A TOOL FOR OPTIMAL MANUAL TEST CASES SELECTION

Lorena do Carmo Caldas (Instituto Federal da Bahia, Bahia, Brasil) – llore.caldas@gmail.com
Antônio M. da S. Pitangueira (Instituto Federal da Bahia, Bahia, Brasil) – antoniomauricio@ifba.edu.br

This paper discuss the development of a web tool called TSuite, created to compose a candidate to solving NP-Complete test cases selection problem, by evaluating its importance and unit costs. The selection supports conditions: time, priority and complexity of the execution test scenarios, considering the time available in the test project, as well as the impact of inserting the new execution test plan within project. Thus, TSuite automates the task of choosing regression test situations, reducing its cost of running and expanding the variety of complex situations, which leads to increased test coverage about the product, the set of selected units. The Tsuite uses a hybrid approach in its structure which involves a specific method and other optimization of Search Based Software Engineering.

Keywords: Software test, selection test cases, optimization, test project, regression test.

UMA FERRAMENTA PARA SELECIONAR CASOS DE TESTE MANUAIS DE FORMA ÓTIMA

Este trabalho aborda o desenvolvimento de uma ferramenta web chamada TSuite, criada para compor uma solução candidata à resolução do problema NP-Completo da seleção de casos de teste, através da avaliação de suas importâncias e custos unitários. A seleção apoia as condições de: tempo, prioridade e complexidade dos cenários de execução do teste, considerando o tempo disponível no projeto de teste, assim como o impacto da inserção do novo plano de execução de teste dentro deste projeto. Dessa forma, o TSuite automatiza a tarefa de escolher situações para teste de regressão, reduzindo seu custo de execução e ampliando a variedade de complexidade das situações, o que conduz ao aumento da cobertura de teste sobre o produto, no conjunto de unidades selecionadas. O TSuite utiliza a abordagem híbrida em sua estrutura o que envolve um método específico e outro de otimização da Engenharia de *Software* Baseada em Buscas.

Palavras-chave: Teste de *software*, seleção de casos, otimização, projeto de teste, teste de regressão.

1. INTRODUÇÃO

Projetos de *software* compreendem atividades voltadas à construção das funcionalidades de um sistema. Dentre os processos que compõem o seu desenvolvimento está o teste de *software*.

As tarefas de teste correspondem aos esforços para controlar a qualidade do produto final entregue ao cliente, o que não implica dizer que atribui soluções ao conjunto (CRISTALLI et. al, 2007). Porém certificam que os critérios contratados estão inclusos no *software* e são acionados corretamente por um ator.

Na computação, a área de Engenharia de *Software* normatiza os aspectos do desenvolvimento de *software*, (FREITAS et. al, 2009). Processos e métodos são definidos como meio de manter um padrão de eficiência e qualidade do *software*. Muitas de suas vertentes utilizam técnicas de otimização para compor tais soluções.

A otimização auxilia na busca de melhores resultados, dado um problema que não possa ser facilmente solucionado diretamente por métodos convencionais, (LAI et. al., 2012) apud (HARMAN e MCMINN, 2010).

O teste de *software* contém atividades que possuem um alto valor de execução, apesar de ter um maior valor agregado, se comparados. Para atestar qualidade de um produto o teste de *software* deve atuar sobre a linha de riscos e defeitos de um projeto de sistema (CRISTALLI et. al, 2007). Essa atividade permeia todas as fases de desenvolvimento e pode despender períodos elevados de operação entre planejamento, execução e entrega. A mínima tentativa de reduzir o tempo de abordagem e alcance dos problemas procurados na ferramenta conduz à minimização da receita de teste e consequente benefício ao projeto de *software*.

Para que o *software* possa ser devidamente testado é necessário observar as condições de operação dele além das ações que poderão ser executadas. Isso para que seja possível analisar a maior parte das funcionalidades envolvidas em seu contexto. A cada rodada de execução dos testes sobre a ferramenta os objetivos e cenários podem variar, fazendo-se necessário encontrar e criar novas situações de teste. Essa tarefa é demorada porque é necessário percorrer todas as funções do sistema de forma a encontrar o maior número de casos que se encaixem no objetivo atual da operação (BARTIÉ, 2002).

O TSuite foi criado com intuito de diminuir os custos da execução da tarefa de seleção de casos de teste considerando cenários de operações que contenham recursos escassos disponíveis e grande volume de dados de teste manuais. Antes de sua criação, seria necessário escolher cada uma das situações manualmente. Tarefa essa dispendiosa, por conta da quantidade de tempo escalada. Essa redução é alcançada porque o TSuite realiza essa função automaticamente.

Por ser uma ferramenta de suporte à tomada de decisão, ela é indicada aos executores de teste que necessitem planejar como testar novamente o *software*, reduzindo o projeto a um conjunto prioritário de unidades de execução de teste. Sua utilização é adequada aos projetos de teste cadastrados na ferramenta de gerência: TestLink. O TestLink é uma ferramenta gratuita e *open source*, mantida pela comunidade de *software* livre Teamst, que oferece os serviços de cadastro, manutenção e suporte à execução de projetos de teste, (TEAMST, 2012).

Assim, ao usar o TSuite o executor poderá listar as situações de teste mais próximas às condições atuais dos recursos disponíveis ao projeto. Ao informar um arquivo contendo o projeto de teste e seus parâmetros obrigatórios, a ferramenta gera um conjunto reduzido das unidades de teste contendo somente as situações prioritárias.

Testes foram realizados para validar os benefícios que o TSuite trás e comprovou-se que ele reduz o tempo de execução da tarefa de seleção manual em aproximadamente 50%. Também,

provou-se que a utilização do TSuite independe do conhecimento total do sistema ou experiência do responsável pela execução da seleção dos casos do teste. Além disso, o TSuite aumenta a quantidade de casos selecionados em até 6,9% porque maximiza a utilização dos recursos disponíveis, por conta da justiça atribuída aos casos de complexidade baixa, com a aplicação do método NSGAII na seleção complementar dos casos. No conjunto das unidades de teste selecionadas, aquelas que possuam complexidade baixa tem sua quantidade ampliada em 16,7%, em relação ao método de solução do problema da Mochila e 55,5%, se o método Aleatório for utilizado, em comparação com o método NSGAII.

