estados junto com pré e pós-condições do diagrama.

Outra base de dados mantém gravados os casos de teste gerados pela ferramenta. Duas tabelas são utilizadas, uma grava os casos de teste gerados a partir do fluxo principal e outra grava os casos de teste gerados a partir do fluxo alternativo

OUTRAS INFORMAÇÕES RELEVANTES:

N/A.

CONCLUSÕES E TRABALHOS FUTUROS:

O trabalho sintetizado neste artigo, realiza a tarefa de "Teste de Software na ausência de Código Fonte", ou seja, Black Box Test. E está orientada para testes baseado em requisitos.

Trabalhos futuros:

- Construir capacidade na futura versão de aceitar UML DST directa e gerar casos de teste e scripts de teste executável por meios automatizados.
- Construir suporte em versões futuras para importar/exportar STD criado por ferramentas similares em um formato próprio de tradução.
- Incorporar a escolha de múltiplas formas formalizadas em termos de diagramas UML variados (DST, Actividade, diagramas de seqüência) como base para gerar casos de teste e scripts de teste.
- Implementar "automação inteligente e de pleno direito" (por meio da erradicação da intervenção manual nos requisitos do engenheiro) dos requisitos de cobertura, como um dos critérios para avaliar a qualidade dos casos de teste gerado pela ferramenta de aplicação.

AVALIAÇÃO DA QUALIDADE:

Questão	Pontuação
Q1	4
Q2	4
Q3	0
Q4	4
Q5	0
TOTAL	12

B.6 - Automatic test generation: a use case driven approach

Formulário de Coleta de Dados					
Id:	06	**Título:**	Automatic test generation: a use case driven approach	**Ano da Publicação:**	2006
Autores:	Nebut, C. ; Fleurey, F. ; Le Traon, Y. ; Jezequel, J.-M.				

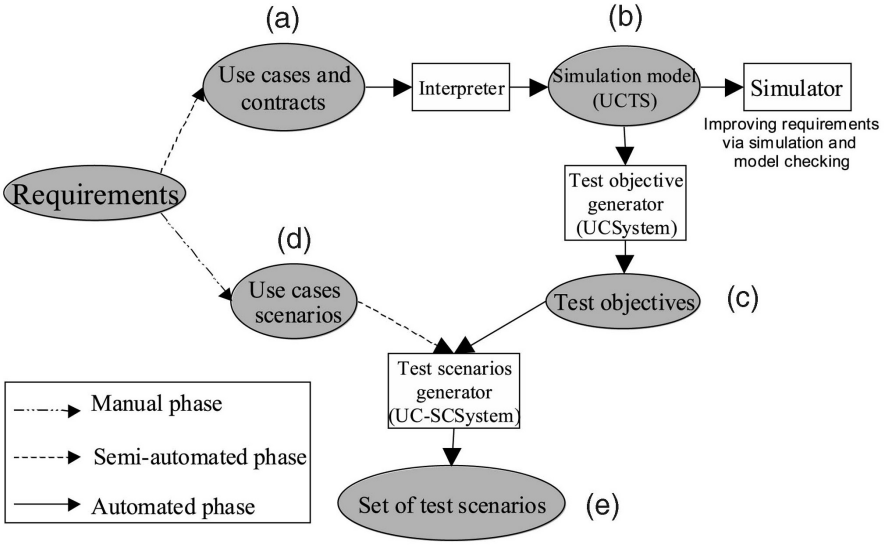
FUNDAMENTAÇÃO E TÉCNICA:

Os autores propõem uma nova abordagem para automatização da geração de cenários de testes de sistemas a partir de casos de uso no contexto de software embarcado orientado a objeto e tendo em conta os problemas de rastreabilidade entre as visões de alto nível e a execução do caso concreto de teste.

