UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA
2008.2

AVALIAÇÃO EMPÍRICA DE UMA FERRAMENTA PARA SELEÇÃO DE TESTES DE REGRESSÃO EM SOFTWARE ORIENTADO A OBJETOS

TRABALHO DE GRADUAÇÃO

Aluno: João Victor Guimarães de Lemos (jvgl@cin.ufpe.br)
Orientador: Marcelo Bezerra D´Amorim (damorim@cin.ufpe.br)

Recife, Novembro de 2008

Assinaturas

Este trabalho de graduação foi resultado do trabalho do aluno João Victor Guimarães de Lemos, sob orientação do professor Marcelo Bezerra D´Amorim, sob o título de “Avaliação empírica de uma ferramenta para seleção de testes de regressão em software orientado a objetos”.

_____

João Victor Guimarães de Lemos (aluno)

_____

Marcelo Bezerra D´Amorim (orientador)

Agradecimentos

Agradeço primeiramente a meus pais. Meu pai, por ser um homem de caráter, íntegro, que sempre fez questão de me elucidar a importância do estudo na vida do ser humano. Minha mãe, por estar comigo em todas as horas, por sempre me entender e me confortar.

Agradeço também a meus irmãos, companheiros dos bons e maus momentos. Meu irmão, aquela pessoa que sempre me proporcionou risadas e alegria, e que me ap0erta o peito de saudade quando lembro das nossas boas conversas de madrugada. Minha irmã, alma afável e bondosa, pessoa simples que me traz muita admiração.

Um agradecimento especial para a minha adorada Denise Paula, a mulher que possui qualidades que toda mulher deveria ter: Beleza, caráter, bom senso, inteligência, ternura, feminilidade e coragem. Obrigado por ter dividido comigo toda a carga que a vida universitária traz, especialmente tratando-se de um curso de engenharia. Quero dizer que espelho minhas atitudes na sua valentia.

Não poderia também deixar de lembrar aqui dos meus amigos e companheiros de sofrimento durante todos esses cinco anos: Brenão, Hugo, Flora, Caio, Tiaguinho, Boneco, Bartolomeu e Aretakis. Foi difícil, desgastante e por muitas vezes angustiante, mas por causa de vocês vou ter muitas histórias para contar no futuro. Em especial, uma menção honrosa aos três primeiros elementos desta lista, já que dividimos o mesmo apartamento durante estes últimos meses.

Sei que vou esquecer-me de alguém (sei também que esta frase é um clichê indispensável em qualquer seção de agradecimentos), mas aqui vai uma lista de pessoas que quero

agradecer por ter me dado apoio em algum momento da minha vida universitária: Professoras Marcília Andrade e Jemima Alves; Professores Cristiano Araújo, Marcelo D´Amorim e Ruy; Sérgio, Salete e Gener; amigos Pedro Igor, Felipe Baudel e Camila Lyra; e finalmente aos colegas de trabalho, que muito me ajudaram na reta final: Paula Bordeiro, Ewerton, Raquel Leite e Arthur Gonzaga.

Resumo

Testes de regressão são testes aplicados a um software que sofreu uma mudança em parte do seu código. Tais testes tem o objetivo de prover ao testador a confiança de dois pontos importantes: (i) As partes modificadas se comportam como esperado e (ii) As partes não-modificadas não foram negativamente afetadas pela introdução de novos segmentos de código.

Para garantir (ii), é necessário re-executar toda a suíte de testes que existia antes de feita a modificação de código (chamada suíte de testes original). No entanto, isto envolve um alto custo computacional, principalmente em ambientes com um rígido controle de versões onde a execução de testes é freqüente.

A fim de reduzir o custo computacional envolvido no processo de execução de testes, existem técnicas de seleção de testes de regressão caracterizadas como "seguras". Estas técnicas visam escolher o menor subconjunto de casos de teste da suíte original, onde cada elemento deste subconjunto pode acusar algum possível erro na versão modificada do software.

O objetivo deste trabalho é analisar o impacto do uso de técnicas automáticas de seleção de testes de regressão num software escrito em linguagens de paradigma orientado a objetos.

Em particular, será analisada uma ferramenta capaz de efetuar seleção automática de testes de regressão, denominada RTSTool, sendo o autor deste presente trabalho um dos autores do software a ser estudado.

O estudo feito aqui oferece como resultado, entre outros, análise de impacto em produtividade no uso da RTSTool ao longo do processo de codificação.

Palavras-chave: Testes de software, Seleção de testes de regressão, Análise de impacto.

Sumário

Índice de figuras

Índice de tabelas

Lista de siglas, abreviações e traduções mais usadas

BCEL	Byte Code Engineering Library
CFG	Control Flow Graph (Grafo de fluxo de controle)
JTopas	Java Token Parser
SIR	Software Infrastructure Repository
XML	Extensible Markup Language

1 Introdução

Testes de regressão são aplicados a programas recentemente alterados, com o objetivo de fornecer confiança que o software se comporta corretamente, e que novos erros não foram introduzidos em partes do código previamente testadas e validadas [3, 4].

A execução de testes de regressão é um processo intrinsecamente custoso [3] devido à necessidade da re-execução de suítes de testes inteiras associadas à versões anteriores do software.

Para reduzir o custo associado a usar testes de regressão, casos de teste são selecionados da suíte que foi usada para testar o software original. Este processo é chamado de seleção de testes de regressão.

Uma técnica de seleção de testes de regressão é classificada como segura se seleciona todos os casos de teste da suíte original que podem expor falhas no programa modificado [1], havendo pelo menos um teste na suíte original que indique uma falha no programa.

Existem técnicas de seleção de testes de regressão, no entanto, que visam selecionar apenas um subconjunto da suíte original e usar este subconjunto para testar a versão modificada do software. Desta forma, o custo da realização de testes de regressão será diminuído de maneira inversamente proporcional ao tamanho do subconjunto delineado.

Uma seleção eficiente (uma que seleciona uma pequena fração dos casos de teste originais) e segura de testes de regressão é particularmente útil em ambientes onde os testes são executados numa freqüência alta ou em ocasiões onde existem numerosos casos de teste, ou mesmo quando o processamento destes casos de teste é excessivamente demorado.

Rothermel e Harrold [4] foram responsáveis pelo desenvolvimento de um algoritmo capaz de efetuar seleção de testes de regressão de forma segura. Este mesmo algoritmo foi adaptado para funcionamento nas linguagens C++ [6] e Java [1].

Com base na técnica descrita em [1], foi desenvolvida uma ferramenta para seleção automática de testes de regressão [7]. O algoritmo de seleção é baseado em [1], no entanto menos preciso. A ferramenta em questão, denominada RTSTool, funciona como um plug-in para a IDE Eclipse [8].

1.1 Objetivos e contexto

O objetivo deste trabalho é analisar o impacto do uso da ferramenta RTSTool num software escrito numa linguagem de paradigma orientado a objetos.

Mais precisamente, o estudo será feito em cima da aplicação do RTSTool em dois outros softwares: O analisador de texto *JTopas* [9] e o processador de textos XML *NanoXML* [10].

Serão realizados experimentos em cima de várias versões dos softwares supracitados. Estas versões foram coletadas da iniciativa SIR (*Software Infrastructure Repository*) [14], repositório com o objetivo de prover softwares-teste para programadores que desejem implementar ou usar mecanismos de teste e depuração.

No total, três versões do *JTopas* e duas versões do *NanoXML* serão investigadas.

De uma maneira mais geral, procurar-se-á investigar parâmetros que remetem ao desempenho de execução da suíte de testes necessária para garantir a consistência do programa, tais como: Número de casos de teste selecionados, porcentagem de casos de teste da suíte original selecionados, redução no tempo de teste do software, número de versões de um mesmo programa testadas, entre outros.

1.2 Organização do documento

Este documento está organizado da seguinte maneira: O capítulo 2 contém uma visão geral sobre a ferramenta RTSTool, além de uma explanação detalhada sobre o algoritmo de seleção de testes de regressão no qual este projeto se baseia.

O capítulo 3 relata a avaliação empírica da ferramenta RTSTool, a metodologia dos experimentos, as métricas usadas na avaliação e apresenta tabelas catalogando os resultados obtidos. Também serão feitas análises destes resultados.

Por fim, o capítulo 4 traz as conclusões obtidas ao longo deste trabalho, tratando também de possíveis trabalhos futuros.

2 A ferramenta RTSTOOL

O projeto RTSTool é um trabalho cuja início deu-se em no primeiro trimestre de 2008, e ainda não completamente finalizado, encontra-se em processo contínuo de desenvolvimento e testes.

A intenção do projeto é incorporar ao ambiente de desenvolvimento de software uma funcionalidade que permita a seleção e execução de testes de regressão, de maneira automática e eficiente.

Foi decidido então, criar a ferramenta RTSTool como uma funcionalidade adicional do ambiente de desenvolvimento Eclipse. Esta decisão foi tomada em virtude do Eclipse ser a IDE mais usada atualmente entre os programadores Java [11] e de prover um excelente suporte a desenvolvimento de plug-ins. O Eclipse permite ao desenvolvedor a possibilidade de personalizar seu ambiente próprio de trabalho de acordo com o projeto que está sendo codificado naquele momento e é uma referência mundial na área de softwares open-source.

Na área de testes de aplicações encontradas para o ambiente Eclipse, a mais difundida é o JUnit, ferramenta de testes unitários para Java. É possível nesta ferramenta manipular tanto casos de teste unitários como suítes de teste completas [15].

A versão do JUnit usada foi a 4.0. Esta versão permite o desenvolvimento mais rápido de testes através do uso de características mais recentes da linguagem Java, como por exemplo, o uso de *annotations.* Tais potencialidades aperfeiçoam significativamente a confecção de casos de teste.

2.1 Técnica para seleção de testes

A maneira pela qual a RTSTool seleciona os testes de regressão é largamente baseada na técnica de seleção de testes de regressão para software codificado em Java, descrita por Harrold et.al. [1].