Neste trabalho, a seção Teste de *Software* trás definições do processo e aborda características que o envolve. A seção Trabalhos Correlatos trás a base e comparação da proposta aqui definida em relação a trabalhos já consolidados. A seção O Método de Seleção descreve o fluxo de execução do TSuite e explica o escopo de sua função de seleção. A seção Testes e Resultados contém as hipóteses e resultados da validação da utilidade da ferramenta. Os tópicos descritos anteriormente estão dispostos respectivamente nesta ordem e são encontrados a seguir.

2. TESTE DE *SOFTWARE*

De acordo com Bartié (2002), teste de *software* é um processo que estrutura passos, após planejá-los, com a intenção de identificar defeitos e falhas no produto. Dito isto, testar é uma atividade onde um sistema ou um componente é verificado múltiplas vezes sob condições objetivas, de forma a gerar resultados específicos. O critério de término do teste é analisado com base no resultado de sucesso ou falha da execução dos cenários de teste.

O teste de *software* vem sendo utilizado com frequência nos projetos de *software* como uma forma de elevar o controle sobre a qualidade do produto disponibilizado ao cliente, (FARZAT e BARROS, 2010). As atividades de teste de *software* costumam consumir grande custo do projeto. Suas execuções demandam tempo e maturidade do processo de desenvolvimento. Na maioria das empresas que produzem *software*, ela faz parte do processo de garantia de qualidade (CRISTALLI et. al, 2007).

Segundo Bartié (2002), a etapa de testes, dentro do esquema citado acima, pode chegar a comprometer 40% dos gastos de todo o processo de desenvolvimento do *software*. Esse valor é atribuído ao tempo gasto na execução da fase de teste, necessário à análise e maturação do sistema, decorrente da correção dos *bugs*. Qualquer forma de redução do escopo das tarefas de teste, sem gerar prejuízos ao processo de qualidade estabelecido, é válida.

De acordo com essa proposta, o TSuite atua na redução de esforços da tarefa de planejamento da fase de execução do teste de regressão, além de propiciar a escolha coerente ao alvo atual do projeto, considerando suas funcionalidades e recursos disponíveis para as executar.

Dado um projeto de teste que contenha diversas execuções passadas, não será mais necessário ao gestor analisar todo o sistema para selecionar manualmente as situações de execução que se enquadrem ao escopo das funcionalidades modificadas e associadas. Com isso, o custo despendido (gasto) na execução da tarefa de escolha de situações de teste sofre redução, já que o executor poderá direcionar esforços a outras atividades.

A seleção manual das condições de teste tem interferência negativa em dois pontos principais. O primeiro é o esforço despendido, algo que está diretamente ligado ao tempo proposto ao exercício da tarefa. Ao considerar o tamanho do projeto, essa é uma atividade que pode levar de poucos minutos a muitas horas de execução. O segundo, que também influencia no

tempo de execução, é a base de escolha das condições. Os critérios da seleção geralmente estão apoiados no conhecimento de regra de negócio do projeto, por parte de seu executor, (FREITAS et. al, 2009).

Outro fator que também pode ser considerado na abordagem do problema aqui relatado é a dificuldade em escolher as situações de acordo com a combinação dos parâmetros dos dados de teste. O cruzamento manual entre todos os critérios de escolha dos casos é uma atividade que demanda muito esforço, algo que é condicionado inclusive pelas características de: entendimento das regras de negócio e experiência profissional do executor. Ainda que existam métodos de filtragem de conteúdo, nas ferramentas utilizadas para manutenção dos dados, sempre será realizado um levantamento incompleto, no sentido de prioridade e diversidade do resultado retornado para a consulta. Ainda considerando esse cenário, a dificuldade aumentaria caso o projeto utilizasse somente o modelo de planilhas para a criação e persistência dos dados.

De acordo com os fatores descritos, o objetivo principal deste trabalho é a criação de uma ferramenta que automatize a tarefa de seleção dos casos de teste para auxiliar na resolução do problema de busca que considere múltiplos objetivos. Isso porque, existe a necessidade de reduzir o tempo de execução da tarefa de seleção de casos de teste, que é dispendiosa, sem que ocorra desprezo da importância dos cenários prioritários ao projeto e da cobertura desses casos por complexidade. Além disso, sua função também é retirar do executor da atividade a necessidade de ter experiência com a execução da tarefa de seleção e a imposição de conhecimento total das regras de negócio do sistema/projeto, porque essas condições concentram em poucos indivíduos a responsabilidade e habilidade de selecionar os casos para a regressão. Sendo assim, o TSuite forma uma base de conhecimento, no momento da tomada de decisão, que influencia diretamente na qualidade do produto.

Confrontando com o método de seleção manual, os principais benefícios em utilizar o TSuite são: minimização do tempo e consequente custo de execução da tarefa, independência do recurso pessoal utilizado na gerência e seleção das situações coerentes com a disponibilidade dos recursos do projeto de teste e funções do sistema.

O custo da qualidade é um fator essencial no projeto de *software* e impacta diretamente na correta execução das tarefas enquadradas na fase de teste. Segundo Myers (2004) e Black (2002) quanto mais cedo ocorrer a descoberta e a correção do defeito no *software*, menos custo ele absorve do projeto. A reflexão que as teorias trazem é de que a prevenção ao erro no projeto e no código deve ocorrer antes da entrega dele ao cliente. Isso para que ele não permaneça no *software* como defeito e venha a ser manifesto no sistema como falha a um usuário, gerando prejuízo e desgaste exacerbados ao projeto.

Considerando isso, o TSuite ajuda a reduzir os custos do planejamento de execução das condições de teste, já que a tarefa de seleção é executada automaticamente. A consequência dessa ação confere em um recurso pessoal disponível para realizar outras atividades.

Apesar do custo de teste ser elevado e por isso condicionar o projeto de teste, outro fator do projeto de *software* que influencia diretamente na qualidade do produto, é a ponderação dos riscos ao projeto, levantados de acordo com a análise de risco. Essa análise considera os fatos que possam causar perdas ao projeto e que por isso necessitam ter suporte.

A execução de teste de *software* tem como função atuar sobre a linha de riscos do *software*. O executor dos testes verifica o comportamento relacionado a cada tipo de situação a que o sistema é submetido. Essa verificação é realizada através da avaliação das regras de negócio e do ambiente de operação do aplicativo.

