



Universidade Federal de Pernambuco

Centro de Informática

Mestrado em Ciência da Computação

Ricson José de Santana

Seleção de Casos de Teste Baseada em Similaridade de Texto e Cobertura de Requisitos

Recife/PE
Fevereiro, 2011

Ricson José de Santana

**Seleção de Casos de Teste Baseada em Similaridade de Texto e
Cobertura de Requisitos**

Dissertação apresentada ao Curso de Mestrado
em Ciência da Computação como requisito
parcial à obtenção do grau de Mestre em
Ciência da Computação.

Orientadora: Profa. Flávia de Almeida Barros

Recife/PE
Fevereiro, 2011

Catálogo na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Santana, Ricson José de

Seleção de casos de teste baseada em similaridade de texto e cobertura de requisitos / Ricson José de Santana - Recife: O Autor, 2011.
77 p. : fig., tab.

Orientador: Flávia de Almeida Barros.

Dissertação (mestrado) Universidade Federal de Pernambuco. Cln. Ciência da computação, 2011.

Inclui bibliografia e apêndice.

1. Engenharia de software. 2. Teste de software. I. Barros, Flávia de Almeida (orientadora). II. Título.

005.1


CDD (22. ed.)

MEI2011 – 029

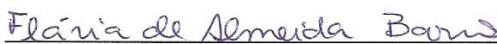
Dissertação de Mestrado apresentada por **Ricson José de Santana** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título “**Seleção de Casos de Teste Baseada em Similaridade de Texto e Cobertura de Requisitos**”, orientada pela **Profa. Flavia de Almeida Barros** e aprovada pela Banca Examinadora formada pelos professores:



Prof. Juliano Manabu Iyoda
Centro de Informática / UFPE

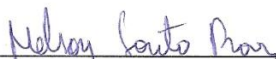


Prof. Eduardo Henrique da Silva Aranha
Departamento de Informática e Matemática Aplicada / UFRN



Profa. Flavia de Almeida Barros
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 24 de fevereiro de 2011.



Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Agradecimentos

Em primeiro lugar agradeço a Deus por conduzir-me em mais uma etapa da minha vida, me proporcionando estímulos para o término desse estudo.

A meus pais, pelo apoio e incentivo recebido durante todo esse período. Sem eles não teria conseguido.

A minha esposa Domícia, por estar sempre ao meu lado, me apoiando e ajudando a concluir este trabalho.

Ao meu irmão pelo companheirismo.

A orientadora e Dra. Flávia Barros pela sua orientação, dedicação, compreensão e paciência, bem como disponibilidade para repassar seus conhecimentos.

Ao professor Ricardo Prudêncio por todo o apoio dado na concepção dos experimentos e apoio na pesquisa sobre *CliqueFinder*.

A Laís Neves e Michelle Silva por todo o apoio na utilização da ferramenta *TaRGeT*.

Aos meus amigos Eduardo Uchoa e Leonardo Lima pelo conhecimento passado durante os momentos de dificuldade da linguagem utilizada.

As minhas amigas Ana Cláudia Monte e Alexandra Dias pela contribuição nos experimentos.

Ao Projeto de Pesquisa Ines ao qual me ofereceu um ambiente para experimentações.

A todos vocês, muito obrigado!

"A melhor maneira de nos prepararmos para o futuro é concentrar toda nossa imaginação e entusiasmo na execução perfeita do trabalho de hoje".

Dale Carnegie

Resumo

Com o objetivo de desenvolver software de qualidade, as empresas cada vez mais investem nas atividades de *Testes de Software*. No entanto, essa é uma atividade de custos elevados e que consome muito tempo. Assim, empresas de software e institutos de pesquisa têm investido na criação de ferramentas de geração automática de Casos de Testes a partir de especificações do software. Essas ferramentas têm por objetivo gerar suítes de teste que exercitem o código completo, a partir de combinações de passos de execução e resultados esperados. Contudo, essas ferramentas geralmente geram uma quantidade excessiva de *Casos de Teste (CTs)*. Quando não há tempo ou recursos disponíveis para executar a suíte completa, os responsáveis pelos testes são obrigados a selecionar um subconjunto da suíte original, que possa ser executado no tempo disponível, porém garantindo a cobertura dos requisitos associados à suíte original.

Este trabalho propõe uma estratégia de *Seleção Automática de Casos de Testes* que utiliza a similaridade de texto dos CTs como critério de seleção, preservando a cobertura de requisitos da suíte original. Foi implementado um protótipo que implementa essa estratégia, o *Sim_TC*. Esse sistema elimina os CTs que atingirem um limiar de similaridade textual determinado pelo analista responsável por testar o software, garantido sempre que os CTs eliminados tenham seus requisitos associados cobertos pelos Casos de Testes restantes. Experimentos realizados aplicaram o *Sim_TC* a duas suítes diferentes de casos de teste, a fim de comparar o desempenho da seleção manual *versus* automática. Nesses experimentos, o desempenho do protótipo foi bastante satisfatório, garantindo a cobertura de requisitos da suíte original. Esse protótipo deverá ser integrado à ferramenta TaRGeT, para geração automática de suítes de casos de teste a partir especificações de casos de uso.

Palavras-chave: Seleção automática de Teste de Software, Similaridade de texto.

Abstract

With the aim of developing software with high quality, companies increasingly invest in Software Testing. However, this is a costly and time-consuming activity. Thus, software companies and research institutes have invested in creating tools for automatic generation of test cases from software specifications. These tools are intended to generate test suites that exercise the entire code, from combinations of execution steps and expected results. However, these tools often generate an excessive amount of Test Cases (TCs). When no time or resources to run the full suite, the responsible for testing are required to select a subset of original suite that can be executed in the time available, but ensuring coverage of the requirements associated with the original suite.

This paper proposes a strategy for automatic selection of Cases Tests using the text similarity of TCs as a criterion for selection, preserving the coverage of the requirements of the original suite. This strategy was implemented in a prototype system, the *Sim_TC*. This system eliminates the TCs that reach a threshold of textual similarity in relation to another TC, however preserving the requirements coverage. That is, a TC is only eliminated if the remaining TCs cover the requirements associated to this TC. The similarity threshold is determined by the analyst responsible for testing the software. Experiments applied the *Sim_TC* two different test cases suites, in order to compare the performance of automatic versus manual selection. In those experiments, prototype's performance was very satisfactory, ensuring requirements coverage of the suite's original. This prototype will be integrated into the TaRGeT tool for automatic generation of test cases suites from use cases specifications.

Key word: Automatic Select Software Testing, Textual Similarity.

Sumário

1	Introdução	13
1.1	Organização do Documento.....	15
2	Teste de Software	16
2.1	Conceitos Básicos	16
2.2	Processo de Testes	18
2.2.1	Planejamento e Controle	19
2.2.2	Análise e Projeto	21
2.2.3	Implementação e Execução	21
2.2.4	Avaliação do Critério de Saída de Testes e Relatório	22
2.3	Técnicas de Testes	22
2.3.1	Testes Caixa Preta (Teste Funcional)	22
2.3.2	Teste Não Funcional	23
2.3.3	Testes Caixa Branca (Teste Estrutural)	23
2.4	Fases de Testes	24
2.4.1	Teste de Unidade	24
2.4.2	Teste de Integração	25
2.4.3	Teste de Sistema	26
2.4.4	Teste de Aceitação.....	26
2.5	Geração Automática de Casos de Testes	26
2.6	TaRGeT	27
2.7	Considerações Finais	30
3	Seleção de Casos de Testes	32
3.1	Seleção baseada em Similaridade de Caminhos.....	33
3.2	Seleção baseada em priorização de Casos de Testes.....	35
3.4	Seleção baseada em Cobertura de Requisitos	37
3.5	Seleção de Casos de Testes baseados no Esforço de Execução	39
3.6	Considerações Finais	41
4	<i>Sim_TC</i> – Seleção de Casos de Testes baseado em Similaridade de Texto e Cobertura de Requisito	42
4.1	Visão Geral do Processo de Seleção de Casos de Teste	42
4.1.1	Detalhamento do <i>Sim_TC</i>	45
4.2	Medida de Similaridade Textual.....	46
4.3	Ordenação dos Pares dos Casos de Testes	50
4.4	Seleção dos Casos de Testes.....	51
4.4	Considerações Finais.....	56
5	Experimentos e Resultados Obtidos.....	58

5.1	Teste de validação do Sim_TC	58
5.1.1	Seleção de Casos de Testes pelo <i>Sim_TC</i>	59
5.2	Estudos de Caso	64
5.2.1	Estudo de Caso 1 - Seleção Manual vs Seleção Automática	65
5.2.4	Estudo de Caso 2 - <i>Sim_TC</i> vs LTSBT	69
5.3	Considerações Finais	70
6	Conclusão	72
6.1	Principais Contribuições	73
6.2	Trabalhos Futuros	73
7	Referências	76

Lista de Figuras

Figura 1 – Processo de Teste Fundamental.....	16
Figura 2 – Fase de Testes [9].....	22
Figura 3 – Teste de unidade: caso de teste, driver e stub. [9].....	23
Figura 4 – Caso de Uso da TaRGeT.....	27
Figura 5 – Caso de testes da TaRGeT.....	28
Figura 6 – Modelo LTS [18].....	31
Figura 7 – Caminhos referentes ao Modelo da Figura 5 [18].....	32
Figura 8 – Matriz de similaridade do modelo da Figura 4 [18].....	32
Figura 9 – Exemplo de Priorização de Casos de Testes [26].....	34
Figura 10 – Cálculo do esforço de uma suíte de casos de testes [27].....	38
Figura 11 - Visão geral do Processo de geração e seleção de CTs.....	41
Figura 12 – Seleção de CTs da suíte original usando Sim_TC.....	43
Figura 13 - Aplicação da Distância de Levenshtein.....	45
Figura 14 - Casos de Testes.....	46
Figura 15 – Similaridade entre os passos dos casos de testes.....	46
Figura 16 - Fórmula de Cálculo de Similaridade [46].....	47
Figura 17 – Exemplo de cálculo de similaridade entre CTs.....	47
Figura 18 - Pseudocódigo – O Sim_TC seleciona aleatoriamente um CT quando há 100% de similaridade de texto entre os CTs.....	50
Figura 19 – Pseudocódigo – O Sim_TC garante que os requisitos do CT que foi cortado estão garantidos pelo CT que foi selecionado, sem perdas.....	51
Figura 20 – Pseudocódigo do Sim_TC, o Caso de Teste A possui o requisito R1 e R2, e o caso de teste B possui os requisitos R3 e R4.....	51
Figura 21 – Grafo dos Pares de Casos de Testes.....	52
Figura 22 – Pseudocódigo do Sim_TC – Seleção entre 3CTs.....	53
Figura 23 – CliqueFinder na Seleção de Casos de Testes.....	54
Figura 24 – Resultado do Sim_TC.....	54
Figura 25 – Tela inicial do Sim_TC.....	58
Figura 26 – Log do Sim_TC: CTs vs Requisitos 1.....	59
Figura 27 – Log do Sim_TC: CTs vs Requisitos 2.....	59
Figura 28 – Log do Sim_TC: Cobertura incompleta dos requisitos.....	60
Figura 29 – Log do Sim_TC: Seleção do 4º par.....	60
Figura 30 – Log do Sim_TC: Seleção do 5º par.....	61
Figura 31 – Log do Sim_TC: Não existe cobertura de requisitos entre os CTs.....	61
Figura 32 – Resultado da Seleção de Casos de Testes.....	62
Figura 33 – Resultado da Seleção de Casos de Testes 2.....	64

Lista de Tabelas

Tabela 1 – Requisito coberto em um suíte de casos de testes.....	36
Tabela 2 – Relação de Requisito x Casos de Testes.....	36
Tabela 3 - Matriz de Casos de Testes.....	43
Tabela 4 - Matriz de Similaridade entre os Pares de Casos de Testes.....	48
Tabela 5 - Ordenação dos Pares de Casos de Testes.....	49
Tabela 6 – Requisitos associados aos CTs da suíte escolhida.....	56
Tabela 7– Template da Suíte de Casos de Testes do Estudo de Caso 1.....	58
Tabela 8 – Lista dos Pares de Casos de Testes com respectivos percentuais de Similaridade de Texto.....	59
Tabela 9 – Cobertura de requisitos entre 3 CTs.....	60
Tabela 10 – Similaridade entre os Pares de Casos de Testes.....	63
Tabela 11 – Percentual de redução da suíte de Casos de Testes.....	65
Tabela 12 – Seleção Automática vs Seleção Manual.....	65
Tabela 13 – Suítes geradas pelo Sim_TC e quantidade de requisitos cobertos por cada uma.....	67
Tabela 14 – Resultado de tempo entre Sim_TC e LTSBT.....	68

1 Introdução

Atualmente é possível verificar o grande investimento das empresas em produzir software de baixo custo [1] e com qualidade, a fim de atender o que foi especificado nos requisitos iniciais, e sobreviver em um mercado competitivo. Os estudos vão desde a elicitação de requisitos até a fase de testes, área de Engenharia de Software cujo principal objetivo é entregar o sistema no menor tempo possível, com menor custo e com alta qualidade [2].

Em Engenharia de Software, é possível verificar que a fase de testes está contida em todo o ciclo de vida do software [3, 4, 5]. Esta fase é responsável por exercitar o software com o objetivo de encontrar erros e verificar se o que foi implementado está de acordo com o especificado no documento de requisito. Contudo, os custos da atividade de testes são muito elevados.

As atividades de teste podem ser divididas de forma simplificada em fases (processos) [6]: Planejamento e Controle, Análise e Projeto, implementação e execução, critério de saída e Relatório da atividade de testes. Tais processos podem ser executados de forma manual, semi-automática ou automática, dependendo da aplicação e dos recursos disponíveis¹. O foco desta pesquisa de mestrado foi em auxiliar na fase de Análise e Projeto de testes.

Existem hoje no mercado ferramentas capazes de gerar automaticamente suítes de testes a partir de alguma especificação (de requisitos ou casos de uso do software) [7, 21, 22, 23]. Contudo, a geração automática pode criar dezenas ou centenas de casos de testes, tornando inviável a execução manual de todos os casos de testes gerados, devido ao alto custo e tempo necessário para a realização dessa tarefa.

No intuito de solucionar esse problema e manter a qualidade do software, torna-se necessário realizar a seleção dos testes mais importantes das suítes geradas. Essa seleção pode ser baseada em critérios diversos, podendo ser realizada de forma manual (com base na experiência do arquiteto de software ou do testador) ou de forma automática [18].

¹ Salientamos que testes de stress e carga de um site são inviáveis de se realizar manualmente. Entretanto, para um teste de interface com o usuário, o teste manual pode ser executado sem grande esforço.

A seleção automática, em geral, tenta verificar a similaridade entre os testes da suíte e eliminar redundâncias. O critério de similaridade pode variar muito, dependendo da estratégia de seleção adotada [8, 18, 24, 26, 29].

Este trabalho de mestrado teve como foco a seleção de Casos de Teste (CTs) a partir de suítes de teste geradas automaticamente. Foi desenvolvida uma estratégia que realiza a seleção automática baseada na similaridade de texto entre os casos de teste, buscando manter a cobertura de requisitos da suíte original. A estratégia elimina os CTs que atingirem um limiar de similaridade textual determinado pelo analista responsável por testar o software, garantido sempre que os CTs eliminados tenham seus requisitos associados cobertos pelos Casos de Testes restantes, com garantia de 100% de cobertura de requisitos. Quando não houver nenhum CT na suíte que cubra os requisitos do teste a ser eliminado, este será mantido na suíte de testes, a fim de garantir cobertura total dos requisitos do software.

Foi desenvolvido um protótipo que implementa essa estratégia, o *Sim_TC*. A medida de similaridade utilizada foi baseada na Distância de Levenshtein [33, 50, 51, 52], que mede a diferença entre duas cadeias de caracteres quaisquer. O algoritmo desenvolvido leva em conta não apenas a diferença entre as palavras no mesmo passo de teste, mas também alterações na ordem de apresentação dos passos do caso de teste.

Os experimentos com o protótipo implementado foram realizados tendo como entrada suítes de testes geradas pela ferramenta TaRGeT [7]. Os resultados foram bastante promissores. A técnica proposta foi comparada com a seleção manual de CTs por dois especialistas, e também com o trabalho de CARTAXO [19]. Em ambos os casos, nossos resultados superaram os resultados obtidos com as outras duas técnicas mencionadas aqui, o que nos dá uma boa indicação de que essa técnica é adequada ao problema de seleção de teste de software.

Ressaltamos, por fim, que não foi encontrado na literatura disponível nenhum trabalho que utilize medidas de similaridade textual para seleção de casos de testes.

A partir deste trabalho, o analista de testes poderá optar por realizar a seleção de casos de testes pelo critério de similaridade de texto e cobertura de requisitos ou realizar a seleção usando outras ferramentas que trabalham com outros critérios, ficando a escolha pelo analista, de acordo com a necessidade do projeto.

Este trabalho foi parcialmente desenvolvido no âmbito do projeto de pesquisa do CIn-UFPE, com o uso de dados do Projeto CIn-Motorola Brasil.

1.1 Organização do Documento

Este documento está dividido em cinco capítulos, além desta Introdução.

- O segundo capítulo destina-se a apresentar os principais conceitos sobre *Testes de Software*, bem como fases e tipos de testes de software. Trata também de abordagem para geração automação de testes.
- O terceiro capítulo aborda a *Seleção de Caso de Testes*. São apresentadas algumas técnicas utilizadas para essa tarefa, e é feita uma análise sobre essa área.
- No quarto capítulo está descrita a técnica aqui proposta para seleção de casos de teste baseado em similaridade de texto e cobertura de requisito, chamada de *Sim_TC*. Nesse capítulo, são apresentados detalhes sobre o algoritmo do *Sim_TC*, que originou um protótipo usado para medir similaridade em suítes de teste geradas a partir de casos de uso, usando cobertura de requisito para nortear um critério de seleção dos casos de testes.
- No quinto capítulo estão os *Experimentos e Resultados Obtidos*. Esse capítulo descreve os experimentos, mudanças ocorridas durante o período de teste para melhorar o desempenho dos resultados e critérios estudados para chegar a um resultado satisfatório.
- Finalmente, o sexto e último capítulo traz a *Conclusão* do estudo realizado, os resultados da pesquisa, além das contribuições ao meio acadêmico e ao projeto de pesquisa. Temos também as limitações da ferramenta, pontos de melhorias e as perspectivas para trabalhos futuros. Em seguida, são apresentadas as referências bibliográficas, finalizando o documento.

