

UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA
CRIPTOGRAFIA E SEGURANÇA DA INFORMAÇÃO

ENGENHARIA DE SEGURANÇA

Ferramentas e técnicas de Code Coverage

André Gonçalves - A80368
Nelson Sousa - A82053
Pedro Freitas - A80975

10 de Maio de 2020

Resumo

No âmbito da Unidade Curricular de Engenharia de Segurança, enquadrada no perfil de especialização de Criptografia e Segurança da Informação, este projeto foi desenvolvido com o objetivo de abordar e explorar conceitos úteis e relevantes desta UC, sendo que se pretende adquirir conhecimentos e consciência do que é Code Coverage assim como algumas técnicas e ferramentas.

Conteúdo

1	Introdução	3
2	Code Coverage	4
2.1	Motivos para se usar Code Coverage	4
3	Técnicas de Testing - Code Coverage	5
4	Ferramentas	7
4.1	C	7
4.2	Python	8
4.3	Java	9
4.4	Javascript	11
5	Conclusão e Análise Crítica	13

1 Introdução

Nos dias que correm existe uma enorme necessidade de produção de novo software onde não é suficiente apenas receber a garantia da sua funcionalidade. Cada vez mais é necessário corresponder às expectativas dos clientes e entregar um produto que apenas é funcional não chega. É esperado, e necessário, que ele cumpra uma variada quantidade de requisitos funcionais e não funcionais, onde todos apontam para um produto final funcional mas com qualidade.

Desta necessidade surgiram documentos e *frameworks* de forma a universalizar parâmetros e fatores que determinam a qualidade de um produto. É aqui que entra a **ISO25000 Portal**, que desenvolveu *ISO/IEC 25000 - System and Software Quality Requirements and Evaluation (SQuaRE) series of standards*. Um conjunto de *standards* que explicitam os atributos que medem a qualidade de um produto de Software para a ISO. Este baseia-se em oito principais fatores:

- Functional Suitability
- Reliability
- Performance efficiency
- Operability
- Operability
- Security
- Compatibility
- Maintainability
- Transferability

Porém, apesar dos atributos acima estarem explícitos como os fatores que medem a qualidade de um software, não existe nenhum método (ou sistema de métodos) que consigam avaliar de uma forma descendente. Por outras palavras, não existe uma maneira de avaliar cada atributo individualmente ou independentemente, nem existe um sistema que possa avaliar todas as medidas com apenas uma técnica ou método.

Por isso surgiu um conjunto de métricas de qualidade de software que nos ajudam a medir e a avaliar a qualidade de um software tendo em conta os atributos já referidos. Daqui surgem as novas métricas:

- **Code coverage**
- Abstract interpretation
- Cyclomatic complexity
- Compiler warnings
- Coding standards
- Code duplication
- Fan out
- Security

Neste documento vamos focar na métrica **Code Coverage** falando do que é, de que forma nos ajuda a avaliar a qualidade de um software e ferramentas de medida da mesma.

2 Code Coverage

Code coverage pode ser definido como uma medida usada para descrever a percentagem de código que é executado quando um determinado conjunto de testes automatizados é executado.

Code coverage é executado pelos desenvolvedores através de testes unitários para verificar a implementação do código de maneira a que quase todas as instruções de código sejam executadas. No geral, um sistema de code coverage coleta informações sobre o programa em execução e combina-as para gerar um relatório sobre a code coverage do conjunto de testes. Code coverage é parte de um ciclo de feedback no processo de desenvolvimento. À medida que os testes são desenvolvidos, a code coverage destaca aspectos do código que podem não estar a ser adequadamente testados e que exigem testes adicionais. Esse loop continuará até que seja atingido algum objetivo especificado.