2.1. Produtos de Teste

Assim como na Computação existem os conceitos de hardware e *software*, surgiu a denominação *testware*, na Engenharia de *Software*. Segundo (BARTIÉ, 2002), *testware* são todos os elementos produzidos por profissionais de teste na etapa de verificação e validação do *software*. A validação corresponde à etapa de estudo e avaliação da regra de negócio modelada ao sistema. Já a verificação referencia a execução do teste de *software*. São exemplos de unidades de *testware*: plano do projeto de teste, *checklists* e *scripts* de execução de teste.

No TSuite, o plano de execução de teste tem seu conceito aplicado apenas de forma representativa, porque a seleção deve ocorrer antes de os casos serem executados. A evidência de uma ou mais suítes, como referência ao modelo atual do sistema, o torna implícito.

Os casos de teste são o *testware* que correspondem aos *scripts* de execução do teste, (IEEE, 2008). Tais *scripts* são a estrutura que descrevem o passo a passo do teste. Por exemplo, considerando os testes funcionais manuais eles são documentos que contém ações enumeradas em ordem linear de raciocínio e execução e resultados esperados relacionados à elas.

O TSuite não é uma ferramenta para criar *scprits* automatizados de condições de testes. No entanto ele visa automatizar a tarefa de selecionar os casos de teste manuais derivados de uma fonte de dados que contenha essas unidades de teste.

Conforme Farzat e Barros (2010) os casos de teste podem ser positivos ou negativos. Os casos de teste positivos cobrem as funcionalidades do sistema realizando operações que levem a certificar em saídas corretas. Ou seja, eles confirmam que o *software* faz o que realmente deveria fazer. Já os casos de teste negativos visam demonstrar que o sistema não faz o que não deveria fazer.

Para Cristalli et. al (2007), as suítes de teste são conjuntos de casos de teste. Esses agrupamentos podem ser criados para identificar módulos do *software* ou "versionamento" dos casos de teste. No segundo caso, eles ganham uma *baseline* própria. As *baselines* contém as suítes de execução de uma iteração do *software*. Para a maioria das ferramentas de gerenciamento de testes, elas fazem parte de um plano de execução de teste.

A figura 1 exibe o formato de um projeto de teste quando ele não está passando por nenhuma execução. Essa é também a forma com que as ferramentas de gerência de *testware* armazenam seus dados, já que seguindo este formato vários planos de execução podem ser criados aproveitando os casos que já sofreram teste. A figura 2 exibe a representação do projeto quando ele já está passando por uma execução.

Figura 1. Formato de um Projeto de teste.
Fonte: (CRISTALLI, 2008).

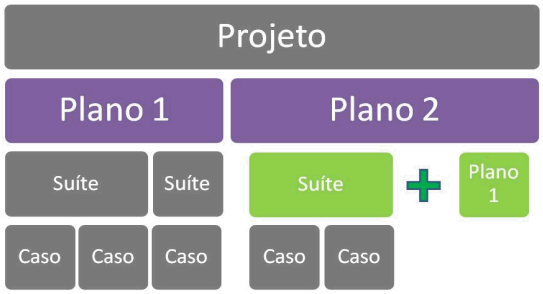

Figura 2. Formato de um Projeto de teste em execução.
Fonte: (IEEE, 2008).

Dentro do projeto de teste tais fatores devem nortear seu planejamento: a probabilidade de ocorrência dos riscos e seus impactos associados.

2.2. Níveis e Tipos de Teste

Conforme Shahid e Harman (2007), o teste de *software* possui propriedades funcionais e não funcionais. As propriedades funcionais envolvem os testes do tipo Funcional. Eles, de acordo com Freitas et. al (2009), são testes realizados para verificar a existência e integridade das funcionalidades que devem estar presentes no *software* que está sendo analisado.

Os testes funcionais geralmente são executados conforme a visão da técnica de Caixa Preta, onde o que se busca é verificar o *software* sem que seja necessário conhecer sua estrutura interna. Ou seja, ele é voltado exclusivamente aos efeitos que a ação do ator sobre o sistema poderá provocar visivelmente. O TSuite utiliza o conceito desse tipo de teste como base e resultado da seleção de casos.

Os testes não funcionais, explica Freitas et. al (2009), são realizados para verificar a resposta do *software* sob as situações que ele pode ser submetido. São itens de observação: desempenho, usabilidade, segurança, acessibilidade, manutenibilidade, recuperação, dentre outros.

Os testes funcionais e os testes não funcionais podem utilizar um ou mais dos seguintes níveis de teste: Teste de Componente, Teste de Integração, Teste de Sistema e Teste de Aceite.

O Teste de Sistema verifica a união das funcionalidades do sistema de uma forma completa. Ele não deve testar particularidades do *software* como trechos de código-fonte ou módulos. Mas sim, a forma de execução – se com sucesso ou falha, conforme o executor siga um fluxo do sistema do início ao fim, (CRISTALLI et. al, 2007). Nesse tipo de teste, deve-se seguir uma sequência de execução lógica dentro de um ou vários módulos do sistema; exemplo: Cadastrar formulário no módulo Pessoa. O objetivo deste tipo de teste é encontrar o máximo de defeitos instalados no programa, antes que ocorra o Teste de Aceite por parte do cliente, (IEEE, 2008).

O Teste de Aceite considera a avaliação do usuário a respeito do sistema. Muitas vezes o próprio cliente realiza esse tipo de teste, pois ele verifica as conformidades implementadas ao sistema em comparação com o projeto contratado. No entanto, demais aspectos do *software* também refletem na aceitação. Ele está relativamente envolvido com os aspectos não funcionais do *software*, embora seja analisados também seus aspectos funcionais, que envolvem a parte concreta ou visual das regras de negócio. Isso, por conta da expectativa do cliente a respeito do sistema. Vale considerar que neste nível de teste, o usuário não busca encontrar defeitos.

Tanto os testes do tipo Funcional quanto do tipo Não Funcional podem ser executados em uma sequência progressiva ou regressiva. Os testes progressivos estão de acordo com a mais nova versão do sistema. Já os regressivos, tem o foco nas versões anteriores à atual. Para esta prática também se dá o nome "reteste". O TSuite seleciona os casos de teste com visão da situação atual do projeto sobre os casos já executados e assim observar os impactos gerados pela nova inserção. Por isso ele é uma ferramenta para apoiar a tomada de decisão na seleção de casos de teste de regressão.

