

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Karen Catiguá Junqueira

**Predição de defeitos usando métricas de
software**

Uberlândia, Brasil

2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Karen Catiguá Junqueira

Predição de defeitos usando métricas de software

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Rodrigo Sanches Miani

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2017

Karen Catiguá Junqueira

Predição de defeitos usando métricas de software

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 11 de Dezembro de 2017:

Rodrigo Sanches Miani
Rodrigo Sanches Miani

Professor
Paulo Henrique Ribeiro Gabriel

Professor
William Chaves de Souza Carvalho

Uberlândia, Brasil
2017

Resumo

Esta monografia analisou a relação entre dois conjuntos distintos de métricas e a ocorrência de defeitos em nível de classe. Para o conjunto de métricas básicas foram analisadas as métricas de Tamanho de Código e de Complexidade, e para o conjunto de métricas orientadas a objeto foram utilizadas quatro das seis métricas propostas por [Chidamber e Kemerer \(1994\)](#). Para realizar o experimento, foi utilizado um software de uma empresa privada. Após a extração e análise das métricas, chegou-se a conclusão que o conjunto das métricas básicas não mostrou ser eficiente como nos trabalhos anteriores. Averiguou-se que, para o trabalho em questão, as métricas propostas por *Chidamber* e *Kemerer* podem influenciar na ocorrência de defeitos.

Palavras-chave: Software, Teste, Métricas, Predição, Defeito.

Lista de ilustrações

Figura 1 – Figura retirado do trabalho da professora Bianca Zadrozny (do Instituto de Computação da UFF) para exemplificar o cálculo da complexidade ciclomática (Extraída de: (SITEMCCABE, 2017))	12
Figura 2 – Valor calculado pelo Eclipse Metrics para a métrica McCabe	13
Figura 3 – Conjunto de métricas extraída pelo Eclipse Metrics	15
Figura 4 – Detalhes de cada métrica exibido pelo Eclipse Metrics	16
Figura 5 – DashBoard da ferramenta SonarQube	17
Figura 6 – Linha de comando de como usar a ferramenta CKJM	17
Figura 7 – Linha do tempo dos artigos relacionados	18
Figura 8 – Página principal Portal Corporativo	22
Figura 9 – Página inicial após a realização do login	22
Figura 10 – Página destinada a realização dos agendamentos de veículo	23
Figura 11 – Página destinada ao cadastro de afastamento dos colaboradores	23

Lista de tabelas

Tabela 1	–	Tabela de probabilidade de defeito por métrica	25
Tabela 2	–	Tabela de Probabilidade de classes com defeitos	26
Tabela 3	–	Tabela de média e desvio padrão dos valores das métricas de classes com defeitos (A) e classes sem defeitos (B)	26
Tabela 4	–	Tabela de Correlações entre as métricas	29

Lista de abreviaturas e siglas

OO	Object-Oriented
CK	Chidamber e Kemerer
DIT	Depth of Inheritance Tree of a Class
NOC	Number Of Children of a Class
CBO	Coupling Between Object Classes
RFC	Response For a Class
LCOM	Lack of Cohesion on Methods
LOC	Lines of Code
WMC	Weighted Methods per Class
CV	Coefficiente de variação

Sumário

1	INTRODUÇÃO	8
2	REFERENCIAL TEÓRICO	10
2.1	Teste de Software	10
2.2	Métricas de Software	11
2.2.1	Métricas Básicas	11
2.2.2	Métricas CK	13
2.3	Ferramentas de Extração de Métricas	14
2.3.1	Eclipse Metrics	15
2.3.2	SonarQube	16
2.3.3	CKJM	17
2.4	Trabalhos Correlatos	17
3	DESENVOLVIMENTO	21
3.1	Metodologia	21
3.2	Resultados	25
3.3	Comparação dos Resultados	30
4	CONCLUSÃO	32
	REFERÊNCIAS	34

1 Introdução

Software é um elemento essencial para o mundo moderno. Vários setores da sociedade, como a indústria, as entidades financeiras e o entretenimento fazem uso constante de softwares. Além disso, a vida útil do software precisa ser prolongada ao máximo, com o objetivo de manter sempre otimizada a relação de custo/benefício. Portanto, de acordo com [Sommerville \(2006\)](#), a engenharia de software é necessária para o funcionamento e manutenção dessas atividades, as quais são muito importantes para a sociedade.