2.1 Motivos para se usar Code Coverage

- Um programa com elevado code coverage, medido como percentagem, teve mais de seu código-fonte executado durante o teste, o que sugere uma menor chance de conter erros de software não detectados em comparação com um programa com baixa percentagem.
- Coverage mostra que código foi executado, e o maior benefício imediato disso é a exposição de mau código. Para além disso, também mostra que partes foram omitidas, o que permite melhorar a robustez através da adição de mais testes para cobrir estes casos.
- Torna o processo de refatorização de código bastante mais fácil.
- Além de provar a exatidão, testes bem criados demonstram o uso adequado de API. Os testes mostram como usar o código e fornecem informações valiosas sobre como as coisas funcionam. O aumento da cobertura de teste expande a documentação interna do projeto, proporcionando mais alavancagem para os developers.
- À medida que o desenvolvimento do produto avança, novos recursos, bem como correções (para os erros levantados durante os testes unitários) são adicionados. Isso significa que iremos ter de adicionar novos testes face às alterações feitas no software durante o desenvolvimento. É importante que os padrões de teste que foram definidos durante o início do projeto sejam mantidos e através da medição pelo code coverage conseguimos garantir que esses padrões são cumpridos.

Code coverage é um aspeto importante das boas práticas de engenharia, no entanto, não prova que os testes estão corretos ou que o código funciona corretamente, esse é o trabalho dos designers e developers.

3 Técnicas de Testing - Code Coverage

Como foi referido no capítulo anterior a avaliação do nível de *Code Coverage* está diretamente relacionado com os testes unitários efectuados durante a fase de desenvolvimento, sendo por isso indispensáveis para a garantia de qualidade do produto.

Estes testes são executados sendo conhecido a estrutura do sistema e usam por isso técnicas baseadas nela, normalmente mais conhecido por técnicas *Black Box*. Estas técnicas começam por identificar uma das estruturas no software e depois criar testes que irão atuar sobre essa mesma estrutura. As estruturas podem ser divididas em três:

- **Nível de Componente:** Avalia a estrutura de um componente. Nesta componente pode ser avaliadas:
 - Statements* - uma linha de código
 - Decisões* - testa comparações (if's)
 - Branches* - ramificações do fluxo do programa devido a decisões
- **Nível de Integração:** Avalia a estrutura em termos de *call tree* - diagrama onde módulos chamam outros módulos
- **Nível do Sistema:** a estrutura pode ser uma estrutura de menus, um processo na camada de negócio ou uma estrutura de páginas web.

A percentagem de *Code Coverage* é calculada então através de testes segundo as estruturas em cima explicitadas. Sendo este um atributo de elevada importância na garantia de qualidade de um software é exigido também à entidade responsável pelo desenvolvimento uma boa qualidade no que toca a estes testes. Para tal é necessário ter em atenção a relação entre o custo de produção dos testes e a quantidade de *Code Coverage* atingível. Por outras palavras, é necessário uma reflexão sobre se valerá a pena realizar um teste que custará bastante à empresa se este aumentar pouco ou nada o nível da cobertura ou então optar por outro teste que tenha maior amplitude de cobertura. Este balanço tende a melhorar consoante a experiência e a maturidade das entidades envolvidas no processo.

Ao analisarmos as definições simples de cada estrutura avaliada nos testes vemos que algumas delas podem causar incompatibilidades. Um exemplo prático disso é quando existe um teste a nível componente, onde um teste simples de *statements* pode implicar uma redução no *code coverage* em *branches*. Vamos agora avaliar o excerto seguinte de código.

```
c = False
if ( a and b ):
    c = True
return c
```

Do excerto em cima apresentado podemos obter o diagrama de fluxo:

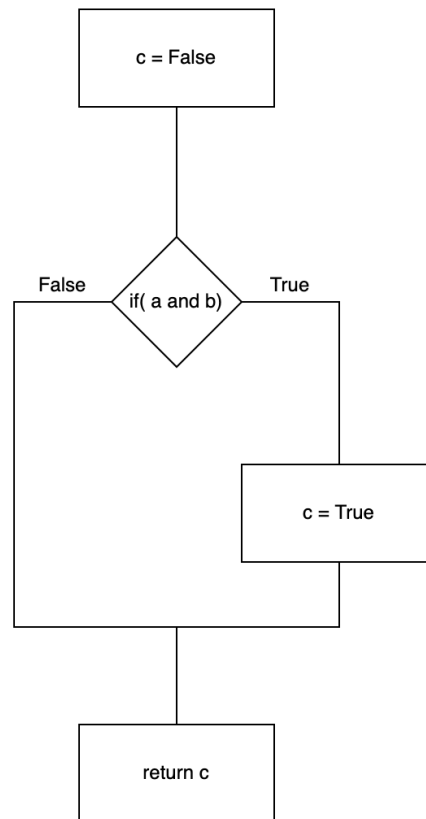


Figura 1: Diagrama de fluxo do excerto de código

Se analisarmos o diagrama facilmente percebemos que se o nosso teste for constituído com os valores de a e de b a **True** obtemos 100% de **Statement Coverage**, porém apenas 50% de **Branch Coverage**.

Para obtermos 100% de Statement Coverage e 100% de Branch Coverage bastaríamos adicionar mais um teste com um dos valores de a ou b a **False**.

Temos de realçar que se o código fosse só aquelas quatro linhas, teríamos um aumento de cobertura de 50% apenas com mais um teste. Porém, se este pedaço estivesse integrado num pedaço de código bem maior, o aumento de cobertura seria muito pequeno e exigiria a construção de mais um teste.

Mais uma vez é necessário um equilíbrio e uma reflexão sobre quais os melhores testes para haver uma maior cobertura do código.

4 Ferramentas

No âmbito do *Code Coverage* existem já varias ferramentas prontas a utilizar, para diferentes linguagens.

Iremos então abordar neste capítulo uma ferramenta para cada uma das linguagens mais populares, isto é, *C*, *Python*, *Java*, *javascript*.

4.1 C

Coco

No âmbito da linguagem *C* uma das ferramentas mais conhecidas é o **Coco**, disponível em <https://www.froglogic.com/coco/>.

Esta ferramenta permite, *function coverage*, *line coverage*, *statment coverage*, *function coverage*, *decision coverage*, e *modified/multiple condition coverage*.

Para além destas funcionalidades de *code coverage* a plataforma contem ainda o seu próprio compilador, emite relatórios de análises em diferentes formatos, e permite a integração com algumas frameworks de teste como por exemplo o **GoogleTest**.

Outra das funcionalidades importantes desta ferramenta é a integração continua, que permite executar o *code coverage* de um software, automaticamente, quando uma nova versão deste é inserida num sistema.