Dentro do método de execução de teste regressivo, existem as formas de cobertura - do sistema - parcial e total, segundo (BARTIÉ, 2002). Com a cobertura total, todos os módulos correspondentes às versões anteriores do *software* precisam ser executados novamente. Na cobertura parcial, busca-se agrupar somente um número de funcionalidades por vez. Ou seja, a execução ocorre para um determinado módulo do sistema. Tal técnica é conhecida como *Smoke Test* ou Teste Básico. Mas ela só ganha essa conotação quando aplicada sob condições de tempo de execução restrito para o *software* em produção, como, por exemplo, o "reteste" das funcionalidades que sofreram alteração após o *software* já estar sendo utilizado no ambiente do cliente, (FARZAT e BARROS, 2010). O *Smoke Test*, portanto, está empregado no contexto do teste de Manutenção.
O TSuite desconsidera o tipo de teste de regressão que será realizada na execução. No entanto, ele tem como foco a execução de teste em iterações.

Para esclarecer os conceitos e fronteiras da produção de *software* e impulsionar a resolução dos problemas considerados pelas técnicas reflexivas citadas acima – planejamento e análise dos fatores de risco - uma nova área da Engenharia de *Software* foi criada. Pesquisas relacionadas às atividades de otimização da Engenharia de *Software* estão contidas em uma área da computação conhecida como *Search Based Software Engineering*, (SOUZA et. al, 2010). A SBSE contém uma subárea de pesquisa que envolve somente os problemas decorrentes das atividades de teste, a *Search Based Software Testing* (SBSE), (FREITAS et. al, 2009).

2.3. O problema da seleção de casos de teste

Yih-farn (1994) é descreve um estudo de caso onde o problema da seleção de casos de testes é questionado. São pré-condições dele: módulos de um programa que possua suítes de teste correspondentes e que esse módulo tenha sofrido alterações. O problema é definido como: encontrar uma suíte de teste reduzida para as modificações ocorridas no programa. Através desse estudo Harman e McMinn (2010), assim como Rothermel e Harrold (1996) explicam que o problema de seleção de casos de teste envolve a release atual e as anteriores de um *software*. Uma release é uma versão do sistema, também conhecida como build. Uma versão pode conter novas partes do sistema e/ou modificações de seus módulos já existentes.

A abordagem de seleção utilizada no TSuite também compartilha desta premissa. Para realizar a análise de impacto originada pela mudança das funcionalidades no *software* é necessário comparar o estado atual dele em relação ao que já foi executado. Isso, para proporcionar uma melhor cobertura de teste sobre ele.

De acordo com Farzat e Barros (2010) a seleção de casos de teste é um problema NP-Completo. No campo da Computação, esse tipo de problema segue sem uma resolução total, já que o bom desempenho em tempo polinomial é uma questão que tende a não ser alcançada. Mas, ao voltar o foco à formulação de um algoritmo de seleção de casos de teste funcionais manuais essa questão pode ser parcialmente abstraída. Isso porque na maioria das empresas, uma ferramenta de gerência de *testware* é utilizada para construir e alterar casos de teste finitos. Tais

programas contém uma base de dados que guardam os atributos desses produtos de trabalho, o que torna o processo de validação passível de ser realizado e facilita a sua busca e restrição de consulta por outras ferramentas de gerência ou extração desses dados.

Outra facilitação é a remoção da obrigatoriedade de geração automática do modelo pelo sistema. Também, a maioria das ferramentas de gerência de *testware* contém um módulo relacionado à geração de relatório, fazendo com que seja possível a extração de uma parte desses dados. Dessa forma, o TSuite torna dinâmica a tarefa de geração de um modelo a cada vez que o *testware* mudar, ficando a cargo do executor informar o projeto e suas alterações. Assim, ocorre rastreio entre o código e os casos de teste manuais.

O problema ainda é mais facilmente solucionado com a atribuição obrigatória de complexidade, custo e tempo de execução, para cada caso de teste candidato à seleção. Essas características possibilitam a conferência de pesos dos casos para que seja realizada a seleção de forma coerente com a necessidade do projeto.

O TSuite utiliza um modelo matemático multiobjetivo na resolução do problema de seleção dos casos de teste. Ele é uma adaptação do modelo encontrado no artigo (FREITAS et. al, 2009) apud (SOUZA et. al, 2010), apresentado na figura 5. As funções que devem ser avaliadas nesse processo são: minimizar a despesa da execução dos testes e maximizar a importância dos casos escolhidos. Cada uma dessas funções segue uma premissa apresentada na figura 3, onde, 1 corresponde ao objetivo de minimização e 2 ao objetivo de maximização. Ambas recebem o parâmetro 't' na unidade 'j'– posição do caso de teste dentro do projeto de teste (tj).

A premissa 1 estabelece as variáveis 'k', 'j' e 't', na somatória que resulta no tempo total gasto durante a execução dos casos. A variável 'k' corresponde ao valor de tempo de um determinado caso de teste, enquanto 'j' é a unidade/quantidade de casos de teste.

Na figura 3 a inequação a) é a representação da restrição do problema. O tempo de execução gasto com a soma de todos os casos de teste deve ser menor que ou igual ao tempo disponível para a tarefa.

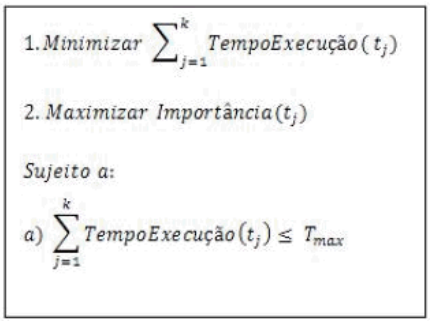

Figura 3. Modelo do problema de seleção dos casos do TSuite.

No trabalho (SOUZA et. al, 2010), antes da seleção é realizado o levantamento de uma base contendo os casos prioritários do projeto. Caso o projeto não tenha recursos disponíveis para selecionar todos, a seleção ótima é executada. No TSuite são selecionados quantos casos prioritários for possível, no tempo disponível para a atividade, utilizando, inclusive, a seleção ótima. Se algum prioritário não for selecionado, isso não será considerado, pois aqueles mais prioritários foram selecionados conforme sua classificação de importância (valor do peso no ranking).

3. TRABALHOS CORRELATOS

A seguir são apresentadas as soluções que possuem relação com o TSuite. Todas têm em comum o fato de terem sido desenvolvidas para tornar automática a seleção de casos de teste para regressão e, portanto, serviram de base à formação da ideia deste trabalho e implementação das funções do TSuite.