Consideremos o caso de um programa (chamado na figura 1 de programa original) que possui uma suíte de testes usada para testá-lo (suíte de testes original). Modificações são feitas no programa criando um programa modificado. As próximas seções detalharão como é feita a seleção da técnica considerando o caso em que a seleção de testes de regressão é realizada pela primeira vez. Após esta explanação, será mostrado como são feitas as seleções subsequentes.

No caso da seleção estar sendo realizada pela primeira vez, esta técnica possui partes distintas e sequenciais.

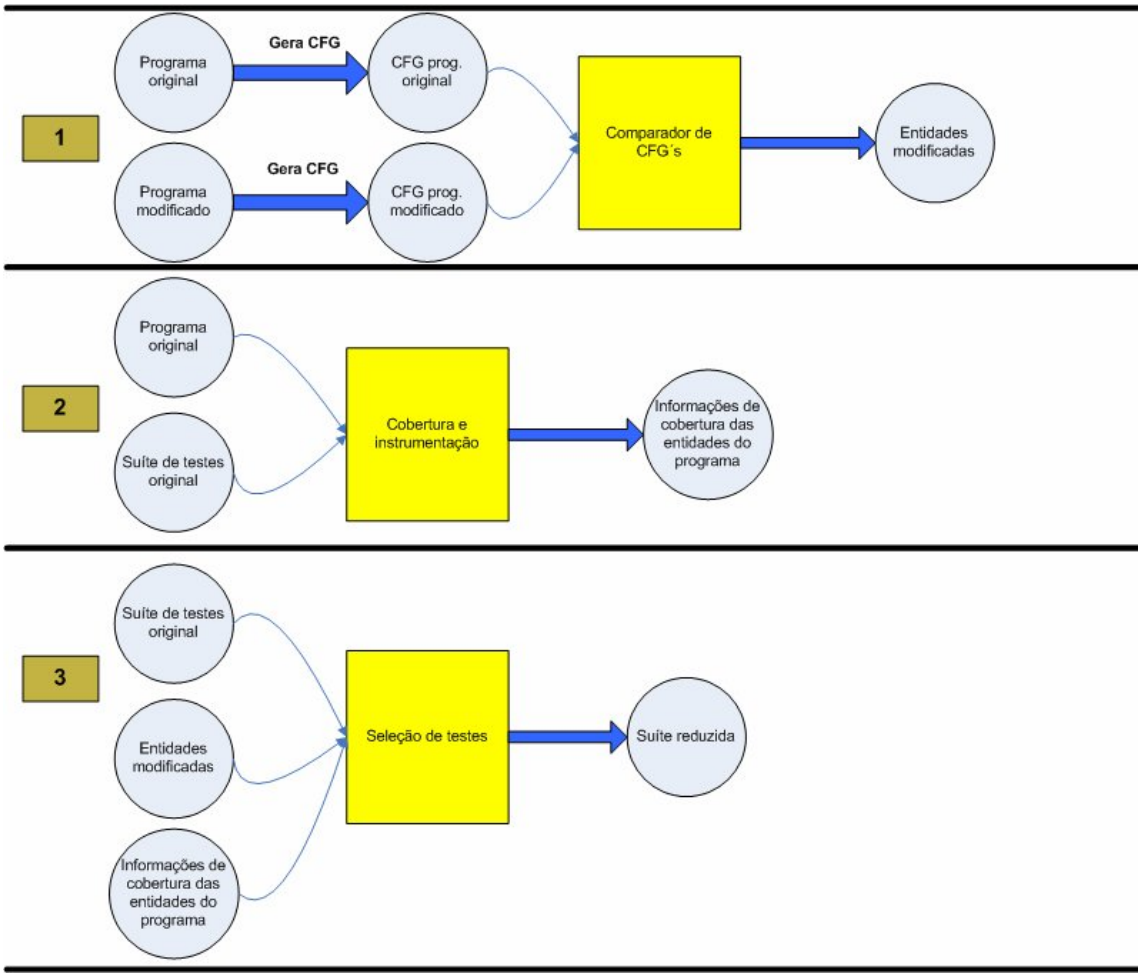

Figura 1 – Técnica de seleção de Rothermel/Harrold

Em (1), a lista de entidades perigosas é obtida através da comparação dos CFG´s gerados. Em (2), o proceso de cobertura é efetuado e em (3) a seleção de testes é realizada usando informações obtidas previamente.

Na primeira parte, as diferentes versões de um programa são modeladas como grafos de fluxo de controle ou CFGs (Control Flow Graphs) e são comparadas a fim de obter informações relativas às diferenças existentes entre versões. Estas diferenças entre duas versões de um programa são identificadas na forma de entidades estruturais de um programa (comandos, desvios, métodos ou condições).

Na segunda parte é efetuado um processo de cobertura da suíte de testes com relação à primeira versão do programa.

Este processo de cobertura guarda as entidades estruturais de um programa que cada caso de teste em particular consegue analisar.

O método usado para realizar o processo de cobertura é chamado de instrumentação. Este método produz uma segunda versão do programa, chamada versão instrumentada, de modo que quando esta versão instrumentada é executada com algum caso de teste, ela armazena a informação de todas as entidades do programa, como comandos e desvios, que são executadas com este caso de teste.

O objetivo desta segunda parte é descobrir as porções de código que são cobertas por cada caso de teste.

Finalmente, a terceira e última parte é responsável pela seleção de casos de teste da suíte de testes original. Esta seleção é feita através das informações disponibilizadas nas duas etapas anteriores.

A técnica supracitada encontra-se sumarizada e ilustrada na figura 1, exibindo as entidades envolvidas na técnica e o relacionamento entre elas.

As seções seguintes abordarão como as três parte supracitadas são realizadas em [1], e como estas são adaptadas para realizar a seleção no caso da ferramenta RTSTool.

2.1.1 Construção dos grafos de fluxo de controle

O primeiro passo da técnica de seleção de testes de regressão descrita em [1] é a representação das diferentes versões de um programa através de um CFG. Rothermel e Harrold utilizaram técnicas de codificação de programas num CFG em muitos de seus trabalhos sobre seleção de testes de regressão [4,6].

Um CFG é construído para cada procedimento P da versão antiga do software e para a sua codificação P’ na nova versão do software. As arestas dos grafos construídos são consideradas possíveis “entidades perigosas”. Estas são escolhidas efetuando um algoritmo que percorre de maneira paralela os CFG´s de P e P´.

Sempre que os alvos das arestas correspondentes diferem, aquela aresta é adicionada ao conjunto de entidades perigosas.

Como exemplo ilustrativo, a figura 2 mostra um programa exemplo denominado *avg* e uma versão modificada *avg'.*

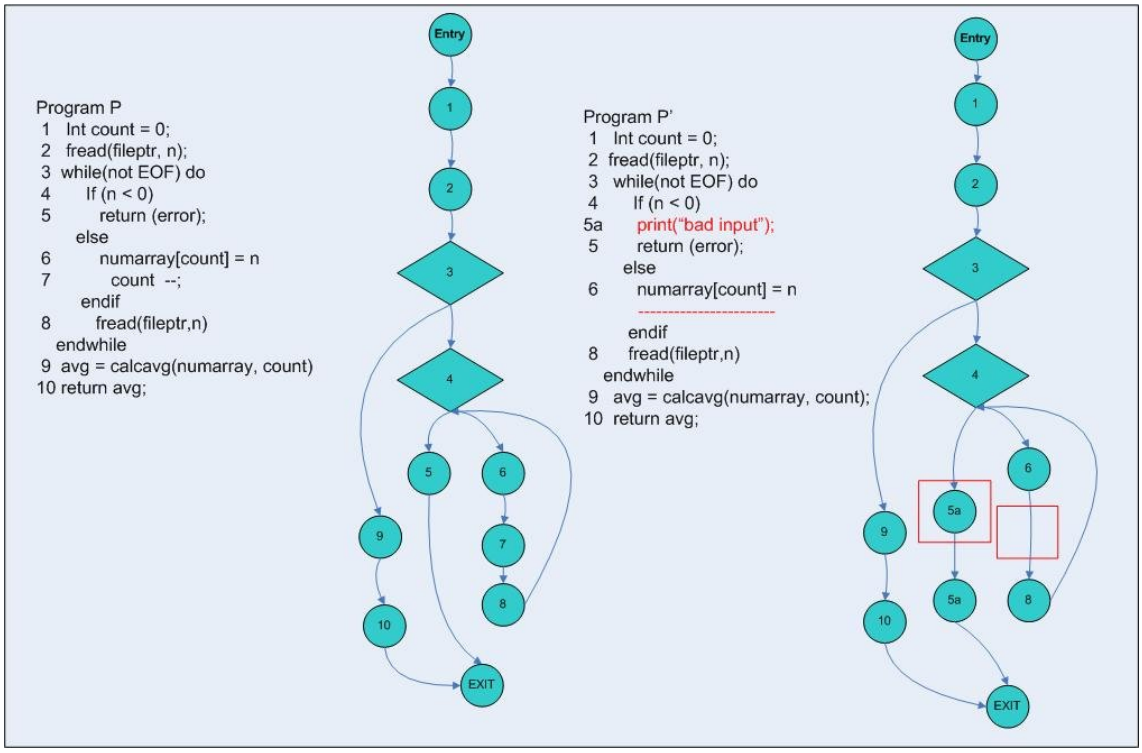

Figura 2 – Construção de CFG´s para programas
De P para P´, duas linhas de código são alteradas, refletindo na inserção do nó 5a e a retirada do nó 7 no grafo da direita.

A diferença entre as versões é que na versão mais recente, o comando 5a foi adicionado e o comando 7 foi retirado.

O algoritmo inicia a sua travessia com os nós de entrada (com o rótulo "entry"), e percorre os grafos paralelamente, perfazendo caminhos até alguma diferença nas arestas é detectada.

Quando o algoritmo considera o nó 4 de avg e o nó 4 de avg', descobre que os alvos da aresta que parte deste nó (com o rótulo "T") diferem nos dois grafos. Portanto, esta aresta é adicionada ao conjunto de entidades perigosas.

