



Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

RAFAEL MORAES DE OLIVEIRA

O PARADIGMA DA COBERTURA DE TESTE COMO MÉTRICA DE
QUALIDADE DE SOFTWARE

ASSIS

2012

RAFAEL MORAES DE OLIVEIRA

O PARADIGMA DA COBERTURA DE TESTE COMO MÉTRICA DE QUALIDADE DE SOFTWARE

Projeto de pesquisa apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Municipal de Ensino Superior de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando: Rafael Moraes de Oliveira

Orientador: Fernando César de Lima

ASSIS

2012

FICHA CATALOGRÁFICA

Oliveira, Rafael Moraes

O paradigma da cobertura de teste como métrica de qualidade de software / Rafael Moraes de Oliveira. Fundação Educacional do Município de Assis – FEMA – Assis, 2012.

68p.

Orientador (a): Fernando César de Lima.

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA.

1. Cobertura de teste. 2. Métrica de qualidade. 3. Teste. 4. Desenvolvimento de software.

CDD: 001.6

Biblioteca da FEMA

PARADIGMA DA COBERTURA DE TESTE COMO MÉTRICA DE QUALIDADE DE SOFTWARE

RAFAEL MORAES DE OLIVEIRA

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis , como requisito do Curso de Bacharelado em Ciência da Computação, analisado pela seguinte comissão examinadora:

Orientador: Fernando César de Lima

Analisador (1): Alex Sandro Romeo de Souza Poletto

ASSIS

2012

RESUMO

A cobertura de teste é uma técnica que auxilia na produção de software de alta qualidade e cada vez mais, mais desenvolvedores estão adotando essa prática. Porém ao fazer uso da cobertura alguns cuidados devem ser tomados, pois a cobertura de teste quando usada de maneira incorreta acarreta em uma falsa métrica de qualidade. Esse trabalho auxilia o desenvolvedor a fazer um melhor uso da cobertura de teste, apontando seus pontos fracos e fortes através de um estudo realizado em um ambiente real de desenvolvimento de software.

Palavras-chave: cobertura de teste; métrica de qualidade; teste; desenvolvimento de software.

ABSTRACT

Test coverage is a technique that helps in the production of high quality software and increasingly more developers are adopting this practice. But to make use of coverage some care must be taken because the test coverage when used incorrectly leads to a false metric of quality. This work helps developers to make better use of test coverage, pointing out their strengths and weaknesses through a study conducted in a real software development.

Keywords: test coverage, quality metrics, testing, software development.

LISTA DE ILUSTRAÇÕES

Figura 1 – Estrutura dos diretórios do Emma-2.0.53.12.....	25
Figura 2 – Emma instalado com sucesso.....	26
Figura 3 – Comando instrumentação <i>on-the-fly</i>	34
Figura 4 – Comando instrumentação <i>offline</i>	35
Figura 5 – Conteúdo arquivo elemma-1.5.3.zip.....	37
Figura 6 – Conteúdo diretório Eclipse.....	38
Figura 7 – Botão EclEmma.....	39
Figura 8 – Aba EclEmma.....	39
Figura 9 – Menu Eclipse.....	40
Figura 10 – Importando os dados da cobertura.....	41
Figura 11 – Selecionando o arquivo de cobertura.....	42
Figura 12 – Classe1.....	43
Figura 13 – Classe2.....	44
Figura 14 – Classe Principal.....	45
Figura 15 – Dados da Cobertura.....	46
Figura 16 – Caso de alto percentual de cobertura.....	49
Figura 17 – Código morto.....	52
Figura 18 - Caso de execução parcial.....	53

LISTA DE TABELAS

Tabela 1 – Opções comando run.....	27
Tabela 2 – Opções comando instr.....	29
Tabela 3 – Opções comando report.....	31
Tabela 4 – Opções comando merge.....	32
Tabela 5 – Cronograma.....	55

LISTA DE ABREVIATURAS E SIGLAS

FEMA	Fundação Educacional do Município de Assis
HTML	Hyper Text Markup Language
ONG	Organização Não Governamental
TI	Tecnologia da Informação
TXT	Arquivo de texto simples
XML	Extensible Markup Language
SO	Sistema Operacional

SUMÁRIO

1. INTRODUÇÃO.....	12
1.1. Objetivo Geral.....	12
1.2. Objetivos Específicos.....	13
1.3. Justificativa.....	13
1.4. Motivação.....	14
1.5. Perspectivas de Contribuição.....	14
1.6. Metodologia de Trabalho.....	14
1.7. Recursos Necessários.....	15
1.7.1. Recursos Bibliográficos.....	15
1.7.2. Recursos de Hardware.....	15
1.7.3. Recursos De Software.....	15
1.8. Estrutura de Trabalho.....	16
2. ENGENHARIA DE SOFTWARE.....	18
3. TESTE DE SOFTWARE.....	20
4. COBERTURA DE TESTE.....	22
5. FRAMEWORK EMMA.....	24
5.1. Instalando O Emma.....	25
5.2. Comandos Emma.....	27
5.3. Comando run.....	27
5.4. Comando instr.....	29
5.5. Comando report.....	31
5.6. Comando merge.....	32
5.7. Realizando a Instrumentação em Aplicações <i>Desktop (on-the-fly)</i>	33
5.8. Realizando a Instrumentação em Aplicações Web (<i>offline</i>).....	34
6.ECLEMMA.....	36
6.1. Instalação.....	36

7. COMO ANALISAR OS DADOS DA COBERTURA.....	43
8. EXPERIMENTO.....	47
8.1. <i>Software</i>	47
8.2. Detalhes Experimento.....	47
8.3. Análise dos dados.....	48
8.4. Benefícios.....	51
8.5. Desvantagens.....	53
9. CONCLUSÃO.....	54
10. CRONOGRAMA.....	55
11. REFERÊNCIAS.....	56
12. APÊNDICE A – ENTREVISTAS.....	57
12.1. Entrevista Fernando César de Lima.....	57
12.2. Entrevista Guilherme de Cleva Farto.....	58
12.3. Entrevista Eduardo Negrão.....	60
12.4. Entrevista Luiz Angelo Franciscatti.....	61
12.5. Entrevista Celso Yamaguti Sobral.....	62
12.6. Entrevista Mariana Budiski De Lima.....	63
12.7. Entrevista Vinícius Dias Oliveira.....	65

1. INTRODUÇÃO

No cenário atual da engenharia de *software*, os testes aplicados sobre um *software* são tão importantes quanto o processo de desenvolvimento, porém ao testar um *software*, é necessário medir sua eficácia e uma forma de conseguir isso é com a cobertura de teste. Existem diversos *frameworks* que auxiliam nessa tarefa, basicamente a cobertura de teste verifica as linhas de código-fonte executadas durante a aplicação do teste, e através dessa verificação cria-se uma medida que pode ser usada como métrica para criar padrões de qualidade, muitos consideram que quanto maior a métrica atingida, melhor também será a qualidade do *software*. Essa metodologia é bastante adotada pelas atuais empresas desenvolvedoras de *software*, mais até onde isso é válido? Seria correto afirmar que o *software* está completamente livre de erros e que atende satisfatoriamente os requisitos do cliente só pelo fato do *software* ter 100% de cobertura? Ou afirmar que o mesmo é falho por não ter atingido percentual máximo? Ou realizar obrigatoriamente os testes até atingir o percentual mínimo?

Esse trabalho busca responder essas e mais questões relacionadas a cobertura de testes, através dos dados coletados a partir de um ambiente real de desenvolvimento de *software*.

1.1. OBJETIVO GERAL

Este trabalho objetiva aplicar a cobertura de teste com o auxílio do *framework* Emma sobre um ambiente real de desenvolvimento de *software*, fazer uma análise detalhada dos resultados obtidos e através dessa análise criar parâmetros para auxiliar no uso correto e sensato da técnica de cobertura de teste independente da linguagem de programação ou do *framework* escolhidos nesse estudo.

1.2. OBJETIVOS ESPECÍFICOS

- a) Explicar o funcionamento do Emma.
- b) Criar um tutorial de como aplicar o Emma sobre qualquer aplicação desenvolvida em Java nos Sistemas operacionais Windows e GNU/Linux.
- c) Ensinar como instalar na IDE Eclipse o Plug-in EclEmma, necessário para visualizar os dados resultantes da cobertura na própria IDE.
- d) Mostrar como interpretar de forma correta os dados gerados pela cobertura de teste.
- e) Por em prática a cobertura de teste usando o Emma em um ambiente de desenvolvimento real.
- f) Levantar questões sobre até onde é válido os resultados da cobertura.
- g) Buscar soluções para essas questões baseado nas experiências e dados recolhidos durante a pesquisa.

1.3. JUSTIFICATIVA

Toda documentação sobre engenharia de software e especialmente o tópico cobertura de teste, revisada até o momento, afirmam que os resultados dos testes obtidos a partir do uso de *frameworks* não são absolutos e que não podem e nem devem substituir os testes de inspeção de código, que é realizado por uma equipe de profissionais de TI devidamente treinados, que analisam minuciosamente o código fonte em questão, linha à linha buscando erros de implementação e de levantamento de requisitos.

No material revisado, a afirmação termina aí, não é explicado e nem exposto, com detalhes, os motivos pelos quais os resultados obtidos da cobertura de teste, não são absolutos, e nem sempre obter 100% de cobertura de teste é sinônimo de

software de alta qualidade.

Esse trabalho visa por a prova essas afirmações, em um ambiente real de desenvolvimento de *software*, e expor as questões, falhas, e soluções resultantes.

1.4. MOTIVAÇÃO

A cobertura de teste é um tema polemico no mundo dos desenvolvedores de *software*, basta uma simples pesquisa realizada na Internet para encontrar centenas de discussões em fóruns sobre esse assunto, mais a grande maioria das afirmações encontradas não possuem embasamento científico, então a surgiu a motivação para realizar esse estudo.

1.5. PERSPECTIVAS DE CONTRIBUIÇÃO

Agregar conhecimento para aqueles que desejam se aprofundar na engenharia de *software*, criar um guia que auxilia o uso da cobertura de teste da forma mais correta possível, ensinar como usar o *framework* Emma integrado a IDE Eclipse, contribuindo diretamente para a comunidade Java e entusiastas da engenharia de *software*.

1.6. METODOLOGIA DE TRABALHO

A metodologia utilizada para o desenvolvimento desse estudo, é a pesquisa bibliográfica de trabalhos acadêmicos, tutoriais sobre cobertura de teste e obras que abordam o assunto como de, Ian Sommerville, um gigante da engenharia de *software*.

O estudo ira coletar dados de um ambiente real de desenvolvimento de *software*, para posterior analise e exposiçao dos resultado obtidos.

1.7. RECURSOS NECESSÁRIOS

1.7.1. Recursos Bibliográficos

Artigos, livros, tutoriais, monografias, que abrangem a cobertura de teste.

1.7.2. Recursos De Hardware

Processador: Dois núcleos de 1.8 ghz.

Memoria RAM: 4 Gigas DDR2

HD: Sata 320 Gigas

Monitor: LCD de 15,4'

1.7.3. Recursos De Software

IDE Eclipse Hélios

Oracle jdk versão 1.6.0_25

Apache Tomcat versão 7.0.26

Emma versão 2.0.5312

Hibernate 4.0

Banco de dados relacional Mysql versão 14.12 Distrib 5.0.51a

Plug-in Eclipse EclEmma versão 1.5.3

Sistema Operacional GNU/Linux

1.8. ESTRUTURA DE TRABALHO

O presente trabalho é composto por 9 capítulos, da forma como segue:

Capítulo 1: Introdução

Na introdução é feita uma breve descrição das questões que se pretende abordar e discutir ao decorrer do trabalho.

Capítulo 2: Engenharia de *Software*

Nesse capítulo é abordado a engenharia de *software*, uma breve introdução sobre seu início e definição.

Capítulo 3: Teste de *Software*

O capítulo discute os conceitos do teste de *software*.

Capítulo 4: Cobertura de Teste

O capítulo explica os conceitos da cobertura de teste.

Capítulo 5: Tutorial Emma

Nesse capítulo será abordado especificamente o *framework* Emma contando sua história e explicando detalhadamente como instalar, executar todas as funções disponíveis e realizar a leitura dos dados coletados.

Capítulo 6: EclEmma

Nesse capítulo será explicado o que é o EclEmma, como fazer a integração com a IDE Eclipse e como usá-lo.

Capítulo 7: Analisar os dados da cobertura

Capítulo explica como é feita a análise dos coletados durante a cobertura.

Capítulo 8: Experimento

Detalhes de como se deu o experimento são tratados nesse capítulo, detalhes como a descrição do *software* alvo dos testes, o método utilizado nos testes, problemas encontrados, benefícios etc...

Capítulo 9: Conclusão

As repostas encontradas após a conclusão do experimento, serão expostas nesse capítulo.