3.1. Estudo de Caso: O algoritmo TestTube

TestTube é uma solução publicada por Yih-farn et. al. (1994) para regredir um conjunto de casos de teste unitários. Essa suíte de teste deve possuir correspondência com o código alterado no *software*. Ele identifica os casos conforme a geração de uma nova versão do sistema.

A figura 4 ajuda a entender a ideia básica por trás do problema de seleção dos casos. Nela, os quadrados representam subprogramas -rotinas- do sistema e os círculos correspondem às variáveis dele. Sendo assim, sem utilizar um programa, no caso o TestTube, todos os casos de teste precisariam ser executados novamente, com a mudança da versão do sistema. Após utilizá-lo, somente o caso 3 precisaria sofrer esta ação, já que ele contempla todas as alterações de código que o sistema recebeu.

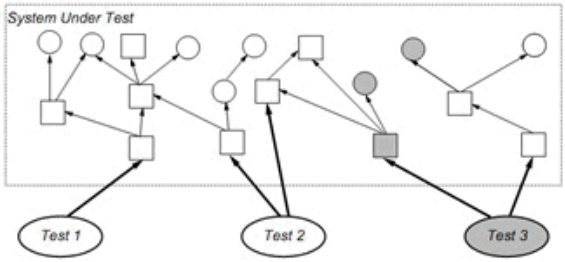

Figure 1: Selective Retesting of a New Version.

Figura 4. Representação do problema de seleção dos casos, a ideia básica do TestTube.
Fonte: (YIH-FARN et. al., 1994)

Como resultado do estudo de caso, percebeu-se uma redução superior a 50% dos casos selecionados para a regressão.

No TestTube, a cada versão do sistema uma nova base de dados do *software*, contendo o mapa de funcionalidades do sistema, é construída. Com isso, é preciso gerar um novo modelo. No TSuite o modelo é gerado conforme a situação que o executor informe à ferramenta, fator que reduz seu custo operacional e o torna uma ferramenta independente de um projeto específico.

3.2. Estudo de caso: Seleção de teste para alterações emergenciais

O trabalho (FARZAT e BARROS, 2010) propõe a criação de uma ferramenta para selecionar casos de teste para uma regressão rápida. Tal regressão é do tipo *Smoke Test*.

Um modelo é criado conforme o incremento de versões do código do sistema, obtido através de suas publicações em um repositório. Assim uma matriz de correspondência entre os

módulos de código-fonte e os casos de teste é criada. Considera-se que para cada escopo de código (funcionalidades do sistema) exista um caso de teste automático em referência.

A proposta do artigo leva em consideração a restrição de tempo imposta à execução de caso de teste. Então, após a seleção de casos de teste que participem da alteração que o código-fonte sofreu, se o tempo restante não for suficiente para concluir a tarefa, um mecanismo de otimização é acionado. Através desse procedimento a seleção passa a obedecer a um período de execução. A escolha é feita contando os casos por sua cobertura e priorização da funcionalidade. Isso após o ator atribuir pesos aos casos positivos e negativos, de forma a somar o valor total de 100%, (FARZAT e BARROS, 2010).

No entanto, essa ideia difere da proposta inserida neste trabalho, pois ela trata somente os casos unitários automatizados e restringe a seleção somente ao contexto da regressão básica. O TSuite não diferencia a seleção por seu tipo de regressão.

3.3. Estudo de caso: Uma abordagem multiobjetiva para o problema da seleção de casos de teste para regressão

O artigo (FREITAS et. al, 2009) apud (SOUZA et. al, 2010) descreve a formatação do problema de seleção de casos de teste através do modelo matemático multiobjetivo. O modelo considera as variáveis de tempo de execução, importância e risco dos casos candidatos à seleção, para realizar duas minimizações e uma maximização, em cada uma de suas três funções objetivos, respectivamente. Os valores são restritos ao tempo de execução dos casos, tempo de execução disponível, precedência e cobertura dos casos.

A primeira função de minimização contém uma somatória do tempo de execução de todos os casos de teste e deve ser menor ou igual ao tempo máximo (restrição a)). A segunda função de minimização avalia o risco de cada um dos casos de teste para ponderar seu grau de impacto, dentro do projeto, e assim, sua importância. A função de maximização serve para selecionar os casos com maior *score* dentro do projeto, dado um contexto.

Os casos de menor risco e maior importância devem ser selecionados, sem que a soma de seus tempos de execução unitários, ultrapasse o tempo máximo disponível no projeto (restrição b)). Assim como define a restrição c), todos os casos devem ser selecionados avaliando sua precedência (relacionada ao risco) e sua cobertura (importância). O modelo é exibido na figura 5.

$$1.\ Minimizar \sum_{j=1}^{k} TempoExecução(t_j)$$

$$2.\ Minimizar \sum_{j=1}^{k} Risco(t_j)$$

$$3.\ Maximizar\ Importância(t_j)$$

Sujeito a:

$$a)\ \sum_{j=1}^{k} TempoExecução(t_j) \leq T_{max}$$

$$b)\ \sum_{h=1}^{p} TempoDisponível(p_h) \leq T_{max}$$

$$c)\ \forall r_i \in R, \forall t_j \in T,$$

$$\left(\exists r \in R \ni r \in precedente(r_i)\ and\ cobertura(t_j, r)\right) \rightarrow t_j \in testCases(r_i)$$

Figura 5. Modelo multiobjetivo para a seleção de casos de teste.
Fonte: (SOUZA et. al, 2010)

O TSuite utiliza este modelo como base para formular sua função específica, na solução do problema de seleção dos casos. No entanto atua de forma resumida, pois a premissa de verificação do risco não é considerada separadamente. Ela está implícita na maximização da importância, onde a análise de impacto é contabilizada sobre o peso unitário dos casos de teste. A restrição de cobertura também ocorre no incremento do peso. Porém, a análise de risco não é restrita à precedência dos casos. Essas características somam à prioridade das suítes do projeto e são contabilizadas como peso dos casos.

A adaptação do modelo também ocorre na restrição da importância, que fica condicionada somente ao critério de peso do caso. Isso porque essa variável representa: a complexidade, tipo e prioridade do caso e prioridade da suíte que o contém, de uma só vez.