De maneira análoga, ao considerar o nó 6 nos dois programas, o algoritmo identifica que o alvo das arestas que partem destes nós diferem, logo a aresta (6,7) é considerada também uma entidade perigosa.

Para gerar os CFG´s correspondentes aos métodos existentes nas classes e efetuar a comparação em busca de mudanças nos programas, a RTSTool se utiliza da biblioteca Java BCEL [12].

O BCEL então é o agente embutido responsável por gerar CFG´s para cada um dos métodos existentes no código do programa que está sendo testado. Após isso, os grafos gerados seguem como entrada para um algoritmo que os compara em busca da detecção das entidades perigosas, ou seja, porções do código que mudaram de uma versão para outra.

Este algoritmo, no entanto, trabalha em menor granularidade que a técnica discutida acima. O conceito de entidade perigosa não é abordado no nível de transferência de fluxo de controle, e sim no nível de método. Deste modo, quando for identificada alguma diferença entre métodos de versões diferentes do software, todo o método é considerado uma entidade perigosa.

Embora esta abordagem pareça excessivamente simplista, estudos indicam que em muitos casos realizar a seleção de testes numa granularidade mais fina (a nível de fluxo de controle) parece não trazer ganhos muito significativos [1].

2.1.2 Construção da matriz de rastro

O segundo passo efetuado pelo algoritmo é a construção da chamada matriz de rastro, ou matriz de cobertura.

Esta matriz é responsável por armazenar informações sobre quais entidades existentes na versão mais antiga do software (não modificada) são cobertas por cada caso de teste.

Após todos os casos de teste terem sido rodados, a informação de cobertura é armazenada na matriz de rastro, que associa cada caso de teste existente na suíte de testes com as entidades que este caso de teste executa.

A tabela 1 exibe uma possível matriz de rastro para o exemplo demonstrado.

Matriz de rastro	
Aresta	**Caso de teste**
(entry,1) , (1,2), (2,3)	**1, 2, 3**
(3,9) , (9,10), (10,exit)	**1, 3**
(3,4)	**2, 3**
(4,5) , (5,exit)	**2**
(4,6) , (6,7) , (7,8) , (8,3)	**3**

Tabela 1- Matriz de rastro

No caso da técnica descrita, em que as entidades do programa são representadas por arestas do CFG, os casos de teste devem estar associados a arestas num grafo de fluxo de controle, representando trechos do código de um método que são cobertos por aquele caso de teste.

A ferramenta RTSTool necessita da capacidade de computar cobertura de código, ou seja, registrar quais trechos de código são exercitados por um teste. Para esta tarefa no RTSTool, a ferramenta *emma* [13] é a responsável por armazenar a matriz de rastro de todos os casos de teste do sistema.

Embora o *emma* seja capaz de registrar cobertura a nível de linhas de comando (dizer quais linhas de comando são executadas por um teste), o nível de cobertura usado na RTSTool é bem mais simples: Simplesmente recorda-se por quais métodos um caso de teste executou. Os métodos das classes do sistema são considerados cobertos (100% de cobertura) ou não cobertos (0% de cobertura).

Portanto, no algoritmo de seleção realizado pela RTSTool as entidades de um programa são representadas por métodos inteiros.

A saída produzida neste passo no caso da ferramenta em estudo será a matriz de rastro que associa os casos de teste e os métodos cobertos por aqueles casos de teste.

2.1.3 Seleção dos casos de teste

Utilizando as informações armazenadas na matriz de rastro de arestas e o conjunto de entidades perigosas computado no primeiro passo do algoritmo de seleção de testes de regressão, o passo final da técnica consiste simplesmente em indexar a matriz usando as entidades perigosas, retornando no final uma lista de casos de teste.

No exemplo demonstrado ao longo desta seção, as entidades perigosas são as arestas (4,5) e (6,7). Desta maneira, buscam-se as entradas na matriz de rastro e conclui-se que os casos de teste 2 e 3 precisam ser re-executados na nova versão do software.

Após a primeira seleção de testes for realizada, não há necessidade de executar toda a suíte de testes (suíte original) para efetuar o processo de cobertura. As novas matrizes de rastro serão construídas após a execução das suítes reduzidas, e serão baseadas em informações de cobertura advindas da execução destas suítes reduzidas.

A figura 3 exibe os processos executados em ordem seqüencial no tempo, sendo o ponto v0 a versão original do programa, v1 a primeira modificação, v2 a segunda modificação e assim por diante.

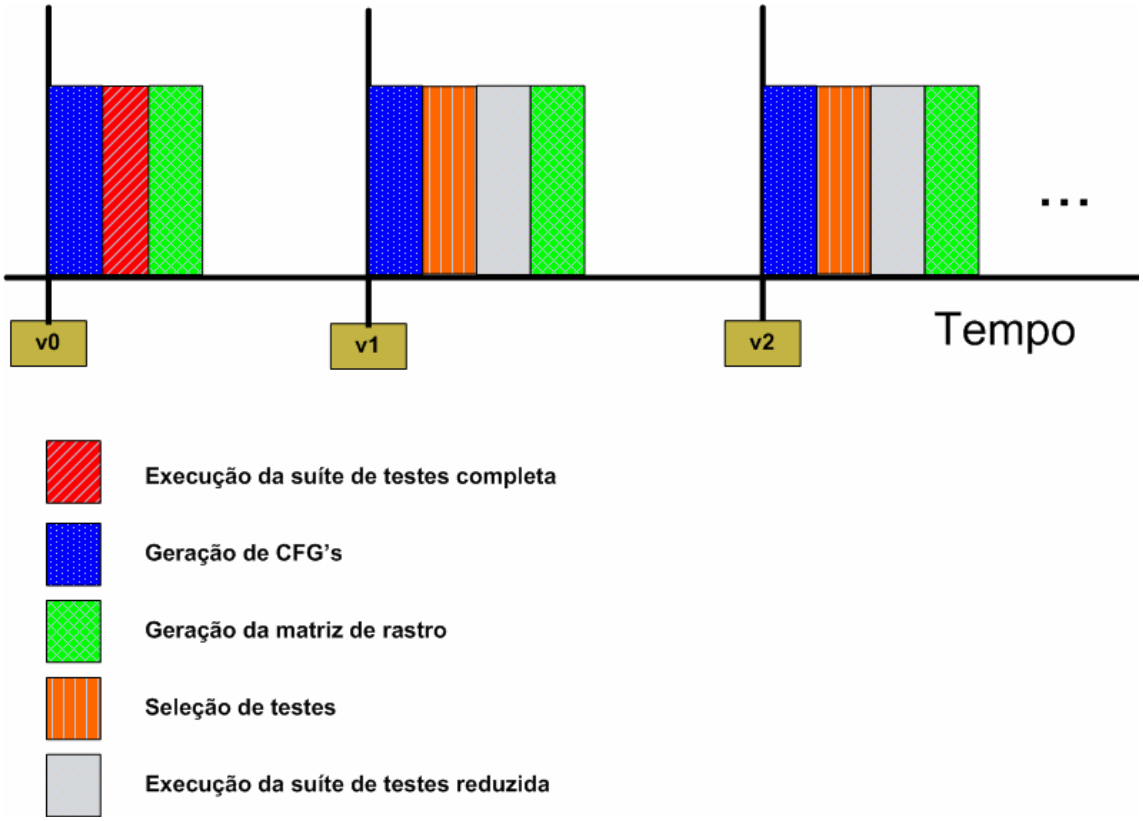

Figura 3 - Execuções sequenciais da técnica de Rothermel/Harrold
Em v0 são computados todos os CFG's e as informações de cobertura do programa original. Para isso todos os casos de teste devem ser rodados. Em v1 (primeiro programa modificado), é feita primeira seleção e executa-se a suíte de testes reduzida. A matriz de rastro é então atualizada com a cobertura dos casos de teste executados na suíte reduzida.

2.2 Funcionamento da ferramenta

Esta seção procura explanar, de modo sucinto, o modo de funcionamento da RTSTool, com destaque na sequência de passos realizados pelo usuário.

2.2.1 Associação de um projeto do Eclipse com o RTSTool

O primeiro passo de uso da RTSTool é habilitar a manipulação de um projeto Java pela ferramenta. O usuário deve indicar o projeto do Eclipse (em Java) no qual ele deseja realizar seleção de testes de regressão.

Para efetuar isto é necessário abrir o menu de ações, pressionando o botão direito do *mouse* sobre o projeto e selecionar a opção "Use RTSTool in this project" (Figura 4).

Feito isto, o projeto estará habilitado a executar todas as funcionalidades da RTSTool.

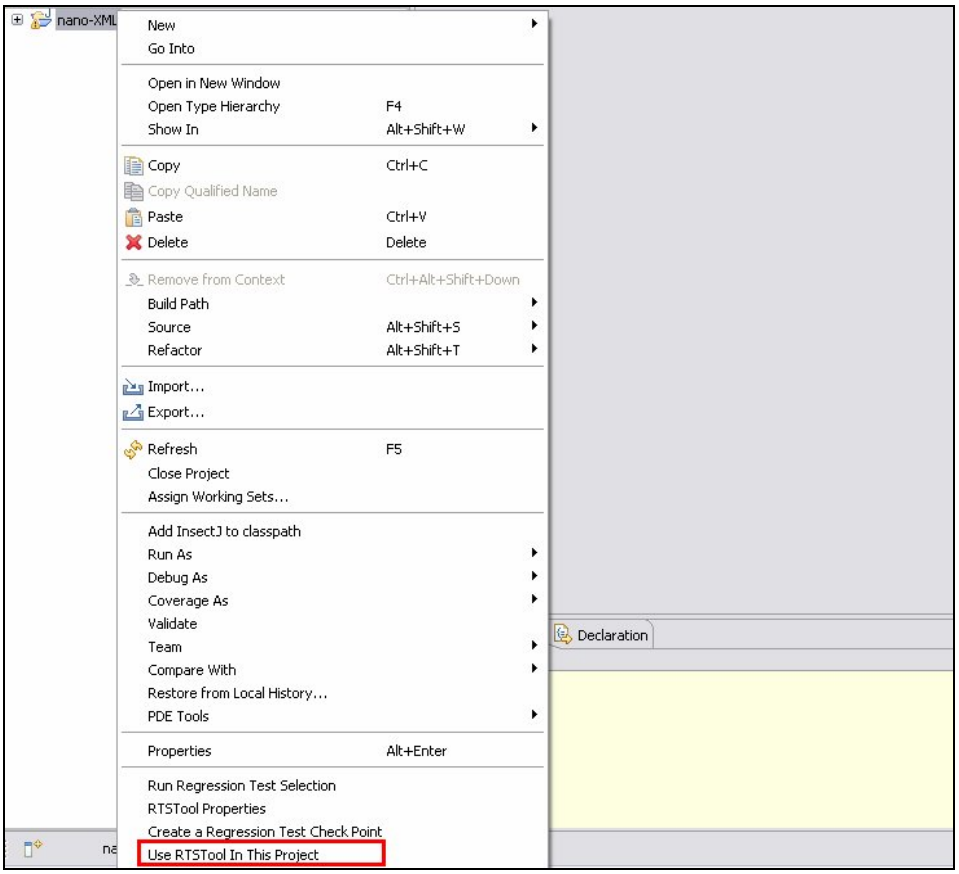

Figura 4 - Associando projeto de Eclipse com a RTSTool

2.2.2 Confeccionando e executando os casos de teste

Os casos de teste da RTSTool devem ser confeccionadas na plataforma de testes Junit 4 [15].