2. ENGENHARIA DE SOFTWARE

Segundo Sommerville (2003, p.05):

A engenharia de *software* é uma disciplina da engenharia que se ocupa de todos os aspectos da produção de *software*, desde os estágios iniciais de especificação do sistema até manutenção desse sistema, depois que ele entrou em operação.

Cabe aos engenheiros de software fazerem os produtos funcionarem, são eles que selecionam as teorias, métodos e ferramentas nas situações apropriadas e sempre buscam solucionar problemas, mesmo quando não existem teorias e métodos de apoio aplicáveis. Os engenheiros também reconhecem a necessidade de trabalhar dentro das restrições organizacionais e financeiras e, assim, buscam soluções que estejam de acordo com essas restrições. Mas a engenharia de *software* não se limita só aos processos técnicos de desenvolvimento de *software*, ela também se estende a atividades como, o gerenciamento de projetos de *software*, o desenvolvimento de ferramentas, métodos e teorias que auxiliem a produção de *software*.

A engenharia de *software* se assemelha as demais engenharias como por exemplo a engenharia civil, segundo Martins (2007 , p.01):

Em muitos aspectos engenharia de *software* pode ser comparada à engenharia civil. Na engenharia civil o projeto, em geral, tem os seguintes

passos: contexto, arquitetura, projeto técnico, construção e validação. Similarmente, num projeto de construção de *software* as etapas são: engenharia de sistema, análise de requisitos, projeto técnico, construção e validação.

A engenharia de *software* é uma disciplina relativamente nova, de acordo com Sommerville (2007, p.04):

A noção de engenharia surgiu pela primeira vez em 1968, em uma conferência organizada para discutir a chamada 'crise do *software*'. Essa crise resultava diretamente da introdução (naquela época) do poderoso *hardware* de computador de terceira geração.

A capacidade de tal *hardware* tornou possível aplicações de computador até então inimagináveis, o *software* resultante desse *hardware* era maior e mais complexo que os sistemas de *softwares* precedentes.

Em suma a engenharia de *software* surgiu com o aumento da complexidade dos softwares e abrange toda a vida útil dos software e sempre prima por qualidade, redução de custo e agilidade o que a torna indispensável ao cenário tecnológico atual.

3. TESTE DE SOFTWARE

Em um projeto de desenvolvimento de *software* uma das atividades que mais consomem recursos é o processo de testes de *software* segundo Martins (2007, p.15):

Um *software* precisa ser testado para descobrir erros que foram feitos durante o projeto e a construção. Os testes são conduzidos através de uma estratégia, que integra os métodos de teste, os passos e os roteiros. A estratégia de teste deve ser planejada sob medida para cada projeto, considerando o tempo que será investido neste trabalho, a disponibilidade de recursos e a tecnologia utilizada na construção do *software*.

Os teste fazem parte do processo de verificação e verificação (V & V), os testes para detecção de defeitos tem a finalidade de expor defeitos latentes em um *software* que ainda está em fase de desenvolvimento constituindo parte do processo de verificação.

Um bom teste de detecção de defeitos deve fazer que o sistema opere de forma incorreta e como consequência, exponha os defeitos existentes, o teste tem que demonstrar a presença e não a ausência de defeitos no *software*.

Também existem os testes de validação, esses se atentam a verificar se o *software* está sendo construído de acordo com a especificação e se atende as necessidades do cliente.

Os testes se dividem no geral em 3 tipos, Teste de caixa-branca (*white-box-testing*) que são teste que consideram os mecanismos internos de um *software* ou

componente também conhecido como teste estrutural e teste de caixa transparente (*glass-box-testing*). Teste de caixa-cinza (*gray-box testing*), teste que ignora os mecanismos internos de um componente e focaliza apenas as saídas geradas em resposta as entradas e condições de execução selecionadas. Teste de caixa-preta (*black-box testing*) os testes que se encaixam nessa modalidade são os que ignoram os mecanismos internos de um sistema e focaliza apenas as saídas geradas em resposta a entradas e condições de execução selecionadas.

Como visto, existem diversos métodos para se testar um *software*. Os testes podem ser feitos manualmente ou de forma automatizada através de códigos escritos pelos próprios programadores envolvidos no desenvolvimento do *software* a ser testado. Inspeções de código também são possíveis ou a adoção de um *framework* disponível no mercado para apoiar essa tarefa, cabe ao engenheiro de *software* decidir qual sera ou quais serão os métodos e ferramentadas adotados.

4. COBERTURA DE TESTE

A cobertura de teste consiste em definir um valor ou valores que indiquem qual foi a abrangência de um teste aplicado sobre um *software*. Os resultados obtidos através da cobertura representam um *feedback* de grande importância aos desenvolvedores de *software*.

Basicamente a cobertura de teste se divide em dois seguimentos, a cobertura baseada em requisitos, e a baseada em códigos a qual será abordada nesse trabalho. A cobertura de teste baseada em códigos mede a quantidade de linhas de código executados durante a aplicação do teste, em relação a quantidade de código não executado, essa modalidade de cobertura pode ser baseada em fluxos de controle (instrução, ramificação ou caminhos) ou fluxos de dados.

A Cobertura sobre fluxo de controle, objetiva testar linhas de código, condições (*if, else, while, for*) de ramificação, caminhos que percorrem o código ou outros elementos do fluxo de controle do *software*.

Na Cobertura sobre fluxo de dados, desta vez o objetivo é testar se os estados dos dados são válidos durante a execução do *software*, por exemplo, se um objeto é declarado antes de ser usado, ou mesmo se está recebendo um tipo de dado correspondente a ele.

A cobertura de teste sobre código gera seus resultados usando a seguinte equação:

Resultado cobertura de teste = $I / Tlic$

Onde **I** é o numero de itens executados expressos como instruções, ramificações e caminhos de código, pontos de decisão do estado de dados ou nomes de elementos de dados. **Tlic** é o número total de itens no código.

Com o resultado dessa equação é possível gerar um percentual que nos possibilita declarar que X% dos casos de teste (**I**) obtiveram uma taxa de sucesso de Y% nessa cobertura de teste.

Os resultados obtidos podem ser comparados a critérios pré-definidos de sucesso e caso os critérios não forem atingidos, também é possível calcular a quantidade de esforço restante para atingir os requisitos definidos.

Hoje em dia, existem diversos *frameworks* que nos possibilita efetuar a cobertura e dentre esses *frameworks* está o Emma, o qual será usado nesse estudo devido as suas virtudes: ser *open-source*, uso simplificado, fácil integração com a IDE Eclipse e seu baixo impacto no tempo de execução do *software* a ser testado.

5. FRAMEWORK EMMA

O *framework* Emma foi desenvolvido por Vladimir Roubtsov e disponibilizado no www.sourceforge.net no ano de 2004 no portal oficial do projeto <http://emma.sourceforge.net>. Pioneira entre as ferramentas de cobertura de teste *open-source* para java. O Emma é um *framework* construído em java, de instalação simples e funciona em modo texto, para usá-lo será necessário acessar o terminal, console ou *prompt* do seu SO.

O Emma tem seu funcionamento baseado em instrumentação de *bytecodes* ou seja, ele lê o conteúdo dos “.class” da aplicação e faz alterações nos mesmos. Essas alterações podem ser realizadas em tempo de execução (*on-the-fly*) ou em alguns casos como em aplicações WEB é necessário instrumentar o seu *classpath* antes da execução, procedimento conhecido como instrumentação *offline*. A instrumentação das classes é o que torna possível ao *framework* realizar seu trabalho, que é identificar quais linhas do código fonte foram executadas durante a realização do teste.

Os dados gerados pelo Emma são calculados a partir da quantidade de linhas de código que foram executadas durante a execução da aplicação. O Emma divide a sua análise em classes, métodos, blocos, e linhas.

No momento da instrumentação o Emma gera um arquivo de metadados que recebe por padrão o nome de *coverage.em*, mas caso seja necessário é possível alterar seu nome. Quando a aplicação é encerrada outro arquivo é gerado o *coverage.ec*. Nesse arquivo está contido todos os dados que foram coletados pelo Emma.

Além de medir a abrangência do teste aplicado, o *framework* nos possibilita exportar os dados resultantes da cobertura (*coverage.em*, *coverage.ec*) para os seguintes formatos de arquivos html, txt e xml, facilitando a visualização da métrica de cobertura dos seus testes. Outra função importante do Emma é a possibilidade de mesclar os dados de diferentes arquivos de cobertura, com isso é possível

modularizar a aplicação dos testes caso o *software* em questão seja de grande porte ou por algum outro motivo que faça ser necessário a reaplicação do teste. Todas as habilidades descritas até aqui serão demonstradas mais adiante.

5.1. INSTALANDO O EMMA

Esse tutorial assume que já exista alguma versão do Java instalado no SO, o primeiro passo é fazer o download do *framework* no seguinte link: <http://sourceforge.net/projects/emma/files/emma-release/2.0.5312/emma-2.0.5312.zip/download>.

Dentro do arquivo se encontra o diretório emma-2.0.5312 e dentro desse diretório temos mais três diretórios, o “doc” onde se encontra a documentação, “examples” onde são encontrados alguns exemplos de uso disponibilizados pelo mantenedor do *framework* e um diretório “lib” onde estão os arquivos emma.jar e emma_ant.jar. Na Figura 1 é possível visualizar a estrutura dos diretórios.

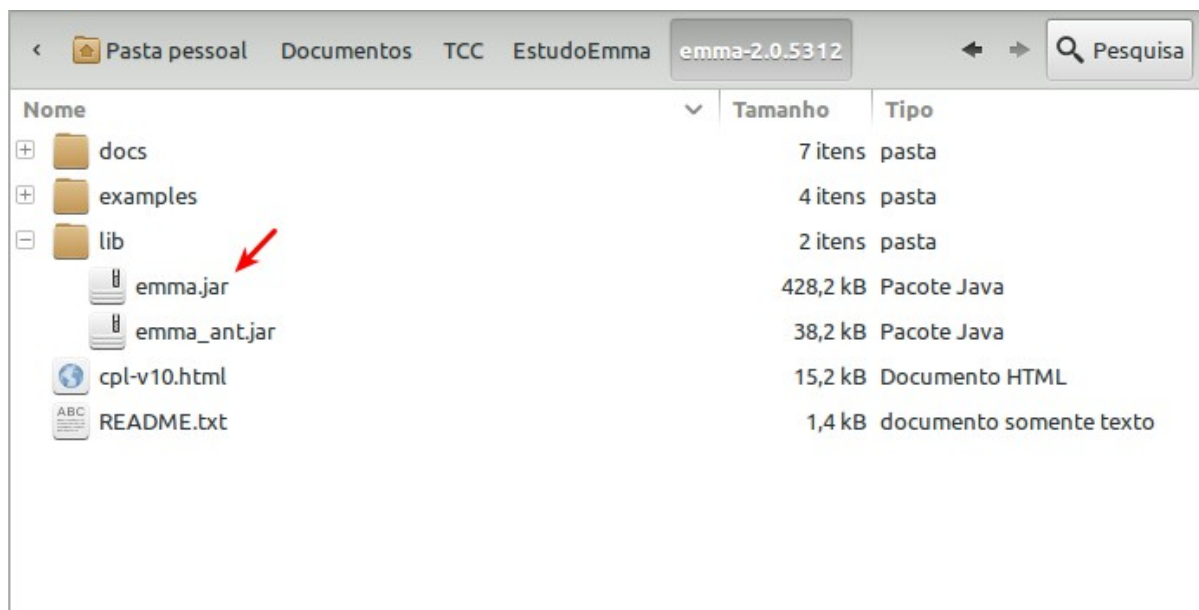
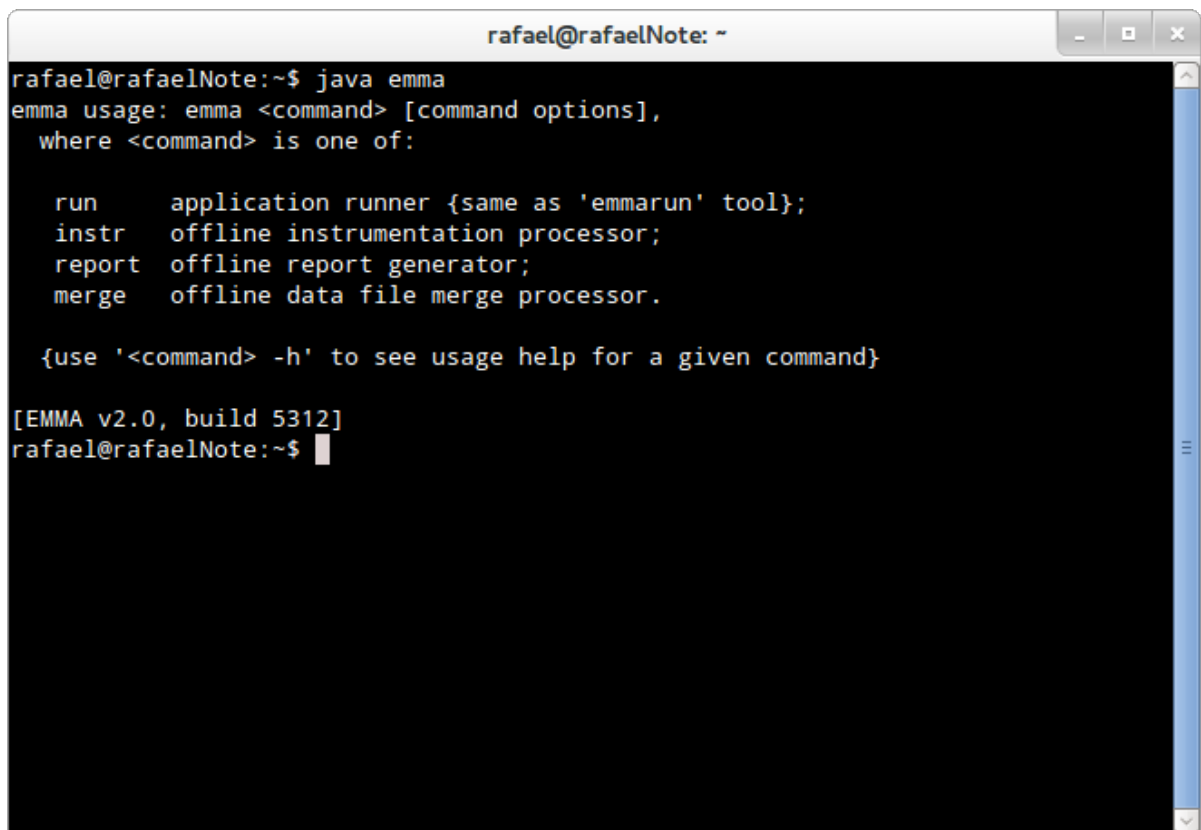


Figura 1 – Estrutura dos diretórios do Emma-2.0.53.12.