Este modelo foi utilizado porque considera a visão do cliente sobre os aspectos do sistema, o que significa dizer que ele está de acordo com o nível de Teste de Aceite. Como o Teste de Sistema possui a estrutura de execução parecida com o Teste de Aceite, utilizando a técnica de Caixa Preta, a visão do cliente é adequada, já que os casos que são o foco deste trabalho são voltados aos testes manuais funcionais de Sistema.

3.4. Estudo de caso: Regressão de testes manuais na prática

O trabalho (DEISSENBOECK et. al, 2011) propõe um estudo de caso baseado na seleção de casos de teste automatizados parcialmente. Eles são assim chamados porque automatizam a sua forma de leitura, por padronizar sua forma de escrita. Isso, para poder verificar a aplicabilidade dos casos de testes manuais, com redução da insuficiência dos resultados de minimização da suíte de teste otimizada automaticamente. No entanto, o artigo não define a forma de fazê-lo.

Uma abordagem complementar à (DEISSENBOECK et. al, 2011) seria fomentar a maneira de escrita e armazenagem dos casos de teste manuais a um nível padronizado. Portanto, com o intuito de abstrair essa etapa de coleta dos dados mantidos na ferramenta de gerência de *testware*, este artigo impõe que: os casos estejam dispostos em suítes distintas. Também, que sejam utilizadas informações classificatórias (*tags*) associadas ao escopo dos casos de teste. Exemplos são: funcionalidade igual à [INCLUIR], complexidade igual à média, etc. Assim como torna obrigatória a descrição de prioridade, complexidade e tempo de execução para cada um dos casos de teste concorrentes na seleção.

Ressalta-se que o TSuite considera que cada caso de teste possui somente uma funcionalidade associada. No caso, ao retratar a edição de um registro, por exemplo, a ação de pesquisa que conduz à edição deve estar contida nos passos de execução, já que só é possível atribuir uma funcionalidade a cada um deles. Com isso, é válido que somente a ação principal esteja informada nesta associação.

O tópico a seguir irá descrever a estrutura do TSuite, informando sua arquitetura e os escopos de código-fonte que contém as principais funções do sistema.
O TSuite utiliza um método específico de abstração do problema de seleção dos casos de teste. Uma rotina específica foi definida para gerar essa seleção. Ela pondera o peso e tempo dos casos, para selecionar o maior número possível deles, de acordo com o tempo disponível ao projeto de teste.

4. O MÉTODO DE SELEÇÃO

O TSuite utiliza os métodos de seleção convencional e de busca ótima para embasar a função de seleção dos casos de teste. Uma rotina específica foi definida para gerar a seleção principal, enquanto que outra, probabilística, é invocada em segunda ordem.

O algoritmo específico é utilizado na primeira parte da seleção dos casos, assim como na restrição do tempo disponível ao projeto, na segunda parte da seleção. O algoritmo probabilístico é utilizado na segunda parte da seleção dos casos. Ele é usado para priorizar os casos dos planos 3 e 4 do projeto, após a seleção convencional por priorização estática, na primeira parte da seleção. A seleção probabilística foi aplicada para oferecer justiça aos casos de menor peso, que podem coincidentemente, em sua maioria, ser casos de complexidade baixa e tipo negativo. A justiça ocorre no aumento da possibilidade de seleção desses casos, em relação aos casos de maior peso. Desta forma, o modelo matemático também sofre divisão, pois a função de maximização da importância dos casos ocorre linearmente (estaticamente), na parte 1 da seleção, ao passo que ocorre quase dinamicamente (pseudo aleatoriamente), na segunda parte.

Utilizando a visão da estrutura em árvore do projeto de teste importado no TSuite, os planos de prioridade 1 e 2 participam da primeira parte da seleção, enquanto que os planos de prioridade 3 e 4 participam da segunda parte.

O algoritmo convencional, desenvolvido para o TSuite, seleciona os casos por seus pesos. A atribuição dos pesos aos casos ocorre em dois momentos. O primeiro momento é a importação do projeto, quando o sistema identifica as características de: tipo, complexidade e prioridade da função associada ao caso, para calcular seu peso inicial. O produto deste cálculo é a soma dos valores relacionados a cada uma dessas características (*tags*).

Se, por exemplo, o caso for do tipo positivo ele ganha o valor 1. Para o tipo negativo o valor do peso permanece inalterado. Os casos de complexidade alta recebem o valor 3, de média o valor 2 e baixa o valor 1. A prioridade da funcionalidade associada ao caso também contribui nessa primeira soma do peso. Como as prioridades só podem variar entre 1 e 5, os casos ganham os valores decrescentes, conforme seja o número de sua prioridade. Assim, a funcionalidade de prioridade 1 agrega o valor 5 ao peso do caso, assim como a prioridade 5 agrega somente 1. Cada uma das 5 funcionalidades só pode ter uma prioridade atribuída, condição essa limitada através da função de Configuração de Tags, no TSuite.

A segunda forma de atribuir pesos aos casos é no momento em que o usuário informa qual(is) a(s) suíte(s), na quantidade máxima de duas(2), participam do plano atual do projeto.

Assim como exibe a figura 6, a primeira faixa compreende os casos contidos nas suítes do plano atual de execução dos testes e participam os valores a partir de 31 até 40. Da segunda faixa participam os casos que possuem associação com o plano atual de execução dos testes e, portanto, são considerados os casos das suítes do plano de execução anterior, no projeto de teste. Seus valores variam entre 21 a 30. A terceira faixa contém os casos associados ao plano anterior de execução, ou seja, são os casos das suítes associadas a partir da segunda ordem, com o plano atual. A faixa varia entre os valores 11 e 20. A quarta e última faixa de valores, que devem estar entre 1 e 10, é integrado por aqueles casos que não possuem relação com nenhum dos outros módulos do plano 1, 2 ou 3, no sistema.

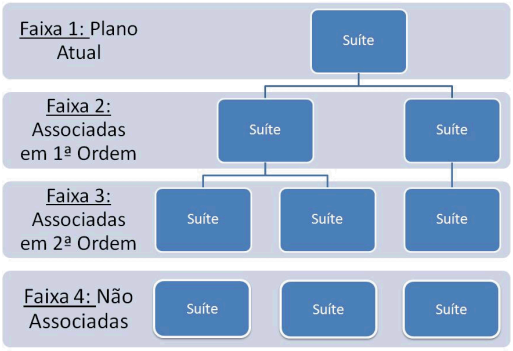

Figura 6. A divisão do Projeto de Teste em faixas de peso para a seleção.