Após a criação de todos os casos de teste, deve-se usar a classe RTSToolSuite através da *annotation* @RunWith. A classe RTSToolSuite é a suíte de testes que contém todos os casos de teste que o programador deseje utilizar em seu projeto.

A figura 5 exemplifica este processo.

```
import org.junit.runner.RunWith;
import com.bjSoft.regressionTestTool.junit4.RTSToolSuite;

@RunWith(RTSToolSuite.class)
@RTSToolSuite.SuiteClasses(value={test1.class, test2.class,
test3.class})
public class MySuite {
}
```

Figura 5 - Código da suíte de testes para um projeto que usa a RTSTool

No exemplo exibido, as classes de teste *test1*, *test2* e *test3* são passadas à suíte de testes como valores.

Para executar a suíte de testes basta incluí-la como um novo executável do Junit, através da opção RunAs -> JunitTest (figura 6).

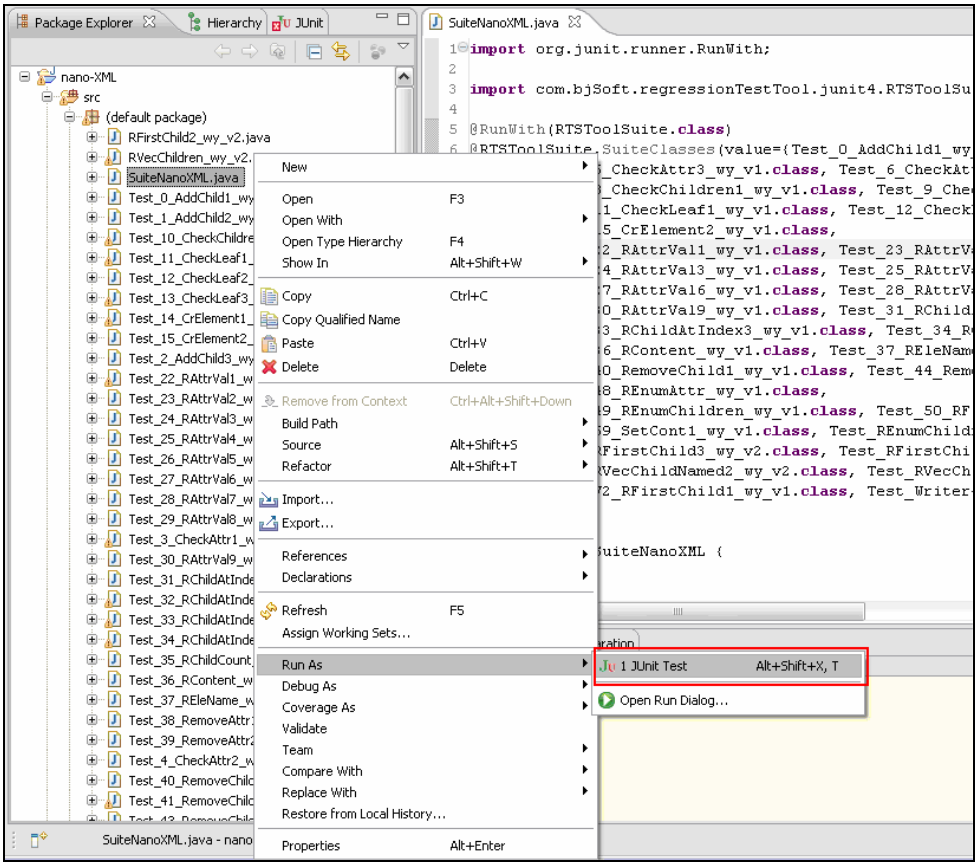

Figura 6 - Executando a suíte de testes no RTSTool

Neste ponto a ferramenta possui um controle de quais testes estão sendo executados.

2.2.3 Determinando ponto de validação da suíte de regressão

Após a execução da suíte completa, caso todos os testes passem (não ocorra nenhuma falha ou erro) é possível gerar um ponto de validação dos casos de teste.

Este ponto de validação será o ponto no qual teremos uma versão estável do programa, a fim de executar a seleção de testes. Qualquer modificação no código feita a partir deste ponto poderá ser usada para efetuar seleção de testes de regressão.

Nesta etapa são computados os CFG´s e dados sobre a cobertura dos casos de teste, de forma semelhante à matriz de rastro de cobertura, explicitada em 2.1.2.

O usuário determina um ponto de validação através da opção “Create a Regression Test Check Point” no menu do projeto em questão, como exibido na figura 7.

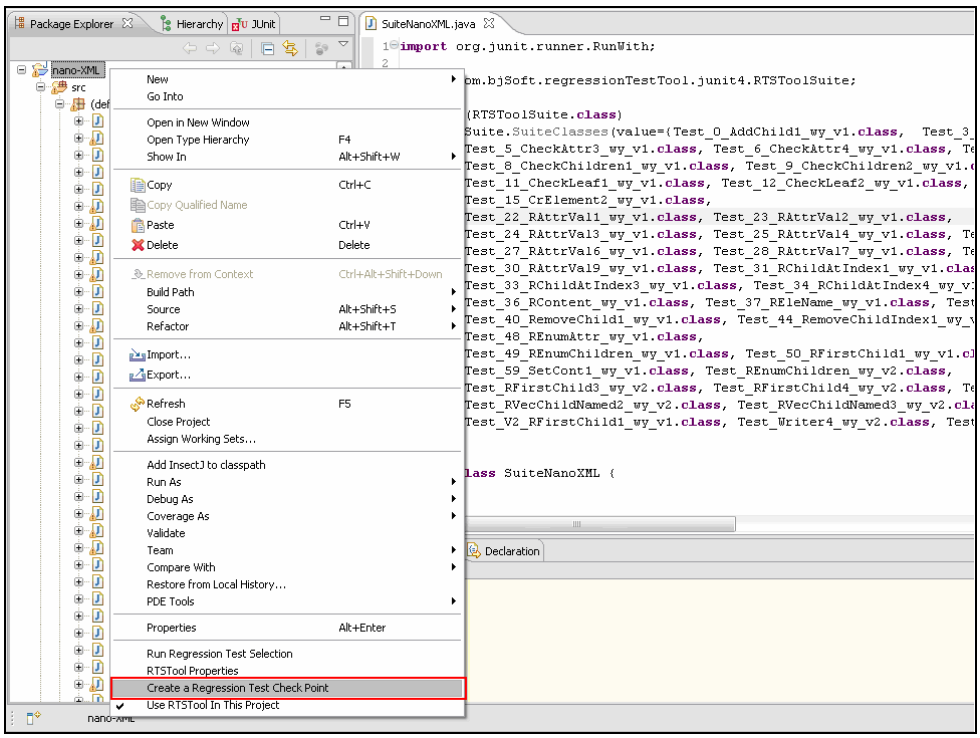

Figura 7 - Determinando um ponto de validação na RTSTool

2.2.4 Executando seleção de testes

Após a determinação de um ponto de validação e de uma alteração no código, a seleção dos testes pode ser executada. Para executar esta seleção, a RTSTool usa as informações obtidas na determinação do ponto de validação e de mudanças ocorridas no código.

No momento em que uma classe do projeto é modificada, um novo CFG é gerado para todos os métodos desta e comparado com os CFG´s existentes (construído na etapa de determinação do ponto de validação). Caso o algoritmo identifique mudanças nestes CFG´s, o método ao qual a mudança pertence será rotulado como entidade perigosa.

Para rod*ar a seleção pasta utilizar a opção* "Run Regression Test Selection*"*, figura 8. Depois de executada a seleção, os resultados se encontram na "Regression Test Selection Result View", figura 9, a qual contém uma lista com todos os testes da suíte, outra com os testes selecionados, o tamanho da suíte original, o tamanho da suíte reduzida e a porcentagem de testes selecionados.