Exemplo

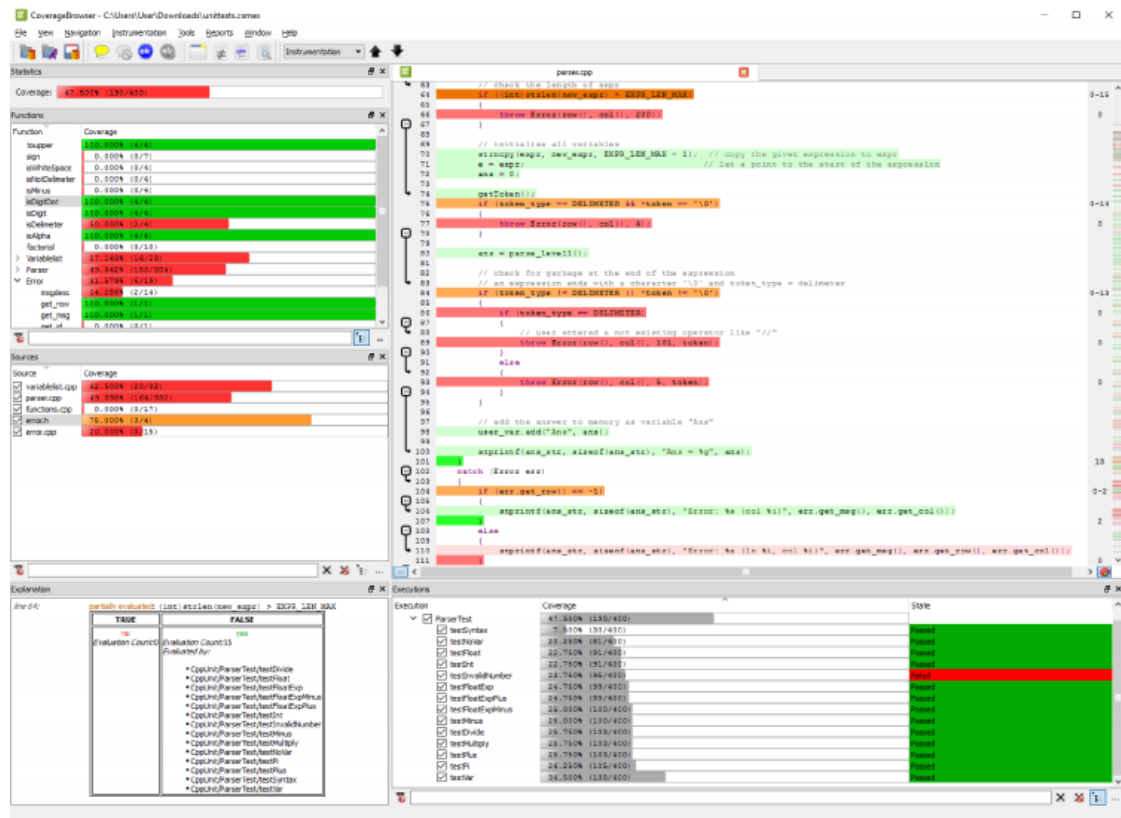


Figura 2: Exemplo do coverage com Coco

Como podemos ver com a ferramenta é possível ver no código fonte as linhas cobertas ou

não, bem como ter uma ideia das funções cobertas, e dos ficheiros. O programa suporta também possibilidade de personalizar as vistas, podendo alterar o que queremos ver no ecrã.

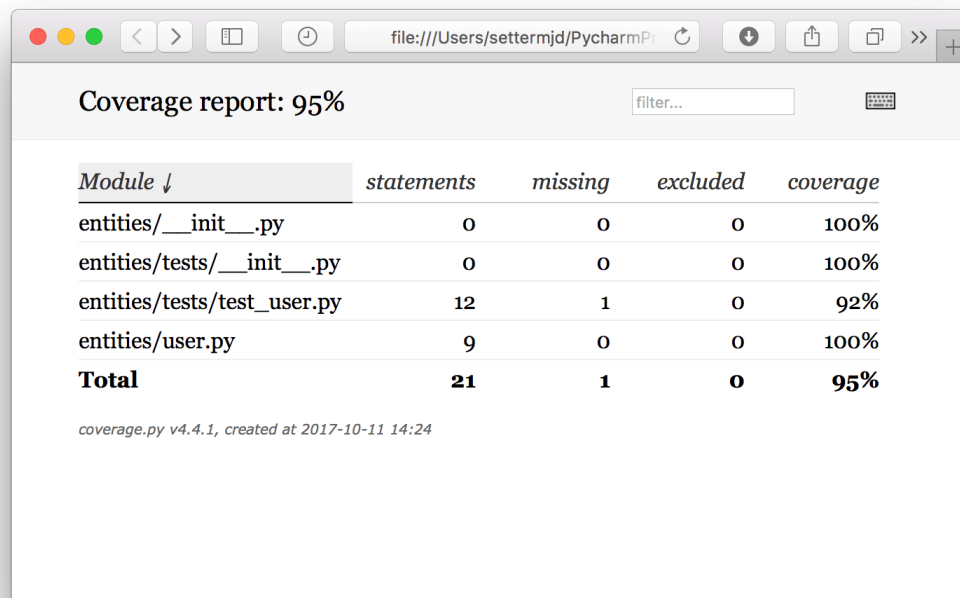
4.2 Python

Para o python a ferramenta mais conhecida existente é o *Coverage.py*, que se encontra em <https://coverage.readthedocs.io/en/coverage-5.1/index.html>. Esta biblioteca permite executar code coverage de código python. Produzindo resultados, em diversos formatos, html, JSON e xml.