O NSGAII é utilizado no TSuite após a seleção dos casos que participam das faixas de peso para seleção 1 e 2. O que significa dizer que ele seleciona os casos contidos nas faixas 3 e 4. O algoritmo NSGAII é considerado uma metaheurística, (PRATAP et. al., 2002). Segundo Freitas et. al, (2009) define-se que algoritmos de metaheurística representam uma classe de algoritmos de busca genéricos.

O mesmo autor explica que a execução de uma atividade metaheurística termina quando uma função traçada é satisfeita por um algoritmo ótimo. A estratégia metaheurística é divida em dois tipos. A otimização mono-objetiva busca completar apenas uma função de satisfação, enquanto que a multiobjetiva é guiada a suprir várias delas, (PIZZOLATO e GANDOLPHO, 2009).

Para chegar a uma solução comum, que atinja a todos os objetivos do problema satisfatoriamente, é preciso evidenciar o conceito de frente de Pareto. De acordo com (SOUZA et. al, 2010), este termo define um conjunto das soluções que não são piores que nenhuma outra, incluindo umas em relação às outras, aos objetivos do problema.

O método frente de Pareto está embasado sobre outro conceito: a dominância. Dentro de um conjunto de resultados retornados pelas funções, aquele que obtiver o maior ou menor resultado, em pelo menos um dos objetivos, em comparação com todos os outros, é o dominante. Desta forma, ele é considerado o melhor.

O algoritmo genético é um algoritmo evolucionário que utiliza como base a teoria da evolução através da seleção natural, descrita por Charles Darwin, onde cada elemento é definido na Engenharia de *Software* como um indivíduo ou cromossomo, (SOUZA et. al, 2010).

Observando a reprodução sexuada das espécies, um indivíduo, pode evoluir relacionando-se e se adaptar ao ambiente através de características próprias ou herdadas que o tornem apto.

O algoritmo genético deve implementar as operações de *crossover* e *mutation*. *Crossover* é o processo de cruzamento entre duas soluções estruturais para gerar duas novas. *Mutation* realiza a atribuição de uma nova característica, comum a todos os cromossomos, quando uma geração possui muitos indivíduos parecidos, para torná-los variados, Freitas et. al. (2009). No TSuite o algoritmo genético foi utilizado para priorizar pelo menos um peso do plano 3 e outro do plano 4. Isso garante a variedade de cenários selecionados, já que o modo específico prioriza os casos de complexidade alta. Dessa forma, ocorre uma melhor distribuição dos recursos e ocorre justiça aos casos de complexidade baixa.

A quantidade de rodadas que o algoritmo é executado corresponde à quantidade de gerações que ele irá produzir até atingir as soluções ótimas. No TSuite este valor foi configurado para 15, para limitar o custo computacional e mantê-lo estável, bem como meio de prevenção à

conversão total das soluções que é o estágio onde o *crossover* torna as soluções muito próximas umas das outras.

A diferença entre os algoritmos NSGAII, de solução ao problema da Mochila e Aleatório serão discutidos na seção Testes e Resultados.

4.1. O fluxo de execução do TSuite

O TSuite foi desenvolvido para selecionar os casos de teste que melhor se enquadrem à situação atual do projeto. Para que a seleção ocorra é necessário ao executor completar uma sequência de passos obrigatórios. Ele precisa informar os dados de entrada como: funcionalidades mais prioritárias do sistema, associação entre os módulos do sistema, suítes que participam do plano atual do projeto, tempo de execução de cada caso de teste, por complexidade, assim como o próprio projeto de teste. Tal projeto deve conter casos com 3 *tags* cada um, correspondentes às características de: tipo, complexidade e funcionalidade.

Já na introdução do sistema o executor deverá informar até 5 *tags* que correspondam às funcionalidades do sistema relacionadas aos casos. A operação é assim disposta para atribuir prioridades às funcionalidades do sistema. Um exemplo seria informar a *tag* [INCLUIR] com prioridade 1, ou seja, a principal. O benefício incutido nessa ação é retirar a obrigatoriedade de atribuir prioridade caso a caso, pois entende-se que somente no momento da seleção é possível assimilar a relevância deles, de acordo com a situação atual do projeto.

O TSuite tem como entrada mandatória que a funcionalidade associada a um caso esteja informada entre colchetes (símbolos '[' ']'). Esse é um padrão estabelecido aqui para que seja possível ao sistema e fácil ao usuário do TSuite diferenciar as funcionalidades das outras informações do *software* que devam ser associadas em forma de *tag*.

A configuração de parâmetros do projeto segue igual formatação da função de configurar funcionalidades, porque abstrai a atividade de informar caso a caso o parâmetro de tempo. Tarefa essa que seria desnecessária, pois entende-se que a base de conhecimento dos recursos do projeto é constituída pela média retirada por um critério da equipe, incluindo da atividade de seleção de casos para regressão. Nesse caso, assume-se o critério da complexidade. Ou seja, esses seriam valores comuns a todos os casos que pertençam a uma das categorias de complexidade: baixa, média e alta.

A informação de complexidade corresponde à quantidade de passos de um caso versus a dificuldade de execução que ele contém. No projeto de teste utilizado para a realização dos testes do TSuite o critério utilizado foi: se o caso possui até 5 passos, ele é de complexidade baixa; se contém até 10 ele é de complexidade média; e se contiver acima de dez 10 passos ele é considerado de complexidade alta. A depender da quantidade de pré-condições ou pós-condições que ele possua essa classificação poderia ser alterada para uma ou outra complexidade. Esse técnica foi adotada na ferramenta por ser um padrão em projetos de teste de médio porte. A quantidade de passos relacionada à complexidade do caso deve refletir a dimensão do *testware* do projeto de teste.

Ao gerar um modelo o usuário realiza as funções de associar suítes e associar funcionalidades à relações de suítes. O passo, que não é obrigatório, permite definir a relação entre as suítes de teste do projeto, gerando indiretamente um modelo do sistema. Uma suíte associada a uma funcionalidade confere na priorização de todos os seus casos que contiverem tal *tag*(funcionalidade). A interface gráfica da ação de gerar modelo é ilustrada na figura 7.