2 Teste de Software

Este capítulo tem por objetivo apresentar alguns conceitos básicos sobre teste de software e suas características no ciclo de desenvolvimento de sistemas de software. A seção 2.1 descreve os conceitos básicos e os princípios de testes de software necessários para o bom entendimento deste texto. A seção 2.2 explica o processo de testes. A seção 2.3 descreve as técnicas de testes, como Teste Funcional (Caixa Preta), Teste Não-Funcional e Teste Estrutural (Caixa Branca). A seção 2.4 aborda as Fases de Testes de Software, explica cada fase do modelo V-Model (Teste de unidade, Teste de Integração, Teste de Sistema e Teste de Aceitação). A seção 2.5 trata da geração automática de Casos de Testes. A seção 2.6 apresenta a TaRGeT. Por fim, a seção 2.7 traz as considerações finais sobre o capítulo.

2.1 Conceitos Básicos

Testar um software é verificar o quanto ele atende aos requisitos funcionais e não funcionais especificados [10]. De acordo com [11] Teste de Software é o processo ou o conjunto de processos responsável por assegurar que o código faz o que foi designado a fazer, e que não faz qualquer coisa aleatória. O software deveria ser previsível e consistente, não oferecendo surpresas para o usuário.

Contudo, testar exaustivamente todo o software, todas as combinações de entradas e condições, é impossível [9], pois a atividade de testes tem um custo alto, principalmente para pequenos projetos, em que o próprio desenvolvedor realiza os testes. Por esse motivo, são estudados processos que garantam a qualidade do software a tipos variados de projetos.

Para uma boa realização dos testes em um software, é necessário um bom planejamento da execução, um plano de testes, e um projeto de testes, como aplicação de técnicas caixa branca, caixa preta, caixa cinza e testes de regressão, usados em fases de Testes diferentes: testes de unidade, de integração, de sistema, de aceitação, de operação, alfa e beta.

O teste de software segue alguns princípios, listados a seguir [9].

- *Testes mostram a presença de defeitos:* testes podem demonstrar a presença de defeitos, mas não a ausência deles.
- *Testes exaustivos são impossíveis:* testar todas as combinações de entrada não é viável.
- *Teste antecipado:* a atividade de testes deve iniciar o mais breve possível no ciclo de vida do *software*.
- *Agrupamento de defeitos:* Defeitos não são igualmente distribuídos, mas são, em sua maioria, agrupados. Se os defeitos são encontrados uns próximos de outros, durante os testes é preciso reagir com flexibilidade.
- *Paradoxo do Pesticida:* se um mesmo teste for repetido várias vezes, e não encontrar mais erros, deve ser revisado e atualizado.
- *Teste depende do contexto:* teste deve ser realizado de forma diferente de acordo com o contexto.
- *A ilusão da ausência de erros:* encontrar e consertar erros não garante que o sistema atende as necessidades do usuário. O uso de protótipos durante o desenvolvimento pode prevenir este problema.

É preciso estabelecer as precondições para a execução do teste, informar as entradas de dados para o sistema, verificar os resultados e compará-los com os resultados esperados que foram especificados, para então determinar se o teste passa ou falha. [15, 30].

Para executar os testes em um sistema, alguns conceitos básicos são necessários para o entendimento de todo o processo de testes:

- *Erro (Error):* Uma ação humana que procede um resultado incorreto. Por exemplo, uma ação incorreta realizado por um desenvolvedor [45].
- *Falta (Bug):* é uma falha em um componente ou definição de dados incorretos em um sistema.

- *Falha (Failure)*: É o desvio, ou mau funcionamento do componente ou sistema em realizar o que foi especificado. Pode ser também a manifestação da falta durante a execução do sistema.

2.2 Processo de Testes

O processo de testes é composto por tarefas, atividades, artefatos, papéis e responsáveis que seguem um padrão para uma melhor aplicação dos projetos de testes. De acordo com [9], o detalhamento das sub-tarefas é necessário para um melhor entendimento, como podemos ver na Figura 1.

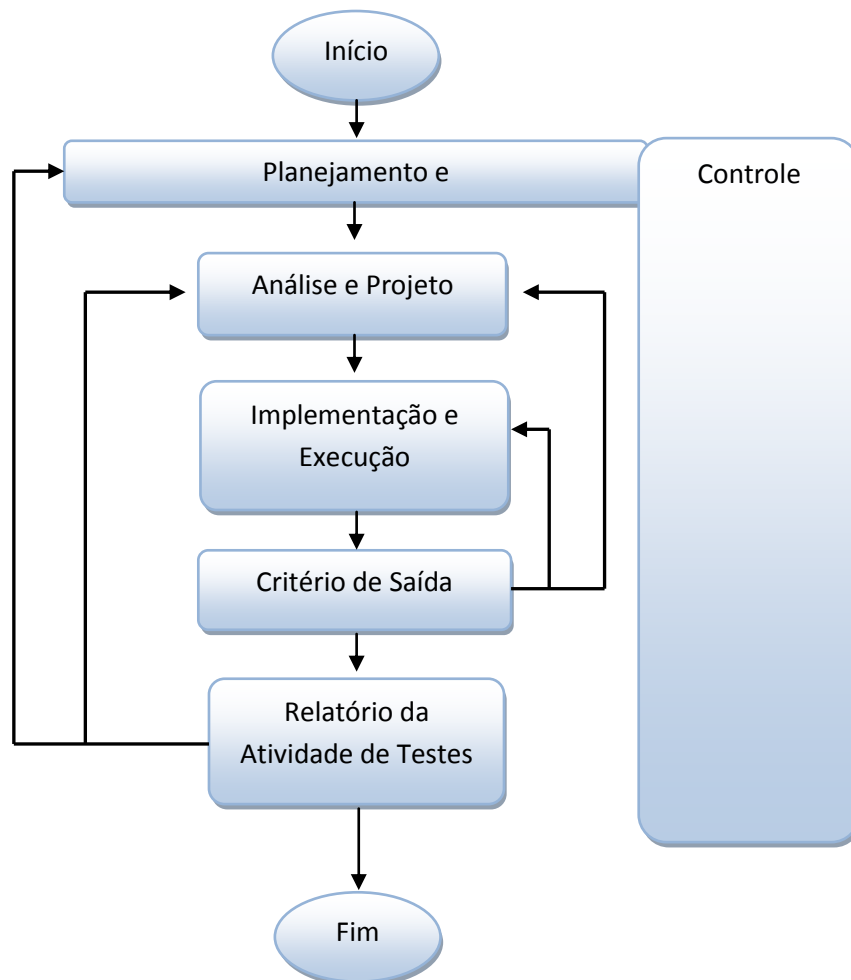


Figura 1 – Processo de Teste Fundamental

2.2.1 Planejamento e Controle

Antes de iniciar a execução dos testes, é preciso realizar seu planejamento. Durante o planejamento, precisamos entender os objetivos e alvos dos clientes, *stakeholders*, e os riscos do projeto [9]. Testar sistemas grandes e complexos requer criação e execução de milhares de casos de testes, testar vários módulos, corrigir erros e empregar pessoas por muito tempo. O planejamento dos testes serve como base para o controle e gerenciamento do processo de testes, criando melhores perspectivas de qualidade para o produto a ser testado.

Conhecidos os objetivos, deve ser elaborado um plano para os testes, onde estão inclusas as políticas e estratégias para os testes, assim como as regras.

Plano de Testes

No plano de testes é descrito o escopo, as abordagens, os recursos e o cronograma das atividades de testes [9]. Ele deve ser elaborado no início do projeto, e deve conter os itens de testes, funcionalidades que serão testadas e não serão testadas, tarefas de testes, quem irá realizar cada tarefa, o ambiente de testes, as técnicas de testes que serão usadas e critérios de entrada e saída que deverão ser usados e eventuais riscos que possam ocorrer.

O Plano deve descrever as estratégias de testes para cada grupo de funcionalidade. Sobretudo, deve abordar as atividades e ferramentas que serão usadas no teste. Dessa forma, deve conter os critérios de sucesso e de falha de cada caso de teste e critério de suspensão e requisitos de reinício.

O Plano de Testes implementa uma política e estratégia de testes, devendo ser verificado durante o planejamento se o plano adere às políticas e estratégias acordadas com *stakeholders*. Abaixo são informados os itens que compõem o plano [17]:

- *Identificação*: identifica o plano de teste com a versão, data do início e término das atividades.
- *Pré-requisitos*: passos necessários para início da execução dos testes.
- *Ambiente*: hardware e software necessários para a execução dos testes.

- *Priorização*: definição da ordem de prioridade de execução dos casos de testes.
- *Testes*: contém os dados necessários para a verificação da qualidade do software.

De acordo com [11], o Plano de testes deve conter os seguintes componentes:

1. *Objetivo*: A definição dos objetivos de cada fase de testes.
2. *Critério de Saída*: definição de quando cada fase de testes está concluída.
3. *Horários*: um calendário com todo o marco de início e fim de cada fase de testes.
4. *Responsáveis*: para cada fase deverá existir um responsável pelo projeto, escrita, execução e verificação dos casos de testes.
5. *Padrões e Biblioteca de Casos de Testes*: são necessários padrões para escrita e armazenamento dos casos de testes.
6. *Ferramentas*: são necessários desenvolver ou adquirir ferramentas de testes, assim como usar e quando serão necessárias.
7. *Tempo de Computador*: deve estar definida no plano os servidores, estações de trabalhos ou aplicações – recursos computacionais - necessários para a execução dos testes.
8. *Integração*: no plano de testes deve estar definida a forma de integração do sistema com partes menores ou com outros sistemas.
9. *Testes de Regressão*: é executado após os testes funcionais ou correções no sistema. O objetivo é verificar se a modificação ou correções de erros no sistema já testado não gerou problemas ou novos erros. Os testes de regressão são a re-execução de subgrupos de testes.

2.2.2 Análise e Projeto

Nesta fase os objetivos de testes são transformados em condições de testes. São criados os procedimentos dos testes (scripts). São definidos as entradas de dados, os resultados esperados e os passos necessários para a realização dos testes [17].

Para a atividade de Análise e Projeto são definidos os documentos:

- *Projeto de Testes*: Descreve com detalhes quais funcionalidades serão testadas e os critérios usados para a execução dos casos de testes. Define quais casos e os procedimentos de testes serão realizados para um dado módulo ou sistema.
- *Casos de teste*: Documento contendo dados de entrada, condições de execução, condições de testes. Determina o modo de exercitar o programa e verificar se está de acordo com o especificado [11]. Elaborado com o objetivo de identificar defeitos na estrutura do software e garantir que os requisitos foram completamente atendidos.

2.2.3 Implementação e Execução

Na implementação e execução dos testes são abordadas as condições de realização dos casos de testes, e a criação do ambiente de testes. Os casos de teste são executados de acordo com o que foi descrito de forma manual ou automática, a fim de validar se o resultado retornado pelo sistema está de acordo com o resultado especificado.

Antes do início dos testes, o ambiente deve ser montado, e os dados de testes devem ser criados para o componente ou sistema em teste. Em seguida, os testes são desenvolvidos e priorizados. São criadas as instruções para a realização dos testes (pré e pós-condições de testes). A seguir, os CTs são agrupados de acordo com as suas pré-condições ou pós-condições, criando as suítes de testes (*test suite*). Por último, é verificado se o ambiente está correto, possibilitando a execução dos testes.

Na execução, são realizadas as suítes de testes e os casos de testes individualmente, seguindo os procedimentos contidos em cada CT. Em seguida, são registrados os resultados, a identificação e versão do sistema em testes, e o resultado atual é comparado com o resultado esperado.

2.2.4 Avaliação do Critério de Saída de Testes e Relatório

A avaliação do critério de saída é uma verificação que avalia se a execução de testes atendeu os objetivos definidos. Deve ser feito em todos os níveis de teste e, baseado no risco do projeto, define até onde a realização dos testes é executada. Assim, um critério de saída pode ser associado ao critério de aceitação. Caso o sistema esteja de acordo com o especificado e aceito pelo cliente, este pode ter os testes encerrados.

Após a execução dos testes, é preciso informar aos *stakeholders* o resultado obtido através do Relatório de Testes, um documento resumido com as atividades e resultados dos testes. Ele contém uma avaliação dos itens testados e o critério de saída. Os envolvidos nos processos devem conhecer que testes foram feitos e o resultado dos testes, para tomar uma decisão sobre o sistema.

2.3 Técnicas de Testes

Existem várias técnicas de testes de software, que são aplicadas de acordo com as fases do desenvolvimento: os *testes estruturais*, chamados de caixa branca, que verificam a implementação do sistema, baseado no código fonte; os *testes funcionais*, caixa preta, que verificam a implementação do sistema baseado na saída de dados usando vários dados de entrada; os *não funcionais*, que focam nas características de qualidade do software; e os *híbridos*, caixa cinza, que unem os *testes estruturais* e *funcionais* para verificar a estrutura interna e a especificação do sistema.

2.3.1 Testes Caixa Preta (Teste Funcional)

Essa técnica de Teste é baseada na análise da especificação das funcionalidades de um componente ou sistema, sendo conhecido como *Teste Funcional* [9]. Esse tipo de Teste considera o comportamento externo do software, comportamento especificado, mas poderá incluir testes não funcionais. Os testes são executados a partir da entrada de dados fornecidos

pelo testador, e tem a saída de dados verificada de acordo com o que foi especificado no caso de teste.

A seleção de casos de testes para os testes funcionais é baseada nos requisitos ou especificação do software a ser testado. Dessa forma, os Testes Funcionais podem ser feitos objetivando a interoperabilidade, segurança, exatidão e conformidade. Eles podem ser feitos a partir de duas perspectivas: baseado em requisitos ou baseado no processo de negócio.

2.3.2 Teste Não Funcional

O teste não funcional foca nas características de qualidade do software produzido ou atributos não funcionais, como segurança, carga, desempenho, entre outros. Os requisitos não funcionais acrescentam qualidade ao sistema, por exemplo, um sistema web que atende a mais de centenas de milhares de solicitações ao mesmo tempo sem perder o tempo de resposta e não cair o servidor.

2.3.3 Testes Caixa Branca (Teste Estrutural)

Os testes Caixa Branca visam verificar a estrutura do código e o comportamento interno do software [11]. Esta estratégia deriva dados de testes a partir da verificação da lógica do programa. O objetivo deste teste é estabelecer uma entrada de dados e percorrer todos os caminhos viáveis existentes no código. Deve verificar partes do código que não foram testadas nos Testes de Caixa Preta (Funcional).

De acordo com [12], uma vantagem do teste de caixa branca é que ele é completo e focado no código produzido. Havendo conhecimento da estrutura interna ou lógica, existe maior probabilidade de se encontrar erros ou danos realizados por parte de um programador. Essa técnica avalia aspectos como: teste de condição, teste de fluxo de dados, teste de ciclos e caminhos lógicos.

2.4 Fases de Testes

De acordo com [9], através do ciclo de vida de software, os testes são compostos por quatro fases (Figura 2):

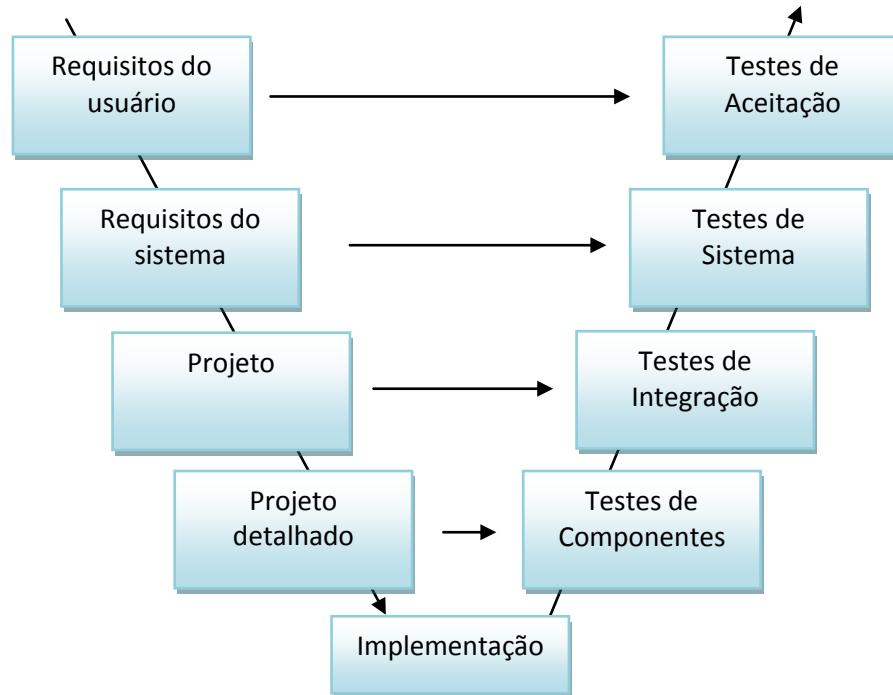


Figura 2 – Fase de Testes [9]

A Figura 2 nos mostra a ordem em que ocorrem as etapas do processo de ciclo de vida do software e relacionada a cada uma, as fases de testes respectivamente. Para os requisitos do usuário, são elaborados e executados os testes de aceitação; Os requisitos do sistema serão verificados e testados pelos testes de sistema; Na fase de Projeto, são realizados os testes de integração, com o objetivo de avaliar a integração entre os componentes do sistema; por fim, o projeto detalhado passa por avaliação dos testes de componentes.

2.4.1 Teste de Unidade

Teste de Unidade é o nível básico de testes [12]. Os testes são focados na menor unidade de código, como componentes, módulos ou funções. Essas unidades são testadas isoladamente a fim de se encontrar erros de código e especificação. Os testes são realizados através da técnica de testes caixa-branca.

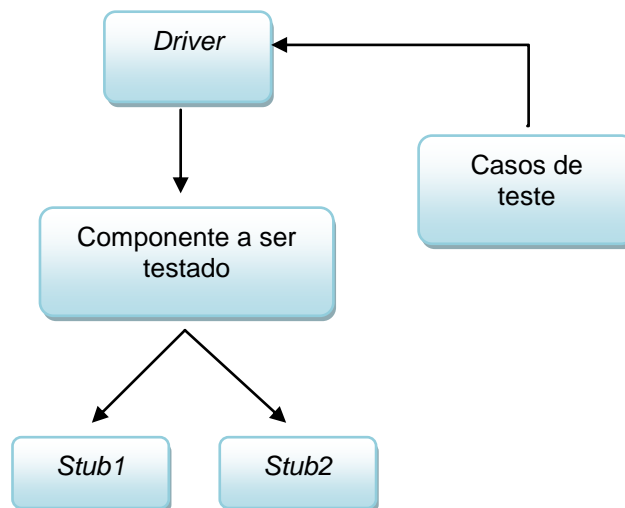


Figura 3 – Teste de unidade: caso de teste, driver e stub. [9]

Para a unidade ser testada, é necessário criar um ambiente de testes utilizando *drivers* e *stubs* (Figura 3). Os *drivers* são utilizados para enviar os dados de entrada ao componente a ser testado. Os *Stubs* são programas que fingem ser componentes utilizados pelo componente a ser testado. Como, em geral, estes componentes não estão prontos no momento do teste de unidade, é preciso implementar um “fake” (stub). Deste modo, é possível simular um ambiente para o módulo a ser testado, suprimindo suas dependências com outros módulos.