Para prosseguir a instalação copie o arquivo `emma.jar` para o diretório “ext” da sua jvm como o SO usado na criação do tutorial é uma distribuição GNU/Linux a jvm se encontra no seguinte caminho “`/usr/lib/jvm/java-6-openjdk-i386/jre/lib/ext`”, se o SO for Windows e o Java tenha sido instalado de forma padrão o caminho será “`C:\Arquivos de programa\Java\jre6\lib\ext`”. Após copiado o jar `emma.jar` para o diretório “ext” da sua jvm, abra o terminal/console caso esteja utilizando o SO GNU/Linux ou o prompt caso esteja usando o Windows, a partir desse ponto trataremos qualquer um deles apenas como terminal, pois não existem diferenças significativas entre eles.

No terminal digite 'java emma' (sem as aspas) e aperte [enter], se os passos anteriores foram executados corretamente aparecerá uma tela conforme apresentada na Figura 2.



```
rafael@rafaelNote: ~  
rafael@rafaelNote:~$ java emma  
emma usage: emma <command> [command options],  
  where <command> is one of:  
  
  run      application runner {same as 'emmarun' tool};  
  instr   offline instrumentation processor;  
  report  offline report generator;  
  merge   offline data file merge processor.  
  
  {use '<command> -h' to see usage help for a given command}  
  
[EMMA v2.0, build 5312]  
rafael@rafaelNote:~$
```

Figura 2 – Emma instalado com sucesso.

Se o texto retornado for diferente ao da Figura 2, é sinal que não houve sucesso na instalação, refazer o procedimento com maior atenção deverá resolver o problema.

5.2. COMANDOS EMMA

Na Figura 2 pode - se ver quatro comandos suportados pelo Emma são eles, run, instr, report e merge. Cada comando executa uma função diferente e cada um tem um conjunto de opção que tornam possível o uso da ferramenta.

5.3. COMANDO RUN

O “run” é o comando usado para fazer a instrumentação *on-the-fly*, ou seja em tempo de execução. A Tabela 1 mostra todas as opções reservadas ao comando emma run.

Opção	Descrição
-cp, -classpath	Opção usada quando a cobertura será aplicada em arquivos zip/jar ou <i>classpath</i> . Ex: java emma run -cp <caminho do <i>classpath</i> ou jar/zip>
-jar	Usado para informar ao <i>framework</i> que a cobertura será realizada sobre um jar executável.

Opção	Descrição
	<p>Ex: java emma run -jar <caminho do jar></p>
-f, -fullmetadata	<p>Informando o caminho do <i>classpath</i> faz a cobertura até das classes que normalmente não tem necessidade de serem cobertas, exemplo classes de <i>frameworks</i> como hibernate, spring, etc...</p> <p>Ex: java emma run -f <caminho <i>classpath</i>></p>
-ix, -filter	<p>Essa opção possibilita criar um filtro, definindo manualmente quais classes de um <i>classpath</i> serão instrumentadas.</p> <p>Ex: java emma run -ix {parametro1, parametro2}</p>
-r, -report	<p>Opção usada para selecionar o formato de dados do relatório de cobertura no caso txt, html, xml.</p> <p>Ex: java emma run -cp <<i>classpath</i>> -r <tipo de dado></p>
-raw	<p>Opção informa ao Emma para criar os dados da sessão</p> <p>Ex: java emma run -raw -cp <caminho <i>classpath</i>></p>
-out, -outfile	<p>Opção usada para definir o arquivo que receberá os dados da cobertura por padrão o arquivo é o coverage.ec.</p> <p>Ex: java emma run -raw -out <caminho>.ec -jar <caminho jar></p>
-merge	<p>Opção usada para mesclar os resultados de coberturas diferentes</p>

Opção	Descrição
	da mesma aplicação gerando o arquivo coverage.es Ex: <code>java emma merge -in coverage.em -in coverage.ec -out coverage.es</code>
-h, -help	Quando usada essa opção ele exibe uma manual de ajuda ao usuário Ex: <code>java emma run -h</code>

Tabela 1 – Opções comando run

5.4. COMANDO INSTR

Faz-se uso do comando instr quando existe a necessidade de se fazer a instrumentação offline do código fonte, ou no caso do Emma dos “.class” da aplicação a ser testada, geralmente usa-se o comando instr em aplicação WEB onde a instrumentação *on-the-fly* não é possível, nunca instrumente a aplicação original pois quando o Emma realiza a instrumentação offline ele faz alterações permanentes nos class da aplicação, sendo assim sempre instrumente uma cópia da aplicação original. A Tabela 2 lista as principais opções disponíveis do comando em questão.

Opção	Descrição
-ip, -cp, -instrpath	Opção usada para instrumentar um classpath, zip/jar, informando o caminho para o arquivo como parâmetro.

	<p>Ex: <code>java instr -ip /caminho/software.jar</code></p>
-d, -dir, -outdir	<p>Usamos essa opção para informar qual diretório receberá o software depois de instrumentado.</p> <p>Ex: <code>java instr -ip /caminho/software.jar -d /caminho/diretorioInstrumentado</code></p>
-out, -outfile	<p>Informa ao Emma qual será o nome do arquivo de metadados resultante da instrumentação, caso a opção não seja usada o <i>framework</i> cria esse arquivo com o nome <code>coverage.em</code> por padrão.</p> <p>Ex: <code>java instr -ip /caminho/software.jar -d /caminho/diretorioInstrumentado -out coberturaDeTeste.em</code></p>
-merge	<p>Essa opção tem a finalidade de mesclar os resultados da cobertura atual com os dados de uma cobertura anterior, desde de que sejam coberturas realizadas no mesmo software.</p> <p>Ex: <code>java instr -ip /caminho/software.jar -d /caminho/diretorioInstrumentado -merge /caminho/coberturaDeTeste.em</code></p>
-ix, -filter	<p>Define quais classes de um <i>classpath</i> serão ou não serão instrumentadas.</p> <p>Ex: <code>java emma run -ix {parametro1, parametro2}</code></p>
-h, -help	<p>Exibe um breve guia de uso do comando em questão.</p>

	Ex: java emma instr -h
--	------------------------

Tabela 2 – Opções comando instr

5.5. COMANDO REPORT

O comando report é utilizado para manipular os dados da cobertura, ao fazer uso desse comando é possível, exportar os dados de cobertura que estão armazenados no coverage.ec ou no coverage.es para três tipos de dados, html, xml e txt. A Tabela 3 descreve em detalhes as opções suportadas pelo comando report.

Opção	Descrição
-in, -input	Opção usada para informar para o <i>framework</i> qual ou quais arquivos de cobertura serão exportados. Ex: java emma report -in /caminho/cobertura.ec
-r, -report	Essa opção informa ao Emma quais ou qual tipo de dado ele deve exportar os dados resultantes da cobertura. Ex: java emma report -in /caminho/cobertura.ec -r html
-sp, -sourcepath	Quando temos todo um diretório com arquivos de cobertura é possível usar essa opção para informar ao <i>framework</i> que o diretório todo deve ser exportado para o tipo de dado informado

	<p>com a opção -r ou -report</p> <p>Ex: java emma report -sp /caminho/meuSourcePath -r xml</p>
-h, -help	<p>Usando essa opção é possível visualizar um manual no próprio terminal sobre o comando report.</p> <p>Ex: java emma report -h</p>

Tabela 3 – Opções comando *report*

5.6. COMANDO MERGE

O comando `merge` tem a finalidade de tornar possível a união de dois arquivos de cobertura, isso é de grande valia pois suponha que o *software* a ser testado é muito grande para ter todos os testes realizados em um mesmo dia, ou simplesmente o programador esqueceu de testar determinada função do *software*, em ambos os casos ao finalizar o *software* o arquivo resultante da cobertura será criado e ao executar o *software* novamente para aplicar o teste ou testes restantes outro arquivo de cobertura será gerado e é aí que deve-se por o comando `merge` em ação pois o mesmo tem a capacidade de mesclar o resultado das duas coberturas em um único arquivo. O comando `merge` vai além da união “física” dos arquivos. Internamente ele funde os dados de todas as coberturas, ou seja, as linhas de código caso uma linha do código não tenha sido executada na primeira cobertura mais na segunda essa linha foi executada no arquivo resultante do `merge` ela constará como executada e com isso todas os percentuais dos arquivos também serão mesclados. A Tabela 4 descreve todas as opção disponíveis ao comando.

Opção	Descrição
-in, -input	Torna possível a informar ao <i>framework</i> quais serão os arquivos de cobertura alvos do merge. Ex: <code>java emma merge -in cobertura1.ec cobertura2.ec</code>
-out, -outfile	Opção usada quando queremos modificar o nome do arquivo resultante do merge, caso não seja feito seu uso será criado por padrão um arquivo de nome <code>coverage.es</code> Ex: <code>java emma merge -in cobertura1.ec cobertura2.ec -out cobertura1+2.es</code>
-h, -help	Ao fazer uso dessa opção um manual de uso do comando <code>merge</code> é exposto no próprio terminal. Ex: <code>java emma merge -help</code>

Tabela 4 – Opções comando *merge*

5.7. REALIZANDO A INSTRUMENTAÇÃO EM APLICAÇÕES *DESKTOP* (*ON-THE-FLY*)

Para simplificar o processo de instrumentação exporte a aplicação para um arquivo “jar” executável que basicamente é um arquivo compactado, abra o terminal e digite o seguinte comando: `java emma run -raw -out <caminho do diretório>/cobertura.ec -jar <caminho do diretório>/nome_seu_jar.jar` e aperte [enter]. Se tudo funcionar corretamente, os arquivos `cobertura.ec` e `cobertura.em` serão criados no diretório

informado através da opção `-out`. A Figura 3 mostra como ficará o comando usado para instrumentar o projeto que será usado como exemplo no decorrer do Capítulo 10.