Para além disso, permite ainda anotar o código com os resultados do coverage feito.

A ferramenta é de fácil instalação, basta usar o pip, e fazer **pip install coverage**, que fica logo pronta a usar.

Exemplo



Coverage report: 95%

Module ↓	statements	missing	excluded	coverage
entities/__init__.py	0	0	0	100%
entities/tests/__init__.py	0	0	0	100%
entities/tests/test_user.py	12	1	0	92%
entities/user.py	9	0	0	100%
Total	21	1	0	95%

coverage.py v4.4.1, created at 2017-10-11 14:24

Figura 3: Exemplo do coverage com Coverage.py - 1

Neste exemplo é possível ver os resultados do coverage com a ferramenta, onde é possível ver para cada ficheiro a percentagem de coverage obtida. É ainda possível, ver o ficheiro em si com o código fonte, onde são sublinhadas a verde as linhas cobertas pelo coverage. Como podemos ver na seguinte figura.



Figura 4: Exemplo do coverage com Coverage.py - 2

4.3 Java

Na linguagem Java uma das ferramentas mais conhecidas é o *Jacoco*, disponível em <https://www.eclemma.org/jacoco/>. Esta ferramenta está direcionada apenas para o *IDE*, Eclipse.

O Jacoco, para além de rápido, mostra logo os resultados do coverage anotados no código fonte. Para além disso a ferramenta é não invasiva, o que quer dizer que não são necessárias modificações ao código para executar o code coverage.

Exemplo

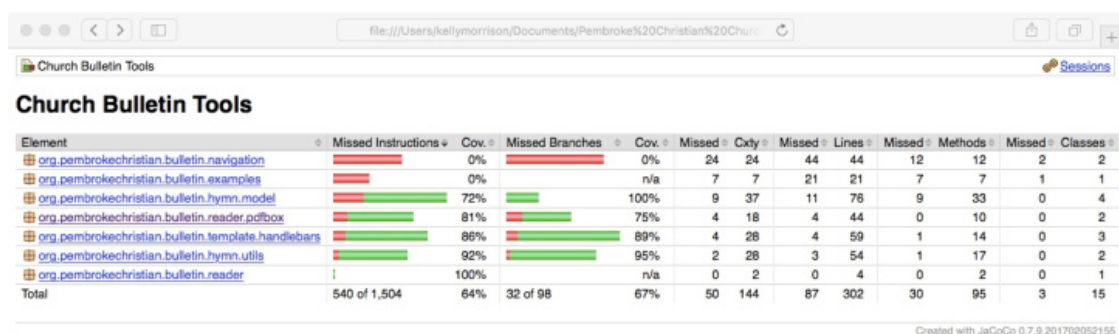


Figura 5: Exemplo do coverage com Jacoco - 1

Como podemos ver na imagem a cima, o code coverage feito em Jacoco, fica organizado num relatório, por package. Onde são apresentados os vários parâmetros de coverage medidos. Também é possível ver o dados do coverage de um ficheiro específico.

```
1. package com.baeldung.testing.jacoco;
2.
3. public class Palindrome {
4.
5.     public boolean isPalindrome(String inputString) {
6.         ◆ if (inputString.length() == 0) {
7.             return true;
8.         } else {
9.             char firstChar = inputString.charAt(0);
10.            char lastChar = inputString.charAt(inputString.length() - 1);
11.            String mid = inputString.substring(1, inputString.length() - 1);
12.            ◆ return (firstChar == lastChar) && isPalindrome(mid);
13.        }
14.    }
15. }
```

Figura 6: Exemplo do coverage com Jacoco - 1

Nesta imagem podemos ver a verde as linhas que foram cobertas pelos testes, a amarelo aquelas que só foram cobertas por determinadas branches, e a vermelho aquelas que não foram cobertas.

4.4 Javascript

Em Javascript uma das ferramentas de code coverage mais utilizadas é o *Istanbul*, disponível em <https://gotwarlost.github.io/istanbul/>. Esta ferramenta faz as operações habituais de coverage disponibilizando relatórios em html. Destacando também os resultados no código.