2.4.2 Teste de Integração

Finalizado os testes de unidade, os componentes devem ser integrados para compor uma função ou outro componente, até que o sistema esteja completo. Nesse momento de integração podem ocorrer outros erros. Os módulos podem não funcionar como esperado, então as unidades são testadas novamente.

O objetivo dos testes de integração é verificar que cada módulo executa corretamente dentro da estrutura, que os modelos de interface estão corretos, e mostrar as falhas nas interações entre os componentes.

2.4.3 Teste de Sistema

Após o teste de integração, todas as funcionalidades do sistema são testadas com base no documento de requisitos. O sistema é completamente testado no ambiente de operação antes dos testes de aceitação. O teste de sistema contém um grupo de testes que verificam a qualidade dos atributos, e assegura que o testes de aceitação ocorrerá relativamente sem problemas [13]. Assim, o teste de sistema verifica que as funções foram desenvolvidas corretamente. Nesta fase, é possível verificar que algumas características não funcionais estão presentes no sistema.

2.4.4 Teste de Aceitação

Depois da finalização dos testes de sistemas, o sistema segue para o teste de aceitação. O teste de aceitação certifica que o sistema satisfaz o requisito original e atende o que foi especificado pelo cliente. Esses testes são executados pelos usuários utilizando a técnica de caixa preta, e podem ser realizados de duas formas: *teste alfa*, realizado pelo cliente no ambiente de desenvolvimento, registrando erros e problemas no uso; e *teste beta*, realizado pelo cliente no ambiente do cliente sem a presença do desenvolvedor [13].

2.5 Geração Automática de Casos de Testes

Como visto anteriormente, o processo de teste é muito custoso para uma fábrica de software. Processo este que demanda tempo e critérios para definir de que forma será aplicada na fábrica, além de analisar alocação de recurso e tipo de testes a ser aplicado no software em desenvolvimento. Somando a isto, temos o fato da geração de casos de testes manual ser onerosa e sujeita a erros.

Para tentar diminuir o tempo e minimizar a probabilidade de surgimento de erros, busca-se utilizar um processo de geração automática de casos de testes. Essa geração consiste em criar casos de testes de forma automatizada a partir de alguma especificação do sistema, como casos de uso [7], modelos UML, diagramas de sequência [14], modelos guiados por protótipos [16] e por modelos formais.

Os casos de testes gerados automaticamente por essas ferramentas tentam atender todos os requisitos especificados no documento usado (casos de uso, modelos, dentre outros). Essas ferramentas geram um número muito grande de casos de testes, a fim de atender todas as combinações possíveis. Com isso, faz-se necessário realizar uma seleção de casos de testes para reduzir a quantidade de casos de testes na suíte que será executada pela equipe de Testes. O processo de seleção, como os métodos para redução de casos de testes, será descrito no Capítulo 3.

2.6 TaRGeT

A TaRGeT (*Test and Requirements Generation Tool*) é uma ferramenta capaz de gerar automaticamente casos de testes a partir de documentos de caso de uso em linguagem natural [7]. Através dessa ferramenta, é possível especificar, editar e visualizar documentos de caso de uso [20], gerando uma suíte de casos de teste a serem manualmente executados. A ferramenta permite também realizar seleção de casos de testes a partir de similaridade de caminho [19].

A ferramenta foi originalmente desenvolvida dentro do Projeto de Pesquisa do CIn/UFPE Motorola, como linha de produto de software. Como entrada, temos um documento do Word (atendendo ao *template* da TaRGeT) com a especificação do caso de uso que descreve o comportamento das *features* a serem testadas. A partir deste documento, é possível:

- Visualizar e navegar pelo caso de uso
- Editar o caso de uso
- Gerar suítes de casos de testes
- Validar documentos de caso de uso (ID duplicados, Referências inexistentes, documentos inválidos)
- Realizar buscas no documento de caso de uso (mecanismo de busca).

A TaRGeT também disponibiliza as opção de utilizar filtros para selecionar casos de testes da suíte gerada, sendo possível realizar a seleção de CTs usando um critério de similaridade de caminho, ou exportar a suíte de CTs em formato .xls ou XML.

Por fim, a ferramenta dispõe do caso de uso da TaRGeT, que descreve todas as funcionalidades presentes na TaRGet, desde a inicialização da ferramenta, passando pelo processo de criação do projeto, importação de dados, geração da suíte de casos de testes e visualização. A partir desse documento é possível entender todas as funcionalidades da TaRGeT, verificar o procedimento necessário para usar cada uma, e tirar dúvidas sobre a sua utilização. Este documento será usado em um dos estudos de caso no Capítulo 5.

Para cada Caso de Uso (ou Use Case - UC) descrito no documento (ver Figura 4), temos:

- Identificação do UC
- Descrição do UC
- Fluxo Principal
- Identificação do Passos para execução
- Ação do usuário
- Estado do sistema
- Resposta do Sistema
- Fluxo Alternativo e/ou Exceção

UC_15 - Fechando um Projeto

Setup: Nada

Fluxo Principal:

Descrição: Fechar um projeto.

De passo: Iniciar

Para passo: Fim

Nº passo	Ação	Estado do Sistema	Resposta do Sistema
1	Clique na opção de "File" na barra de Menu.	TaRGeT é inicializada e um projeto é aberto.	A opção de "Close Project" is ativada.
2	Selecione a opção de "Close Project". [FR_TARGET_0035]		Uma mensagem é mostrada perguntando se o usuário deseja fechar o projeto.
3	Clique no botão de "Yes".		O projeto é fechado. O projeto em branco é exibido. [FR_TARGET_0225]

Fluxo Alternativo

Descrição: Cancelar o fechamento do Projeto.

De passo: 2

Para passo: Fim

Nº passo	Ação	Estado do Sistema	Resposta do Sistema
3	Clique no botão "No".		A janela é fechada e a ferramenta volta ao projeto.

Figura 4 – Caso de Uso da TaRGeT

A Figura 5, nos mostra um caso de teste gerado pela TaRGeT após o processo de geração automática de caso de testes. Este caso de teste é composto por:

- Id do caso de testes
- Caso de uso
- Requisitos
- Setup
- Notas
- Condições iniciais
- Passos
- Resultados esperados
- Condições Finais

Descrição	Passos	Resultados Esperados
Func_001	Nada	
Caso de uso:	6378#UC_01	
Requisito:	FR_1513,FR_1429	
Setup:	Nada	
Condições Iniciais	Modo Airplane está ligado	
Notas	Caso de Teste gerado por TaRGeT system	
Passos		
1	Ligue o celular	Uma mensagem é recebida durante a sequência de ligação
2	Selecione a opção de Airplane Mode	O celular será ligado em Airplane Mode
Condições Finais	Nada	

Figura 5 – Caso de testes da TaRGeT

2.7 Considerações Finais

O Teste de software é essencial em fábricas de software. E, para a eficácia desta atividade, é essencial que haja um processo de testes bem definido, com plano de testes e casos de testes criados com alocação de recursos, atividade de execução, atendendo o tempo disponível.

Neste capítulo, vimos a fundamentação teórica para entender o que é teste: conceitos e princípios básicos de testes; criar e executar todas as atividades do processo de testes, como o planejamento, execução, saída e relatório da execução; entender e definir em que momento usar a técnica de testes ideal: Teste Funcional, Não Funcional ou Estrutural; as fases que compreendem o processo: teste de unidades, integração, sistemas, aceitação e regressão e uso de ferramentas que auxiliam na execução dos testes.

As ferramentas de testes auxiliam na geração da documentação, minimizam o risco de erros ao criar ambientes para a atividade, assim como há um ganho de tempo. Das ferramentas e técnicas disponíveis na literatura, focamos nas ferramentas de geração automática de casos de testes. Elas auxiliam os engenheiros a gerarem exaustivamente casos de testes com todas as

combinações possíveis. A saída dessa ferramenta, as suítes de casos de testes serão utilizadas em nossa pesquisa na seleção dos casos de testes.

3 Seleção de Casos de Testes

No capítulo anterior, vimos que o processo de testes, apesar de indispensável ao desenvolvimento de um software de qualidade, é muito custoso. Um dos maiores gargalos é o custo de execução das suítes de teste, que são, em geral, muito longas. Mesmo as suítes geradas manualmente tendem a ser longas, a fim de exercitar o código completo e cobrir todos os requisitos especificados para o software. Este problema se agrava com a geração automática de suítes de teste, como visto no capítulo anterior.

Uma prática adotada pelas fábricas de software a fim de viabilizar a etapa de testes é a seleção dos casos de teste mais importantes (ou representativos) da suíte original. Isso ocorre na fase de planejamento dos testes, levando-se em conta o tempo disponível para a execução dos testes, a infra-estrutura e os recursos disponíveis. O objetivo dessa seleção é executar o maior número possível de casos de testes, de forma a atender todos os requisitos do software, exercitando o maior número de caminhos.

Essa seleção pode ser baseada em critérios diversos, como cobertura de código, de requisitos, similaridade entre os testes, etc. A seleção pode ser realizada de forma manual (com base na experiência do arquiteto de software ou do testador) ou de forma automática. A seleção manual é muito custosa e lenta, sendo ainda passível de erros. Assim, a seleção automática de CTs aparece como a alternativa na solução deste problema.

A seleção automática, em geral, verifica a similaridade entre os testes da suíte, a fim de eliminar redundâncias. O critério de similaridade pode variar muito, dependendo da estratégia de seleção adotada. Contudo, é importante ter em mente critérios de corte que preservem características importantes da suíte de testes original, como por exemplo, a cobertura de requisitos.

Visando minimizar o problema de redundância e manter a mesma cobertura de requisito, Jones e Harrold [29] sugerem duas técnicas: priorização dos casos de testes e redução dos casos de testes.

Este capítulo apresenta métodos e técnicas usadas na seleção de casos de teste.

A Seção 3.1, a seguir, apresenta a seleção de CTs baseada em similaridade de caminhos. A Seção 3.2 descreve a seleção baseada em priorização de casos de testes. A Seção 3.3 apresenta seleção baseada em cobertura de requisitos, e a Seção 3.4 mostra a seleção de casos de testes baseados no esforço de execução. Por fim, a Seção 3.5 traz as considerações finais deste capítulo.

3.1 Seleção baseada em Similaridade de Caminhos

Esta técnica considera a similaridade entre os caminhos percorridos pelos casos de testes como critério de corte. Aqui, cada CT é comparado com todos os outros da suíte original, e um CT é descartado quando atinge um percentual de similaridade nos caminhos percorridos em relação a outro CT da suíte [9, 18, 19, 31]. Esse limiar de similaridade é determinado pelo analista de testes.

Na Figura 6, abaixo, temos um exemplo de um modelo LTS [19]:

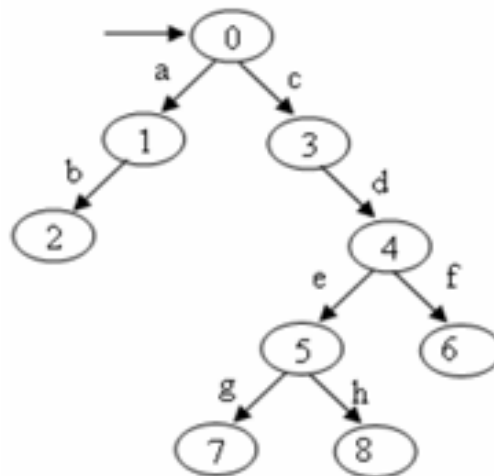


Figura 6 – Modelo LTS [19]

A partir deste modelo (Figura 6), usando a técnica de similaridade de caminhos, são obtidos 4 caminhos que o teste percorre, conforme podemos observar na Figura 6.

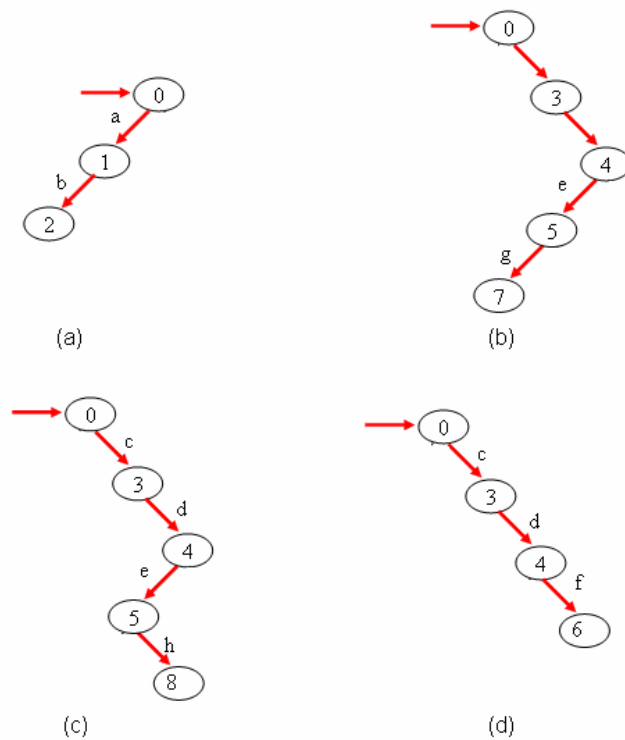


Figura 7 – Caminhos referentes ao Modelo da Figura 6 [19]

Considerando, por exemplo, um percentual de similaridade de caminho de 25%, este método de seleção de CTs cortará 3 caminhos para atender o percentual desejado, dado que temos 4 caminhos na suíte de testes original.

Para realizar a seleção, o LTS monta uma matriz de similaridade 4x4, pois temos 4 caminhos (ver Figura 8) e calcula o tamanho de cada um dos caminhos.

	a	b	c	d
a	0	0	0	0
b		0	3	2
c			3	2
d				2

Figura 8 – Matriz de similaridade do modelo da Figura 6 [18]

A abordagem proposta por Cartaxo et al. [19], LTS-BT, elimina o maior valor da matriz de similaridade e em seguida utiliza o critério do tamanho do caminho para realizar a seleção. Neste exemplo hipotético, temos o caminho b e c, que possuem 3 passos similares e ambos possuem o mesmo tamanho do caminho, $|b|=4$ e $|c|=4$, logo, um dos caminho é escolhido aleatoriamente.

No próximo passo da seleção, o LTS elimina o caminho “d”, pois possui um menor caminho, em comparação a “c” – $|c|=4$ e $|d|=3$. Por fim, é verificado o tamanho do caminho entre “a” e “c”, $|a|=2$ e $|c|=4$, eliminando o caminho “a” por ser menor.

Neste critério de seleção, similaridade de caminho, a cobertura de requisitos não é verificada antes de realizar a eliminação de um caminho, podendo o sistema realizar uma escolha infeliz, cortando um caminho que cubra mais requisitos.

Apesar dos problemas de possível falta de cobertura de requisitos, esse método ajuda o engenheiro de testes a obter um subconjunto de CTs que tenham caminhos em comum com os testes cortados, reduzindo a suíte original a fim de se encaixar no tempo e recursos disponíveis no projeto.

3.2 Seleção baseada em priorização de Casos de Testes

Na seleção baseada em priorização de casos de testes, são estabelecidos critérios para ordenar a execução dos casos de testes da suíte. Para pontuar os casos de testes, os testadores de software podem usar critérios diversos [24, 26], como acervos existentes, documentação do software, histórico de execução, julgamento de especialistas, entre outros critérios. Vale lembrar que priorização é a ordem de execução dos CTs da suíte; e Seleção é a escolha (anterior a execução). Nem sempre o critério de Seleção é o mesmo do de priorização. Portanto, nem sempre priorização serve para a seleção.

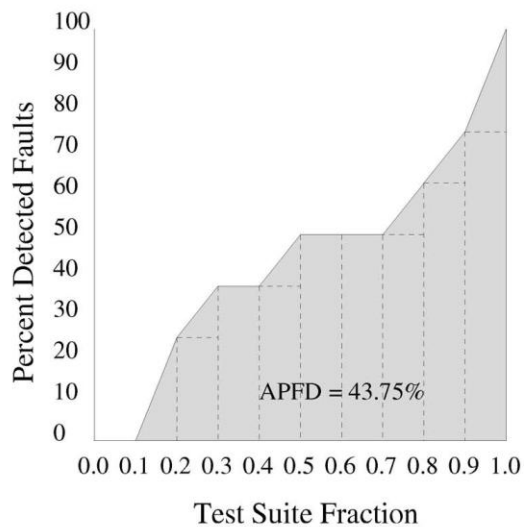
Os critérios de priorização podem ter pesos associados, que determinam a importância de cada critério na ordenação dos testes. O propósito é atingir o objetivo do teste o mais cedo possível. Quando há restrição de tempo, essa ordenação garante a execução dos casos de testes com maior prioridade, diminuindo o risco de grandes falhas escaparem [25].

A Figura 9 mostra um exemplo de priorização de casos de testes, conforme [26].

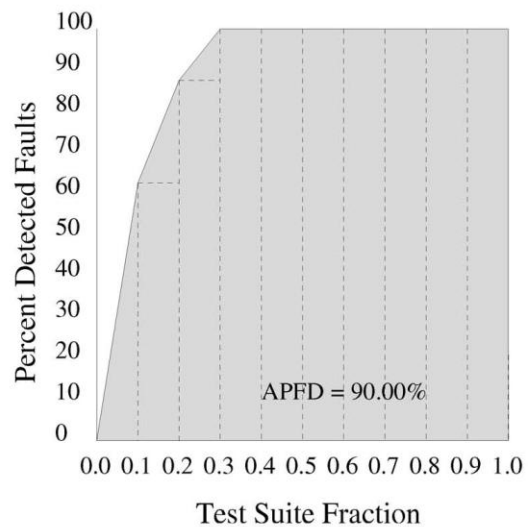
test	fault							
	1	2	3	4	5	6	7	8
A								
B	x	x						
C	x	x	x					
D		x	x					
E	x							x
F								x
G		x						
H				x				x
I	x	x	x	x	x			
J					x	x	x	

(a)

Test Case Order T1: A-B-C-D-E-F-G-H-I-J Test Case Order T2: I-J-E-B-C-D-F-G-H-A



(b)



(c)

Figura 9 – Exemplo de Priorização de Casos de Testes [26]

No exemplo acima, a suíte composta com 10 casos de testes é executada na ordem original, resultando no gráfico da Figura 9 (b), e em seguida, temos o resultado da execução da suíte com os casos de testes priorizados – Figura 9 (c). Podemos observar que, com a execução de apenas 2 casos de testes (I e J), é alcançada a cobertura de 90% das falhas do software através do uso da métrica do APFD, que mede a média ponderada do percentual de

faltas detectadas sobre vida da suíte de testes. Quanto maior o valor, melhor o resultado de detecção de faltas.