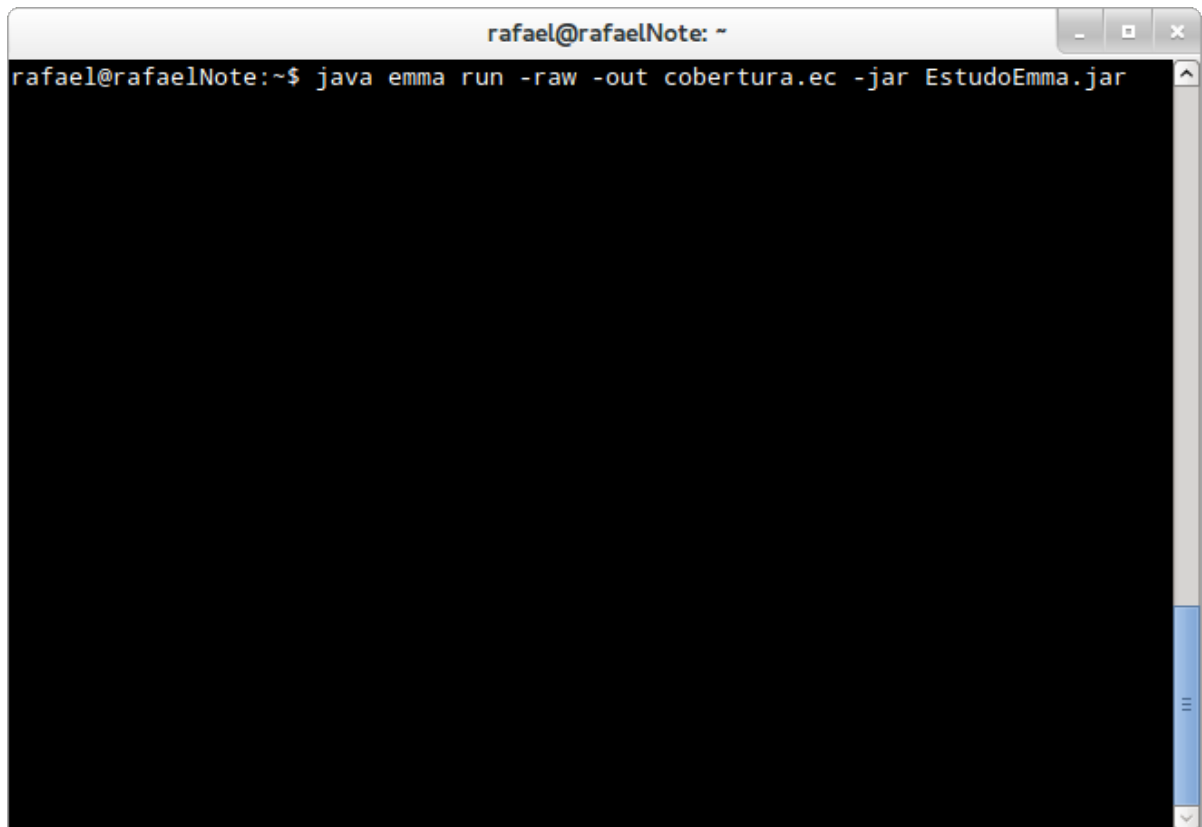
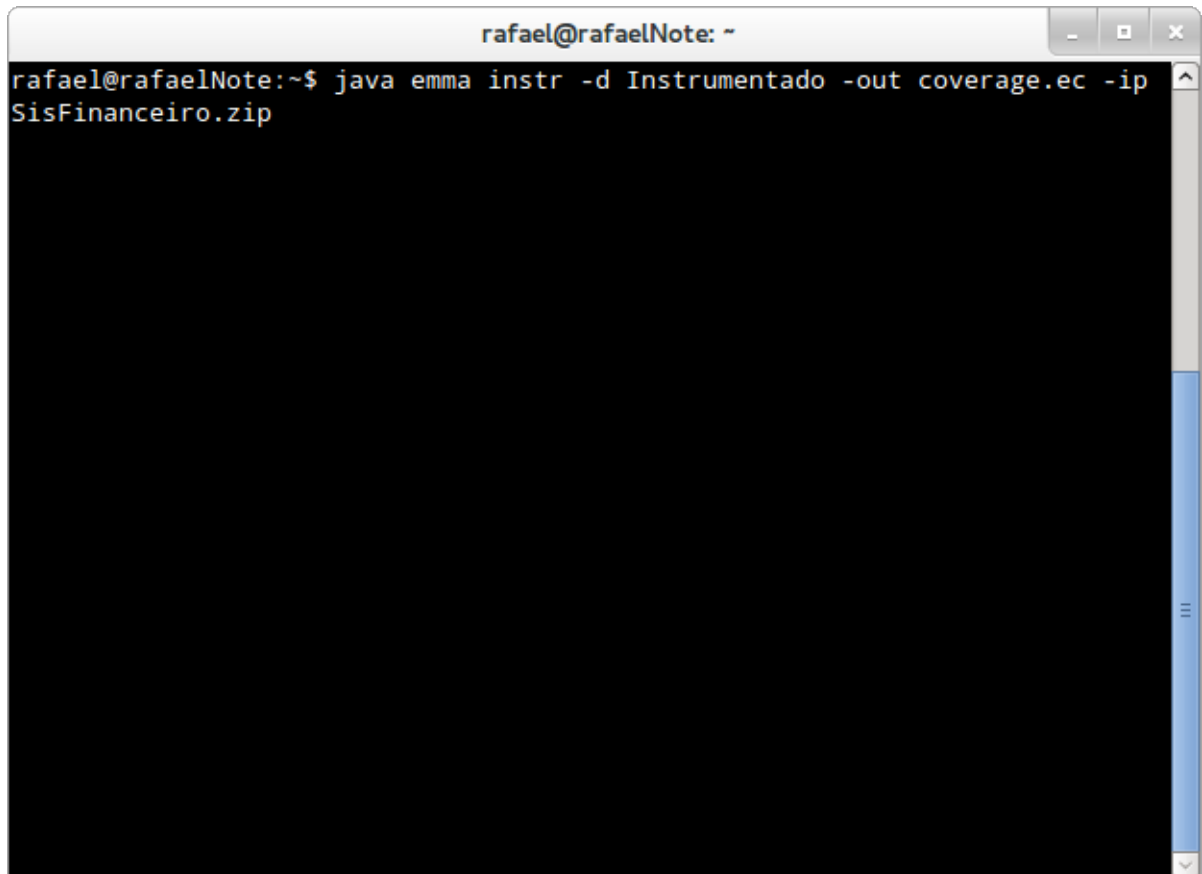
A screenshot of a terminal window titled "rafael@rafaelNote: ~". The terminal shows the command "java emma run -raw -out cobertura.ec -jar EstudoEmma.jar" being entered at the prompt "rafael@rafaelNote:~\$". The terminal background is black, and the text is white. The window has standard Linux window controls (minimize, maximize, close) in the top right corner.

Figura 3 – Comando instrumentação *on-the-fly*

5.8. REALIZANDO A INSTRUMENTAÇÃO EM APLICAÇÕES WEB (*OFFLINE*)

Novamente para simplificar, exporte o seu projeto web para um pacote war, mude a extensão de arquivo.war para arquivo.zip abra o terminal e digite o seguinte: `java emma instr -d <caminho diretório> -ip <caminho diretório>/seu_projeto.zip` e aperte [enter], após a conclusão da instrumentação os `.class` instrumentados foram criados e salvos no diretório informado na opção `-d`. Agora extraia o conteúdo do arquivo.zip e substitua os `.class` originais pelos `.class` instrumentados pelo Emma. Agora faça o

deploy da aplicação instrumentada no Tomcat, após a execução da aplicação o arquivo de cobertura estará armazenado no diretório “bin” do Tomcat. A Figura 4 exibe o comando necessário para instrumentar a aplicação que será usada no desenvolvimento desse estudo.

A screenshot of a terminal window titled "rafael@rafaelNote: ~". The terminal shows a command being entered: "rafael@rafaelNote:~\$ java emma instr -d Instrumentado -out coverage.ec -ip SisFinanceiro.zip". The command is split across two lines. The terminal has a black background with white text. The window title bar includes standard Linux window controls (minimize, maximize, close) and the user's name and host information. The terminal output is currently empty, showing only the command prompt and the entered command.

```
rafael@rafaelNote:~$ java emma instr -d Instrumentado -out coverage.ec -ip  
SisFinanceiro.zip
```

Figura 4 – Comando instrumentação *offline*

6. ECLEMMMA

EclEmma um *framework* para cobertura de código Java para a IDE Eclipse, disponível sob a licença pública Eclipse. Originalmente o EclEmma foi baseado no Emma, nesse trabalho será usado os EclEmma para auxiliar na análises dos dados coletados pelo Emma na bancada do Eclipse, o *framework* ira colorir em verde o *background* das linhas de códigos executadas em amarelo as linhas executadas parcialmente e de vermelho as linhas que não foram executadas além de exibir os dados padrão da coleta do Emma.

6.1. INSTALAÇÃO

Antes de instalar é necessário fazer o download do EclEmma versão 1.5.3 nesse endereço:

http://sourceforge.net/projects/eclEmma/files/01_EclEmma_Releases/1.5.3/eclEmma-1.5.3.zip/download, a versão não deve ser superior a 1.5.3 pois versões mais recentes do *Plug-in* deixaram de ser compatíveis com os dados coletados pelo Emma.

Apos o término do download, dentro do arquivo pode-se encontrar dois diretórios, *plugins* e *features* ambos visualizados na Figura 5.

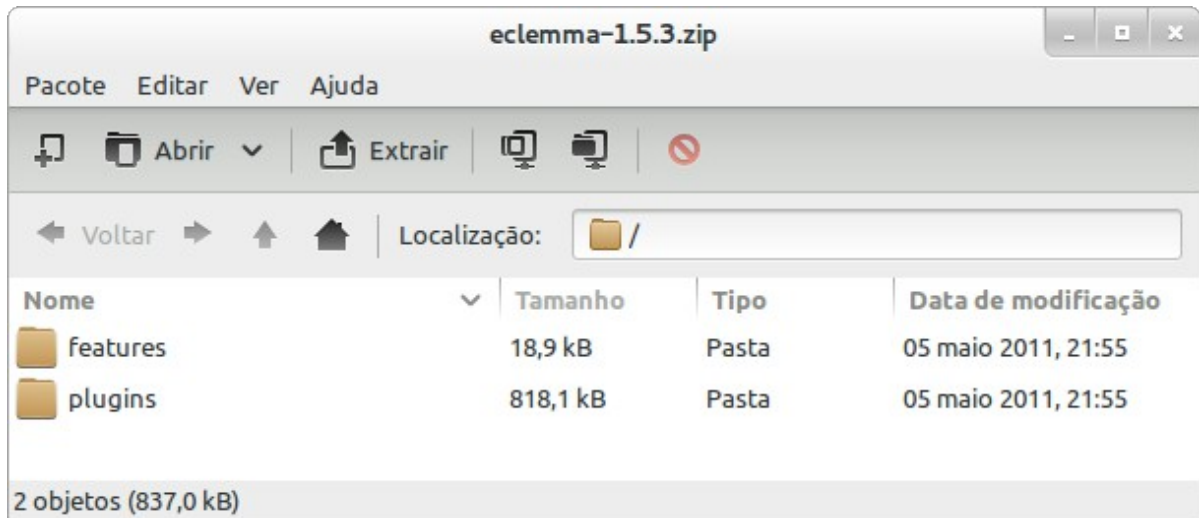


Figura 5 – Conteúdo arquivo eclEmma-1.5.3.zip

Copie os dois diretórios visualizados na Figura 5 e navegue até o diretório onde se encontra instalado seu Eclipse, dentro deste diretório existem dois diretórios que também possuem o nome *features* e *plugins* como pode ser visto na Figura 6.



Nome	Tamanho	Tipo
+ about_files	6 itens	pasta
+ apache-tomcat-7.0.27	11 itens	pasta
+ configuration	8 itens	pasta
+ dropins	0 item	pasta
+ features 	205 itens	pasta
+ p2	3 itens	pasta
+ plugins 	1.042 itens	pasta
+ readme	1 item	pasta
about.html	18,9 kB	Documento HTML
artifacts.xml	320,0 kB	Documento XML
eclipse	63,0 kB	executável
eclipse.ini	425 bytes	Configuration Settings
epl-v10.html	16,5 kB	Documento HTML
icon.xpm	9,0 kB	imagem XPM
libcairo-swt.so	266,2 kB	biblioteca compartilhada
notice.html	9,1 kB	Documento HTML

Figura 6 – Conteúdo diretório Eclipse

Cole o conteúdo copiado anteriormente, de modo a mesclar os diretórios *features* e *plugins* do EclEmma com os diretórios já existentes no diretório do Eclipse.

Agora é só abrir o Eclipse e mais um botão foi adicionado a barra de ferramentas como é possível ver na Figura 7.

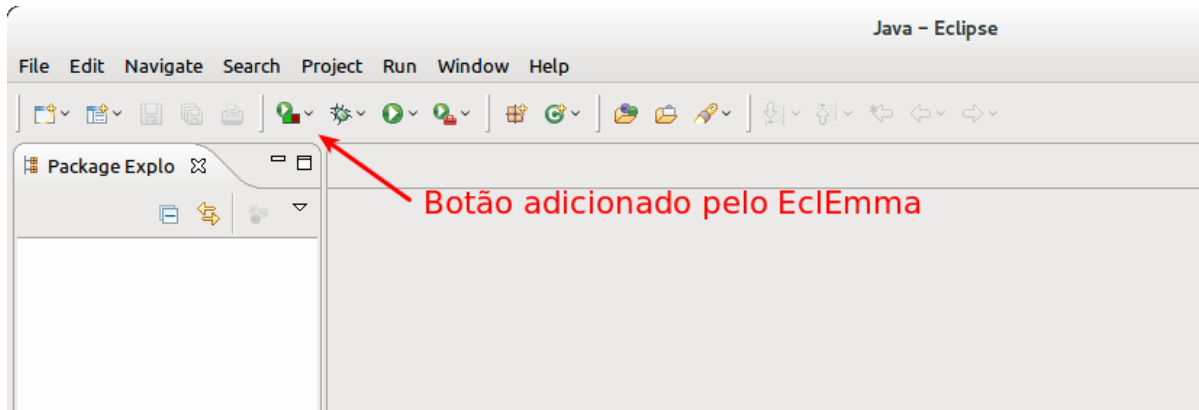


Figura 7 – Botão EclEmma

O Plug-in também adiciona uma aba de nome *coverage* que normalmente se localiza próxima a aba console, a Figura 8 exibe a posição exata da mesma.