A Engenharia de Software pode ser definida como a área da Ciência da Computação que se dedica à construção de programas computacionais e da sua documentação necessária à sua operação e manutenção ([SOMMERVILLE, 2006](#)). Um dos campos mais importantes da Engenharia de Software é denominado Testes de Software. A atividade de teste não pode garantir a ausência de defeitos no software, todavia, seu objetivo principal é revelar a presença de defeitos.

A realização dos testes de software faz parte de um processo no qual um software é avaliado a fim de identificar erros que foram gerados inadvertidamente no momento em que o sistema foi projetado ou construído ([PRESSMAN, 2010](#)). Tal processo de encontrar defeitos em softwares pode se tornar complexo e oneroso. Desta forma, como uma medida de redução de tempo de trabalho, de esforço da equipe e de custos, a detecção de defeitos antecipada torna-se uma grande aliada para a área de testes de software. Segundo [Basili et al. \(1996\)](#), buscar e corrigir problemas de software depois da entrega é, em média, cem vezes mais caro do que buscar e encontrar os mesmos problemas durante as atividades de requisitos ou de projeto.

Um dos desafios atuais na área de testes de softwares é a identificação de trechos do código que podem influenciar na ocorrência de erros. Para ajudar nesta identificação, as métricas de software podem ser usadas em modelos de previsão de falhas para melhorar a qualidade do software, de modo a prever a localização da mesma ([RADJENOVIC, 2013](#)). A fim de constatar quais são esses trechos e o porquê deles resultarem em algum defeito, este trabalho tem o objetivo de realizar uma análise referente a relação entre algumas métricas de Software e a ocorrência de defeitos. Ao decorrer desta monografia, serão mostradas quais métricas foram escolhidas e o porquê destas escolhas.

Alguns estudos já foram feitos para identificar a relação entre os defeitos de software e as métricas de software. Como exemplo, o trabalho proposto por [Lucca, Vilela e Jino \(2007\)](#) mostra a relação entre as métricas de tamanho e de complexidade com a quantidade de defeitos de software. Os autores apresentaram resultados de um experimento com um programa que contém defeitos reais documentados. Os dados coletados

apresentaram que, para o programa em questão, há maior probabilidade de conter defeitos nos módulos maiores e mais complexos do que nos menores e menos complexos.

A finalidade do presente trabalho é de analisar se os resultados encontrados podem ser aplicados em softwares reais, ou seja, se as métricas selecionadas para fazer a predição de defeitos realmente irão ajudar a identificar a ocorrência de defeitos no software testado. Existem outros estudos que abordam o mesmo assunto, porém a maioria dessas pesquisas envolve o uso de bases de dados públicas, como *Firefox* (CHOWDHURY; ZULKERNINE, 2011), *Windows Vista* (BASILI et al., 1996) e o software da Agência Espacial Europeia (*Space*) (LUCCA; VILELA; JINO, 2007). Neste trabalho, foi explorada a relação entre as métricas de software e a ocorrência de defeitos em sistemas computacionais comerciais.

O objetivo geral deste trabalho é utilizar os conceitos de métricas de Software para avaliar a ocorrência de defeitos. Para tanto, foi preciso pesquisar ferramentas que seriam necessárias para a extração das métricas de software, selecionar um sistema alvo para realização dos testes e documentação de defeitos. Assim sendo, com os defeitos documentados e as métricas coletadas, pode-se efetuar uma comparação entre as métricas encontradas no sistema selecionado e os defeitos registrados.