Segundo [28], a ordem em quem os CTs são executados pode ter um impacto significativo no tempo total de execução de testes. A priorização de Albertins et al. [28] dos CTs pode ser realizada manual ou automaticamente, e em ambas levam em conta a reutilização de dados: quanto mais os dados são reutilizados entre os testes, mais rápido a sequência é executada. A seleção automática oferece a vantagem de reduzir o tempo de criação das suítes com testes priorizados. Além disso, para suítes de tamanho pequeno, a priorização automática é capaz de encontrar a melhor sequência, depois tentar todas as combinações possíveis de casos de testes. Contudo, independentemente da priorização ser realizada manual ou automaticamente, há, se for bem feita, um ganho no tempo de execução da suíte.

O método de priorização foca nos casos de testes que podem encontrar falhas de alto risco mais cedo no software, de acordo com os critérios selecionados pelo analista. Através da priorização dos CTs, é possível realizar a seleção de CTs, pois em projetos em que o tempo é curto, os CTs são executados à medida do tempo e recursos (testadores) disponíveis. Porém, o conjunto de CTs priorizado pode conter casos de testes semelhantes, com alto percentual de similaridade, resultando em perda de tempo e aumento dos custos do projeto.

De acordo com Jones e Harrold [29], pesquisadores têm investigado algoritmos de redução de suíte de testes que identifica a suíte de CTs que prover a mesma cobertura da suíte original, usando como critério, algoritmo de priorização de CTs.

Por fim, é importante observar que se o engenheiro de testes ou a ferramenta usada nesse processo não levar em conta os requisitos do software, os casos de testes que cobrem estes requisitos podem não ser executados.

3.4 Seleção baseada em Cobertura de Requisitos

O objetivo da cobertura de requisitos em testes de software é que exista pelo menos um caso de testes relacionado ao requisito. Cada caso de testes cobre “parcialmente” um requisito, sendo necessários vários casos de testes para cobrir o requisito.

O objetivo da atividade de testes é exercitar o maior número viável de caminhos do software, e garantir que ele está de acordo com o que foi especificado nos requisitos iniciais. Contudo, alguns casos de testes podem cobrir um ou mais requisitos, gerando assim, redundância nos testes.

Como podemos observar na Tabela 1, um caso de teste pode cobrir um ou mais requisito, neste caso, iniciados por “FR_*” (*Feature*). Da mesma forma que um requisito pode estar presente em vários casos de testes, podendo ocorrer redundâncias.

Requisitos de Software						
	FR_1513	FR_1429	FR_1514	FR_1428	FR_1427	FR_1425
CT 1	x	x				
CT 2		x	x			
CT 3	x			X		
CT 4			x	X		
CT 5					x	
CT 6						x

Tabela 1 – Requisito coberto em um suíte de casos de testes

A Tabela 2 traz um exemplo de uma suíte com 6 casos de testes, onde podemos observar a relação de Requisito x Casos de Testes. Essa tabela possibilita detectar as redundâncias entre requisitos e casos de testes. Aqui, podemos ver que é possível realizar a seleção de casos de testes, criando uma nova suíte que mantenha o mesmo nível de cobertura de requisitos. Dessa forma, é possível eliminar o caso de teste 2, pois os casos de testes 1 e 4 cobrem os requisitos deste.

Requisito	CT- Casos de Testes
FR_1513	CT 1, CT 3
FR_1514	CT 2, CT 4
FR_1429	CT 1, CT 2
FR_1428	CT 3, CT 4
FR_1427	CT 5
FR_1425	CT 6

Tabela 2 – Relação de Requisito x Casos de Testes

A seleção baseada em cobertura de requisitos garante que todos os requisitos especificados serão cobertos pelos casos de testes selecionados, e que no mínimo um caminho será exercitado. Porém, para requisitos complexos, um caso de teste não garante 100% da cobertura deste requisito, sendo necessários vários casos de testes para atender todo o requisito.

3.5 Seleção de Casos de Testes baseados no Esforço de Execução

O objetivo deste método de seleção é montar a suíte candidata de CTs a partir do tempo de execução estimado de cada CT da suíte original.

A estimativa de tempo de execução de uma dada suíte de CTs é calculada a partir da análise de cada caso de teste, e da atribuição e totalização dos pontos de execução dos CTs, estimando assim da quantidade de horas necessárias para a execução de toda a suíte de testes.

Para o cálculo do esforço (tempo) de execução, é necessário, primeiramente, analisar cada passo do caso de teste, e identificar as características que representam os requisitos funcionais e não funcionais. De acordo com cada característica encontrada, atribui-se um nível de esforço (baixo, médio e alto) e, a seguir, cada passo recebe os pontos de execução. Por fim, todos os pontos de execução associados a cada passo são somados, representando o grau de complexidade e tamanho do caso de teste.

A partir dos históricos de execução e medidas de pontos de execução, pode-se estimar o tempo em segundos de cada caso de teste (ver Figura 10, retirada de [27]).

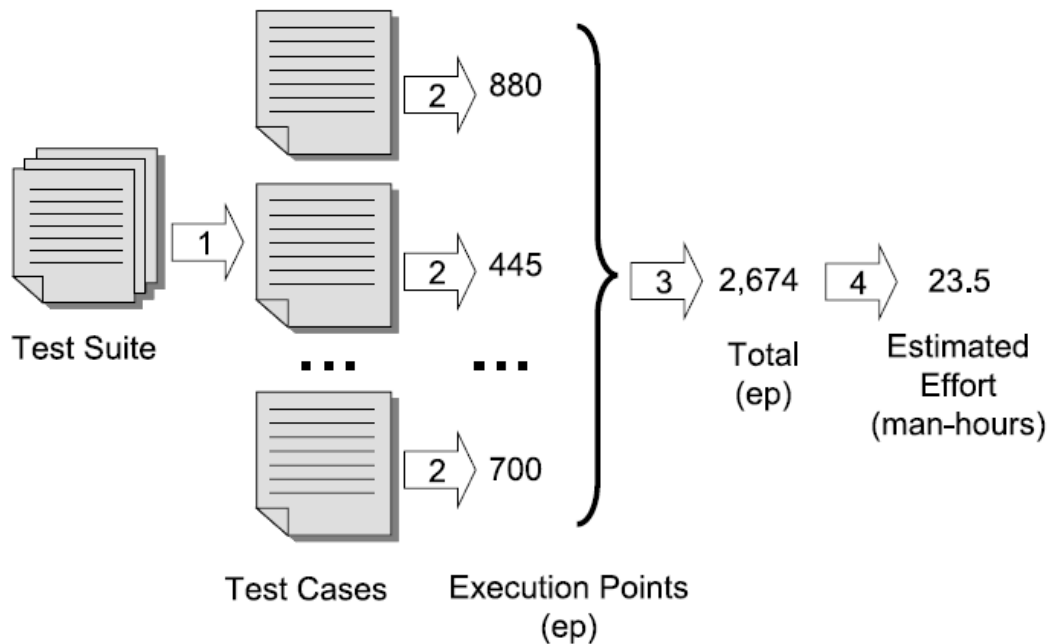


Figura 10 – Cálculo do esforço de uma suíte de casos de testes [27]

Neste método, a seleção dos CTs é realizada usando-se o tempo de execução como restrição para montar a suíte de saída. Dado o tempo disponível para execução dos testes, o algoritmo seleciona CTs até completar o tempo disponível, de acordo com algum critério de escolha definido pelo engenheiro de teste.

O trabalho apresentado em [8] realiza a seleção de CTs com base na cobertura de requisitos, utilizando o custo de execução como restrição de seleção. Isto é, um CT só é selecionado para suíte sendo montada se o seu tempo de execução, somado ao tempo acumulado dos CTs já selecionados, não exceder o tempo total estipulado pelo engenheiro.

Aqui, várias suítes candidatas são montadas ao mesmo tempo, e a suíte escolhida será aquela que respeita o tempo estipulado e cobre maior quantidade de requisitos. Esse trabalho utiliza o método Otimização por Enxame de Partículas (PSO - *Particle Swarm Optimization* [48, 49]) para criar e evoluir as suítes candidatas.

Apesar de utilizar o critério de cobertura de requisitos, esse trabalho não leva em conta a similaridade entre os casos de testes, podendo criar uma suíte com casos de testes muito semelhantes, ao invés de conter casos de testes que executem caminhos diferentes. Uma solução seria combinar as abordagens mencionadas anteriormente para realizar uma melhor seleção de casos de testes.

3.6 Considerações Finais

Como visto, a geração automática de casos de testes cria suítes muito extensas que, na prática, são inviáveis de serem executadas devido ao alto custo, quantidade de recursos e tempo necessários para a execução. Para minimizar este problema, existem diversos métodos de seleção de casos de testes.

Estes métodos auxiliam os engenheiros de testes na seleção de casos de teste que atendam os requisitos do software, encontrem o maior número de erros possíveis e sejam executados dentro do tempo estimado no projeto. A seleção visa obter os melhores casos de testes da suíte, encontrando o maior número de falhas e provendo qualidade o software a ser testado.

Contudo, cada método de seleção isolado não atende as demandas das fábricas de software: obter suítes de casos de testes que atendam o requisito de tempo, recursos disponíveis, tempo de execução e cobertura de requisito; e gerar a suíte perfeita. No capítulo 4 veremos uma proposta de seleção baseada em similaridade textual e cobertura de requisito que busca oferecer mais uma boa opção de método para redução de suítes longas sem perdas.

4 *Sim_TC* – Seleção de Casos de Testes baseado em Similaridade de Texto e Cobertura de Requisito

Como visto no Capítulo 2, as ferramentas de geração automática de casos de testes geram exaustivamente uma suíte com todos os casos de testes possíveis a partir de alguma especificação do software. Claramente, essas suítes tornam-se inviáveis de serem executadas e re-executadas em uma fábrica de software, devido aos prazos curtos e ao custo da execução manual de testes. Por esse motivo, várias pesquisas estão focando na seleção automática de casos de testes que atendam um ou mais critérios de seleção, como visto no Capítulo 3.

Este capítulo apresenta uma estratégia de *Seleção Automática de Casos de Testes* que utiliza a similaridade de texto dos CTs como critério de seleção, preservando a cobertura de requisitos da suíte original. Essa estratégia foi implementada em um sistema protótipo chamado de *Sim_TC*.

A Seção 4.1 apresenta uma visão geral do processo de geração e seleção de casos de teste pelo *Sim_TC*. Como dito, as suítes de teste usadas como entrada pelo *Sim_TC* são geradas pela ferramenta TaRGeT a partir de casos de uso do software a ser testado.

A seguir, a Seção 4.2 descreve em detalhes a técnica de medição da Similaridade Textual proposta neste trabalho, que foi baseada na Distância de Edição ou Distância de Levenshtein [32, 33, 50, 51, 52]. A Seção 4.3 descreve o modo como os pares de casos de testes são ordenados e pré-selecionados, de acordo com o percentual de similaridade definido pelo analista de testes. A Seção 4.4 apresenta a Seleção de Casos de testes e os critérios usados, além do uso da técnica *CliqueFinder*. Por fim, a Seção 4.5, conclui com as considerações finais.

4.1 Visão Geral do Processo de Seleção de Casos de Teste

A Figura 11 apresenta uma visão geral do processo de geração de suítes de teste e posterior seleção de casos de testes com base na similaridade de texto e cobertura de requisitos. Neste processo, podemos observar as seguintes etapas:

1. A criação da suíte de testes a partir do caso de uso, realizada ferramenta TaRGeT;

2. A seleção de casos de teste pelo *Sim_TC* a partir da suíte de casos de testes usando critérios de Similaridade textual e preservando a Cobertura de requisitos;
3. Geração do subconjunto de TCs.

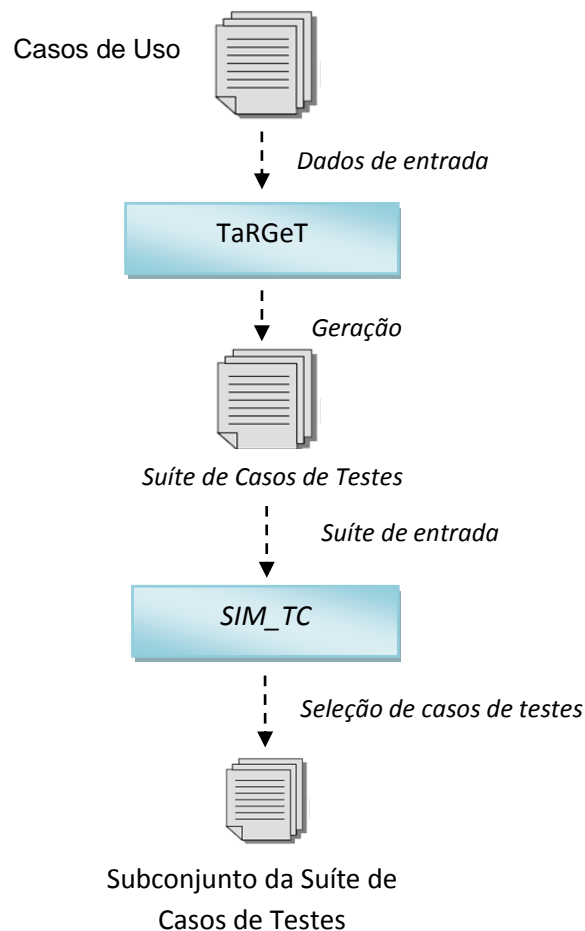


Figura 11 - Visão geral do Processo de geração e seleção de CTs

Vale ressaltar aqui que existe um procedimento de seleção de CTs já implementado na ferramenta TaRGeT [19]. Contudo, tal algoritmo utiliza similaridade de caminhos em CTs para realizar os cortes, enquanto o Sim_TC utilizará o percentual similaridade textual entre CTs. O Capítulo 5 (experimentos) apresenta um estudo de caso no qual os dois procedimentos foram comparados, tendo sido observado um ganho de cobertura de requisitos por parte da nossa abordagem.

A seleção dos CTs usando o *Sim_TC* se realiza através das seguintes fases (Figura 12):

1. Leitura dos casos de testes na suíte de entrada;
2. Criação da matriz de CTs, que relaciona cada CT com todos os outros presentes na suíte de entrada. Trata-se de uma matriz triangular, que vai futuramente armazenar o

valor de similaridade entre dois testes quaisquer da suíte de entrada (mais detalhes a seguir);

3. Cálculo da Similaridade entre os pares de CTs;
4. Ordenação dos pares CTs com base no valor de similaridade calculado;
5. Seleção e corte de CTs com maior taxa de similaridade com outro teste que vai permanecer na suíte, considerando o limiar de similaridade pré-estabelecido, e mantendo a cobertura de requisitos. Esse limiar é determinado pelo analista responsável pelos testes do software. Utiliza-se aqui também a técnica *CliqueFinder* para selecionar um terceiro caso de teste para auxiliar o algoritmo do Sim_TC na seleção de casos de testes.

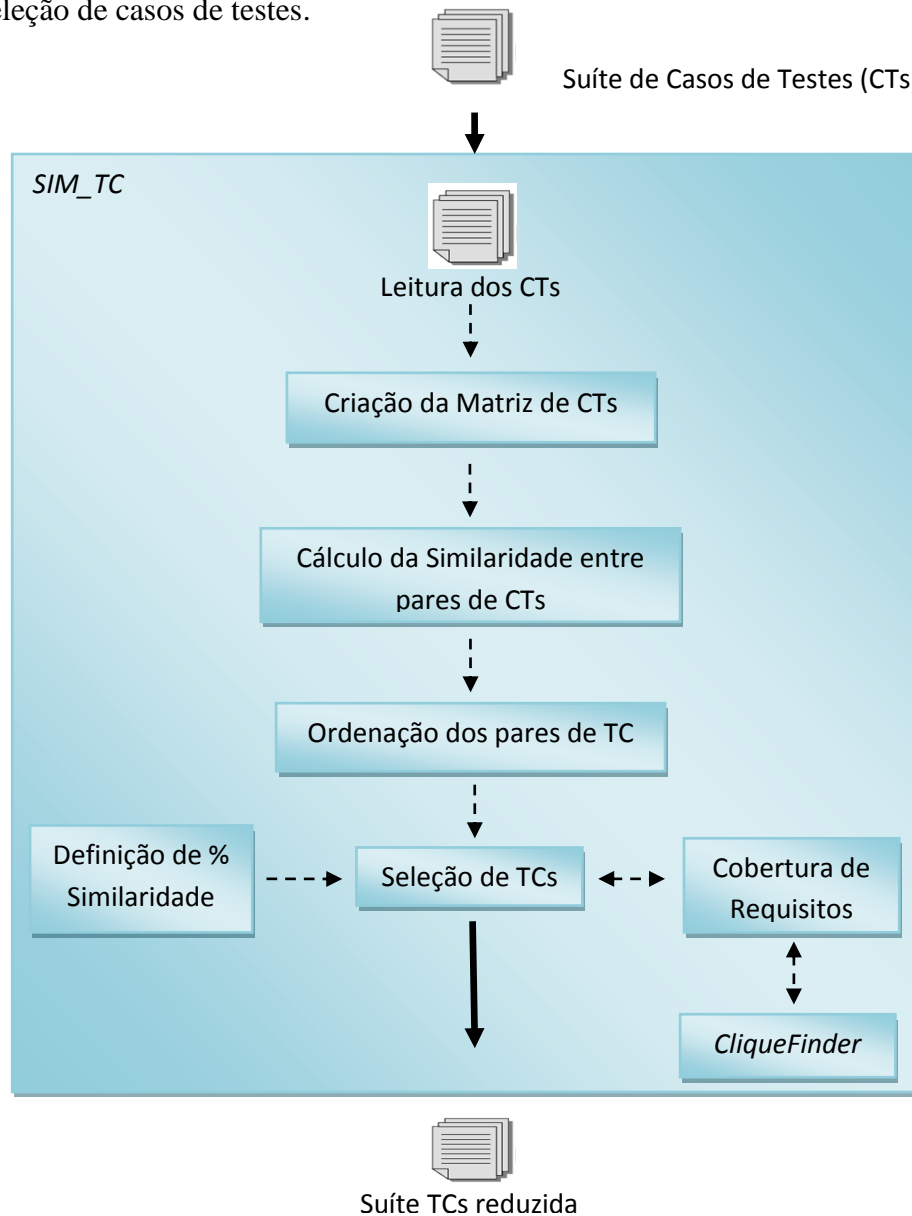


Figura 12 – Seleção de CTs da suíte original usando *Sim_TC*

4.1.1 Detalhamento do *Sim_TC*

O primeiro passo do processo de seleção realizado pelo *Sim_TC* é obter a suíte de caso de teste, que vem escrita no formato .xls seguindo o padrão de documentos da TaRGeT. O *Sim_TC* carrega todos os casos de testes, obtendo informações de:

- ID do caso de Teste
- Requisitos associados
- Condições Iniciais
- Ações
- Resultado esperado
- Condições Finais

Com os dados carregados, o *Sim_TC* irá montar a matriz que relaciona todos os casos de teste da suíte $T = \{CT1, CT2, CT3, \dots, CT_{Total}\}$, conforme Tabela 3, a seguir. Será criada uma matriz triangular inferior, para não repetir as combinações de pares e não comparar o caso de teste com ele mesmo.