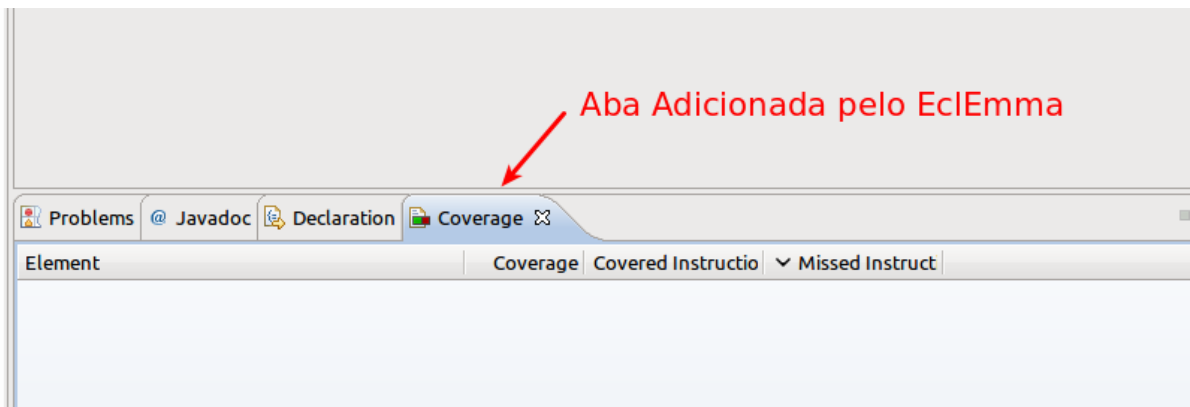


Figura 8 – Aba EclEmma

Caso a aba não tenha aparecido, segure a tecla Ctrl e aperte a Tecla 3, uma janela similar a da Figura 9 surgirá e no campo para digitação escreva, *coverage* e aperte [enter]. Agora a aba deve ter sido adiciona.

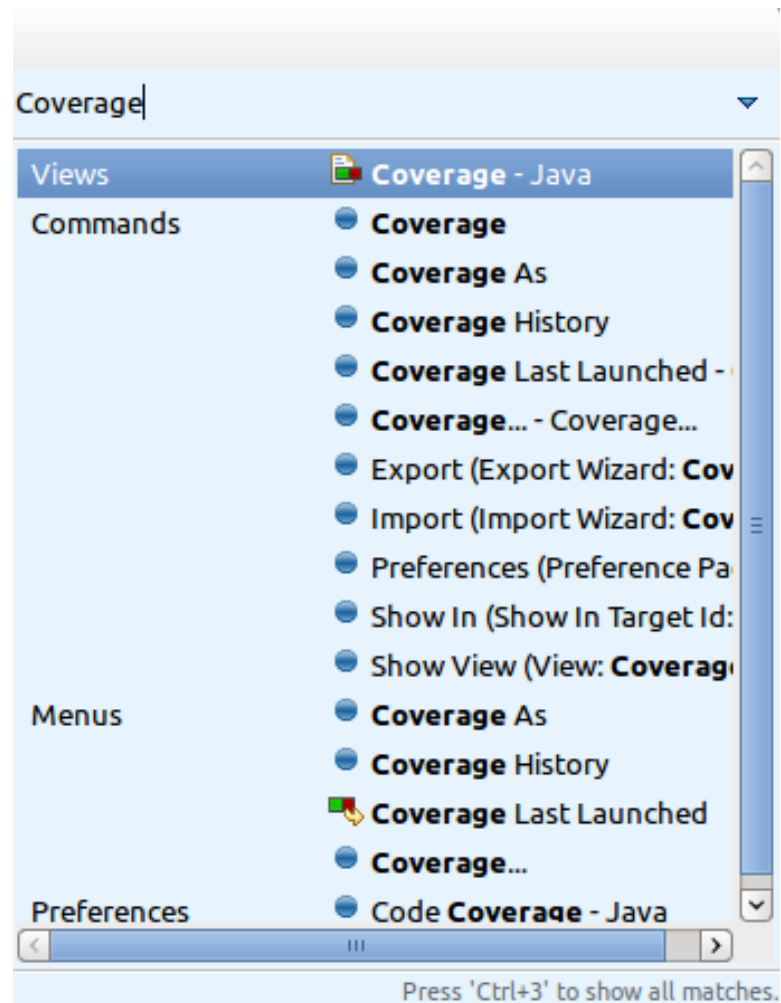


Figura 9 – Menu Eclipse

Com o EclEmma corretamente instalado, a instrumentação *on-the-fly* pode ser realizada simplesmente clicando no botão exibido na Figura 9, o arquivo de cobertura será gerado automaticamente e salvo no diretório do seu projeto, para realizar a instrumentação offline continua sendo necessário o uso do terminal.

Para visualizar graficamente as linhas executadas durante a aplicação do teste, devemos importar o arquivo gerado com os dados da cobertura (coverage.ec) que como dito anteriormente está localizado no diretório raiz do seu projeto. Para abrir o arquivo de cobertura na bancada do Eclipse clique com o botão direito do mouse na área mostrada pela Figura 10 e selecione a opção *import session*.

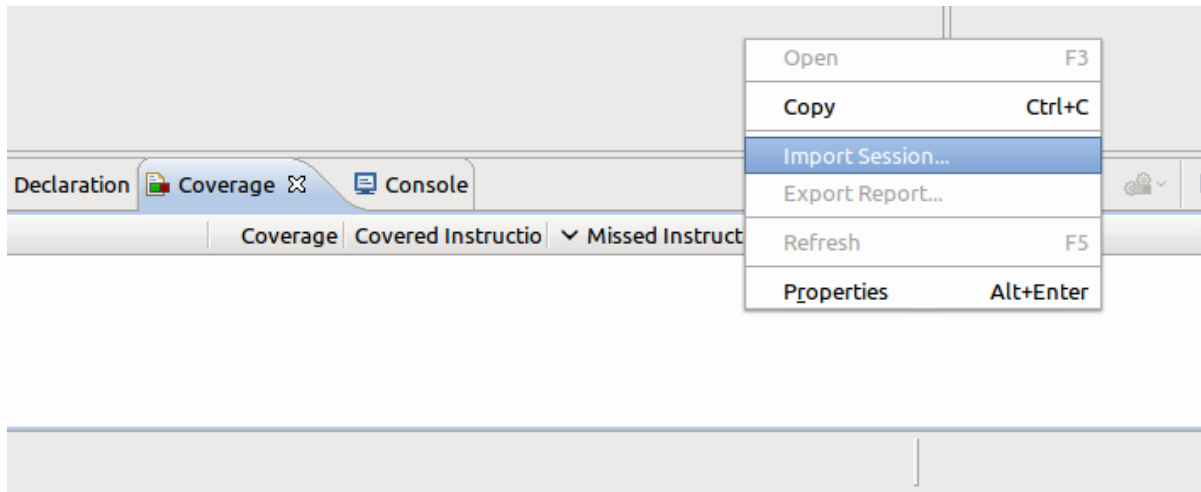


Figura 10 – Importando os dados da cobertura.

Apos o clique outra janela será aberta, e através dessa janela navegue até onde se encontra o arquivo de cobertura, marque o projeto alvo do teste e clique no botão *finish* como mostrado na Figura 11.

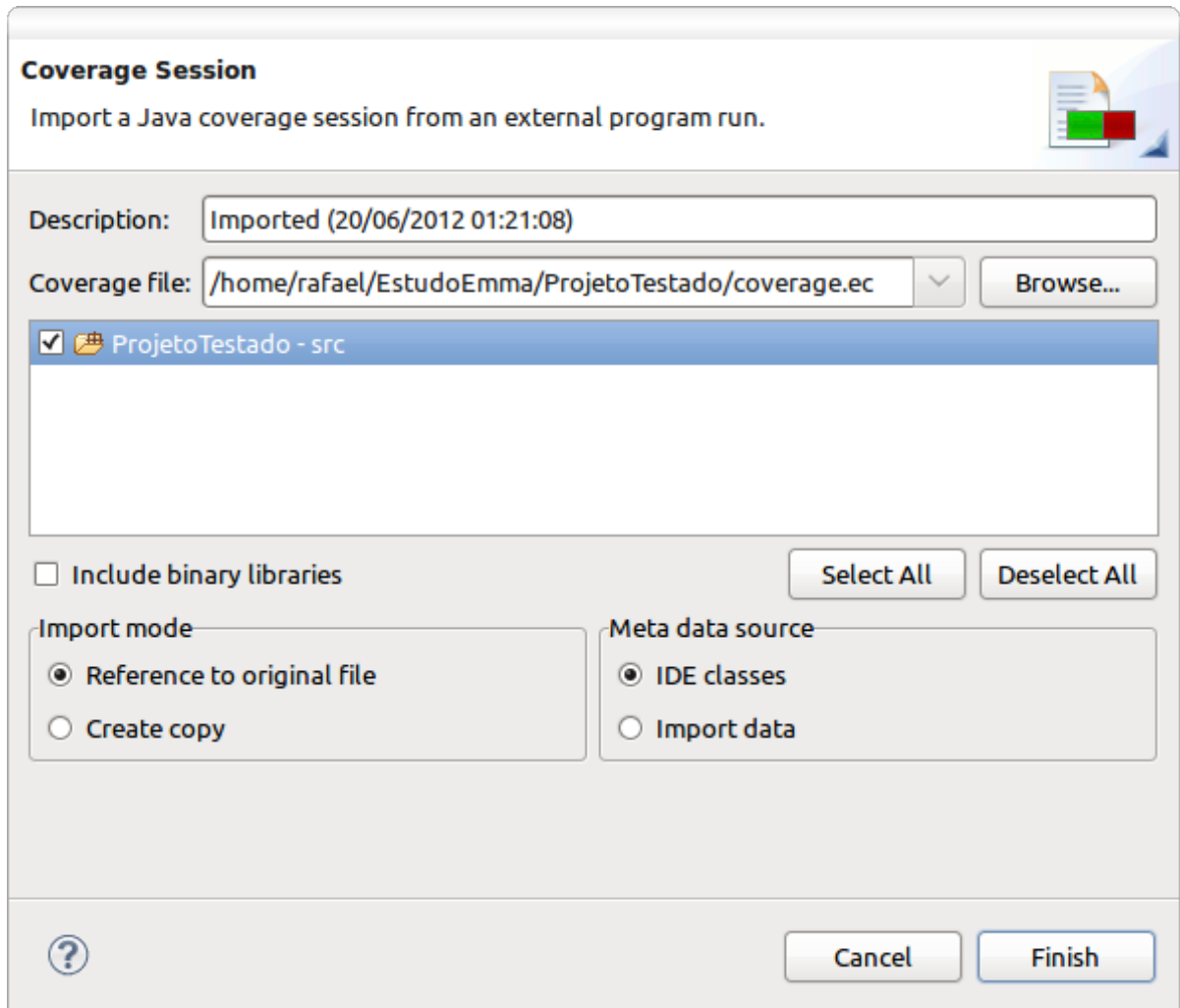
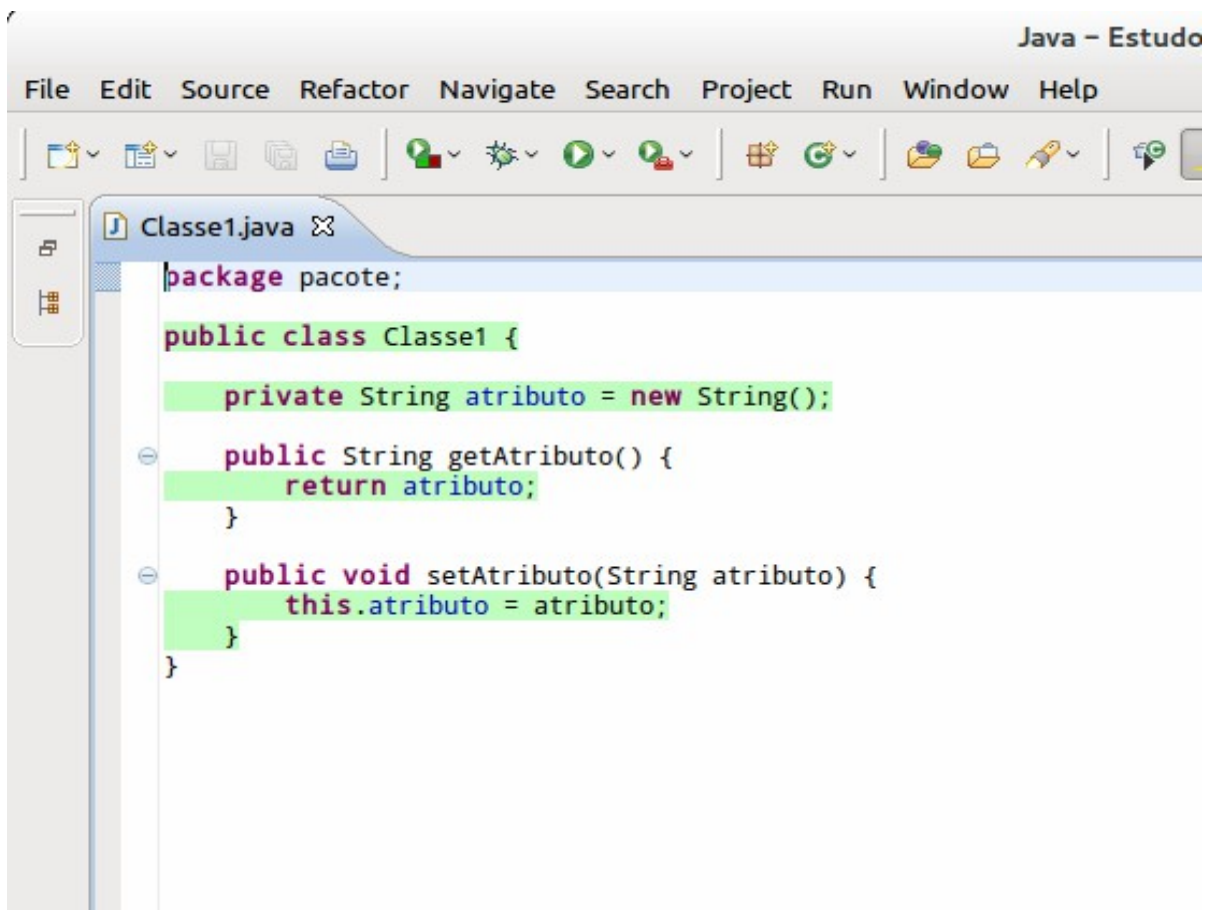