Figura 7. Passo Gerar Modelo, da aplicação TSuite

Assim como o passo Gerar Modelo, a função de Selecionar Casos é constituída por partes. A evidência do plano realiza o cadastro do plano atual e dos demais, o que pode incluir o plano de integração.

Das etapas aqui relatadas, somente os passos Gerar Modelo e Exportar Projeto são opcionais. No entanto a tarefa de Gerar Modelo é o único meio de aplicar a análise de impacto no momento da execução da seleção dos casos, o que implica dizer que, sem informá-lo, essa análise será desconsiderada. A tabela 1 lista todos os passos contidos no TSuite e as explica em resumo, o que pode ser observado no diagrama de caso de uso exibido na figura 8.

Tabela 1. Ações da ferramenta web TSuite

Módulo do sistema	Objetivo
Login e Cadastro de usuário	Autenticar Usuário no sistema
Configurar Tags	Atribuir prioridade aos casos que possua uma funcionalidade associada a ele
Configurar Parâmetros	Informar os custos unitários dos casos de teste
Importar Projeto	Extrair as suítes e casos de teste do projeto de teste
Gerar Modelo	Criar um modelo virtual do projeto de teste
Selecionar Casos	Restringir o projeto de teste a um conjunto prioritário de situações do sistema
Exportar Projeto	Disponibilizar ao TestLink a suíte de regressão gerada

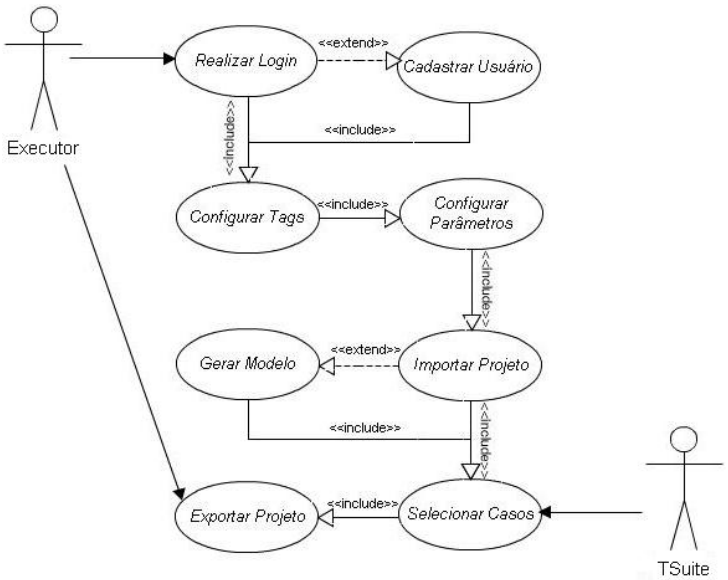

Figura 8. Diagrama de caso de uso do TSuite

5. TESTES E RESULTADOS

O cenário para a realização dos testes de validação do TSuite foi o centro de teste de uma empresa de tecnologia da informação. O teste foi constituído de duas partes, a primeira com foco sobre a vantagem em utilizar a forma automática na seleção dos casos em relação ao método manual. A segunda parte evidencia a necessidade em priorizar os casos dos planos 3 e 4 para oferecer justiça aos casos de complexidade baixa, dentro dessas faixas.

O método de seleção utilizado no primeiro teste foi puramente convencional, com a intenção de oferecer condições de comparação com a segunda etapa de testes.

O arquivo XML utilizado na importação do projeto de teste é uma projeção dos dados de um sistema real para gerenciamento de serviços sociais. Os casos de teste receberam as *tags* de: tipo, complexidade e funcionalidade, para atender às necessidades de importação da ferramenta.

A validação foi realizada levando em consideração três tipos de executores de teste. Os dois primeiros grupos foram formados por profissionais de teste com experiência na realização da tarefa de seleção e execução de casos de teste manuais através da ferramenta TestLink. A diferença entre eles é que, a primeira equipe foi constituída pelos indivíduos que possuem entendimento sobre as regras de negócio do projeto importado na ferramenta, enquanto que o segundo não. A terceira equipe foi formada por profissionais que possuem o nível de experiência inferior aos grupos citados anteriormente, mas que obteve acesso às regras do sistema.

O teste foi constituído de duas etapas e aplicado para três equipes, com cada equipe contendo 2 profissionais de teste. Cada uma das fases ocorreu em períodos distintos, mas contou com o ambiente de operação e pessoal iguais. No primeiro teste realizado os grupos receberam a atividade de selecionar manualmente os casos de teste de regressão que possuíssem maior relevância de acordo com o objetivo do plano de execução atual do projeto. Foi dado o tempo de 30 minutos para ambos realizarem a atividade de selecionar uma suíte de teste prioritária em meio a 63 casos de teste de complexidade baixa ou alta. O projeto de teste modelado não possuía casos de complexidade média, pois eles foram retirados para evidenciar ainda mais a desvantagem em utilizar a abordagem puramente convencional. É perceptível na tabela 3 que os casos de complexidade baixa estão em menor quantidade do que aqueles de complexidade alta.

Complexidade do caso de teste	Tempo(min)
Baixa	12
Média	24
Alta	48
	*
Total(Tempo)	8h

Tabela 2. Recursos disponíveis ao Teste 1.

Tipo de Ex.	Variáveis	Equipe Verde	Equipe Azul	Equipe Vermelho
Manual	Tempo(min)	22	30	30
	Qtd.	12	22	14
	Variedade	7-A|10-B	5-A|17-B	6-A|8-B
	Resto(min)	84	296	156
TSuite	Tempo(min)	19	15	16
	Qtd.	13	13	13
	Variedade	9-A|4-B	9-A|4-B	9-A|4-B
	Resto(min)	0	0	0

Tabela 3. Resultados obtidos para o Teste 1.

Foi obrigatório levar em consideração também os casos de integração entre esse módulo e os demais. O resultado segue a forma das hipóteses 1 e 2 e esquema de resposta ilustrados nas tabelas 3 e 4.

Tipo de Ex.	Variáveis	Equipe Verde	Equipe Azul	Equipe Vermelho
Manual	Tempo(min)	Satisfatório	No Limite	No Limite
	Qtd.	Suficiente	Muito	Pouco
	Resto(min)	Desperdício	Desperdício	Desperdício
TSuite	Tempo(min)	Melhorou	Melhorou	Melhorou
	Qtd.	Suficiente	Suficiente	Suficiente
	Resto(min)	Ideal	Ideal	Ideal