Matriz de Casos de Testes										
	CT 1	CT 2	CT 3	CT 4	CT 5	CT 6	CT 7	CT 8	CT 9	CT 10
CT 1	-									
CT 2	x	-								
CT 3	x	x	-							
CT 4	x	x	x	-						
CT 5	x	x	x	x	-					
CT 6	x	x	x	x	x	-				
CT 7	x	x	x	x	x	x	-			
CT 8	x	x	x	x	x	x	x	-		
CT 9	x	x	x	x	x	x	x	x	-	
CT 10	x	x	x	x	x	x	x	x	x	-

Tabela 3 - Matriz de Casos de Testes

O *Sim_TC* começa comparando o primeiro caso de teste da suíte com todos os outros, exceto com ele mesmo, calculando a similaridade textual entre eles, através da função de cálculo de similaridade extraído da TaRGeT. Esta função de cálculo de similaridade foi

desenvolvida e implementada na TaRGeT. Foi realizado o reuso da função e adaptada a necessidade do *Sim_TC*, neste caso, calcular a similaridade de texto.

Finalizado o primeiro passo, o segundo caso de teste é selecionado e submetido ao emparelhamento com toda a suíte, excetuando-se ele e o caso de teste anterior. O processo é realizado com todos os CTs seguintes. Desta forma, o *Sim_TC* evita que haja comparações duplicadas, ganhando tempo de execução. Esta etapa se encerra quando o algoritmo encontra o caso de teste $CT_{Total-1}$, o penúltimo caso da lista, uma vez que não é necessário comparar o último caso de teste com ele mesmo.

A partir dessa matriz, o *Sim_TC* cria uma lista de pares de CTs ordenados pelo valor da sua similaridade, do mais similar para o menos similar. Cada par que tiver seu valor de similaridade maior que o limite estipulado pelo analista é candidato a corte. Dado um par de CTs de similaridade acima do limiar, o *Sim_TC* escolhe cortar o teste do par que cobrir menos requisitos. Porém, o corte só é feito se os requisitos do teste escolhidos forem cobertos por algum teste que irá permanecer na suíte final, a fim de preservar a cobertura de requisitos.

Esse processo se repete até que os pares restantes não apresentem mais similaridade dentro do limiar escolhido.

4.2 Medida de Similaridade Textual

A principal maneira de se medir similaridade entre duas *strings* quaisquer é conhecida como ‘distância de edição’. A distância de edição é uma medida do esforço de se transformar uma *string* A em uma outra *string* B, levados em consideração seus elementos/caracteres e a ordem entre eles.

A métrica mais utilizada para medir similaridade entre documentos textuais é a distância de edição conhecida como Distância de Levenshtein[32, 33, 50, 51, 52]. A métrica consiste em calcular o número mínimo de operações necessárias para transformar uma *string* A em outra B. Essas operações são: *inserções, remoções e substituições de um caractere*. Esta é uma métrica bem estabelecida [33], com amplas aplicações, como exemplo: agrupamento de consultas em mecanismos de busca [34, 36], processamento de imagens, dentre outros [35, 37, 38, 39].

A Figura 13 mostra um exemplo do cálculo dessa distância entre as *strings* ‘tráfego’ e ‘tráfico’.

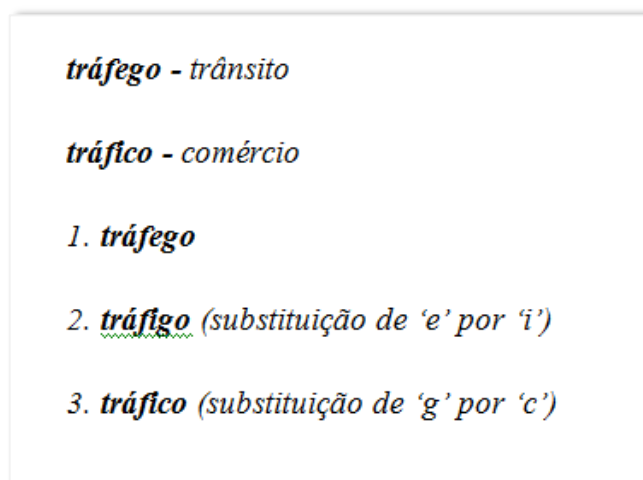


Figura 13 - Aplicação da Distância de Levenshtein

No exemplo da Figura 13 a distância Levenshtein entre as palavras “tráfego” (trânsito) e “tráfico” (comércio) é 2, pois com duas operações conseguimos transformar uma palavra na outra, e não há maneira de o fazer com menos de duas substituições.

O cálculo de similaridade de texto realizado pelo *Sim_TC* foi implementado e está presente na ferramenta TaRGeT² [7]. Esta função foi reutilizada e adaptada para medir similaridade textual entre CTs, utilizando informações contidas nos campos ações e resultado esperado. Utilizamos apenas os campos de “ações” e “resultado esperado”, pois, além de alguns casos de testes podem vir sem a informação do campo “condição inicial”, os dois campos contêm as principais informações necessárias para a execução do CT pelos testadores. O algoritmo implementado na TaRGeT foi criado com o fim de simplificar a manutenção de mudanças ocorridas nas suítes de teste geradas por essa ferramenta, fazendo o emparelhamento entre CTs novos e antigos (que tenham sofrido alguma alteração) [7]. Esse algoritmo foi baseado na distância de Levenshtein.

A Figura 14 traz um exemplo de dois CTs semelhantes, a fim de ilustrar o cálculo realizado pelo *Sim_TC*.

² Apenas o algoritmo de similaridade textual foi extraído da TaRGeT, demais fases do processo de seleção do *Sim_TC*, como a criação da matriz, ordenação dos pares e seleção de CTs foram implementados nesta dissertação.

Descrição	Passos	Resultados Esperados
Func_001	Nada	
Caso de uso:	6378#UC_01	
Requisito:	FR_1513 FR_1429	
Setup:	Nada	
Condições Iniciais	Modo Airplane está ligado	
Notas	Caso de Teste gerado por TaRGet system	
Passos		
1	Ligue o celular	Uma mensagem é recebida durante a sequência de ligação
2	Selecione a opção de Airplane Mode	O celular será ligado em Airplane Mode
Condições Finais	Nada	

Descrição	Passos	Resultados Esperados
Func_002	Nada	
Caso de uso:	6378#UC_01	
Requisito:	FR_1513 FR_1428	
Setup:	Nada	
Condições Iniciais	Modo Airplane está desligado	
Notas	Caso de Teste gerado por TaRGet system	
Passos		
1	Ligue o celular	Uma mensagem é recebida durante a sequência de ligação
2	Selecione a opção de Airplane Mode	O celular será ligado em Airplane Mode off
Condições Finais	Nada	

Figura 14 - Casos de Testes

O cálculo do Sim_TC é baseado na similaridade entre os passos (as ações e resultados esperados) dos CTs. O Sim_TC calcula a similaridade entre os passos do CT1 $\{p_1, p_2, \dots, p_n\}$ e os demais passos do CT2 $\{p'_1, p'_2, \dots, p'_n\}$, como podemos ver na Figura 15.

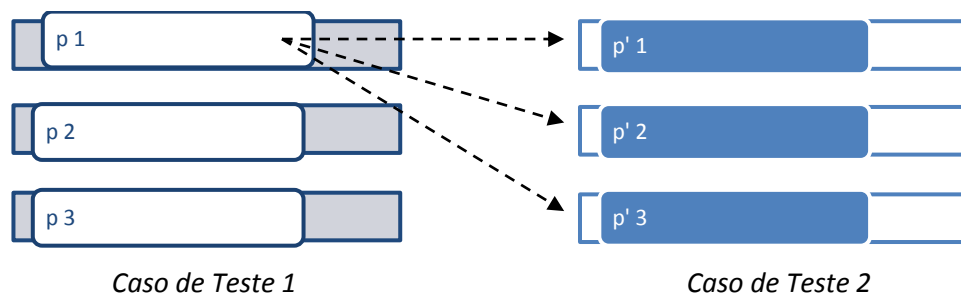


Figura 15 – Similaridade entre os passos dos casos de testes

O algoritmo também verifica a ordem (sequência) dos passos dos CTs sendo comparados. Caso seja constatada a similaridade textual entre o texto, mas a sequência seja diferente (isto é, se tiver havido alguma inversão dos passos), o percentual de similaridade

será penalizado por um valor simbólico, pois o ajuste dos pesos é complexo, o que requer o conhecimento de um especialista.

O *Sim_TC* recebe os valores de similaridade entre os passos dos casos de teste CT_1 e CT_2 , calcula a similaridade geral entre eles, e adiciona o resultado à matriz de CTs vista anteriormente. A primeira rodada de cálculo se encerra quando o algoritmo calcula a similaridade textual entre o CT_{n-1} e CT_n .

A Figura 16 mostra a fórmula para calcular o percentual de similaridade textual utilizada pela TaRGeT e pelo *Sim_TC*, onde N é o número de passos do CT_1 que estão idênticos no CT_2 , verificação realizada através do cálculo da distância de Levenshtein; $S1$ é o número de passos do CT_1 e $S2$ é o número de passos do CT_2 , onde são dividido por 2, para realizar a média de número de passos.

$$\text{Similaridade}(CT_1, CT_2) = \left(\frac{\frac{N}{S1 + S2}}{2} \right)$$

Figura 16 - Fórmula de Cálculo de Similaridade [46]

De acordo com os CTs da Figura 14, calculamos a similaridade de texto entre [CT_1 , CT_2], onde ambos possuem 4 passos. Através do cálculo de distância de Levenshtein, é possível calcular o valor da variável N da fórmula citada na figura 15 e constatar que entre o passo 2 do CT_1 e CT_2 existe uma divergência de 1 palavra no resultado esperado, neste caso, o N é igual a 3,9232 (o valor de: 3 – temos 2 passos + 1 resultado esperado iguais; e 0,9232 pela palavra “ off” diferente).

Neste exemplo, a similaridade textual entre os dois CTs será de 98,08%, conforme vemos na figura 17.

$$\text{Similaridade}(CT_1, CT_2) = \left(\frac{\frac{3,9232}{4 + 4}}{2} \right) = \frac{3,9232}{4} = 0,9808 = 98,08\%$$

Figura 17 – Exemplo de cálculo de similaridade entre CTs

Encerado o primeiro ciclo, o próximo passo é comparar o CT_2 com o restante da suíte de casos de teste, exceto com o CT_1 , pois já foi calculada a similaridade textual na primeira rodada. O cálculo de similaridade de toda a suíte encerra quando o CT_{n-1} é comparado com o último caso de testes, neste caso o CT_n . Na Tabela 4, vemos um exemplo da matriz de similaridade com o percentual de similaridade textual entre os pares de casos de testes de uma dada suíte.

Matriz de Similaridade										
-	ID: 1	ID: 2	ID: 3	ID: 4	ID: 5	ID: 6	ID: 7	ID: 8	ID: 9	ID: 10
ID: 1	-									
ID: 2	32.3%	-								
ID: 3	32.15%	75.0%	-							
ID: 4	39.2%	18.34%	20.01%	-						
ID: 5	25.27%	13.75%	13.1%	75.0%	-					
ID: 6	26.39%	13.03%	13.1%	64.5%	89.5%	-				
ID: 7	21.12%	10.42%	10.48%	51.61%	71.6%	80.0%	-			
ID: 8	20.22%	11.0%	10.48%	60.0%	80.0%	71.6%	91.61%	-		
ID: 9	37.72%	17.37%	17.07%	86.0%	63.2%	63.2%	50.56%	50.56%	-	
ID: 10	28.29%	13.03%	12.8%	64.5%	80.48%	90.98%	60.0%	60.0%	75.0%	-

Tabela 4 - Matriz de Similaridade entre os Pares de Casos de Testes

4.3 Ordenação dos Pares dos Casos de Testes

Calculada a similaridade de todos os pares de casos de testes, estes serão dispostos em ordem decrescente, de acordo com o percentual de similaridade (Tabela 5). Esse percentual, definido pela analista de testes, é usado para definir quais casos de testes serão considerados pelo processo de seleção de CTs.

Os pares de casos de testes com percentuais próximo dos 100% são mais similares, sendo os primeiros a serem considerados para corte. Já os pares de CTs com percentuais de similaridade abaixo do limiar estabelecido não são considerados pelo procedimento de corte.

O percentual de similaridade definido pelo usuário é parametrizado, e pode ser alterado a qualquer momento. A partir da definição desse percentual, o *Sim_TC* seleciona os

pares de casos de testes que apresentem percentual maior ou igual ao definido pelo analista. Posteriormente, esses casos de testes serão analisados na seleção dos casos de testes.

Na Tabela 5, temos outro exemplo hipotético de uma seleção para o qual foi definido um limiar de 80% de similaridade textual pelo analista de testes. Os pares de casos de testes que possuem um percentual inferior ao definido são descartados por esse procedimento.

Casos de Testes	% Similaridade
[1, 12]	100,00%
[14, 15]	94,28%
[7, 8]	91,61%
[18, 19]	91,61%
[5, 11]	90,98%
[5, 6]	89,50%
[10, 11]	89,50%
[4, 9]	86,00%
[4, 13]	86,00%
[9, 13]	84,13%
[8, 18]	83,55%
[7, 19]	83,55%
[19, 20]	83,34%
[5, 17]	82,48%
[6, 11]	80,48%
[5, 10]	80,48%
[6, 7]	80%
[5, 8]	80%
[17, 18]	80%

Tabela 5 - Ordenação dos Pares de Casos de Testes

4.4 Seleção dos Casos de Testes

Nesta fase, os pares de testes com o percentual igual ou superior ao limiar definido pelo analista serão considerados para corte, de acordo com a similaridade textual entre si e a cobertura de requisito.

Para os pares de CTs que possuem a similaridade de texto baixa, abaixo do limiar de similaridade de texto definido pelo analista, e possuem o mesmo requisito, estes não tem os CTs eliminados e ambos permanecem na suíte, como exemplo, temos o CT1 com o passo1: Tirar uma foto, e requisito R1; e o CT2 com o passo2: capturar uma figura, com o requisito R1. O requisito é o mesmo, o testador entende que deverá tirar uma foto (figura), mas a similaridade de texto entre os casos de testes é baixa.

Para definir a cobertura de requisito entre os pares de casos de testes, verificam-se quantos e quais requisitos cada CT está cobrindo, pois um caso de teste pode estar associado a um ou mais requisitos. Os casos de testes da suíte são gerados a partir de um caso de uso que atende um ou vários requisitos. Quando esses casos de testes são gerados, herdam as informações de casos de uso e requisitos, informações necessárias na seleção do *Sim_TC*.

Veremos a seguir alguns critérios usados no processo de seleção de CTs:

1. Se o percentual de similaridade do par de casos de teste for igual a 100%, o *Sim_TC* irá selecionar um caso de teste aleatoriamente (ver Figura 18). Por serem similares, os CTs devem conter os mesmos requisitos. A exceção para dois casos de testes serem 100% similares textualmente e não estarem associados aos mesmos requisitos, seria o caso da geração ser realizada manualmente, o que estaria passivo a erros.

```
Se (Similaridade [TC_A, TC_B] == 100%) {
    Então Selecione um caso de teste aleatoriamente;
}
```

Figura 18 - Pseudocódigo – O *Sim_TC* seleciona aleatoriamente um CT quando há 100% de similaridade de texto entre os CTs.

2. Caso o par de Casos de Testes não atenda a primeira condição, de 100% de similaridade textual, o *Sim_TC* verifica se atende a segunda condição: o percentual de similaridade do par está dentro do limiar definido pelo analista. O segundo passo desta condição é verificar a cobertura de requisitos. Em um exemplo hipotético, o CT A possui 4 requisitos associados e o CT B possui 2 requisitos. O sistema verifica se os requisitos do caso de teste com maior número de requisitos atendem (cobrem) os requisitos do caso de teste com menor número de requisito (Figura 19). O CT A cobre os requisitos R1, R2, R3 e R4; o CT B cobre os requisitos R1 e R3. Como o caso de testes A cobre os requisitos do caso de teste B, há 100% de cobertura de requisitos de B (R1 e R3). O *Sim_TC* elimina o caso de teste B e seleciona o caso de teste A. Desta forma, a seleção deste caso de teste garante a cobertura dos requisitos

R1 e R3, além dos requisitos R2 e R4. Após cada verificação, os requisitos de cada CT não são armazenados para os passos posteriores.

```

Se (similaridade [TC_A , TC_B ] > parâmetro){
    Se (Requisito TC_A cobre Requisito TC_B){
        Então selecione o caso de teste A
    } Senão Se (Requisito TC_B cobre Requisito TC_A){
        Então selecione o caso de teste B
    }
}

```

Figura 19 – Pseudocódigo – O *Sim_TC* garante que os requisitos do CT que foi cortado estão garantidos pelo CT que foi selecionado, sem perdas.

3. Caso a verificação anterior, de 100% de cobertura de requisitos entre o par de casos de testes, não se confirme, o *Sim_TC* irá verificar o percentual de cobertura de requisitos. Quando não há uma cobertura entre os requisitos presentes nos casos de testes do par (Cobertura Requisitos = 0%), por exemplo, o caso de teste A cobre os requisitos R1 e R2, e o caso de teste B cobre os requisitos R3 e R4, o *Sim_TC* não eliminará nenhum caso de teste, mantendo os dois CTs na suite de saída. Como visto na Figura 20, o algoritmo do *Sim_TC* verifica que não há cobertura de requisito entre os casos de testes e seleciona os Casos de Testes A e B.

```

SE (Similaridade [TC_A, TC_B] > parâmetro{
    SE (Requisito TC_A cobre Requisito TC_B) {
        Então selecione o caso de teste A;
    } Senão SE (Requisito TC_B cobre Requisito TC_A){
        Então selecione o caso de teste B;
    } Senão SE (Requisito TC_A  $\cap$  Requisito TC_B =  $\emptyset$ ){
        Então selecione casos de testes A e B;
    }
}

```

Figura 20 – Pseudocódigo do *Sim_TC*, o Caso de Teste A possui o requisito R1 e R2, e o caso de teste B possui os requisitos R3 e R4

4. Caso a verificação da cobertura de requisito anterior não se confirme, o *Sim_TC* realizará a última verificação. Verificação esta que ocorre quando a cobertura de

requisito não alcança os 100% necessários para a eliminação de um CT, porém existe alguma interseção entre os requisitos cobertos pelos CTs do par. Neste caso, a cobertura é $> 0\%$ e $< 100\%$, ou seja, pelo menos um requisito de um caso de teste é coberto pelo outro caso de teste do par, mas existem outros requisitos que não são cobertos, impossibilitando a simples seleção de um caso de teste e conseqüente eliminação do outro. Veja exemplo na Figura 20. Aqui, o CT X cobre os requisitos R1 e R2, e o CT Y cobre os requisitos R2, R4 e R5. Assim, temos a cobertura de requisitos de 50% de Y em relação a X.