O método desenvolvido pelos autores é baseado em um modelo de caso de uso que desvenda as muitas ambigüidades dos requisitos escritos em linguagem natural. Foram construídos casos de uso UML reforçados com contratos (baseado nas pré e pós-condições dos casos de uso). Os autores propuseram tornar esses contratos executáveis, escrevendo-os na forma de expressões lógicas no nível de requisitos. Baseado nestes requisitos mais formais, mas ainda em alto nível, definiu-se um modelo de simulação de casos de uso. Desta forma, uma vez que os requisitos sejam escritos em termos de casos de uso e contratos, eles podem ser simulados em ordem para checar sua consistência e corretude. O modelo de simulação é também usado para construir explicitamente um modelo de todas as seqüências válidas de casos de uso e, a partir daí, extrair caminhos relevantes utilizando critérios de cobertura. Esses caminhos são chamados objetivos de teste. A geração dos objetivos de teste a partir dos casos de uso constitui a primeira fase da abordagem. A segunda fase tem como objetivo gerar cenários de teste a partir desses objetivos de teste.

A contribuição deste artigo é gerar testes a partir da formalização dos requisitos do sistema, no contexto de software embarcado orientado a objeto. O objetivo maior da abordagem é diminuir o trabalho na geração de teste por meio da

automação e transferir o esforço para a atividade de especificação. O objetivo é gerar eficientemente casos de teste detectando falhas em softwares embarcados.



A primeira fase do método (passos ‘a’ até ‘c’ na figura 1), tem como objetivo gerar os objetivos de teste a partir da visão de caso de uso do sistema. Casos de uso não correspondem à forma como as funções principais serão implementadas no sistema, casos de uso são visões globais do sistema. É por isso que casos de testes não podem ser gerados apenas a partir dos casos de uso. Os autores, então, propõem uma abordagem de requisitos por meio de contratos. Estes contratos não são executados antes ou depois da execução de um caso de uso, mas são usados para inferir a ordenação parcial correta de funcionalidades que o sistema deve oferecer.

Quando parâmetros são especificados para um caso de uso, contratos são especificados na forma de pré e pós-condições envolvendo esses parâmetros. Os contratos de caso de uso são expressões lógicas de primeira ordem combinando predicados com operadores lógicos. Um predicado tem um nome, uma variedade e um conjunto (potencialmente vazio) de parâmetros formais digitados. Os predicados são usados para descrever fatos no sistema.

Uma vez que a lógica booleana é usada, um predicado ou é verdadeiro ou é falso, mas nunca indefinido.

A partir dos casos de uso e seus contratos, um protótipo de ferramenta (UC-System), constrói um modelo de simulação (passo ‘b’ na figura 1) e gera seqüências corretas de casos de uso (passo ‘c’ na figura 1). Em seguida, tal seqüência correta de casos de uso é chamada de objetivo de teste. Como mostrado na figura 1, o modelo de caso de uso pode ser simulado. Esta fase de simulação permite ao engenheiro de requisitos checar e possivelmente corrigir os requisitos antes dos testes serem gerados.

A segunda fase (passos ‘c’ até ‘e’ da figura 1) tem como objetivo gerar os cenários de testes. Um cenário de teste pode ser diretamente utilizado como um caso de teste executável, ou pode precisar que o testador faça algumas alterações adicionais, caso algumas mensagens ou parâmetros estiveram faltando. Para ir do objetivo de teste para os cenários de teste, são necessárias informações adicionais, especificando as trocas de mensagens envolvidas entre o ambiente e o sistema. Tais informações podem ser anexadas a um dado caso de uso na forma de diversos artefatos: diagramas de seqüência, máquina de estados ou diagrama de atividades. Todos esses artefatos descrevem os cenários que correspondem ao caso de uso. Por questões de simplicidade, os autores lidaram apenas com diagramas de seqüência que são chamados cenários de casos de uso. O princípio da transformação de objetivos de teste para cenários de teste, consiste de substituir cada caso de uso do objetivo de teste por um dos seus cenários de caso de uso, utilizando a ferramenta protótipo UC-SCSystem. Os artefatos necessários para aplicar nessa abordagem são casos de uso com o reforço de contratos e cenários anexados a estes casos de uso. A partir destas entradas a geração de cenários de teste é automática.

A abordagem dos autores propõe associar contratos, ou seja, pré e pós-condições, para cada caso de uso, na forma de expressões lógicas.

Os objetivos de teste são gerados usando os casos de uso. Eles têm então que serem transformados em seqüências válidas de chamadas e resultados esperados do sistema em teste.