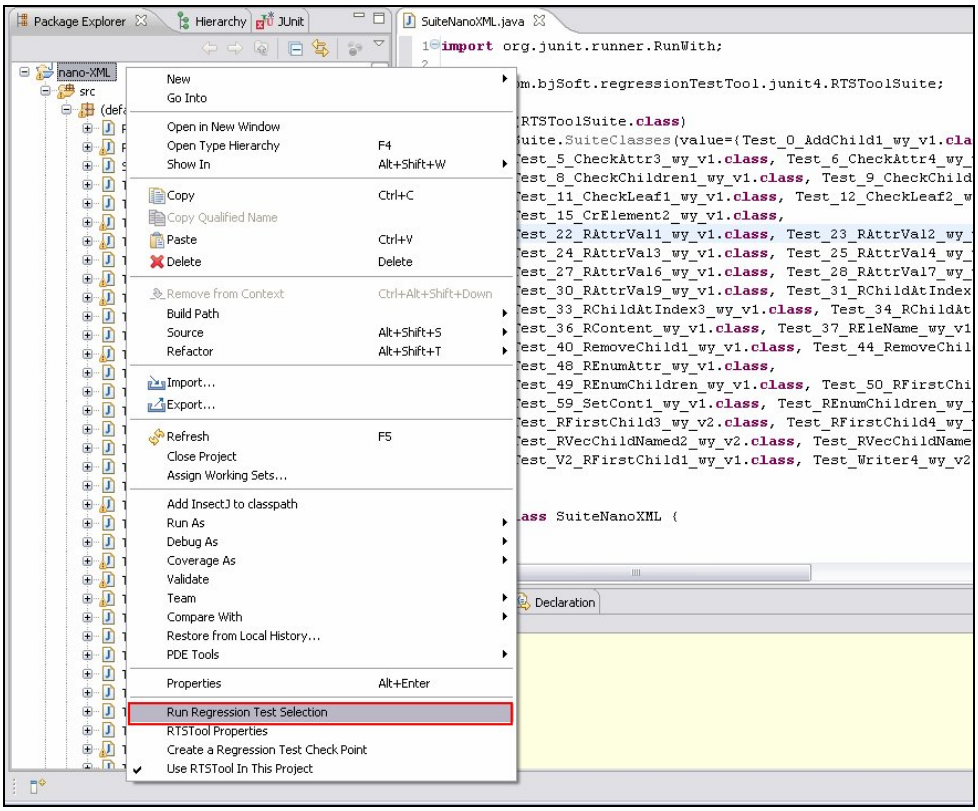

Figura 8 - Executando seleção de testes na RTSTool

Após a seleção, o usuário pode rodar apenas os testes selecionados ou rodar todos os testes. Para rodar apenas a suíte reduzida basta escolher a opção “Use JUnit with test selection”. Desmarcando esta opção o JUnit fica habilitado a rodar todos os testes da suíte.

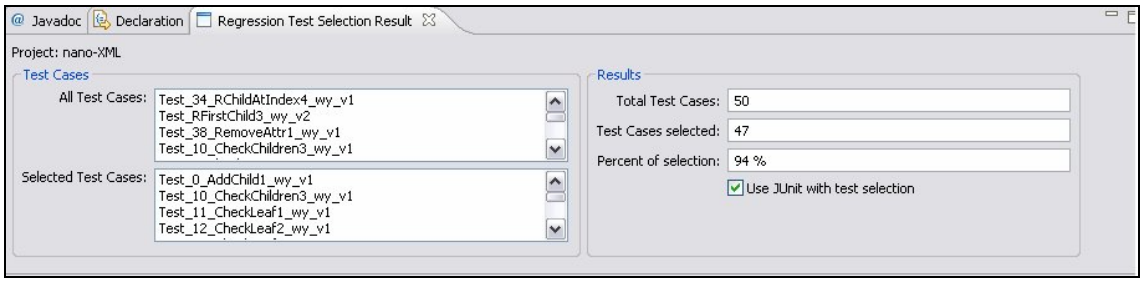

Figura 9 - Exibição dos testes selecionados

A figura 10 exibe a execução da suíte de testes usando a seleção de testes de regressão. Os três testes que não foram executados são aqueles não selecionados, sendo ignorados pela plataforma de execução do Junit.

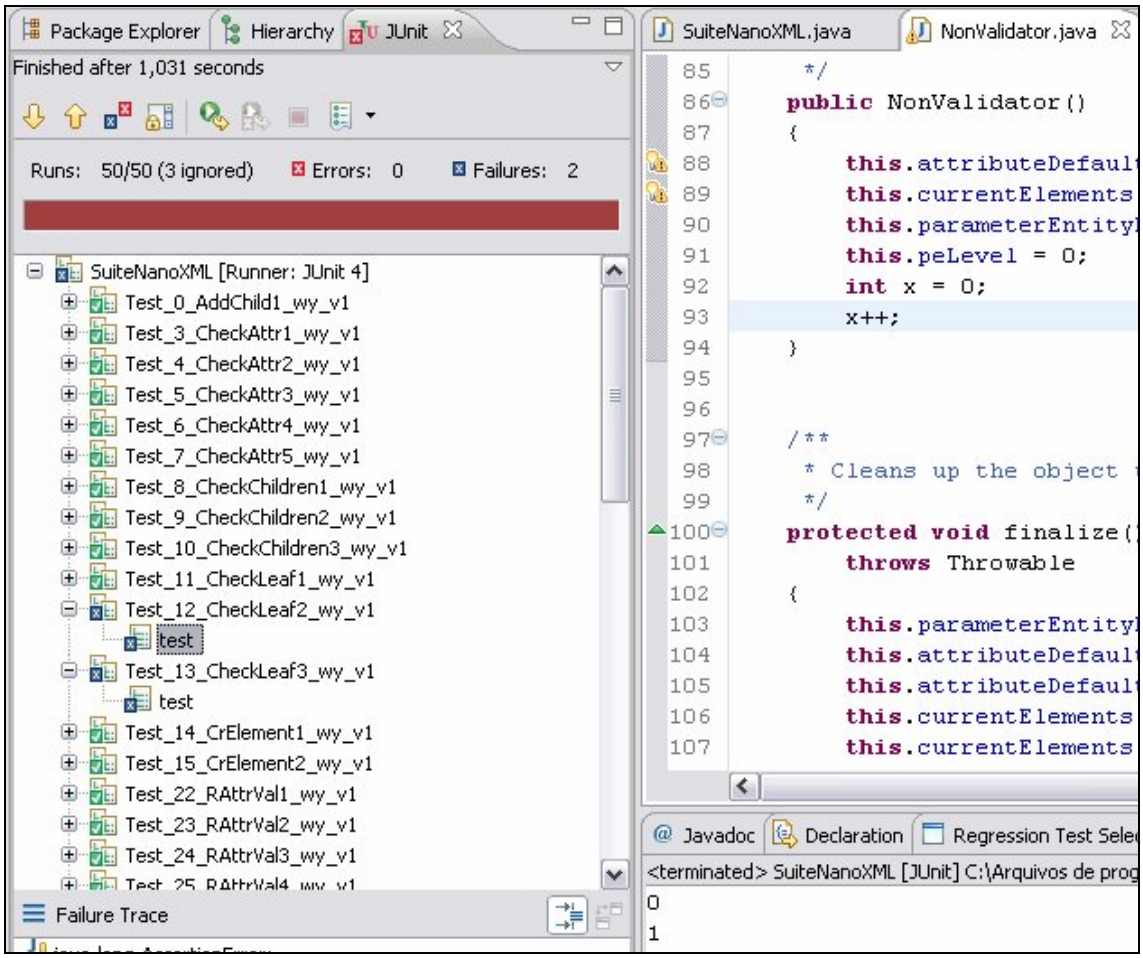

Figura 10 - Resultado de execução da suíte reduzida

3 Avaliação empírica

De modo a avaliar a eficiência da ferramenta em diversos aspectos, foram realizados experimentos nos quais a RTSTool foi usada para efetuar a seleção de testes de regressão em projetos do mundo real escritos na linguagem Java.

Os experimentos foram realizados sobre dois softwares: O analisador de texto *JTopas* (Java Token Parser) [9] e o processador de textos XML NanoXML [10].

Os dois softwares supracitados foram extraídos da iniciativa SIR [14], um repositório de artefatos construído com o propósito de dar suporte a desenvolvedores que desejem realizar experimentos rigorosos e controlados com relação à análise e testes de programas.

3.1 Estrutura de um artefato no SIR

Um projeto do SIR é constituído de várias versões seqüenciais de um software. Estas versões correspondem a versões de um projeto real.

Cada uma destas versões possui uma série de casos de teste associados, usados pelos desenvolvedores para testar a aplicação.

Estas versões possuem também um *script* que injeta modificações nos arquivos fonte do projeto. Estas falhas são artificiais, ou seja, não foram realmente cometidas por um programador ao longo do processo de codificação. Ao invés disso, elas são inseridas em partes específicas do código, com o objetivo de observar o comportamento da suíte de testes.

Estas injeções geram modificações no código que podem alterar o resultado da execução da suíte de testes, resultando em falhas ou erros em um subconjunto dos testes. No entanto, uma

injeção pode não produzir novos resultados na execução dos testes, significando que não havia nenhum teste da suíte que cobria aquela parte do código em particular.

Deste modo, usando várias versões de um artefato e também o *script* de injeção, é possível obter uma sequência de versões, e em cada uma destas, uma série de sub-versões. Cada uma destas sub-versões é referenciada por um identificador de falha.

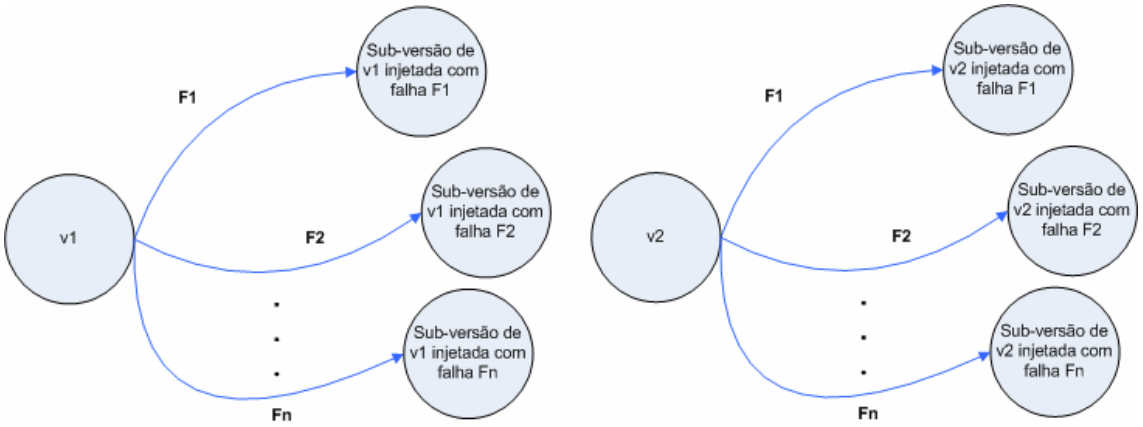

Figura 11 - Estrutura de um artefato no repositório SIR.
Uma versão v1 de um programa é disponibilizada em sua forma original e em n sub-versões através das injeções de falha F1, F2, ... Fn. Uma outra versão v2 também possui várias sub-versões obtidas através da injeção de diferentes falhas, F1, F2, ... , Fn.