Nessas situações onde não é possível eliminar um CT de forma simples, o *Sim_TC* usa o método de *CliqueFinder* [40, 41, 44, 47]. Com *CliqueFinder*, o *Sim_TC* tenta localizar um terceiro caso de testes que auxilie o sistema a decidir CT selecionar. No caso apresentado acima, o *Sim_TC* adia a decisão de qual CT será cortado, ‘guarda’ o par de CT (X,Y), e continua o processo de seleção seguindo a lista de pares de CTs. Quando o sistema encontra um CT que faz par com dois casos de testes armazenados na lista – por exemplo, (X, Z) e (Y, Z), e cujo percentual de similaridade textual está dentro do limite estabelecido, é verificada a cobertura de requisito desses dois pares, a fim de definir qual CT será selecionado, como vemos na Figura 21.

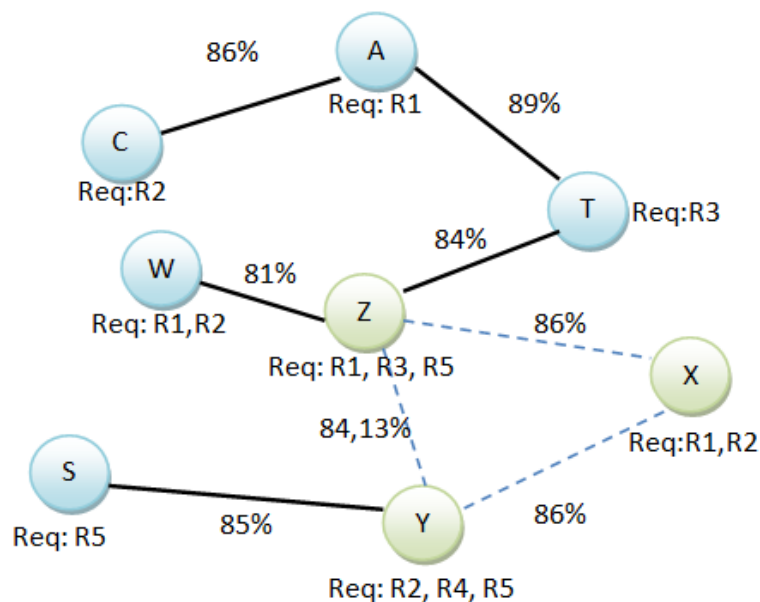


Figura 21 – Grafo dos Pares de Casos de Testes

O *Sim_TC* usa o CT Z para decidir pela seleção do caso de teste X ou Y. O CT Z cobre os requisitos R1, R3 e R5. O sistema verifica que os requisitos do caso de teste X (R1 e R2)

são cobertos pelo caso de teste Y (R2) e pelo caso de teste Z (R1). Desta forma, o *Sim_TC* seleciona o caso de teste Y dentre o par [X, Y] e o caso de teste Z, eliminando o caso de teste X, e garantido assim a cobertura de requisito (Figura 22).

```

Se (similaridade [ TC_X , TC_Y ] > parâmetro){
...
  Senão se (cobertura_requisito TC_X, TC_Y > 0% & <100 %){
    Então armazene o par [ TC_X, TC_Y ]
  }
...
Próximos pares de casos de teste
...
Se (similaridade [ TC_X , TC_Z ] > parâmetro){
  Se (Requisito TC_X cobre Requisito TC_Z)
    Então armazene o par [ TC_X, TC_Z ]
}
Se (similaridade [ TC_Y , TC_Z ] > parâmetro){
  Se (Requisito TC_Y cobre Requisito TC_Z)
    Então armazene o par [ TC_Y, TC_Z ]
}

```

Figura 22 – Pseudocódigo do *Sim_TC* – Seleção entre 3CTs

Outra visão do processo pode ser vista na Figura 23, onde não há uma cobertura completa dos requisitos entre os CTs, e o *Sim_TC* seleciona um CT diferente dos CTs do par para realizar o *CliqueFinder*. É importante notar que os pares em que o terceiro CT está contido têm o percentual superior ao limiar definido pelo analista. Caso o terceiro CT tenha o percentual de similaridade de texto menor que o definido pelo o analista, este não será usado no *CliqueFinder*.

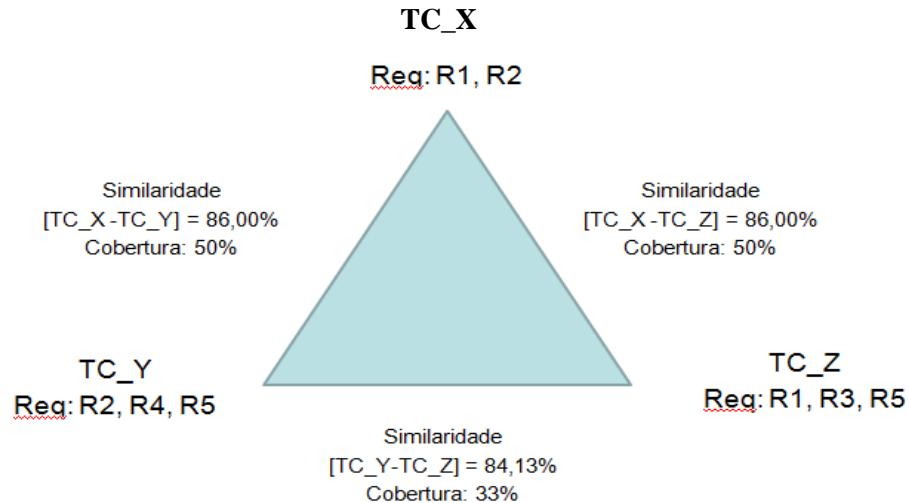


Figura 23 – *CliqueFinder* na Seleção de Casos de Testes

Após o processo de seleção, a suíte de casos de testes selecionados pelo *Sim_TC* é apresentada ao usuário, conforme Figura 24. Além da lista disponível em tela, o sistema também possibilita ao usuário salvar o resultado no formato .txt.

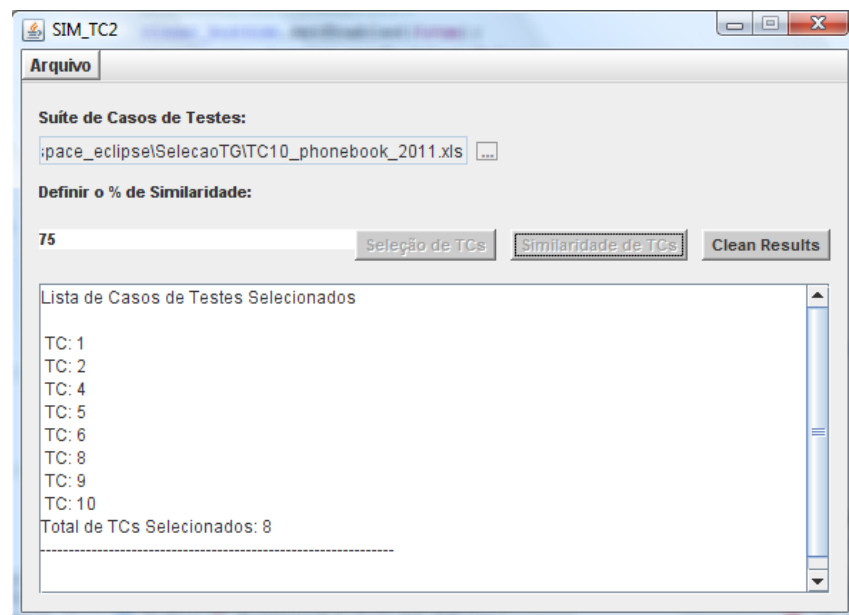


Figura 24 – Resultado do *Sim_TC*

4.4 Considerações Finais

Neste capítulo apresentamos uma visão geral e o pseudocódigo do *Sim_TC*. Mostramos todo o fluxo, desde a entrada da suíte de casos de testes até a seleção dos casos de teste, gerando uma nova suíte. A seleção visa diminuir a grande quantidade de casos de testes gerados pela TaRGeT [7], porém garantindo a cobertura dos requisitos associados à suíte.

Para isso, o *Sim_TC* seleciona os casos de testes cujo percentual de similaridade de seus pares é igual ou superior ao limiar definido pelo Analista de Testes.

A seleção é concluída quando todos os pares de casos de testes contidos na lista definida pelo analista passaram pelo processo de seleção, e todos os requisitos contidos na suíte estão presentes, isto é, representados por pelo menos um caso de teste.

O protótipo descrito neste capítulo apresenta uma técnica única de seleção de casos de testes: seleção e corte utiliza o critério de Similaridade Textual, com garantias de Cobertura de Requisitos. O *Sim_TC* também será integrado à TaRGeT [7], tornando-se mais uma opção de critério de seleção de CT, além da seleção já existente [19], que considera apenas o tamanho da suíte de CT desejado.

O próximo capítulo apresenta os experimentos realizados para validar o *Sim_TC*.

5 Experimentos e Resultados Obtidos

O capítulo anterior apresentou em detalhes o processo de seleção realizado pelo *Sim_TC*, desde a entrada da suíte de casos de testes, até a seleção dos CTs que vão formar nova suíte com os testes selecionados obedecendo os critérios de similaridade textual e cobertura de requisito.

Inicialmente, este capítulo apresenta o passo a passo de um teste realizado para validar o correto funcionamento do sistema, verificando se todas as condições (critérios) de seleção presentes no algoritmo do *Sim_TC* são executadas de acordo com o que foi especificado no Capítulo 4 (Seção 5.1).

A seguir a Seção 5.2 apresenta dois estudos de caso realizados para avaliar a qualidade dos nossos resultados, comparando nosso processo com o processo manual de seleção de CTs realizado por analistas/especialistas em teste; e com a seleção automática realizada pela LTSBT [19], integrada à ferramenta TaRGeT (visto no Capítulo 2).

A Seção 5.3 traz as considerações finais sobre o trabalho aqui apresentado.

5.1 Teste de validação do *Sim_TC*

Como dito, o objetivo desse teste foi verificar se os critérios de seleção vistos no capítulo anterior estão corretamente implementados no *Sim_TC*. Assim sendo, buscou-se percorrer todos os caminhos possíveis do algoritmo, a fim de testá-lo.

Para isto, selecionamos uma suíte de casos de testes da *feature Phonebook* para dispositivos móveis. A suíte é composta por 10 casos de testes funcionais que têm por objetivo executar manualmente o código do software a ser testado nas condições de: *criar um novo contato e procurar um contato na agenda*.

Cada caso de testes está associado a um ou mais requisitos. Cada requisito está presente em um ou mais CT, conforme Tabela 6.

Casos de Testes	Requisitos Associados
CT 01	TRS_11111_101, TRS_11111_115
CT 02	TRS_111166_102, TRS_11111_101
CT 03	TRS_111166_102
CT 04	TRS_111166_102, TRS_111166_103
CT 05	TRS_111166_103, TRS_11111_115
CT 06	TRS_111166_103, TRS_111166_116
CT 07	TRS_11111_110, TRS_11111_104, TRS_111166_105
CT 08	TRS_11111_110
CT 09	TRS_11111_104, TRS_111166_105
CT 10	TRS_11111_110

Tabela 6 – Requisitos associados aos CTs da suíte escolhida

O limiar de similaridade textual estipulado para este experimento foi de 70%. Assim, pares de CTs com similaridade abaixo de 70% não serão considerados para corte.

Outros limiares foram testados, porém, para a suíte de CTs escolhida, os liminares abaixo de 70% não permitiam cortar mais CTs do que esse limiar permite, devido à obrigatoriedade de se manter a cobertura de requisitos da suíte original, ou seja, o objetivo é haver 100% de cobertura de requisitos. Assim sendo, para essa suíte de CTs, o liminar de corte mais baixo é de 70% de similaridade. Limiares mais altos podem ser aplicados aqui, porém permitem poucos cortes de CTs, não causando uma redução satisfatória no tamanho da suíte de testes. O limiar de similaridade de texto fica a cargo da necessidade do analista, ele poderá escolher um limiar mais baixo ou mais alto, dependendo da quantidade de CTs que o mesmo deseja na suíte selecionada.

5.1.1 Seleção de Casos de Testes pelo Sim_TC

Inicialmente, o *Sim_TC* lê a suíte de casos de testes no formato xls conforme *template* da Tabela 7, e recebe o limiar de similaridade de texto estipulado pelo testador, neste caso, de 70%. Em seguida, inicia-se o processo de seleção (botão “Realizar Seleção” na Figura 25).

Test Case Name	Case Description	Step Description	Expected Results
Func_0001	Caso de uso:	01	
	Requisitos:	TRS_11111_101, TRS_11111_115	
	Configuração:	Este é a configuração do caso de uso principal.	
	Condições Iniciais:	1) A aplicação Agenda está instalada no fone. 2) Existe bastante memória no fone para inserir um novo contato.	
	Notes:	Caso de Teste gerado automaticamente pela TaRGiT.	
	Procedimento de Teste (Número do passo):		
	1	Inicialize a aplicação “Agenda”.	A aplicação “Agenda” é exibida.
	2	Selecione a opção “Novo Contato”.	O Formulário de Novo Contato é exibido.
	3	Preencha o nome do contato. Preencha o número do fone.	O formulário do Novo Contato é preenchido.
	4	Confirme a criação do contato.	Um novo contao é criado na aplicação “Agenda”.
	Condições Finais:	Nada.	

Tabela 7– *Template* da Suíte de Casos de Testes do Estudo de Caso 1

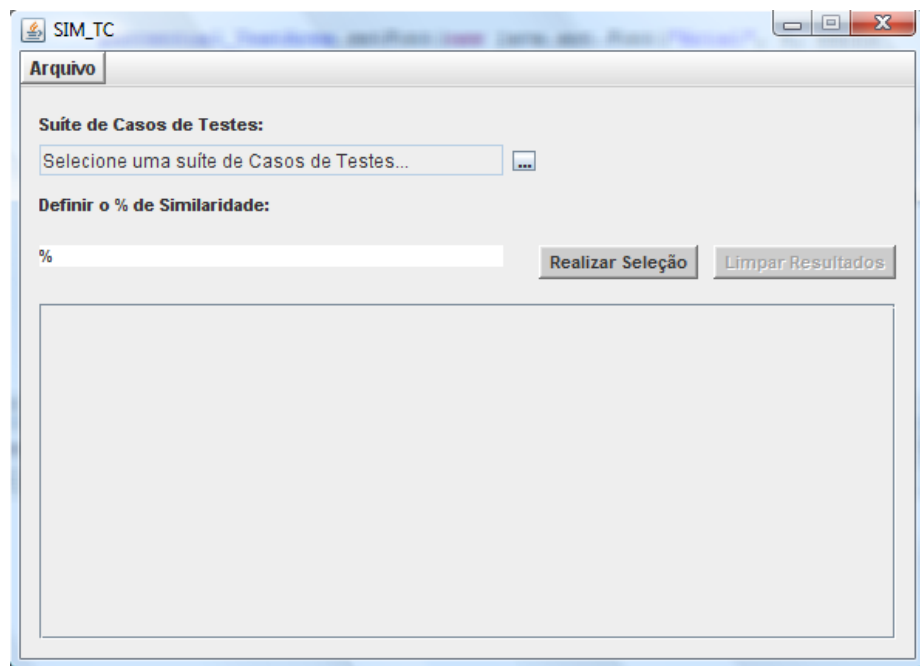


Figura 25 – Tela inicial do *Sim_TC*

A seguir, o *Sim_TC* cria uma matriz 10x10 para armazenar os valores de similaridade textual entre todos os CTs da suíte original. Após o cálculo de similaridade de texto, o algoritmo descarta os pares que apresentam o percentual abaixo do limiar definido, e armazena os pares restantes em uma lista, em ordem decrescente de similaridade (Tabela 8).

Nº do Par	Pares de Casos de Testes	% Similaridade de texto
1	[8 , 10]	100.0 %
2	[2 , 3]	94.51 %
3	[1 , 5]	90.39 %
4	[3 , 4]	86.54 %
5	[2 , 4]	83.34 %
6	[6 , 8]	80.68 %
7	[6 , 10]	80.68 %
8	[5 , 6]	78.47 %
9	[1 , 6]	73.34 %
10	[4 , 9]	73.3 %

Tabela 8 – Lista dos Pares de Casos de Testes com respectivos percentuais de Similaridade de Texto

Dada a lista de pares de CTs, o processo de seleção é iniciado baseando-se na cobertura de requisitos. O primeiro par de CTs é examinado, e é constatado que, além dos 100% de similaridade entre os CTs 8 e 10, eles possuem os mesmos requisitos. Nesta situação, um dos dois casos é selecionado aleatoriamente, como podemos observar na Figura 26.

CTs [8, 10]: 100.0 %
 Requisitos do CT 8: TRS_11111_110
 Requisitos do CT 10: TRS_11111_110
 CT: 8 cobre 10

Figura 26 – Log do *Sim_TC*: CTs vs Requisitos 1

A seguir, o segundo par de casos de testes é verificado: [CT₂ , CT₃] possuem 94.51 % de similaridade de texto. O sistema verifica que o CT 2 possui 2 (dois) requisitos e que um deles cobre um requisito do CT 3. O CT 2 é então selecionado (ver Figura 27).

CTs [2, 3]: 94.51 %
 Requisitos do CT 2: TRS_111166_102,
 TRS_11111_101
 Requisitos do CT 3: TRS_111166_102
 CT: 2 cobre 3

Figura 27 – Log do *Sim_TC*: CTs vs Requisitos 2

No terceiro par da lista, encontramos uma situação em que não há 100% de cobertura de requisitos entre os CTs (Figura 28). Apenas um requisito é compartilhado pelos dois CTs (TRS_11111_115).

CTs [1, 5]: 90.39 %
 Requisitos do CT 1: TRS_11111_101, TRS_11111_115
 Requisitos do CT 5: TRS_111166_103,
 TRS_11111_115
 cobre apenas: 1 requisito

Figura 28 – Log do *Sim_TC*: Cobertura incompleta dos requisitos

Como visto no Capítulo 4, para esses casos o processo de seleção e corte é adiado até que o *Sim_TC* tenha mais informações sobre os outros CTs da suíte, para então realizar um corte sem perdas de requisisto. O *Sim_TC* busca na lista de pares de CTs um terceiro CT que faça par com os dois CTs do par sendo analisado no momento, e então verifica a cobertura de requisitos desse novo CT. Este procedimento é realizado com o auxílio do *CliqueFinder*.

Neste exemplo, o *Sim_TC* busca de um terceiro CT que faça par com o 3º par da lista ([CT₁, CT₅]). O CT₆ é encontrado, pois ele faz par com CT₁ e CT₅ ([CT₅, CT₆] = 78,47% e [CT₁, CT₆] = 73,34 %). Os requisitos do CT₆ são analisados, e é constatado que o par [CT₁, CT₆] cobre totalmente os requisitos do CT₅ (Tabela 9). Com base nessa constatação, os casos de testes 1 e 6 são selecionados, e o caso de teste 5 é eliminado.