Figura 11 – Seleccionando o arquivo de cobertura.

7. COMO ANALISAR OS DADOS DA COBERTURA

Visando facilitar o entendimento de como analisar os dados obtidos com a cobertura, foi criado um projeto *desktop* que contém três classes com fim puramente explicativo e aplicado a cobertura de teste conforme as instruções dadas no capítulo anterior.

Na Figura 12, pode-se ver a “Classe1” que é composta por apenas um atributo e os métodos get e set, o *background* das linhas estão na cor verde pois todas elas foram executadas e assim a classe obteve 100% de cobertura.



```
Java - Estudo
File Edit Source Refactor Navigate Search Project Run Window Help
Classe1.java
package pacote;

public class Classe1 {

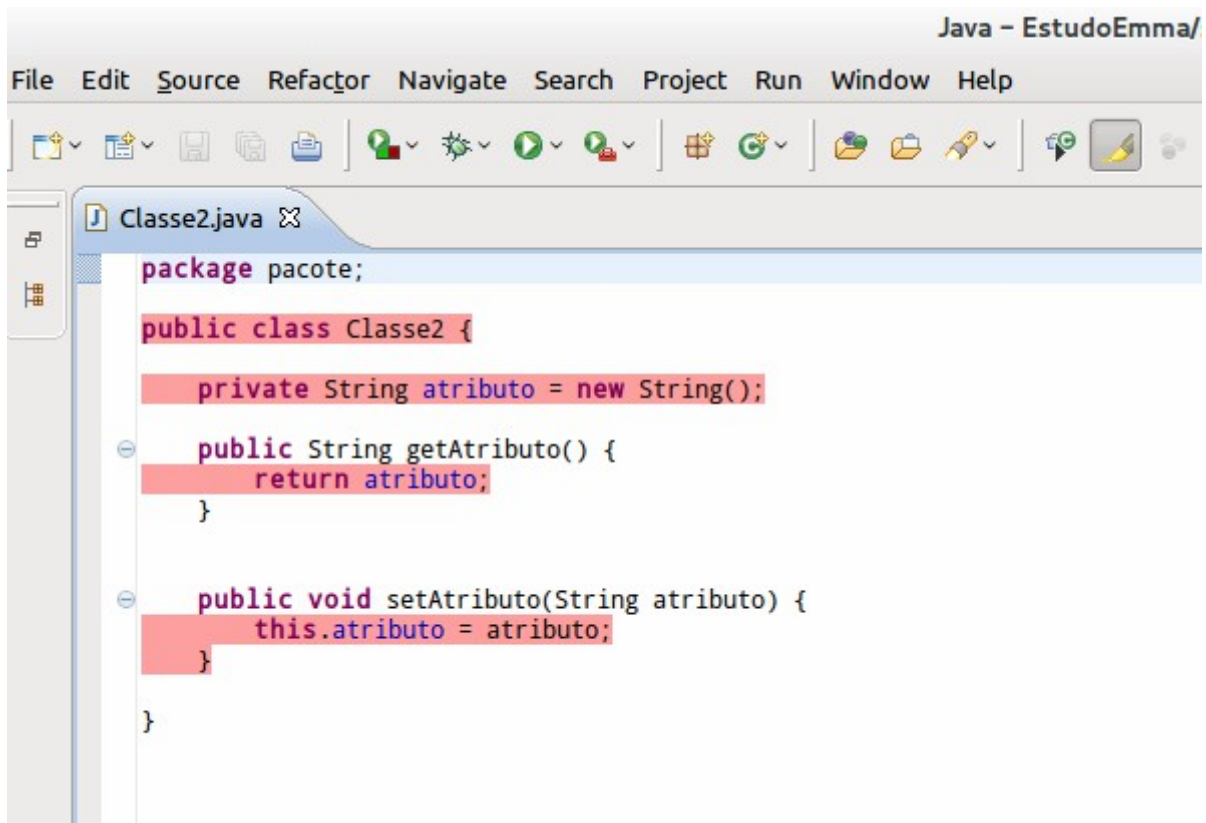
    private String atributo = new String();

    public String getAtributo() {
        return atributo;
    }

    public void setAtributo(String atributo) {
        this.atributo = atributo;
    }
}
```

Figura 12 – Classe1

A Figura 13, exibe a Classe2 que também é composta de apenas um atributo e os métodos de get e set, propositalmente essa classe nunca será instanciada e como consequência nenhuma das suas linhas serão executadas, resultando assim em 0% de cobertura nesta classe e todas as suas linhas terão a cor vermelha em seu *background*.



```
Java - EstudoEmma/
File Edit Source Refactor Navigate Search Project Run Window Help
Classe2.java
package pacote;
public class Classe2 {
    private String atributo = new String();
    public String getAtributo() {
        return atributo;
    }
    public void setAtributo(String atributo) {
        this.atributo = atributo;
    }
}
```

Figura 13 – Classe2

A ultima classe a ser exposta é a Principal, nela está a *void main* onde foi criado uma instancia da Classe1 e acessado todos os métodos implementados pela mesma. O objeto criado a partir da Classe1 é usado para imprimir uma mensagem no console do Eclipse e propositalmente foi implementado um “if” que será executado parcialmente resultando em 61,5% de cobertura nessa classe.

As linhas executadas parcialmente tem seu *background* na cor amarela, o Emma

entende como executado parcialmente um *for* ou *while* que, não tenha sido executado até o valor total de sua condição de parada, ou um *if* que não teve todas as suas condições contempladas o qual é o caso do *if* da Figura 14.

```
1 package pacote;
2
3 public class Principal {
4
5     public static void main(String[] args) {
6
7         Classe1 mensagem = new Classe1();
8
9         mensagem.setAtributo("Entendendo os dados da cobertura");
10
11        System.out.println(mensagem.getAtributo());
12
13        if(mensagem.equals(null) && mensagem.equals("Entendendo os dados da cobertura")){
14            System.out.println("Jamais serei impresso");
15        }
16
17    }
18 }
19
```

Figura 14 – Classe Principal

As métricas calculadas pelo Emma podem ser visualizadas na aba *Coverage*, como é ilustrado na Figura 15.

Element	Coverage	Covered Instructio	Missed Instructior	Total Instructions
▼ EstudoEmma	55,4 %	31	25	56
▼ src	55,4 %	31	25	56
▼ pacote	55,4 %	31	25	56
▼ Classe2.java	0,0 %	0	15	15
▼ Classe2	0,0 %	0	15	15
● setAtributo(String)	0,0 %	0	4	4
● getAtributo()	0,0 %	0	3	3
▼ Principal.java	61,5 %	16	10	26
▼ Principal	61,5 %	16	10	26
● main(String[])	69,6 %	16	7	23
▼ Classe1.java	100,0 %	15	0	15
▼ Classe1	100,0 %	15	0	15
● getAtributo()	100,0 %	3	0	3
● setAtributo(String)	100,0 %	4	0	4

Figura 15 – Dados da cobertura

Como dito anteriormente, o Emma mediu a quantidade de instruções de cada classe, verificou quantas linhas foram executadas e calculou um percentual, para cada método, classe, pacote, e projeto nos proporcionando uma visão extremamente detalhada de como a aplicação está sendo executada, como previsto a Classe1 e todos os seus métodos obtiveram 100% de cobertura, a Classe2 obteve 0%, a Principal obteve 61,5%, o método *main* teve 69,6% e como o projeto só possui um *package* então tanto o pacote como o projeto obtiveram o mesmo percentual de cobertura 55,4%. Caso o projeto possuía mais pacotes os mesmos teriam seus percentuais calculados individualmente.

8. EXPERIMENTO

No decorrer desse capítulo é abordado os detalhes de como transcorreu o experimento. São esclarecidos detalhes do *software* alvo da cobertura e a metodologia adotada para coleta e análise dos dados.

8.1 SOFTWARE

O *software* alvo do experimento se trata de um sistema de gestão financeira totalmente projetado para auxiliar a gestão financeira de ONGs. O projeto não possui fins lucrativos e é mantido pelo Rede Ciranda, organização composta pela união de todas as entidades filantrópicas do município de Assis/SP em parceria com a FEMA e patrocinado pela Telefônica|Vivo que no Brasil destaca-se como o maior conglomerado empresarial no segmento de telefonia fixa, móvel, transmissão de dados e internet, TV paga e atacado.

O *software* busca facilitar o processo de prestação de contas das ONGs, a seus respectivos patrocinadores e parceiros, havendo grande aceitação por parte das entidades, o sistema será disponibilizado no portal do software público Brasileiro, portal criado em abril de 2007 para compartilhar softwares de interesse público e tratar o software como um bem público.

8.2. DESTALHES DO EXPERIMENTO

O experimento se deu aplicando a cobertura de teste sobre o sistema descrito no capítulo 8.1. Não foi usado nenhum *framework* ou técnica de automatização de testes, o único *framework* utilizado foi o Emma para permitir a visualização da

quantidade de código-fonte executada durante a execução dos teste.

Os testes foram todos aplicados manualmente pela equipe de desenvolvedores, cujo a prática só foi possível pois o sistema é de médio porte. Inspeções de código também foram adotadas para complementar os dados recolhidos pelo Emma.

Os testes foram aplicados a medida que os módulos saíram da fase de desenvolvimento, problemas e soluções encontradas com o auxílio da cobertura de teste foram detalhadamente registradas para posterior análise onde será levantado questões sobre um percentual ideal de cobertura, esforço gasto com cobertura de teste e custo benefício.

Como complemento aos dados coletados, entrevistas foram realizadas com diferentes profissionais de tecnologia da informação, as entrevistas podem ser encontradas no Apêndice A.

8.3. ANÁLISE DOS DADOS

Com a análise dos dados no decorrer do experimento e auxílio das entrevistas feitas com diferentes desenvolvedores de *software*, muitas questões foram levantadas as quais serão todas discutidas abaixo.

Um das questões levantadas durante o experimento é: Até onde é válido o esforço para atingir 100% de cobertura; pois no caso de uma transação mal sucedida com o banco de dados quem se encarrega de fazer o *rollback* (função que possibilita ao banco de dados cancelar a transação mal sucedida e voltar ao estado anterior da mesma) é o banco de dados; o programador no caso do experimento só precisa colocar a rotina de transação dentro de um bloco try/catch e o mesmo se encarregará de disparar a instrução de *rollback* localizada dentro do bloco catch, caso exista algum problema na persistência dos dados; ou seja são rotinas que certamente já foram testadas até a exaustão por seus desenvolvedores. Ficar testando situações similares a essa até atingir 100% de cobertura é algo que

demanda muito esforço e tempo da equipe envolvida considerando que 40% dos gastos no desenvolvimento de software são com testes. (SOMMERVILLE, 2006)

Claro que uma rotina que tenha atingido um alto percentual de cobertura é sinal que os testes aplicados cobriram grande parte do código-fonte, mesmo isso não significando que todas as possibilidades lógicas foram esgotadas; pois por mais que uma aplicação tenha sido testada a exaustão no máximo foram cobertos 70% de todos os caminhos lógicos. (GLASS, 2002)

Outro ponto que o desenvolvedor deve ficar atento é a completa ineficácia da cobertura de teste em apontar erros lógicos, uma vez que uma rotina pode estar completamente fora da necessidade do cliente e mesmo assim obter auto percentual de cobertura, essa situação é algo que se mostrou completamente possível de ocorrer o que gera no desenvolvedor uma falsa métrica de qualidade.