No final desta monografia foi constatado que, diferentemente do esperado, as métricas básicas utilizadas não mostraram ser tão eficientes como nos trabalhos estudados. Também constatamos que, dentro do conjunto de métricas propostas por Chidamber e Kemerer (1994), a que mais se destacou foi a métrica de Acoplamento entre objetos.

Este trabalho foi organizado em capítulos. No segundo capítulo, é apresentado a revisão dos conceitos relacionados a esta monografia, como testes de software, métricas de software além de apresentar as ferramentas utilizadas para a extração de métricas. Ainda no segundo capítulo, é apresentado uma revisão bibliográfica dos trabalhos relacionados. O terceiro capítulo descreve a metodologia desenvolvida, desde a escolha do programa até a extração das métricas escolhidas. Em seguida, no referido capítulo, é realizada uma análise acerca dos resultados extraídos e as devidas conclusões para cada análise também são apresentadas. Finalmente, o último capítulo apresenta as considerações finais desta monografia, comparando os resultados encontrados com outros trabalhos relacionados, e as melhorias encontradas.

2 Referencial Teórico

Neste capítulo são apresentados os conceitos relevantes para a realização deste trabalho e os trabalhos correlacionados. Primeiramente, os conceitos básicos são explicados para o entendimento desta dissertação, como teste de software e métricas de software. Por fim, é feita uma breve síntese dos artigos que influenciaram na escrita deste trabalho.

2.1 Teste de Software

Teste de software é um processo que envolve as atividades, estáticas ou dinâmicas, do ciclo de vida de um software. Esse processo concentra-se no planejamento, na preparação e na avaliação dos produtos de software para demonstrar que eles satisfazem os requisitos especificados, que estão aptos para cumprir sua função e que estão livres de defeitos ([GRAHAM et al., 2008](#)).

Para realizar testes em um software precisa-se escolher uma estratégia de teste que irá definir o roteiro que deve ser seguido. Segundo [Pressman \(2010\)](#), algumas estratégias de teste de software têm sido propostas na literatura e certas características são genéricas e compartilhadas por tais estratégias. Por exemplo: (a) a execução de revisões técnicas formais; (b) os testes iniciam a nível dos componentes e diminuem de granularidade em direção ao teste do sistema como um todo; (c) a cada etapa, determinadas técnicas de teste são mais adequadas do que outras; (d) os testes são conduzidos pelo desenvolvedor de software e por uma equipe de teste independente.

A atividade de teste de software pode ser realizada de várias maneiras. Existe um modelo de teste que é conhecido como testes caixa-branca ou estruturais, que são utilizados para testar a estrutura interna do software. Este tipo de teste analisa o código-fonte através de casos de testes, que são conjuntos de condições que determinam se um requisito ou um caso de uso está sendo satisfeito. Outro modelo de teste é conhecido como teste de caixa-preta, ou teste funcional. Tal teste é realizado sem acesso ao código-fonte do software, em que são fornecidas apenas algumas entradas para uma dada funcionalidade do software e então são comparadas as saídas geradas com resultados previamente conhecidos ([EUGÊNIO; BEZERRA, 2008](#)).

Neste trabalho, foram feitos testes de caixa-branca e de caixa-preta. Para realização dos testes de caixa-branca, foram implementados, na linguagem Java, diversos testes unitários utilizando o *framework* [JUnit \(2017\)](#), o qual se baseia em testar a menor parte do código garantindo uma maior qualidade do software. Este tipo de teste é usado para mensurar e avaliar a estrutura interna de um software.

Já para a execução dos testes de caixa-preta, foi utilizada a ferramenta [Selenium \(2017\)](#), que auxilia na automatização dos testes realizados por meio de navegadores. Este tipo de teste funciona da seguinte maneira: para cada grupo de determinadas entradas em cada módulo do sistema, as saídas geradas são comparadas com os resultados esperados. Na seção de resultados desta monografia, será efetuada uma análise dos defeitos que os testes de caixa-branca e caixa-preta encontraram.