Caso de Teste	Requisitos	
CT 1	TRS_11111_101	TRS_11111_115
CT 5	TRS_111166_103	TRS_11111_115
CT 6	TRS_111166_103	TRS_111166_116

Tabela 9 – Cobertura de requisitos entre 3 CTs

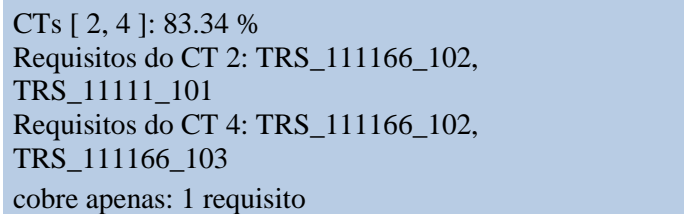
Continuando o processo de seleção a partir do ponto de interrupção (par 3 comentado acima), o *Sim_TC* analisa então os pares 4 e 5.

No 4º par da lista ([CT₃, CT₄]), o *Sim_TC* verifica que possuem 86.54% de similaridade de texto. O sistema verifica que o CT 4 possui 2 (dois) requisitos e que um deles cobre um requisito do CT 3 (ver Figura 29).

CTs [3, 4]: 86.54 %
 Requisitos do CT 3: TRS_111166_102
 Requisitos do CT 4: TRS_111166_102,
 TRS_111166_103
 CT: 4 cobre 3

Figura 29 – Log do *Sim_TC*: Seleção do 4º par

No 5º par da lista ([CT₂, CT₄]), com 83,34% de similaridade de texto, o Sim_TC verifica que ambos os CTs possuem 2 (dois) requisitos associados e apenas 1 (um) requisito dentre eles é similar. O Sim_TC busca de um terceiro CT que faça par com o 3º par da lista e que este esteja dentro do limiar informado pelo analista, entretanto, não o encontra. Nesta situação, os 2 (dois) CTs são selecionados (ver Figura 30).

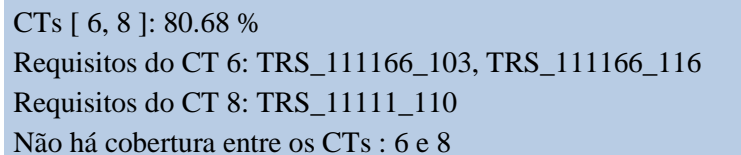


```

CTs [ 2, 4 ]: 83.34 %
Requisitos do CT 2: TRS_111166_102,
TRS_11111_101
Requisitos do CT 4: TRS_111166_102,
TRS_111166_103
cobre apenas: 1 requisito
  
```

Figura 30 – Log do *Sim_TC*: Seleção do 5º par

A seguir, o 6º par é analisado. Neste par não existe nenhuma interseção entre os requisitos associados aos CTs. Assim, o Sim_TC seleciona (mantém) os dois CTs na suíte de saída, como podemos verificar na Figura 31.



```

CTs [ 6, 8 ]: 80.68 %
Requisitos do CT 6: TRS_111166_103, TRS_111166_116
Requisitos do CT 8: TRS_11111_110
Não há cobertura entre os CTs : 6 e 8
  
```

Figura 31 – Log do *Sim_TC*: Não existe cobertura de requisitos entre os CTs

Concluída a seleção o *Sim_TC* exibe na tela o resultado da nova suíte de casos de testes. A nova suíte consiste nos CTs selecionados, e oferece a garantia de que todos os requisitos na suíte original estão cobertos aqui também (Figura 32).

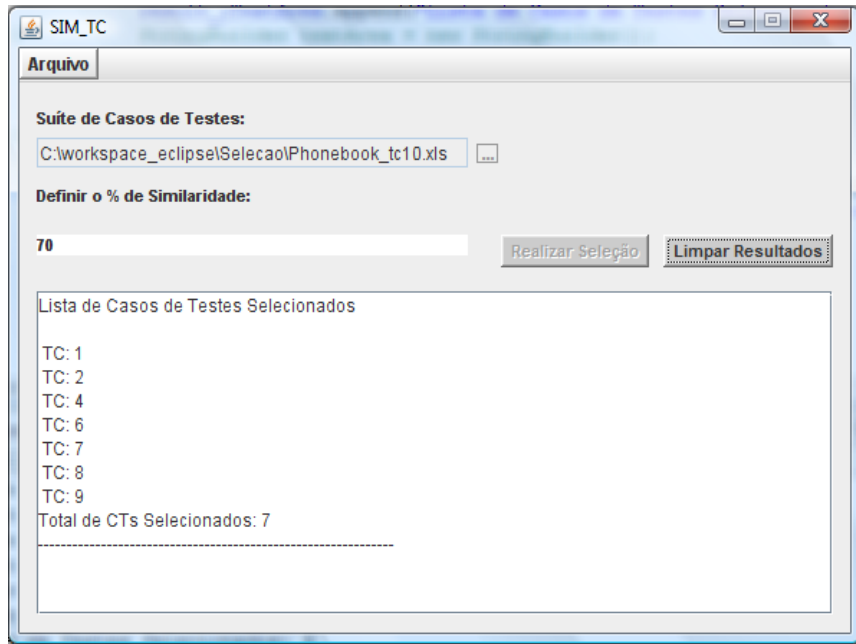


Figura 32 – Resultado da Seleção de Casos de Testes 1

Com esse estudo de caso, foi possível verificar que o *Sim_TC* está funcionando a contento para o exemplo, realizando a seleção de CTs de acordo como que foi definido no Capítulo 4.

5.2 Estudos de Caso

Os estudos de caso aqui apresentados tiveram como objetivo avaliar a qualidade dos nossos resultados, comparando o *Sim_TC*:

- (1) com o processo manual de seleção de CTs realizado por analistas/especialistas em teste (Seção 5.2.1); e
- (2) com a seleção automática realizada pela LTSBT, integrada à ferramenta TaRGeT(Seção 5.2.2). A fim de possibilitar essa comparação, utilizamos aqui uma suíte de casos de testes funcionais gerada automaticamente pela TaRGeT.

Em cada um desses experimentos, foi utilizada uma suíte diferente, bem como limiares de similaridade variados. Detalhes sobre os estudos de caso encontram-se a seguir.

5.2.1 Estudo de Caso 1 - Seleção Manual vs Seleção Automática

Dado os detalhes sobre a suíte de CTs utilizada no experimento, explanada no Capítulo 2, veremos os resultados da seleção automática, realizada pelo Sim_TC com esta suíte. A seguir, veremos uma comparação entre a seleção automática e a seleção manual de CTs a partir da mesma suíte de entrada.

Seleção de Casos de Testes pelo Sim_TC

A suíte utilizada neste experimento contém 20 Casos de Testes gerados pela TaRGeT, onde cada CT é composto por: ID do CT, um ou mais requisitos associados, Condições Iniciais, Ações, Resultado Esperado, Condições Finais.

A partir dessa suíte, o *Sim_TC* realiza seu o processo completo de seleção, como visto na seção anterior. A matriz de CTs é criada, e as similaridades entre os CTs é calculada. A seguir, a lista de pares de CTs é montada em ordem decrescente de similaridade, com um total de 190 pares (Tabela 10).

Nº do Par	Pares de CT	% Similaridade
1	[14 , 15]	94.28 %
2	[7 , 8]	91.61 %
3	[18 , 19]	91.61 %
4	[12 , 13]	91.38 %
5	[5 , 11]	90.98 %
6	[6 , 10]	90.98 %
7	[5 , 6]	89.5 %
8	[10 , 11]	89.5 %
9	[4 , 9]	86.0 %
10	[4 , 13]	86.0 %
11	[9 , 12]	84.49 %
12	[9 , 13]	84.13 %
13	[7 , 19]	83.55 %
14	[8 , 18]	83.55 %
15	[4 , 12]	83.34 %
16	[19 , 20]	83.34 %
...

Tabela 10 – Similaridade entre os Pares de Casos de Testes

A partir dessa lista, o *Sim_TC* inicia o processo de seleção utilizando apenas os pares de casos de testes com o percentual maior ou igual ao limiar definido, neste caso 90%. Como sempre, o critério de cobertura de requisito será respeitado.

O limiar utilizado aqui foi alto (90%) porque, para limiares mais baixos, a quantidade de pares de casos de testes a serem analisados aumentaria muito, tornando o processo de seleção manual mais lento. Devido a esse mesmo problema (custo de tempo do processo manual), utilizamos aqui uma suíte pequena, quando comparada à do experimento a seguir, que conta com 216 TCs.

A seleção é finalizada quando o *Sim_TC* verifica a cobertura do último par de casos de testes com o percentual de similaridade maior que o limiar definido pelo analista. Para o cenário proposto com uma suíte de 20 casos de testes e um percentual de 90% de similaridade temos como resultado uma nova suíte com 14 Casos de Testes (Figura 33).

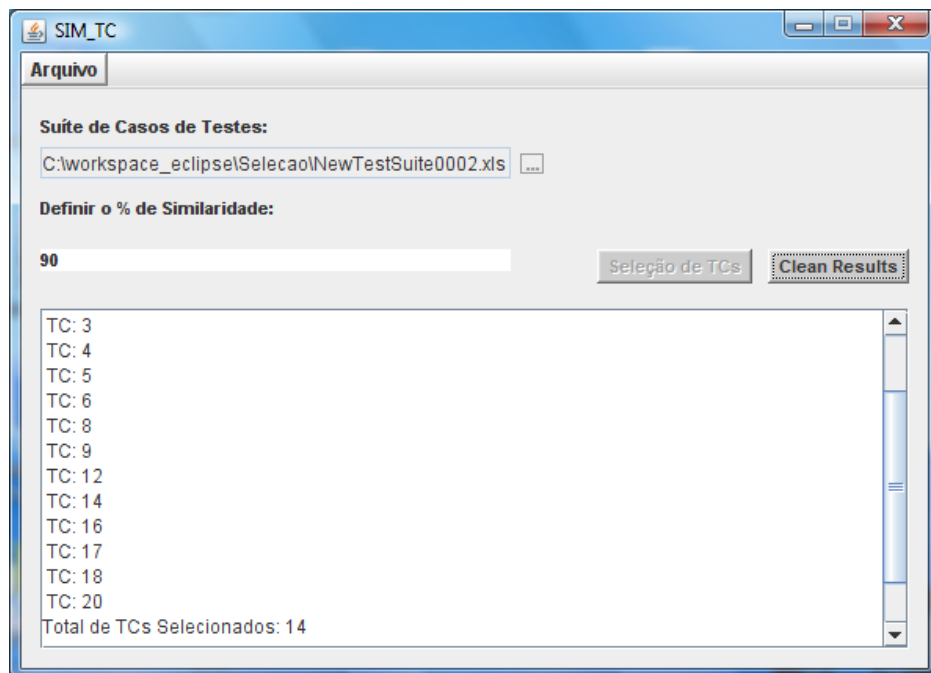


Figura 33 – Resultado da Seleção de Casos de Testes 2

Neste caso, obtivemos uma redução de 30% da suíte original (6 casos de testes), com 100% dos requisitos cobertos, pois os requisitos dos CTs eliminados estão cobertos no restante da suíte.

A fim de testar o resultado da seleção com outros limiares e verificar a cobertura de requisitos, realizamos testes com percentuais distintos (Tabela 12). Para um limiar de 85% de

similaridade textual, tivemos uma redução de 7 CTs na suíte de TCs original, selecionado 13 casos de testes para a nova suíte e garantindo todos os requisitos presentes na suíte original. Para o limiar de 80% de similaridade, tivemos uma redução de 9 CTs da suíte, selecionado 11 de um total de 20 CTs, como pode ser visto na Tabela 11.

Limiar de Similaridade	Quantidade de CTs Selecionados	Quantidade de CTs Descartados
90%	14	6
85%	13	7
80%	11	9

Tabela 11 – Percentual de redução da suíte de Casos de Testes

À medida que diminuímos o limiar de similaridade textual, o *Sim_TC* eliminará um número maior de CTs, gerando suítes menores de CTs, e garante a cobertura de todos os requisitos.

Seleção Manual vs Seleção Automática

A mesma suíte utilizada na execução do *Sim_TC* apresentada acima foi submetida a uma avaliação manual por 2 (dois) analistas de testes experientes de empresas distintas, no qual não tiveram contato entre si, para que fosse realizada a seleção manual de CTs. Cada analista recebeu a suíte de CTs original, e foi solicitado a selecionar apenas 14 desses testes. Eles deveriam eliminar os 6 CTs mais similares aos que permaneceram na suíte final, garantindo ainda que os CTs selecionados iriam cobrir os requisitos dos CTs eliminados.

Como dito, utilizamos aqui uma suíte pequena devido ao longo tempo necessário para fazer a seleção manual, e devido ao pouco tempo de que os analistas dispunham para realizar esse experimento.

A Tabela 12 traz o resultado comparativo entre a seleção manual e a seleção realizada pelo *Sim_TC*.

Colunas1	CTs Selecionados	CTs eliminados	Tempo de execução
Analista 1	1, 2, 3, 5, 7, 8, 10, 11, 12, 13, 14, 16, 18, 20	4, 6, 9, 15 , 17, 19	90 min
Analista 2	1, 2, 3, 5, 7, 8, 9, 12, 13, 15, 16, 18, 19, 20	4, 6, 10 , 11 , 14, 17	110 min
<i>Sim_TC</i> 90%	1, 2, 3, 4, 5, 6, 8, 9, 12, 14, 16, 17, 18, 20	7, 10 , 11 , 13, 15 , 19	5 seg

Tabela 12 – Seleção Automática vs Seleção Manual

Nesta tabela, podemos verificar os pontos positivos e negativos das seleções realizadas comparando-se com a seleção automática. Observamos que, entre a seleção manual e automática, houve divergências entre alguns casos de testes selecionados.

Os resultados mostram que o *Sim_TC*, com o limiar de similaridade de 90%, garantiu a cobertura de 100% dos requisitos da suíte original. Dos 14 CTs selecionados pelo protótipo, 2 CTs não foram selecionados por nenhum dos analistas, o que nos faz concluir que o resultado do *Sim_TC* : O subconjunto de CTs gerado pelo *Sim_TC* equivale a 85% das duas suítes geradas pelos dois analistas, ou seja, dos 14 CTs selecionados pelo *Sim_TC*, 12 CTs estão contidos nas suítes selecionadas pelos analistas.

É notório que os dois analistas escolheram 3 CTs iguais para a eliminação { 4, 6, 17}. Para o CT6 eliminado pelos analistas, o *Sim_TC* optou pelo CT7 por seleção aleatória, pois ambos possuem os mesmos requisitos. Em relação aos CTs 4 e 17, o *Sim_TC* não os analisou devido ao fato da similaridade de texto deles estarem abaixo do limiar de 90% selecionado para o experimento.

Devido a alta similaridade de texto entre os CTs, a seleção manual realizada pelos dois analistas eliminou 3 CTs iguais, no total de 6, e também garantiu a cobertura de requisito da suíte original. Entretanto, sua execução gastou um tempo de 90 minutos e de 110 minutos, para o analista 1 e analista 2, respectivamente, além de requerer que esses analistas tivessem um conhecimento prévio de testes de software e requisito de software. Para a mesma seleção, o *Sim_TC* gastou apenas 5 segundos, e também garantiu a presença de todos os requisitos.

Caso usássemos um limiar de 80% de similaridade textual usando o *Sim_TC*, teríamos a seleção de 11 CTs, conforme Tabela 12. Entretanto, a realização da seleção manual usando o mesmo limiar de similaridade textual, obrigaria os analistas a realizar a eliminação de um número maior de CTs, o que seria passivo de erros na garantia de cobertura de requisitos apenas lendo os CTs.

À medida que o analista deseja gerar um subconjunto de CTs menor da suíte original, através da seleção manual, é necessário um esforço (tempo) maior, além de um conhecimento prévio de testes e experiência na realização da atividade.

5.2.4 Estudo de Caso 2 - *Sim_TC* vs LTSBT

Neste último experimento, realizamos uma comparação entre o *Sim_TC* e o LTSBT, esta última já integrada à TaRGeT. Para isto, uma nova suíte de casos de testes foi gerada pela TaRGeT a partir do mesmo caso de uso, contendo 216 casos de testes associados a 54 requisitos. Foi possível utilizar aqui uma suíte bem maior do que a do experimento anterior, uma vez que os dois processos deste estudo de caso são automáticos. Assim sendo, o tamanho da suíte não inviabiliza o experimento (como seria o caso no estudo anterior –seleção manual). Com uma suíte maior, foi possível obtermos resultados estatisticamente mais confiáveis.

Para enriquecer este estudo de caso, usamos vários limiares de similaridade de caminho no LTSBT, e de similaridade de texto no *Sim_TC*.

Podemos observar na Tabela 13 que, à medida que diminuímos o limiar de similaridade, as suítes geradas passam a conter um número menor de CTs. Entretanto, o *Sim_TC* permanece com a mesma quantidade de requisitos cobertos pela suíte original. A menor suíte que o *Sim_TC* pode gerar neste caso é com 119 CTs, abaixo disto, seria impossível garantir todos os requisitos presente na suíte original. Enquanto que o LTS-BT deixa de garantir requisitos nas suítes geradas à medida que diminuímos o limiar de similaridade de caminho.

Se caso o analista deseje selecionar uma suíte com 100% de cobertura de requisitos e de tamanho mínimo possível, basta informar o limiar de similaridade igual a 0%.

Limiar de Similaridade	Sim_TC		LTS-BT	
	Quantidade de CTs selecionados	Requisitos Cobertos	Quantidade de CTs selecionados	Requisitos Cobertos
90%	164	54	194	54
75%	130	54	162	54
55%	122	54	119	52
25%	120	54	54	48
10%	119	54	22	27

Tabela 13 – Suítes geradas pelo *Sim_TC* vs *LTS-BT* e quantidade de requisitos cobertos por cada uma.

A comparação entre os dois processos de seleção se baseou na observação do subconjunto de CTs gerado por cada processo. Foi verificada a cobertura de requisitos de cada processo para diferentes tamanhos da suíte de saída.

Se observarmos o resultado do *Sim_TC* e LTSBT com base na quantidade de CTs da suíte gerada após a seleção (por exemplo, um conjunto de 119 CTs), veremos que a suíte selecionada pelo *Sim_TC* permanece com a mesma quantidade de requisitos original, enquanto o LTSBT deixa de cobrir 2 (dois) requisitos, conforme mostra a tabela 14.