Os autores propõem derivar cenários de teste a partir de objetivos de teste usando cenários de casos de uso. Cada caso de uso é documentado por seus

contratos e cenários ilustrando como o sistema tem que ser estimulado pelos atores a fim de executar o caso de uso e verificar como o sistema deve reagir ao estímulo. Esses cenários são expressos com diagramas de seqüência UML.

DIFICULDADES:

A técnica desenvolvida pelos autores estabelece a criação de contratos que junto com os casos de uso serão utilizados para gerar automaticamente os casos de teste. Porém os autores citam algumas dificuldades na criação dos contratos, são elas:

- Dificuldade de construir um modelo de contrato de caso de uso - O analista de requisito precisa ser preciso e rigoroso na semântica atribuída a cada casos de uso, portanto tornando mais difícil a construção. Além disso, podem existir dependências entre os predicados de cada contrato e essa abordagem não trata isso.

- Tipos Numéricos - Os requisitos que podem ser expressos com contratos nos casos de uso são os de alto nível, por exemplo, eles não são adequados para lidar com tipos de dados complexos (incluindo cálculos aritméticos, por exemplo).

- Restrições nas pós-condições - Nesta abordagem as pós-condições precisam ser determinísticas.

Outras dificuldades que podem ser identificadas são: Inconsistências entre os predicados e os contratos, bem como subespecificação ou erros nos requisitos.

MELHORES PRÁTICAS:

N/A.

FERRAMENTAS UTILIZADAS:

UC-System- Constrói um modelo de simulação a partir dos casos de uso e seus contratos e gera seqüências corretas de casos de uso.

UC-SCSystem- Transforma os objetivos de teste para cenários de teste, substituindo cada caso de uso do objetivo de teste por um dos seus cenários de caso de uso.

OUTRAS INFORMAÇÕES RELEVANTES:

N/A.

CONCLUSÕES E TRABALHOS FUTUROS:

A idéia da abordagem apresentada é diminuir o trabalho para geração de teste por meio da automação e transferir o esforço para a atividade de especificação. No trabalho, foi apresentada uma cadeia completa e automatizada para geração de casos de teste a partir de requisitos formalizados. Esses casos de teste são gerados em dois passos principais: ordenações de casos de uso são deduzidas a partir de contratos de casos de uso e então cenários de casos de uso são substituídos por cada caso de uso para produzir casos de teste. Enquanto no primeiro passo, os modelos de casos de uso lidam com questões de alto nível. No segundo passo, a complexidade dos dados é levada em consideração com o uso dos cenários de casos de uso. A abordagem proposta assume a disponibilidade de uma formalização dos requisitos, que tem que ser feita manualmente. A definição deste modelo pode ser bastante difícil e sujeita a erros, enquanto a eficácia dos testes gerados, depende fortemente da qualidade deste modelo.

AVALIAÇÃO DA QUALIDADE:

Questão	Pontuação
Q1	4
Q2	4
Q3	4
Q4	4
Q5	4
TOTAL	20

B.7 - An Automatic Tool for Generating Test Cases from the System's Requirements

Formulário de Coleta de Dados					
Id:	07	**Título:**	An Automatic Tool for Generating Test Cases from the System's Requirements	**Ano da Publicação:**	2007
Autores:	Rosziati Ibrahima; Mohd Zainuri Saringat; Noraini Ibrahim; Noraida Ismailb.				

FUNDAMENTAÇÃO E TÉCNICA:

O ciclo de vida convencional do desenvolvimento de software é composto pela análise, design, desenvolvimento e teste. No entanto, atrasar os testes até o último estágio fará com que o custo do teste e o custo para corrigir os erros sejam maiores.

O artigo aborda testes de software mostra a presença de falhas e propõe uma ferramenta automática de teste para gerar casos de teste automaticamente.

Os autores argumentam que casos de teste são manualmente gerados a partir de casos de uso e esse trabalho manual requer mais tempo e é mais tedioso. No entanto, existem muitos pesquisadores estudando caminhos para automatizar o processo de gerar casos de teste. Os autores argumentam que alguns desses pesquisadores propõem um esquema que combina o processo de configuração, a execução de teste e a validação de teste em um simples programa de teste para testar o comportamento de classes orientadas a objeto. O programa de teste é gerado automaticamente usando os casos de teste e as especificações fechadas de classes.