2.2 Métricas de Software

Pesquisas desenvolvidas desde o início do século XX têm mostrado a utilidade de métricas de software como fonte de informação para modelos de predição de módulos defeituosos ([SUBRAMANYAM; KRISHNAN, 2003](#)). A medição do software preocupa-se com a derivação de um valor numérico ou o perfil para um atributo de um componente de software, sistema ou processo ([SOMMERVILLE, 2006](#)).

Segundo Sommerville ([SOMMERVILLE, 2006](#)), um dos objetivos de se medir software é fornecer a capacidade de fazer julgamentos, a longo prazo, sobre a qualidade de software, ao invés de realizar revisões. Ou seja, que sistemas fossem aprovados ou reprovados de acordo com os valores de um conjunto de métricas.

[Pressman \(2010\)](#) afirma que durante as atividades de teste e de manutenção de software, as métricas fornecem informações detalhadas que ajudam a localizar áreas de instabilidade potencial nos módulos do sistema. Nos últimos anos, aumentou substancialmente o uso de métricas de software durante o desenvolvimento de um sistema. Por exemplo, as métricas de [Chidamber e Kemerer \(1994\)](#) foram utilizadas para diversos estudos, como predizer a propensão de erros que uma classe pode ter ([JULIANO, 2014](#)).

2.2.1 Métricas Básicas

Para a execução desta monografia, foram escolhidas algumas métricas tradicionais, como a de tamanho e a de complexidade. Ambas estão sendo analisadas neste trabalho, pois mostraram ser relevantes na detecção de defeitos de software, como será ressaltado no capítulo 3 e na seção 2.4. Para a métrica de tamanho, será utilizado *LOC - Line of Code*. Já para a complexidade, a métrica de *McCabe* ([MCCABE, 1976](#)) será utilizada.

A métrica de linhas de código possui a vantagem de ser uma métrica objetiva. Basta analisar a quantidade de linhas de código e obtém-se o valor do que foi produzido. Contudo, para realizar uma comparação entre linhas de código de sistemas diferentes, deve-se levar em consideração a linguagem em que o sistema foi produzido. Por exemplo, uma mesma função pode ser implementada na linguagem Java com apenas 100 linhas de código, enquanto na linguagem C podem ser usadas 200 linhas para implementar a mesma função.

Já a métrica de complexidade ciclomática, proposta por *McCabe*, mede a quantidade de caminhos de execução independentes a partir do código fonte. Essa complexidade é computada através de um grafo de fluxo de controle (Figura 2), onde os nós são os comandos, e as arestas entre os nós indicam se um comando pode ser executado após o outro. Uma estratégia de teste de software proposta por *McCabe* é testar cada caminho independente de um programa, ou seja, testar todos os caminhos formados pelo grafo de fluxo de controle, de forma que a quantidade de casos de teste será a complexidade ciclomática do programa.