Um exemplo de que o resultado obtido na cobertura não é absoluto pode ser visto na Figura 16. O método exibido na figura é responsável por persistir os dados do lançamento de um débito do cliente e entre esse dados estão várias datas que precisam ser persistidas como, data de compra, data de pagamento, data de vencimento e todas são validadas impedido que o usuário informe uma data impossível de existir.

Element	Coverage	Covered Instructio	Total Instructions
● getUltimasBaixas()	0,0 %	0	3
● getUsuarioBaixa()	0,0 %	0	3
● getUsuarioEstorno()	0,0 %	0	3
● getUsuarioLancamento()	0,0 %	0	3
● lancarDebito()	96,6 %	282	292
● prepararConsultarDebitosAprovados()	0,0 %	0	24
● prepararConsultarDebitosLancados()	0,0 %	0	24
● prepararDetalhes()	0,0 %	0	58

Figura 16 – Caso de alto percentual de cobertura

Como pode ser visto o método “lancarDebito()” obteve 96,6% de cobertura nos testes aplicados o que fez a equipe de desenvolvedores acreditar que essa rotina estava livre de erros; porem na fase de implantação um problema nesse método foi

revelado, ele permitia que o usuário informasse uma data de compra maior que a de data de pagamento algo que nesse caso geraria inconsistências nos dados.

Muitos outros casos similares ocorreram durante o experimento, os quais não são detalhados; pois todos foram resultados de falha humana seja na implementação, teste ou levantamento de requisitos.

Outra questão levantada pelo estudo foi se o uso da cobertura de teste facilitaria o desenvolvimento de *software*, para essa questão a resposta foi sim, pois com a cobertura é possível visualizar exatamente quais são as linhas exercitadas durante os testes e com esse mapeamento o desenvolvedor pode criar testes não redundantes e mais eficazes impactando em economia de tempo e esforço.

Outro incógnita era se ao usar a cobertura de teste a qualidade da aplicação seria maior e novamente a resposta foi sim, pois a aplicação de testes é uma tarefa custosa, que demanda de uma boa fatia de tempo e recursos do projeto, a visão e agilidade proporcionada pela cobertura resulta em economia de tempo e esforço da equipe envolvida, possibilitando a elaboração de testes mais eficientes e até mesmo a realização de um maior número de testes.

Obter 100% de cobertura é sinônimo de *software* sem falhas? Com toda certeza não, pois a cobertura de teste consiste em verificar quais partes do código-fonte foram executadas durante a bateria de testes, é perfeitamente possível elaborar testes fracos e que não testam o que é realmente necessário simplesmente para atingir 100% de cobertura. Claro que isso não é falha da técnica de cobertura de teste, fica evidente que quando se obtêm 100% de cobertura e mesmo assim falhas são encontradas após a fase de testes, é resultado de falha humana na fase de testes ou nos levantamentos de requisitos.

Qual a métrica ideal de cobertura? O estudo não encontrou um percentual de cobertura e nem uma equação que gerasse um percentual ideal para cada situação. O que ficou evidente é que quanto maior o percentual de cobertura melhor; mesmo isso não significando estar completamente livre de erros e muito menos estar atendendo satisfatoriamente o seu cliente. Alguns fatores devem ser levados em consideração para definir uma métrica ideal e os fatores apontados pelo

experimento são: tempo disponível para fase de teste, o número de pessoas envolvidas na fase de teste, a importância da rotina na aplicação e a experiência dos integrantes da equipe. Cabe a equipe ponderar todos esses fatores e definir a métrica ideal para aquela situação, em suma o que vale nesse caso é o bom senso e comprometimento da equipe.

8.4. BENEFÍCIOS

O maior benefício da cobertura de teste é a capacidade de mapear quais linhas do código-fonte foram exercitadas durante a bateria de teste, e assim o desenvolvedor tem uma visão exata do que melhorar nos testes aplicados e reaplicar somente os testes que entender como necessário após analisar os dados da cobertura, essa otimização da fase de teste impacta diretamente ao custo e tempo de desenvolvimento do projeto, o que é de extrema importância no padrão de desenvolvimento de software atual.

A cobertura mostrou-se também bastante útil em auxiliar a detecção de métodos e condições (*if, for, while*) que por algum motivo deixaram de ter a necessidade de existir na aplicação, mas que por falta de atenção do programador ou algum outro motivo os mesmos não foram removidos. Como consequência desse mapeamento gerado pela cobertura um código-fonte mais otimizado e simples é construído. Um exemplo de código morto apontado pelo *framework* pode ser visto na Figura 17, no caso os dois métodos marcados em vermelho só foram utilizados no início do desenvolvimento para dar suporte a outra rotina que estava em desenvolvimento; mas com a finalização do módulo em questão deixou de ser necessário e muito provavelmente senão fosse a cobertura pelo apontado o mesmo não seria removido.

```

public class LogoffMB extends HttpServlet{

    public String voltar(){
        return "principal.xhtml";
    }

    public String redirecionar(){
        return "/restrito/paginaRestrita.xhtml";
    }

    public String logout(){

        FacesContext context = FacesContext.getCurrentInstance();

        for (String bean : context.getExternalContext().getSessionMap().keySet()) {
            context.getExternalContext().getSessionMap().remove(bean);
        }

        UtilMB.setUser(null);

        HttpSession session = (HttpSession) context.getExternalContext().getSession(false);
        session.invalidate();

        return "login.xhtml";
    }
}

```

Figura 17 – Código morto

Outro fantástico benefício que vem junto com a cobertura é a melhoria dos algoritmos que implementam as regras de negócio, através da capacidade do *framework* de apontar as linhas que foram executadas parcialmente, cujo o mecanismo que define a execução parcial foi explicado no Capítulo 7, essa capacidade não é exclusiva do Emma atualmente muitos *frameworks* de cobertura também possuem funcionalidade similar. A melhoria surge quando uma linha ou condição lógica foi marcada em amarelo, o que significa execução parcial, é sinal que as condições não estão bem implementadas, então a equipe envolvida no desenvolvimento sempre analisava novamente a rotina e não encontrando problema, era feita uma consulta aos requisitos levantados e quase sempre a reavaliação acarretava em mudanças no algoritmo implementado ou em casos mais raros até alteração dos requisitos após consulta ao cliente aumentado a eficiência da aplicação. Um exemplo ocorrido no experimento pode ser visto na Figura 16, como o *if* da figura era muito longo uma parte foi omitida. Esse *if* é responsável por controlar

a renderização de alguns componentes na tela e parte de uma rotina maior responsável por criar uma consulta personalizada a base de dados, como foi executado parcialmente o mesmo foi reanalisado e condições excessivas foram encontradas e corrigidas nesse caso não foi necessário consultar novamente os requisitos.

```
if(opSelecionadaPrimeiroSelect.equals("dataCompra") || opSelecionadaPrimeiroSelect.equals("dataLancamento") ||  
seleccionadoPesquisarPorData = true;  
seleccionadoPesquisarPorDescricao = false;  
exibirBtnPesquisar = false;
```

Figura 18 – Caso de execução parcial

Observou-se também que as rotinas que obtiveram maior percentual de cobertura foram as que menos apresentaram problemas no início da implantação do *software*, algo que estimula a continuidade do uso da cobertura de teste.

8.5. DEFICIÊNCIAS

A cobertura de teste durante o experimento não apresentou falhas, existiram falhas mas todas foram ocasionadas por falha humana seja durante a implementação, teste ou levantamento de requisitos. Mas como a cobertura de testes não apresentou deficiências e mesmo assim falhas chegaram até o cliente? Deve-se atentar que a proposta da cobertura é simplesmente marcar no código-fonte quais são as linhas executadas e não executadas, durante a execução do teste; ou seja sua proposta não é verificar se o teste aplicado está realmente testando a rotina em busca de falhas e muito menos se a rotina testada condiz com os requisitos levantados ou mesmo validar os requisitos levantados.

9. CONCLUSÃO

No decorrer do experimento ficou evidente que ao fazer uso da cobertura de teste a qualidade do software aumenta visivelmente, pois seu uso ajuda a criar testes realmente eficazes e dinâmicos graças ao mapa criado pelo teste de cobertura.

Outro ponto positivo para a cobertura é sua capacidade de ajudar o desenvolvedor a criar um código-fonte mais simples, e otimizado uma vez que a essência da cobertura de teste é fazer um mapeamento das linhas executadas e não executadas do código-fonte.

Mais um ponto revelado pelo estudo é que é melhor usar a cobertura de teste como uma indicadora do que não foi testado, ao invés de ser usada para indicar o que foi testado. Em suma o que vale mesmo é a elaboração de testes realmente eficazes que testem se as rotinas implementadas atendem as necessidades do cliente sem gerar inconsistências nas informações manipuladas pelo mesmo.

O estudo mostrou que uma métrica de cobertura padrão para avaliar a qualidade do software não existe, nesse caso vale o bom senso, pois ao longo do estudo muitas rotinas que tiveram auto nível de cobertura causaram problemas em alguns pontos, outro ponto que ficou evidente é que nem sempre é válido o esforço para atingir 100% de cobertura e muito menos criar testes fracos simplesmente para exercitarem todas as linhas da aplicação.

10. CRONOGRAMA

2012	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov
Levantamento bibliográfico									
Estudo <i>frameworks</i>									
Desenvolvimento pré-projeto									
Coleta dos dados									
Desenvolvimento qualificação									
Apresentação qualificação									
Análise dos dados									
Desenvolvimento TCC									
Defesa									

Tabela 5 - Cronograma

11. REFERÊNCIAS BIBLIOGRÁFICAS

MARTINS, José Carlos Cordeiro. **Técnicas para gerenciamento de projetos de software**. Rio de Janeiro: Brasport, 2007. 456 p.

DEITEL, Paul J.; DEITEL, Harvey M.. **Java como programar**. São Paulo: Pearson Prentice Hall, 2001. 1114 p.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: McGraw-Hill, 2006. 720p.

SOMMERVILLE, Ian. **Engenharia de software**. São Paulo: Pearson, 2007. 568p.

FILHO, Wilson P P. **Engenharia de software: fundamentos, métodos e padrões**. São Paulo: LTC, 2009. 1256p.

GLASS, Robert L.. **Facts And Fallacies of Software Engineering**. Internacional: Addison Wesley, 2002. 224 p.

Documentação Emma Disponível em: < <http://emma.sourceforge.net/intro.html> >
Acesso em : 1 de mar. 2012.

Documentação EclEmma Disponível em: <
<http://www.eclEmma.org/userdoc/index.html> > Acesso em: 15 de mar. 2012.

12. APÊNDICE A – Entrevistas

12.1. ENTREVISTA FERNANDO CESAR DE LIMA

1 – Fale um pouco sobre você, nome, formação acadêmica, situação atual profissional.

Meu nome é Fernando Cesar de Lima, sou graduado e pós graduado pela Fundação Educacional do Município de Assis. Atualmente, além de professor na área de Ciências da Computação, sou empresário no ramo de informática, especificamente, no ramo de desenvolvimento de software. Atuamos a mais de 10 anos em várias regiões do País.

2 – Você faz uso da cobertura de teste em seu cotidiano profissional?

Sim, realizamos.

3 – O seu empregador exige que seja feito uso da cobertura de teste?

Fazemos a exigência quando se trata de rotinas complexas.

4 – Como a sua empresa considera os dados gerados pela cobertura de teste, existe uma meta mínima de cobertura?

Não, pois geralmente exigimos que determinado algoritmo possua todas as suas linhas executáveis testadas. Porém, não temos a exigência de cobertura de teste para o projeto todo.

5 – Você considera a cobertura de teste uma boa métrica de qualidade?

Com certeza, pois como citamos, existem rotinas que são complexas por natureza, onde sem o auxílio da cobertura de testes, o programador não teria como causar todos os fluxos possíveis de tal rotina.

6 – Alguma vez você já se deparou com algum problema em uma aplicação que

tenha atingido alto percentual de cobertura?

Sim, pois a cobertura de testes ao nosso ver, assegura a execução técnica de uma rotina, porém combinação de valores de parâmetros assim como ausência de seus valores nem sempre são previsíveis e mesmo simuláveis.

7 - Qual a sua opinião sobre cobertura de teste de código, considera ser uma técnica auxiliar na produção de software de boa qualidade?

Se usada com moderação, com certeza é uma ferramenta extremamente importante. Porém, existe a ressalva de que um projeto não está salvo de falhas devido ao fato de ter atingido um alto percentual de testes.