Exemplo

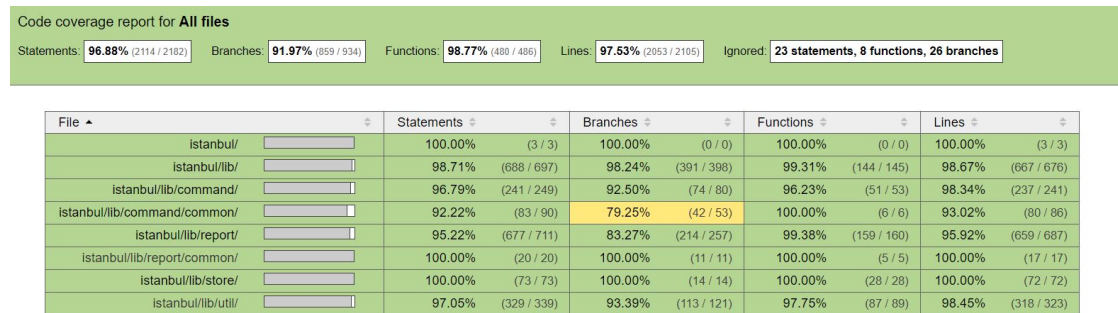


Figura 7: Exemplo do coverage com Istanbul - 1

Como podemos ver pela imagem, o relatório, em html, apresenta primeiramente o coverage de todas as subdiretórias do projeto. Mostrando a percentagem dos statments alcançados, das branches, das functions, e das lines cobertas pelo coverage.

Podemos ainda ter uma visão mais aprofundada sobre cada ficheiro, clicando na sua respectiva directoria.

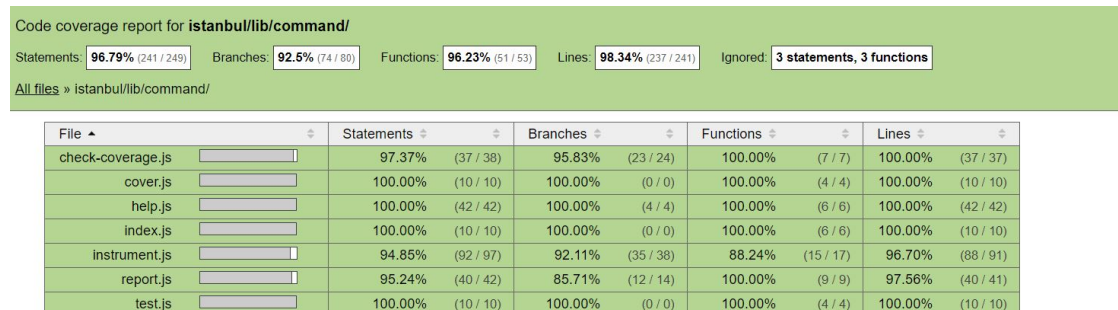


Figura 8: Exemplo do coverage com Istanbul - 2

Esta vista apresenta nos as mesmas medidas da vista anterior, mas em relação a cada ficheiro de uma certa directoria do projeto.

Por ultimo podemos ver o coverage de cada ficheiro em especifico, onde as linhas cobertas estão a verde e as que não foram cobertas estão a vermelho.



Figura 9: Exemplo do coverage com Istanbul - 3

5 Conclusão e Análise Crítica

Após a realização de mais um projeto de pesquisa voltamos a reter ensinamentos muito importantes e práticos para o mundo empresarial.

Nesta área da produção de software é costume ouvir-mos falar de qualidade de software e este trabalho ajudou-nos a compreender melhor como essa avaliação é feita e permitiu-nos aprofundar um tema que está relacionada a uma área muito cobiçada que é o Software Testing. Este projeto ajudou-nos assim a perceber algumas técnicas e ferramentas utilizados e pudemos adquirir conhecimentos bastante práticos e úteis.

Damos assim concluído o nosso trabalho prático, saindo do mesmo felizes e confiantes que os objetivos foram cumpridos com sucesso.