Sim_TC vs LTSBT		
Ferramentas	CTs selecionados	Requisitos Cobertos
Sim_TC	119	54
LTSBT	119	52

Tabela 14 – Resultado de tempo entre *Sim_TC* e LTSBT

Como visto, o *Sim_TC*, independente do limiar definido pelo analista, garante em 100% que todos os requisitos contidos na suíte original de CTs estará representado por pelo menos um CTs do subconjunto de CTs gerado.

O problema da nossa abordagem é que, se houver pouca redundância na suíte original, a suíte final não terá seu tamanho muito reduzido. Assim sendo, concluímos que a melhor escolha, entre o *Sim_TC* e o LTSBT, depende da necessidade do testador no momento. Se o testador necessita realmente reduzir muito sua suíte, mesmo sabendo que haverá perda de cobertura de requisitos, ele deve optar pelo uso do LTSBT. Caso o testador não possa perder cobertura, ele deve optar pelo *Sim_TC*.

5.3 Considerações Finais

Neste capítulo apresentamos testes e exemplos de seleção automática de CTs usando *Sim_TC*, e dois estudos de casos comparando seu desempenho com a execução manual, e com a ferramenta LTSBT.

O teste do sistema realizou a seleção de casos de testes a partir da suíte *feature* de *Phonebook* onde foi possível testar as condições presentes no algoritmo. Foi possível observar o funcionamento de todo o processo de seleção, como a criação da matriz de CTs, a lista de pares de CTs e seus respectivos percentuais de similaridade, o critério utilizado para a seleção – cobertura de requisitos –, e o novo conjunto de casos de testes gerado pela seleção

automática. Vimos também o funcionamento do *CliqueFinder* quando não há 100% de cobertura entre os requisitos dos pares de CTs analisados.

O primeiro estudo de caso revelou resultado 80% similar da seleção manual versus a automática. Contudo, é preciso considerar os altos custos de tempo e recursos do processo de seleção manual realizada por analistas de testes (que devem ter experiência e conhecimento especializado para realizar o trabalho). Além disso, o processo manual é passível de erro.

Por fim, comparamos o Sim_TC com o LTSBT. Enquanto em todas as suítes geradas pela seleção automática do *Sim_TC* cobriam 100% dos requisitos da suíte original, o LTSBT perde a cobertura à medida em que o limiar de similaridade diminui, deixando de cobrir alguns requisitos.

6 Conclusão

Este trabalho de mestrado teve como foco principal desenvolver um método de seleção automática de casos de testes baseado em similaridade de texto e cobertura de requisito – o *Sim_TC*.

A partir dos estudos realizados da literatura relacionada ao tema, verificamos uma lacuna onde poderíamos dar uma boa contribuição para as pesquisas na área: não encontramos nenhum trabalho de seleção de CTs baseado em similaridade textual, e que preserve a cobertura de requisitos da suíte original.

O *Sim_TC* recebe como entrada uma suíte de CTs a ser reduzida, e o limiar de similaridade de texto definido pelo analista de teste. Todos os pares de teste com similaridade igual ou maior à estipulada pelo analista serão considerados para seleção. Será cortado o teste que cobrir menos requisitos.

Quando não há 100% de cobertura de requisitos entre os CTs do par sendo analisado, o sistema busca um terceiro CT na suíte que auxilie na escolha do CT a ser eliminado, de maneira que o terceiro CT vai cobrir os requisitos do par junto com o CT que será selecionado para permanecer na suíte. Esta técnica se mostrou bastante eficaz, satisfazendo nossos propósitos.

Os experimentos apresentados no Capítulo 5 demonstram o correto funcionamento do *Sim_TC*, bem como suas vantagens e desvantagens em relação a técnica de seleção manual, e uma técnica de seleção automática implementada na ferramenta TaRGeT.

A presente pesquisa, apesar de apresentar resultados satisfatórios, apontou que ainda existe a necessidade de aprofundamento de estudos acerca da técnica de seleção de CTs baseada em similaridade de texto. A Seção 6.1 ressalta as principais contribuições deste trabalho, e a Seção 6.2 traz indicações de trabalhos futuros visando dar continuidade a esta pesquisa, bem como superar as limitações da nossa abordagem.

6.1 Principais Contribuições

As principais contribuições deste trabalho são:

- Estudo aprofundado da literatura relacionada a Seleção de Casos de Teste;
- Utilização da técnica de Distância de Edição (Distância de Levenshtein) como base para o algoritmo de seleção de casos de testes;
- Criação de um sistema protótipo para seleção de casos de testes que utiliza a similaridade de texto para corte, porém preservando a cobertura de requisito;
- Utilização do *CliqueFinder* para auxiliar na escolha do CT a ser eliminado, quando não há 100% de cobertura dos requisitos entre os testes do par sendo analisado;
- Realização de experimentos comparando o *Sim_TC* à técnica manual de seleção, e a uma técnica automática implementada dentro da TaRGeT. Em ambos os casos, o *Sim_TC* apresentou algumas vantagens desejáveis à atividade de seleção de CTs.

6.2 Trabalhos Futuros

A partir de uma análise deste trabalho, apresentamos abaixo algumas propostas para dar continuidade ao projeto:

Diversificação dos Formatos dos Dados de Entrada

Atualmente, os dados de entrada do *Sim_TC* são originados das suítes de casos de testes geradas pela TaRGeT. Essas suítes seguem um *template* rígido.

O objetivo deste trabalho futuro é dar mais liberdade ao formato de entrada de dados no *Sim_TC*, de modo que ele seja capaz de receber CTs definidos em outros tipos de dados e padrões mais variados (como xml e outros formatos). Assim sendo, será possível utilizá-lo não apenas acoplado à TaRGeT, mas também como ferramenta *standalone*.

Inclusão de novos critérios de seleção no Sim_TC

Analisando o processo de seleção, observamos que existem situações em que os dois critérios para a realização da seleção de CTs (similaridade textual e cobertura de requisitos) não são suficientes para garantir uma boa escolha. Nos casos de pares de CTs que apresentam o percentual de similaridade de texto maior que o definido pelo analista, e cobrem os mesmos requisitos, o *Sim_TC* realiza uma escolha aleatória. Seria interessante para esse caso que houvesse um terceiro critério para realizar a seleção, como a técnica de seleção por esforço de execução [8] e priorização de casos de testes [26].

Permitir o usuário definir os requisitos da suíte selecionada

O sistema permitirá que o analista defina quais os requisitos que deverão estar presentes no subconjunto da suíte gerada pela ferramenta, ao invés de sempre tentar 100% de cobertura de requisitos. Deste modo, o analista definirá o limiar de similaridade de texto e os requisitos do quais ele deseja para realizar a seleção de casos de testes.

Comparar com técnicas de redução de suíte

Comparar a seleção automática de casos de testes realizados pelo *Sim_TC* com ferramentas e técnicas de redução de suíte de casos de testes. O objetivo é verificar a qualidade do resultado entre as ferramentas.

Integração do Sim_TC com a TaRGeT

Atualmente, o *Sim_TC* recebe como entrada suítes de CTs seguindo o *template* da TaRGeT, além de usar parte do algoritmo de similaridade de texto implementado nessa ferramenta. Contudo, resta ainda incorporar esse sistema à ferramenta geral, como acontece com o LTSBT, oferecendo aos seus usuários da TaRGeT mais uma opção de seleção de casos de testes.

Utilização de Clustering para grandes suítes

Para tratar suítes de casos de testes muito extensas, o processo de seleção aqui descrito pode não ser muito eficaz. Neste caso, vislumbramos o uso de um algoritmo de *Clustering* [42, 43], para gerar um subconjunto de CTs semelhantes, aplicar o *Sim_TC* a esses subconjuntos. O critério de similaridade aqui seria de cobertura de requisitos, e o *Sim_TC* então permaneceria usando a similaridade textual.

Realização de Novos Estudos de Casos (Avaliação da Seleção)

Os dois estudos de casos apresentados nos mostraram o resultado da aplicação da ferramenta em situações reais na realização da seleção de casos de testes. Entretanto, precisamos submeter o *Sim_TC* a suítes com características diferentes das utilizadas aqui – por exemplo, suítes muito extensas, ou com baixo numero de redundâncias.

É necessário verificar também a eficácia da nossa técnica em relação às outras alternativas. Isto é, é preciso verificar se o *Sim_TC* é capaz de selecionar testes que de fato detectam mais falhas e erros no software sendo testado, em relação às outras técnicas de seleção de CTs.

Essas direções de trabalhos futuros permitirão que novos estudos sejam realizados a partir do trabalho aqui apresentado.

7 Referências

- [1] DALAL, S. R.; KARUNANITHI; JAIN, A.; LEATON, N.; LOTT, C. M.; PATTON, G. C.; HOROWITZ, B. M.. Model-Based Testing in Practice. Proceedings of the ICSE'99, p. 285-294, 1999.
- [2] GIMENES, M. S.; WEIS, G. M.; HUZITA, E. H. M.. Um Padrão para Definição de um Gerenciador de Processos de Software. Proceedings of the 2nd Workshop IberoAmericano de Engenharia de Requisitos Y Ambientes Software. San Jose, Costa Rica, Ideas'99 Memorias, 1999, vol. 1, San Jose: Instituto Tecnológico de Costa Rica, 1999. Disponível em: <http://www.larces.uece.br/~jeff/padroes/v2.pdf>. Acessado em: 01/08/2010.
- [3] BEIZER, B.. Black-Box Testing: Techniques for Functional Testing of Software and Systems. John Wiley & Sons, 1995.
- [4] BRIAND, L., and LABICHE, Y.. A UML-based approach to system testing. Software and Systems Modeling. SpringerLink, vol. 1 (1), pp. 10-42, 2002.
- [5] CRAIG, R. D. and JASKIEL, S. P.. Systematic Software Testing. Boston: Artech House, vol. 35, p. 81, 2002.
- [6] LINZHANG, W., JIESONG, Y., XIAOFENG, Y., XUADONG, L. AND GUOLIANG, Z.. Generating Test Cases from UML Activity Diagram based on Gray-Box Method. In: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), p. 284-291, 2004.
- [7] BORBA, Paulo; TORRES, Dante; MARQUES, Rafael; WETZEL, Luiz. TaRGeT: Test and Requirements Generation Tool. In Motorola's 2007 Innovation Conference, Software Expo Session, Lombard, Illinois, USA, October 5, 2007.
- [8] Souza, L.S.; PRUDENCIO, R. B. C. ; BARROS, F. A.. A Constrained Particle Swarm Optimization Approach for Test Case Selection. In: The 22nd International Conference on Software Engineering and Knowledge Engineering, 2010, Redwood City - San Francisco. Proc of the 22nd International Conference on Software Engineering and Knowledge Engineering (to appear), v. 1. p. 1-6, 2010.
- [9] GRAHAM, Dorothy; VEENENDAAL, Erik Van; EVANS, Isabe; BLACK, Rex. Foundations of Software Testing. London-UK: Thompson, p. 5-68, 2007.
- [10] SOMMERVILLE, IAN. Engenharia de Software. 8ª edição. São Paulo: Pearson Addison – Wesley, p. 42-56; 355-371, 2007.
- [11] MYERS, Glenford J., WILEY, John & SONS,. The Art of Software Testing. 2, Nova Jérsei, 2004.
- [12] LEWIS, William E. Software Testing and Continuous Quality Improvement. Second Edition. Auerbach Publications, 2005.
- [13] PRESSMAN, Roger S. Software Engineering - A practitioners Approach. IE-Mcgraw-Hill, 2001.

- [14] CARTAXO, E.; MACHADO, P.; NETO, F.; OURIQUES, J.. Test Case Selection Using Similarity Function. Workshop on Model-based Testing in 37th Annual Congress of the Gesellschaft fuer Informatik. Bremen: Germany, September, p. 381-386, 2007.
- [15] JORGENSEN, P. C. Software Testing: A Craftsman's Approach. 2 ed. CRC Press, 2002.
- [16] Geração Automática de Casos de Teste CSP Guiada por Propósitos. Disponível em: <http://www.citeulike.org/group/3126/article/220766>. Acessado em 20/10/2010.
- [17] The Institute of Electrical and Electronics Engineers. IEEE Std 829: Standard for Software Test Documentation. New York: IEEE Computer Society, September, 1998.
- [18] XIE, TAO; NOTKIN, D. Tool-assisted unit test selection based on operational violations. Automated Software Engineering. Proceedings. 18th IEEE International Conference on. Dept. of Comput. Sci. & Eng., Washington Univ., Seattle, WA, USA, p. 40-48, 2003.
- [19] CARTAXO, E.; ANDRADE, W.; NETO, F.; MACHADO P.. LTSBT: A Tool to Generate and Select Functional Test Cases for Embedded Systems. *23rd Annual ACM Symposium on Applied Computing (SAC'2008)*, p.1540-1544, New York, NY, USA, 2008.
- [20] HORI, Érica A. A.. UCSCNL – A Controlled Natural Language for Use Case Specifications. 2010. Dissertação (Mestrado em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco.
- [21] OSTRAND, Thomas J. and BALCER, Marc J. The category-partition method for specifying and generating functional tests. Published in: Magazine Communications of the ACM. vol. 31, ACM New York, NY, USA, pp. 676-686, June, 1988.
- [22] AMARAL, Ana Silvia M. S. do; VIJAYKUMAR, N. L. & MARTINS, Eliane. Geração Automática de Casos de Teste de Conformidade para Software de Aplicações em Protocolos de Comunicação. Disponível em: <http://mtcm18.sid.inpe.br/col/lac.inpe.br/worcap/2003/10.29.11.18/doc/finalll2003.pdf> Acessado em: 21/10/2010.
- [23] GOULART, Marli Silva & MACHADO, Rodrigo Prestes. Uma Ferramenta para Geração de Casos de Testes para Web Services. Disponível em: <http://prestesmachado.com.br/artigos/MarliGoulart.pdf>. Acessado em 26/10/2010.
- [24] FEIJIS, L. M. G.; GOGA, N.; MAUW, S. & TRETSMANS, J.. Test Selection, Trace Distance and Heuristics. Testing of Communicating Systems XIV. Kluwer Academic Publishers, p.267-282, 2002.
- [25] MEDEIROS, Aline; COSTA, Cleyverson & VASCONCELOS, Alexandre. Método de Seleção de Casos de Teste de Regressão Baseado em Risco. Disponível em: <http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/artigos/pos/spg1.pdf>. Acessado em 20/10/2010.
- [26] ELBAUM, Sebastian; MALISHEVSKY, Alexey G. and ROTHERMEL, Gregg. Test Case Prioritization: A Family of Empirical Studies. IEEE Trans. on Software Engineering, vol. 28, no. 2, p.159-182, feb. 2002.

- [27] ARANHA, E.; BORBA, P. Estimation Model for Test Execution Effort. *International Symposium on Empirical Software Engineering and Measurement (ESEM)* Madri, Espanha, p. 107-116, Setembro, 2007.
- [28] ALBERTINS, Lucas; ARANHA, Eduardo; IYODA, Juliano and SAMPAIO, Augusto. Test Case Prioritization based on Data Reuse: An Experimental Study. In the Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement. Lake Buena Vista, Florida, USA, p. 279-280, October, 2009.
- [29] JONES, James A.; HARROLD, Mary J.. Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage. IEEE Computer Society Washington, DC, USA, pp. 195-209, 2003.
- [30] ABRAN, Alain; MOORE, James W. Guide to the Software Engineering Body of Knowledge (SWEBOK)' Tech. Rep., Joint IEEE-ACM Software, 2004. Disponível em: www.swebok.org. Acessado em: 21/10/2010.
- [31] LIN, J. and YEH, P.. Automatic test data generation for path testing using GAs. *Information Sciences*, 131(1-4):P. 47–64, 2001.
- [32] NAVARRO, G. A guided tour to approximate string matching. *ACM Computing Surveys*, vol. 33, no. 1, pp. 32-88, Mar. 2001.
- [33] WAGNER R. A.; FISCHER, M. J.. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.
- [34] WEN, J.-R.; NIE, J.-Y., & ZHANG, H.-J.. Clustering user queries of a search engine. In *Proceedings of the 10th International Conference on World Wide Web*, p. 162–168, New York, USA, 2001.
- [35] KRUSKAL, J. B.. An overview of sequence comparison: time warps, string edits, and macromolecules. *SIAM Review*, 25(2):201–237, 1983.
- [36] Search clusty. Disponível em: <http://clusty.com/>. Acessado em: 15/12/2009.
- [37] CLARKE, W.A.; FERREIRA, H.C. Further results on insertion/deletion correcting codes. Rand Afrikaans Univ., Aucklandpark, South Africa-This paper appears in: *Information Theory, International Symposium on Issue*, p. 373, 2002.
- [38] REN, F.; DEGEN, Huang. Extended super function based Chinese Japanese machine translation Xiao Sun. Dept. of Inf. Sci. & Intell. Syst., Tokushima Univ., Tokushima, Japan This paper appears in: *Natural Language Processing and Knowledge Engineering, International Conference on Issue*, p. 1-8, 2009.
- [39] YUJIAN, LI; BO, Liu. A Normalized Levenshtein Distance Metric. This paper appears in: *Pattern Analysis and Machine Intelligence*, p. 1091-1095, IEEE Transactions on Issue Date: June 2007.
- [40] Kick Java. Disponível em: <http://kickjava.com/src/org/jgrapht/alg/BronKerboschCliqueFinder.java.htm>. Acessado em: 13/07/2010.

- [41] BronKerboschCliqueFinder. Disponível em: <http://www.jgrapht.org/javadoc/org/jgrapht/alg/BronKerboschCliqueFinder.html>. Acessado em: 13/07/2010.
- [42] MCQUEEN, J.. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 281–297, 1967.
- [43] KAUFMAN L.; ROUSSEEUW, P.J.. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.
- [44] SAMUDRALA R., MOULT J.. A Graph-theoretic Algorithm for comparative Modeling of Protein Structure; *J.Mol. Biol.* pp. 287-302, 1998.
- [45] The Institute of Electrical and Electronics Engineers. IEEE Standard Glossary of Software Engineering Terminology. Disponível em: <http://ieeexplore.ieee.org/servlet/opac?punumber=2238>. IEEE Computer Society More Year:1990, Date of Current Version: 06/08/2002.
- [46] TaRGeT Requirements Specification. Disponível em: http://twiki.cin.ufpe.br/twiki/pub/TestProductLines/TaRGeTProductLine/TaRGeT_Requirements_Specification.pdf Acessado em: 13/12/2010.
- [47] BRON, C. & KERBOSCH, J.. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16, 575-577, 1973.
- [48] KENNEDY, J.; EBERHART, R. C.. Particle Swarm Optimization. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 1942–1948, 1995.
- [49] KENNEDY, J.; EBERHART, R. C.. A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, pp 4104–4109, 1997.
- [50] V. LEVENSHTAIN. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707-710. Original in Russian in *Doklady Akademii Nauk SSSR*, 163(4): 845-848, 1965.
- [51] MORGAN, H.L. Spelling correction in systems programs. *Comm. ACM* 13, 2, pp 90-94. 1970.
- [52] V. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1:8-17, 1965.