Porém a proposta dos autores é baseada no trabalho feito por Heumann (2001), que trata da geração manual de casos de teste usando casos de uso e fluxos de eventos. A proposta dos autores é automatizar o processo de geração de casos de teste para reduzir os custos de teste e economizar o tempo de gerar os casos de teste manualmente. Por isso, esta pesquisa irá utilizar diagramas de caso

de uso com base nos requisitos do sistema para gerar os casos de testes automaticamente.

Os requisitos do sistema são normalmente capturados durante a fase de análise. Os requisitos são classificados em funcionais e não-funcionais. Os requisitos funcionais se preocupam com as tarefas que o novo sistema irá executar enquanto que os requisitos não funcionais se preocupam com todas as restrições do novo sistema. Os requisitos funcionais descrevem o que o sistema deve fazer e esses requisitos são usualmente representados usando a especificação do sistema antes do sistema ser implementado. Os casos de uso são usados para definir os requisitos do sistema. Os casos de uso representam as funcionalidades do sistema Na maioria das vezes, cada caso de uso é convertido em ma função representando as tarefas do sistema. Embora, possa se converter um ou mais casos de teste de cada caso de uso.

DIFICULDADES:

Na abordagem, não foram considerados os requisitos não-funcionais uma vez que esses requisitos apenas se preocupam com as restrições do sistema e como o sistema é implementado e não o que o sistema deve fazer.

MELHORES PRÁTICAS:

N/A.

FERRAMENTAS UTILIZADAS:

A ferramenta que os autores batizaram de GenTCase (Gerador de Caso de Teste), pode ser usada para o layout do diagrama de caso de uso de qualquer sistema. Este diagrama de caso de uso representa um requisito funcional do sistema. A ferramenta também é capaz de gerar automaticamente casos de teste de sistemas de acordo com o diagrama de caso de uso que tenha sido formado anteriormente.

A ferramenta é desenvolvida usando orientação a objeto com C++. Ela tem 3 componentes principais: Workspace->Engine->Test Cases

O workspace é usado pelo usuário para fornecer os requisitos do sistema por meio de um diagrama de caso de uso, fluxo de eventos e diagrama de seqüência. No workspace uma caixa de ferramenta é usada para criar, editar e exibir os diagramas de casos de uso, fluxo de eventos e diagramas de seqüência.

Uma vez que os diagramas de caso de uso tenham sido finalizados, o usuário pode desenvolver o fluxo de eventos e o diagrama de seqüências usando o espaço de trabalho fornecido. Então o usuário pode gerar os casos de teste usando o gerador da ferramenta. A Engine irá pegar todos os casos de uso, checar a consistência do fluxo de eventos com os diagramas de seqüência e procurar as palavras-chave usadas na base de dados fornecida. A base de dados consiste da maioria das palavras-chave padrão de um caso de uso. Uma vez que o caso de uso utilizado encontre a palavra-chave dentro da base de dados, a engine gerará seus respectivos casos de teste de acordo com seu caso de uso.

Os casos de testes gerados automaticamente pela ferramenta são exibidos na tela e podem ser armazenados em arquivos .txt.

OUTRAS INFORMAÇÕES RELEVANTES:

N/A.

CONCLUSÕES E TRABALHOS FUTUROS:

GenTCase é uma ferramenta que é capaz de gerar casos de teste automaticamente de acordo com os requisitos do sistema. Os requisitos do sistema são transformados em um diagrama de caso de uso, fluxo de eventos e diagrama de seqüências. O fluxo de eventos e o diagrama de seqüências são usados como informação adicional dar maior consistência e validade para a geração de casos de teste. A proposta da ferramenta e reduzir os custos de teste dos sistemas e economizar o tempo que seria gasto com a produção manual de casos de teste.

A ferramenta tem limitações, como a incapacidade de utilizar requisitos não-funcionais na geração de casos de teste.

Para trabalhos futuros, os autores têm a intenção de estender a ferramenta para que ela possa trabalhar com diagramas de atividades e com isso gerar casos de testes mais eficientes.

AVALIAÇÃO DA QUALIDADE:

Questão	Pontuação
Q1	2
Q2	2
Q3	0
Q4	4
Q5	1
TOTAL	9