3.2 Métricas de avaliação

Na execução dos experimentos, algumas métricas específicas foram perseguidas, de forma a avaliar a capacidade da RTSTool de se comportar bem em alguns parâmetros buscados na seleção de testes de regressão.

3.2.1 Precisão

A precisão de uma seleção de testes mede a fração de testes selecionados que indicam falhas.

Esta métrica indica a capacidade da seleção de apenas selecionar testes que sejam relevantes ao testador no processo de depuração de seu código.

Para calcular a precisão, deve-se efetuar a seleção de testes e, dentre os casos de teste escolhidos, deve-se observar a fração dos testes que falham.

Uma precisão de 100% (ou 1) indica que todos os testes selecionados falham em sua execução, enquanto que uma precisão baixa indica que dentre os casos de teste selecionados, poucos falham.

Uma definição mais formal de precisão é mostrada abaixo.

Definição (Precisão de uma seleção de testes de regressão):

Seja um programa P com uma suíte de testes S constituída de N casos de teste. Após uma modificação em P, é realizada uma seleção de n casos de teste (n <= N). Destes n casos de teste, p falham e q não falham, de modo que p + q = n. A precisão desta seleção é caracterizada como p/n.

3.2.2 Recall

A métrica do recall mede a capacidade de uma seleção de selecionar os testes que falham numa suíte de testes.

Um recall de 100% (ou 1) indica que todos os testes que falham na execução da suíte original foram selecionados.

Definição (Recall de uma seleção de testes de regressão):

Seja um programa P com uma suíte de testes S constituída de N casos de teste. Destes, P falham e Q executam com sucesso, de modo que P + Q = N. Após uma modificação em P, é realizada uma seleção de n casos de teste (n <= N). Destes n casos de teste, p falham e q não falham. O recall desta seleção é caracterizada como p/P.

Uma técnica de seleção segura é definida como aquela que seleciona todos os testes que podem detectar um erro. Logo ela deve selecionar todos aqueles testes da suíte original que falham, possuindo o recall com valor 1.

Existe também a possibilidade de não haver nenhum teste na suíte original que falhe. Neste caso, o valor do recall não é definido (pois usando a definição, haver-se-ia um denominador nulo), mas esta seleção também enquadra-se na definição de seleção segura.

3.2.3 Taxa de redução

A taxa de redução de uma seleção de testes é uma métrica relativamente simples, que visa indicar a redução do tamanho da suíte de testes selecionada em relação à suíte de testes do programa original.

Embora seja desejável que a taxa de redução de uma seleção seja a maior possível, uma alta taxa de redução não necessariamente significa uma seleção de boa qualidade.

Se numa seleção de testes obtida uma alta taxa de redução for obtida juntamente com baixos índices de precisão e recall, os testes obtidos não serão muito úteis no sentido de encontrar falhas no código.

Em termos de porcentagem, é possível definir a taxa de redução de uma seleção em termos formais.

Definição (Taxa de redução de uma seleção de testes de regressão):

Seja um programa P com uma suíte de testes S constituída de N casos de teste. Após uma modificação em P, é realizada uma seleção de n casos de teste (n <= N). A taxa de redução desta seleção é caracterizada como 1 – (n/N).

A precisão é uma métrica que está relacionada à corretude de uma seleção, enquanto que o recall está associado à completude da seleção.

De um modo geral, procura-se aumentar a precisão, o recall e a taxa de redução, a fim de obter seleções mais corretas, completas e eficientes.

3.3 Avaliação experimental 1- JTopas

A primeira série de experimentos foi realizada usando o projeto JTopas (*Java Token Parser*).

JTopas é uma biblioteca da linguagem Java usada para processar dados textuais. Suas classes e interfaces atualmente permitem o reconhecimento e processamento básicos de textos.

Processadores de texto de diferentes naturezas podem ser construídos usando esta biblioteca, como por exemplo, um

reconhecedor de textos de linha de comando, um leitor de arquivos, um interpretador de protocolo IP (*Internet protocol*) ou um leitor de páginas HTML (*Hyper Text Markup Language*) [9].

Esta biblioteca possui tal flexibilidade devido a configuração dinâmica de suas classes e separação bem delineada entre as tarefas de reconhecimento e processamento de texto.

3.3.1 Organização dos experimentos

O JTopas está organizado em 4 diferentes versões seqüenciais, cada uma delas possui um conjunto de casos de teste associado.

A tabela 2 relaciona as características de cada versão do projeto.

Organização do projeto JTopas		
Versão	**Quantidade de classes**	**Quantidade de casos de teste**
V0	17	8
V1	19	30
V2	21	32
V3	43	53

Tabela 2 - Organização do projeto JTopas

Para cada versão do JTopas (excetuando-se a v0) foi usado um *script* que altera partes do código fonte do programa, propositalmente inserindo pequenas falhas de lógica.

Estas falhas, como já foi frisado, são artificiais, ou seja, são inseridas de maneira calculada e ordenada, de modo a causar um erro de lógica numa parte do programa já prevista. Estas falhas não foram resultado de um processo natural de codificação por parte de um programador.

Desta maneira, para cada versão do JTopas a partir da v0, teremos outras várias sub-versões, geradas pela injeção de falhas no programa.

3.3.2 Realização dos experimentos

Para a avaliação empírica da ferramenta usando o JTopas, duas etapas distintas foram realizadas, de modo a analisar as métricas de interesse: Precisão, recall e taxa de redução da ferramenta.

Em cada versão do programa, a partir da v1 original (sem modificações de código), foram injetadas falhas de código nas classes do programa, obtendo assim sub-versões modificadas, identificadas pelo rótulo da falha que as gerou (F1, F2...Fn).

Procurou-se avaliar a qualidade da seleção que a RTSTool faz usando nesse caso as métricas de precisão e recall.

Primeiro, para cada sub-versão com injeção de falhas, catalogou-se a quantidade de falhas registradas usando toda a suíte de testes.

Após isso, a seleção de casos de teste da RTSTool foi feita, obtendo-se uma suíte reduzida. Esta suíte reduzida foi executada e registrou-se o número de falhas que nela ocorreram. Desta maneira foi possível calcular a precisão e o recall naquela seleção.

O processo acima descrito foi efetuado para todas as versões e sub-versões do JTopas. O resultado dos experimentos está detalhado abaixo na tabela 3.

Na tabela de resultados, procurou-se detalhar, além das constatações experimentais (Tamanho das suítes original e reduzida e falhas nas suítes original e reduzida), as métricas de precisão, recall e taxa de redução da seleção.

A taxa de redução encontra-se explicitada em parênteses, ao lado da indicação de tamanho da suíte reduzida.

Por fim, a coluna da extrema direita da tabela de resultados indica uma média aritmética das três métricas analisadas (taxa de redução, precisão e recall).

O processo de experimentação para diferentes versões do JTopas é no geral bastante semelhante. Uma diferença consiste nas injeções de falhas relativas às diferentes versões: São diferentes em número e afetam partes diferentes do código.

Por exemplo, para a versão 1 (v1), existem 10 injeções de falha, gerando um total de 10 sub-versões, enquanto que existem 12 injeções de falha para a versão 2 (v2) e apenas 8 para a versão 3 (v3).

Foram omitidos das tabelas de resultados os experimentos em que nenhum teste da suíte original falhava após uma modificação no código. Este tipo de resultado ocorreu com relativa freqüência, mas, não acrescentando nenhum debate de caráter científico aos experimentos, foram ignorados.

Nos experimentos da versão 1, por exemplo, esta situação ocorreu nas sub-versões F4, F7 e F8.

Experimentos JTopas (versão 1)							
ID falha	**Tamanho da suíte original**	**Falhas na suíte original**	**Tamanho da suíte reduzida**	**Falhas na suíte reduzida**	**Precisão**	**Recall**	**Média**
F1	30	2	3 (0.90)	2	0.66	1	0.85
F2	30	3	3 (0.90)	3	1	1	0.97
F3	30	2	2 (0.93)	3	1	1	0.98
F5	30	2	14(0.53)	2	0.143	1	0.56
F6	30	1	0 (1.00)	0	N/D	0	--
F9	30	3	3 (0.90)	3	1	1	0.97
F10	30	3	5 (0.83)	3	0.6	1	0.81
Experimentos JTopas (versão 2)							
ID falha	**Tamanho da suíte original**	**Falhas na suíte original**	**Tamanho da suíte reduzida**	**Falhas na suíte reduzida**	**Precisão**	**Recall**	**Média**
F1	32	2	3 (0.91)	2	0.66	1	0.86
F3	32	4	4 (0.88)	4	1	1	0.96
F7	32	1	1 (0.94)	1	1	1	0.98
F10	32	1	1 (0.94)	1	1	1	0.98
F11	32	1	1 (0.94)	1	1	1	0.98
F12	32	1	1 (0.94)	1	1	1	0.98
Experimentos JTopas (versão 3)							
ID falha	**Tamanho da suíte original**	**Falhas na suíte original**	**Tamanho da suíte reduzida**	**Falhas na suíte reduzida**	**Precisão**	**Recall**	**Média**
F4	53	2	20(0.62)	2	0.1	1	0.57
F7	53	1	2 (0.96)	1	0.5	1	0.82

Tabela 3 - Resultado dos experimentos para o JTopas

3.3.3 Análise dos resultados

Através dos resultados obtidos na seção anterior, podemos fazer análises mais detalhadas da qualidade da seleção de testes realizada pela ferramenta.