12.2. ENTREVISTA GUILHERME DE CLEVA FARTO

1 – Fale um pouco sobre você, nome, formação acadêmica, situação atual profissional.

Meu nome é Guilherme de Cleva Farto, sou formado em Bacharelado em Ciência da Computação pela FEMA (2010) e possuo pós-graduação em Engenharia de Componentes Java pela FIO e TNT Educacional (2011). Atualmente sou analista e desenvolvedor Java Web na PRX – Software and Services; empresa do grupo TOTVS. Também sou professor universitário do curso de Bacharelado em Ciência da Computação pela FEMA (2012) nas disciplinas de Tópicos Avançados (4º ano) e Eletrônica Digital (2º ano).

2 – Você faz uso da cobertura de teste em seu cotidiano profissional?

Sim. Para cada atividade ou solicitação que é enviada à fábrica de software (em Assis/SP) é realizado, pelo analista, desenvolvedor e equipe de qualidade testes relacionados à determinada tarefa.

3 – O seu empregador exige que seja feito uso da cobertura de teste?

Sim. Por trabalharmos diretamente sobre as diretrizes da TOTVS (empresa global de

TI) somos obrigados a utilizar normas e abordagens de testes. Mesmo quando não era obrigado, já realizávamos as atividades de testes por saber da necessidade e compromisso de entrega de produtos devidamente testados e validados.

4 – Como a sua empresa considera os dados gerados pela cobertura de teste, existe uma meta mínima de cobertura?

A meta sempre é 100%: solicitação especificada, codificada, validada e entrega no prazo. Claro que, como qualquer outra fábrica de software, é possível que haja atrasados em quaisquer etapas (análise, desenvolvimento, testes e expedição), mas o objetivo é cumprir todas as exigências para que o produto seja entregue conforme acordado no início do projeto e com nenhum (ou o mínimo possível) de refluxo.

5 – Você considera a cobertura de teste uma boa métrica de qualidade?

Sim. Cada vez mais devemos nos atentar à etapa de testes ou qualidade, que, comumente, é deixada de lado. Não basta apenas realizar a especificação, implementação e testes unitários, devemos utilizar de novas metodologias para manter a aplicação, e projeto como um todo, dentro das expectativas, tanto para o cliente quanto para a empresa fornecedora de software. Apesar de ser difícil medir ou mensurar software, com certeza a cobertura de teste auxilia na gestão da qualidade do software ou produto final.

6 – Alguma vez você já se deparou com algum problema em uma aplicação que tenha atingido alto percentual de cobertura?

Devido a constante execução de testes, torna-se bem incomum solicitações de desenvolvimento apresentar altos índices de refluxo durante a execução de testes de qualidade. Mesmo assim, estamos em constante atenção quanto a possíveis lacunas ou “brechas” durante a análise e desenvolvimento de atividades, evitando, ao máximo, apresentar altos percentuais de falhas durante as fases de testes e expedição.

7 - Qual a sua opinião sobre cobertura de teste de código, considera ser uma técnica auxiliar na produção de software de boa qualidade?

Sim, conforme mencionado anteriormente, a realização de testes é essencial para

que todas as demais etapas (análise, especificação e desenvolvimento) sejam realizadas com sucesso. Um software não devidamente testado é uma grande falha no processo “industrial” dentro de uma fábrica de software capaz de trazer grandes consequências tanto para o cliente quanto para a empresa desenvolvedora de aplicações.

12.3. ENTREVISTA EDUARDO NEGRÃO

1 – Fale um pouco sobre você, nome, formação acadêmica, situação atual profissional.

Meu nome é Eduardo Negrão. Sou coordenador de T.I, atuando na coordenação de uma equipe de Quality Assurance (Q.A.) e uma equipe de desenvolvimento de software. Sou pós-graduado em Engenharia de Software.

2 – Você faz uso da cobertura de teste em seu cotidiano profissional?

Atualmente não faço, mas já utilizei a nível acadêmico e em projetos profissionais no passado.

3 – O seu empregador exige que seja feito uso da cobertura de teste?

Não exige atualmente.

4 – Como a sua empresa considera os dados gerados pela cobertura de teste, existe uma meta mínima de cobertura?

Não existe nenhuma meta a respeito, visto que o nível de maturidade dos processos de software que temos na empresa em que trabalho não permitem que a qualidade do software seja compatível com a velocidade de entrega.

5 – Você considera a cobertura de teste uma boa métrica de qualidade?

Sim, com certeza. Uma alto percentual de cobertura de testes proporciona maior qualidade ao software, pois dá confiabilidade na geração de alterações no código, alta e rápida manutenibilidade e evita erros humanos no processo de testes

funcionais/manuais.

6 – Alguma vez você já se deparou com algum problema em uma aplicação que tenha atingido alto percentual de cobertura?

Sim, pois, como já é sabido, não existem aplicações com percentual zero de bugs. No entanto, os problemas encontrados foram unicamente ocasionados por falhas humanas não previstas nem mesmo nos requisitos documentados para o software.

7 - Qual a sua opinião sobre cobertura de teste de código, considera ser uma técnica auxiliar na produção de software de boa qualidade?

Sim, considero. Acredito que todo software com alta complexidade ou que seja muito dinâmico (com alterações ocorrendo a todo o momento, instabilidade dos requisitos e etc...) deveria considerar um bom percentual de cobertura de teste de código para garantir um nível de qualidade sempre constante.

Em se tratando de testes de unidade, como visto nas práticas de *Test Development-Driven* (TDD), uma boa cobertura de testes de código se mostra ainda mais importante, visto que ela não somente garante testes de boa qualidade, mas um design de projeto, arquitetura, um código mais limpo e projetos mais íntegros e profissionais.

12.4. ENTREVISTA LUIZ ANGELO FRANCISCATTI

1 – Fale um pouco sobre você, nome, formação acadêmica, situação atual profissional.

Conclui o curso de Análise e Desenvolvimento de Sistemas na Fema, fui estagiário no período do curso, o TCC (Trabalho de Conclusão do Curso) foi desenvolvido um sistema desktop na linguagem Java, após concluído consegui emprego em Florianópolis SC como desenvolvedor Java, depois de algum tempo vim para São Paulo como Desenvolvedor Oracle.

Nome: Luiz Angelo Franciscatti

Minha formação acadêmica foi em 2010 na Fema em Análise e Desenvolvimento de Sistemas.

Situação profissional: Atualmente estou como Analista de Sistema e Desenvolvedor Oracle na empresa R-Dias uma consultoria de varejo.

2 – Você faz uso da cobertura de teste em seu cotidiano profissional?

Sim, a toda tarefa concluída tem que fazer teste.

3 – O seu empregador exige que seja feito uso da cobertura de teste?

Sim, o próprio ambiente exige : Homologação, teste e produção.

4 – Como a sua empresa considera os dados gerados pela cobertura de teste, existe uma meta mínima de cobertura?

Considera muito importante, sim tudo é gerado em base do tempo de cada fase do projeto a ser desenvolvido.

5 – Você considera a cobertura de teste uma boa métrica de qualidade?

Sim

6 – Alguma vez você já se deparou com algum problema em uma aplicação que tenha atingido alto percentual de cobertura?

Sim, algumas vezes, porém concluída .

7 - Qual a sua opinião sobre cobertura de teste de código, considera ser uma técnica auxiliar na produção de software de boa qualidade?

Minha opinião sobre o teste é muito importante tanto para a qualidade como para o desempenho da produção a todo momento testamos nossa aplicação, com isso obtemos resultados positivos.

12.5. ENTREVISTA CELSO YAMAGUTI SOBRAL

1 – Fale um pouco sobre você, nome, formação acadêmica, situação atual

profissional.

Olá, meu nome é Celso Yamaguti Sobral, ex-aluno da FEMA, e atualmente trabalho numa empresa de comércio eletrônico em Fort Lauderdale, Florida.

2 – Você faz uso da cobertura de teste em seu cotidiano profissional?

Teoricamente sim, pois faço teste de todos os novos componentes, módulos e programas que desenvolvo. Mas não como deveria ser. A empresa é focada em vender produtos, então em alguns casos a qualidade não é tão importante quanto a velocidade de desenvolvimento.

3 – O seu empregador exige que seja feito uso da cobertura de teste?

Não.

4 – Como a sua empresa considera os dados gerados pela cobertura de teste, existe uma meta mínima de cobertura?

5 – Você considera a cobertura de teste uma boa métrica de qualidade?

Esse tema é novo para mim, mas toda a parte de métrica de qualidade é melhor do que a falta da mesma.

6 – Alguma vez você já se deparou com algum problema em uma aplicação que tenha atingido alto percentual de cobertura?

7 - Qual a sua opinião sobre cobertura de teste de código, considera ser uma técnica auxiliar na produção de software de boa qualidade?

Considero qualquer técnica de teste um ótimo processo para assegurar um software de boa qualidade.

12.6. ENTREVISTA MARIANA BUDISKI DE LIMA

1 – Fale um pouco sobre você, nome, formação acadêmica, situação atual profissional.

Meu nome é Mariana, estou no ultimo ano do curso de ciência da computação na Fundação Educacional do Município de Assis. Desenvolvo softwares em Java como estagiária em um projeto financiado pela Fundação Telefônica em prol das entidades sociais de Assis. Atualmente, estou desenvolvendo um Sistema para Gestão Financeira que será usado pelas entidades sociais da cidade que já está em fase de implantação.

2 – Você faz uso da cobertura de teste em seu cotidiano profissional?

Não faço uso da cobertura de teste no meu cotidiano e só senti a necessidade de realizar a cobertura depois de acompanhar o trabalho da faculdade de um amigo sobre o tema.

3 – O seu empregador exige que seja feito uso da cobertura de teste?

Não. Meu empregador não tem conhecimento sobre desenvolvimento de software e por isso não faz exigências como forma de desenvolvimento ou testes, isso fica a critério da equipe de desenvolvimento.

4 – Como a sua empresa considera os dados gerados pela cobertura de teste, existe uma meta mínima de cobertura?

Não. A empresa onde trabalho não exige metas para cobertura de testes.

5 – Você considera a cobertura de teste uma boa métrica de qualidade?

Não considero a cobertura de testes uma boa métrica de qualidade de software porque ela não cobre falhas no levantamento de requisitos e não consegue sinalizar possíveis erros de lógica de programação.

6 – Alguma vez você já se deparou com algum problema em uma aplicação que tenha atingido alto percentual de cobertura?

Não, nunca utilizei a cobertura de testes em aplicações que desenvolvi.

7 - Qual a sua opinião sobre cobertura de teste de código, considera ser uma técnica auxiliar na produção de software de boa qualidade?

Considero uma técnica auxiliar mas não absoluta para desenvolvimento de software

de qualidade, pois com a cobertura de testes é possível evitar blocos de códigos como getters e setters nunca utilizados e com isso eliminá-los, porém não sinaliza na aplicação erros como blocos try..finally mal implementados que podem causar muitos problemas.

12.7. ENTREVISTA VINICIUS DIAS OLIVEIRA

1 – Fale um pouco sobre você, nome, formação acadêmica, situação atual profissional.

Meu nome é Vinicius Dias Oliveira, me formei na FEMA em 2009 no curso de Ciência da Computação e atualmente estou cursando Engenharia e Arquitetura de Software como pós graduação na Universidade Gama Filho. Atuo como engenheiro de software, trabalhando com desenvolvimento e testes de software.

2 – Você faz uso da cobertura de teste em seu cotidiano profissional?

Sim, no projeto em que trabalho fazemos TDD – Test Driven Development e uma das ferramentas de teste utilizadas é de cobertura de código.

3 – O seu empregador exige que seja feito uso da cobertura de teste?

Sim, antes da entrega de um requisito, uma das etapas é atingir um certo nível de cobertura do código nos testes.

4 – Como a sua empresa considera os dados gerados pela cobertura de teste, existe uma meta mínima de cobertura?

Sim, trabalhamos com o percentual de 90% de cobertura do código de produção escrito para um requisito.

5 – Você considera a cobertura de teste uma boa métrica de qualidade?

Com certeza, a cobertura por si só não garante um software sem falhas, mas ajuda muito o desenvolvedor a prevenir erros como os de validação de valores e fluxo do algoritmo implementado.

6 – Alguma vez você já se deparou com algum problema em uma aplicação que tenha atingido alto percentual de cobertura?

Sim, escrever um software sem falhas hoje é muito difícil, a cobertura de código ajuda a visualizar quais partes do código não estão sendo testadas durante testes unitários, mas é fundamental que outras fases de testes sejam implementadas e automatizadas.

7 - Qual a sua opinião sobre cobertura de teste de código, considera ser uma técnica auxiliar na produção de software de boa qualidade?

A cobertura de código é definitivamente uma técnica que auxilia o desenvolvedor a evitar problemas básicos no código, diminuindo a possibilidade de erros serem encontrados pelo usuário final, mas é uma técnica que deve fazer parte da fase de testes, e não a única técnica utilizada para garantir a boa qualidade do software.