As linhas da tabela preenchidas na cor verde indicam que a ferramenta teve o comportamento ideal, com precisão e recall máximos.

De um modo geral, os experimentos mostram que a RTSTool possui um recall alto e precisão variável, embora em muitos casos baixa.

Isto pode ser explicado pela própria técnica de seleção que a ferramenta usa. Quando qualquer comando de um método é modificado na versão falha, a ferramenta marca o método inteiro como modificado.

Caso haja vários casos de teste cobrindo aquele método, todos eles serão selecionados. Isto pode acarretar na seleção de testes que não estão diretamente relacionados à falha, causando uma perda de precisão.

Os casos em que a ferramenta consegue uma precisão de 100% correspondem à situação em que apenas um caso de teste cobre o método ao qual a falha pertence. Nesta situação ocorre também que nenhum outro método coberto por algum caso de teste chama o método que possui a falha.

Já o recall de 100% em quase todos os casos em que esta métrica pode ser avaliada decorre do fato da RTSTool marcar o método em que a falha ocorreu como modificado. Se algum caso de teste da suíte original cobre o método, este caso será selecionado. Existe um experimento em que o recall não é de 100%, e este será analisado posteriormente.

Existem casos em que os testes não são capazes de capturar os defeitos da versão falha. Não se trata de uma situação incomum, e neste caso reportamos a precisão com valor 0 (zero) e recall com valor indefinido (--).

Para completar a análise, no restante desta seção serão realizadas análises dos resultados específicos de cada versão.

Versão 1

Nas versões falhas F4, F7 e F8 a suíte de testes não consegue detectar a falha no código alterado. No entanto, F4 e F8 são situações em que, apesar de introduzida uma falha, a RTSTool não considera nenhum método como alterado.

- Das dez situações analisadas, três reportam uma seleção ideal, de precisão e recall máximos (F2, F3 e F9).

- F6 é uma situação excepcional, na qual existe um teste que falha na suíte, mas a ferramenta não o seleciona. Isto acarreta no único experimento realizado no projeto JTopas no qual o recall não é máximo.

- RTSTool apresentou uma precisão bastante baixa em F4 pelo fato da modificação ter sido feita num método exaustivamente coberto pelos casos de teste, mas só causar falha numa pequena parcela destes.

- RTSTool teve uma precisão 0 (zero) em F7 pois é uma situação em que uma modificação no código não gera falhas na execução da suíte de testes, mas a

modificação no código ocorre em um método que é coberto por vários casos de teste.

- F1, F5 e F10 são situações mais comuns, em que a RTSTool seleciona apenas uma parcela dos testes que falham na suíte original, mostrando a pouca precisão da ferramenta. Isto é evidenciado principalmente em F5.

Com relação à situação excepcional encontrada em F6, foi observado que o algoritmo de comparação de CFG´s usado pela ferramenta não consegue capturar esta mudança em particular.

A situação retratada em F6 corresponde ao único caso encontrado nos experimentos no qual o recall não é igual a 1. Isto significa que existe um teste que falha na suíte original e este não é selecionado. Portanto, isto descaracteriza a seleção realizada pela RTSTool como segura, de acordo com a definição pré-estabelecida [16].

Versão 2

F2, F4, F5, F6, F8 e F9 são situações em que a suíte de testes não consegue detectar a falha no código alterado.

- Os resultados dos experimentos para a versão 2 são um pouco diferentes do que observamos na tabela passada. Devido à própria natureza dos casos de teste, os resultados foram bastante favoráveis à eficiência da ferramenta.

- F2, F7, F10, F11 e F12 foram identificados como seleções ideais.

- F1 é a única sub-versão na qual apenas uma parcela dos casos de teste da suíte que falham é selecionada.

Versão 3

Na maioria dos experimentos desta versão a suíte de testes não foi capaz de detectar a falha produzida. Apenas nos experimentos F4 e F7 isto não aconteceu.

- Na maioria dos experimentos desta versão a suíte de testes não foi capaz de detectar a falha produzida. Apenas nos experimentos F4 e F7 isto não aconteceu.

- A seleção realizada em F4 possui baixa precisão pois, assim como a seleção realizada em F4 na versão 1, a injeção de falha é feita num método com cobertura extensiva, mas que causa apenas poucos casos de teste a falhar.

- Novamente a baixa precisão e o alto recall da ferramenta são evidenciados, com destaque para a seleção feita em F4, com precisão de apenas 10%.

Redução da suíte de testes

O objetivo em avaliar a métrica da taxa de redução é determinar o quão a suíte de testes pode ser reduzida usando a técnica implementada pela RTSTool.

Para cada versão do JTopas, a partir da v1, foi executada a seleção de casos de teste usando a ferramenta. Os resultados destas seleções estão explicitados na tabela 4.

Seleção de testes para o projeto JTopas		
Versão	**Testes na suíte original**	**Quantidade de testes selecionados**
V1	23	21
V2	30	6
V3	32	10

Tabela 4 - Seleção de testes para o projeto JTopas

Dos resultados exibidos acima, pode-se facilmente calcular a taxa de redução de todas as seleções, considerando que as versões do JTopas são seqüenciais e usando a definição mostrada na seção 3.2.3 deste documento.

As taxas de reduções são mostradas na tabela 5 e ilustradas no gráfico abaixo (Figura 11).

Taxas de reduções para as seleções feitas no projeto JTopas	
Versão	**Taxa de redução obtida na seleção**
V1	8,7%
V2	80%
V3	68,75%

Tabela 5 - Taxas de reduções para as seleções feitas no JTopas

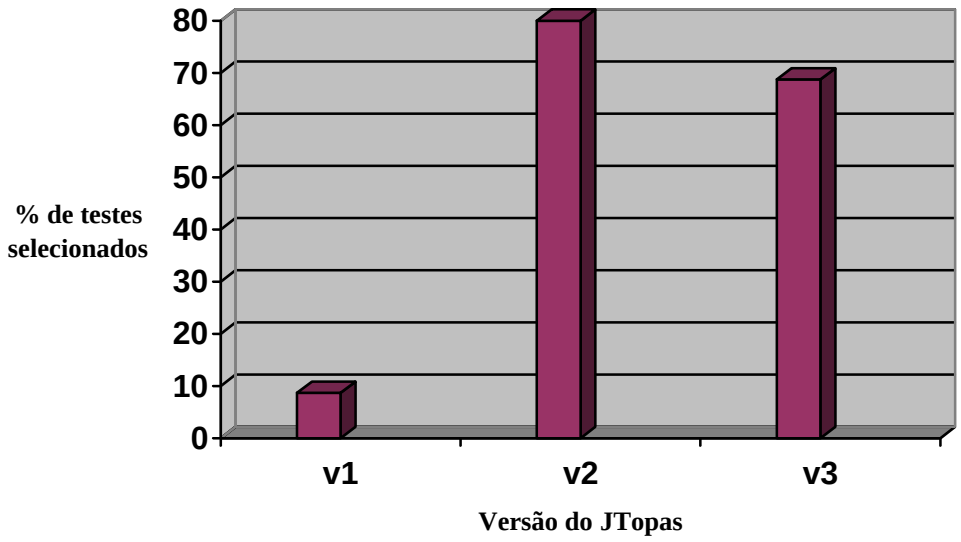

Figura 12 - Taxas de reduções para as seleções feitas no JTopas (Gráfico)

O gráfico mostra que a taxa de redução para as diferentes versões varia significativamente entre versões.

A baixíssima taxa de redução da versão 1 pode ser explicada pelo fato de que as mudanças de código que foram feitas para se chegar a esta versão foram quase todas cobertas por casos de teste.

Desta forma, para assegurar a corretude das mudanças, faz-se necessário a re-execução de quase todos os casos de teste.

O contrário pode ser observado no caso da segunda seleção, na qual as mudanças de código para se chegar a esta versão não foram expressivas o suficiente, de modo que apenas 20% dos casos de teste existentes na versão anterior foram selecionados.

Poder-se-ia afirmar que os casos de teste não estão cobrindo uma boa parcela das classes do projeto, por isso que o processo de seleção agrupou tão poucos itens.

Este não é o caso, pois o projeto JTopas, em todas as suas versões possui uma alta cobertura de métodos. Pode-se verificar este fato experimentalmente usando uma ferramenta de análise de cobertura, como por exemplo, o *emma* [13].

3.4 Avaliação experimental 2- NanoXML

A segunda série de experimentos foi realizada com o projeto NanoXML.

NanoXML é um pequeno processador de arquivos XML. É um software grátis, *open source*, sem interfaces gráficas e sem o uso de bibliotecas externas [10].

XML é uma meta-linguagem de marcação largamente utilizada para o armazenamento, estruturação e transporte de dados. Em função de suas características, tem desempenhado um papel de fundamental importância para a melhoria das condições de descrição das informações na Web.

3.4.1 Organização dos experimentos

O NanoXML está organizado em 6 diferentes versões seqüenciais, cada uma delas possui um conjunto de casos de teste associado.

Para cada versão (excetuando-se a v0) existe um *script*, semelhante ao usado no projeto JTopas, que altera partes do código fonte do programa, com o intuito de inserir falhas de lógica que sejam capturadas pelos testes.

Embora o SIR disponibilize 6 (seis) versões diferentes deste projeto, nos experimentos serão utilizadas apenas duas versões, a segunda e terceira versões considerando a ordem seqüencial (v1 e v2, respectivamente).

A tabela 6 relaciona as características das versões aproveitadas para os experimentos.

Organização do projeto NanoXML		
Versão	**Quantidade de classes**	**Quantidade de casos de teste**
V1	16	67
V2	19	50

Tabela 6 - Organização do projeto NanoXML

3.4.2 Realização dos experimentos

A realização dos experimentos para as versões do seguiu a mesma metodologia utilizada na experimentação com o projeto JTopas.

Para as versões v1 e v2 do NanoXML, foram injetadas modificações no código para causar falhas nos testes. Foram injetadas no total 7 (sete) falhas para v1 e mais 7 (sete) falhas para v2, resultando num total de 14 (quatorze) sub-versões.

As métricas analisadas foram exatamente as mesmas que foram observadas nos experimentos do JTopas: Precisão, recall, taxa de redução e uma média aritmética destas três.

Nesta série de experimentos, ao contrário do ocorrido com os experimentos do JTopas, o *script* de injeção de falhas não foi usado em todos os casos. De modo a evitar situações em que a suíte de testes não detecta falhas no código alterado, a maioria das falhas foi inserida manualmente.

A injeção de falhas manuais foi feita alterando trechos cruciais do código, de modo a forçar uma situação em que a suíte de testes original acusasse ao menos uma única falha. Isto foi realizado em todas as situações da tabela, com exceção de F7 na v1 e de F3 e F6 na v2.

Os experimentos F3 e F6 da versão 2 do NanoXML foram executados sem que a suíte de testes original falhasse uma única vez, portanto estas linhas foram ignoradas dos experimentos.

O resultado dos experimentos está detalhado na tabela 7.

Experimentos NanoXML (versão 1)							
ID falha	Tamanho da suíte original	Falhas na suíte original	Tamanho da suíte reduzida	Falhas na suíte reduzida	Precisão	Recall	Média
F1	67	63	63 (0.06)	63	1	1	0.69
F2	67	57	62 (0.07)	57	0.92	1	0.66
F3	67	1	1 (0.98)	1	1	1	0.99
F4	67	57	63 (0.06)	57	0.904	1	0.65
F5	67	63	63 (0.06)	63	1	1	0.69
F6	67	63	63 (0.06)	63	1	1	0.69
F7	67	1	63 (0.06)	1	0.016	1	0.36
Experimentos NanoXML (versão 2)							
ID falha	Tamanho da suíte original	Falhas na suíte original	Tamanho da suíte reduzida	Falhas na suíte reduzida	Precisão	Recall	Média
F1	50	47	47 (0.06)	47	1	1	0.69
F2	50	47	47 (0.06)	47	1	1	0.69
F4	50	8	50 (0.00)	8	0.16	1	0.39
F5	50	4	43 (0.14)	4	0.09	1	0.41
F7	50	2	3 (0.94)	2	0.66	1	0.87

Tabela 7 - Resultados dos experimentos para o NanoXML

3.4.3 Análise dos resultados

Nesta série de experimentos, como todas as falhas foram injetadas com o objetivo de causar pelo menos uma falha detectável pelos testes, não houve nenhuma situação em que ocorreram 0 (zero) falhas na suíte original.

Esta série de experimentos continua a demonstrar o que havia sido observado nas primeiras experimentações: A ferramenta RTSTool possui uma precisão irregular e um recall com valor 1 em quase todas as situações. Excetuam-se aquelas em que o recall não é definido ou situações excepcionais, como pode ser observado na sub-versão F6 da versão 1 para o projeto JTopas.

O natureza do projeto NanoXML difere um pouco daquela observada no JTopas.

No JTopas, as classes do projeto possuíam relativamente poucos casos de teste associados. Ou seja, geralmente um método

era coberto apenas por alguns casos de teste. Isto acarretava na seleção de poucos casos de teste após a injeção de uma falha.

Este fato pode ser facilmente observado na tabela 2 (Resultados dos experimentos do JTopas. Nos resultados, é visível que a taxa de redução é geralmente bastante alta.

Já no caso do projeto NanoXML, existem poucas classes e muitos casos de teste. As classes são exaustivamente cobertas pelos casos de teste. São comuns situações em que um mesmo método é coberto por vários casos de teste.

Esta natureza do NanoXML acarreta uma baixa taxa de redução nas seleções, já que após uma modificação num método, uma boa quantidade de casos de teste serão selecionados.

No entanto, ao mesmo tempo em que há uma diminuição significativa da taxa de redução, ocorre um aumento notável na precisão. Isto é facilmente explicável pois como um método é coberto por um grande número de casos de teste, uma falha neste método causa a falha de todos os métodos que o cobrem.

A experimentação com o projeto NanoXML mostrou um número excessivamente alto de seleções consideradas ideais (com recall 1 e precisão 1), principalmente na primeira versão (v1).

A análise feita aqui será complementada com o levantamento de alguns pontos observados nas versões experimentadas.

Versão 1

- Todos os casos apresentados possuem alta precisão, com a exceção de F7. Isto ocorre pois esta modificação causa falha em apenas um dos testes.

- F3 é a situação mais próxima do ideal possível, com um valor de média de 0.99. É uma situação na qual existe apenas um caso de teste para o método modificado, e

esta modificação causa uma falha que é detectada pela ferramenta.

Versão 2

- F4 e F5 são exceções à análise feita nesta seção. Tratam-se de situações em que um método é coberto por muitos casos de teste, mas a modificação do código causa falha em apenas alguns deles.

4 Conclusão

Seleção de testes de regressão é uma atividade que contribui bastante para o desenvolvimento de software de qualidade.

Testar o software após cada modificação realizada provê garantia ao desenvolvedor da diminuição da possibilidade de erros.

Nesse contexto, este trabalho apresenta um estudo empírico objetivando a análise de resultados de uso de uma ferramenta de seleção de testes de regressão em projetos reais.

4.1 Principais constatações

O uso da ferramenta RTSTool em projetos reais acarreta numa seleção de testes de regressão de boa qualidade com relação ao parâmetro de recall.

Isto significa que, excetuando-se situações extraordinárias, a seleção de testes efetuada pelo RTSTool selecionará todos os testes que falham na suíte de testes após realizada uma modificação no código do programa.

Com relação aos parâmetros de taxa de redução (fração dos testes que foram ignorados na seleção) e precisão (quantidade dos testes selecionados que indicam uma falha no programa), pode-se dizer que estes são variáveis de acordo com a natureza do projeto.

Foi possível constatar, por exemplo, que nas experimentações com o projeto JTopas, a taxa de redução era bastante significativa, enquanto que nas experimentações com o projeto NanoXML, a taxa de redução era geralmente baixa. Isto se deve ao fato do JTopas apresentar pouca cobertura das classes (nenhum ou poucos casos de teste cobrem os métodos da aplicação) enquanto que no NanoXML os métodos da aplicação geralmente são cobertos por uma porção de casos de teste.

Experimentalmente, foi possível constatar que a situação na qual a RTSTool faz a melhor seleção é aquela em que os testes são confeccionados de maneira a estabelecer uma relação de apenas um teste por método da aplicação. Este tese deve ser capaz de detectar qualquer falha que possa ocorrer naquele método.

Tal constatação pôde ser observada na realização dos experimentos com o projeto NanoXML, na versão 1. A linha F3 indica uma situação ideal, em que existe apenas um teste cobrindo o método modificado, e este teste falha após a injeção de uma falha lógica no código.

4.2 Perspectivas

Com a conclusão deste trabalho, é possível vislumbrar uma série de possibilidades de complementação e expansão do mesmo, tanto no tocante à possíveis melhorias da ferramenta como em possíveis experimentações e avaliações a qual esta pode ser submetida.

A lista abaixo evidencia alguns possíveis trabalhos futuros relacionados à RTSTool.

- Mudança no método de cobertura da RTSTool, considerando mudanças ao nível de *statements* (linha de código)

- Aperfeiçoamento do algoritmo de comparação de CFG´s, de modo a caracterizar a técnica de seleção realizada pela ferramenta como segura.

- Geração de casos de teste automática para aqueles métodos identificados sem cobertura

- Aperfeiçoamento do algoritmo de comparação para CFG´s (Control Flow Graphs)

- Aperfeiçoamento da interface com o usuário

- Experimentação com uma gama maior de projetos e um número maior de versões

- Experimentação com softwares que possuam *bugs* naturalmente introduzidos, evitando-se o estudo com falhas artificiais

- Experimentação com um projeto de grande porte, com uma quantidade significativamente maior de classes e casos de teste

Referências

[1] HARROLD, Mary Jean; JONES, James A.; LI, Tongyu; LIANG, Donglin; Regression Test Selection for Java Software, Proceedings of the ACM Conference on OO Programming, Systems, Languages, and Applications (OOPSLA '01), 2001.

[2] Desikan, Srinivasan, A test methodology for an effective regression testing, www.stickyminds.com, acesso em 28/09/2008.

[3] G. Rothermel and M. J. Harrold. Empirical studies of a safe regression test selection technique. IEEE Transactions on Software Engineering, June 1998.

[4] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. IEEE Transactions on Software Engineering, June 1997.

[5] T.L. Graves, M.J. Harrold, J-M Kim, A. Porter and G. Rothermel. An empirical study of regression test selection techniques. 20th International Conference on Software Engineering, April 1998.

[6] G. Rothermel, M.J. Harrold, and J. Dedhia. Regression test selection for C++ programs. Journal of Software Testing, Verification, and Reliability, June 2000.

[7] B. Bezerra. Um Plug-in do Eclipse© para seleção de teste caixa-branca. Trabalhos de graduação, Cin UFPE, Novembro 2008.

[8] Eclipse – an open development platform, www.eclipse.org , acesso em 03/09/2008.

[9] JTopas – Java tokenizer and parser tools, http://jtopas.sourceforge.net/jtopas/, acesso em 01/11/2008.

[10] NanoXML 2.2.1 – a small non-validating parser for Java, http://nanoxml.sourceforge.net/orig/, acesso em 01/11/2008.

[11] Artigo sobre o Eclipse IDE, http://www.planet-index.org/blog/software/eclipse-most-popular-java-ide.html, acessado em 02/11/2008.

[12] Bcel – the byte code engineering library, http://jakarta.apache.org/bcel/, acesso em 12/10/08.

[13] Emma – a free Java code coverage tool, http://emma.sourceforge.net/, acesso em 19/10/2008.

[14] SIR – Software-artifact infrastructure repository, http://sir.unl.edu/portal/index.html, acesso em 09/09/2008.

[15] An early look at JUnit 4, http://www.ibm.com/developerworks/java/library/j-junit4.html?ca=dgr-lnxw01JUnit4, acesso em 11/11/2008.

[16] G. Rothermel, M.J. Harrold. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, Aug 1997.