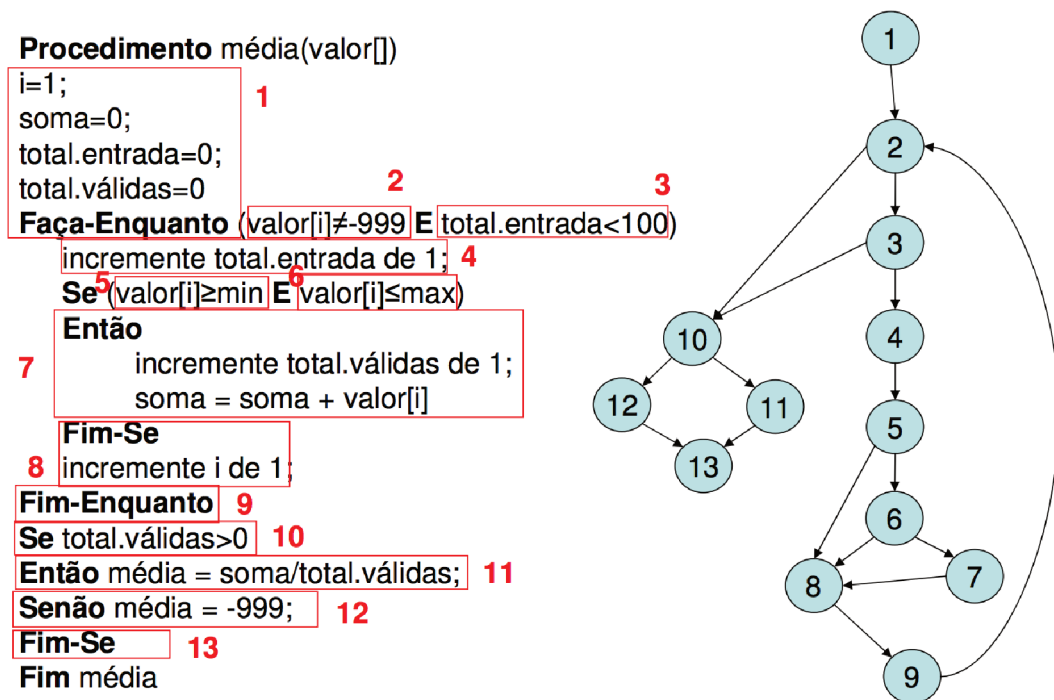
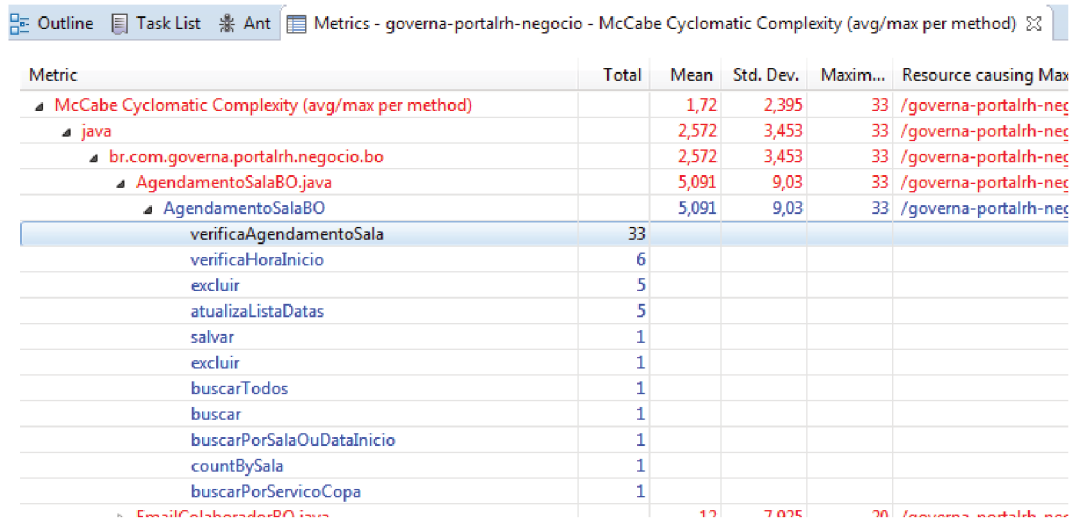


Figura 1 – Figura retirado do trabalho da professora Bianca Zadrozny (do Instituto de Computação da UFF) para exemplificar o cálculo da complexidade ciclomática (Extraída de: (SITEMCCABE, 2017))

Para a realizar a extração desta métrica, a ferramenta *Metrics* (2016) foi utilizada para auxiliar no cálculo desta métrica. Na Figura 2, pode-se ver os valores calculados para a métrica em uma classe do sistema que foi utilizado para realizar os testes desta monografia (como será apresentado na secção 3). Observa-se que a ferramenta apresenta os valores da métrica para cada método dentro da classe analisada.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum
▲ McCabe Cyclomatic Complexity (avg/max per method)		1,72	2,395	33	/governa-portalrh-negocio
▲ java		2,572	3,453	33	/governa-portalrh-negocio
▲ br.com.governa.portalrh.negocio.bo		2,572	3,453	33	/governa-portalrh-negocio
▲ AgendamentoSalaBO.java		5,091	9,03	33	/governa-portalrh-negocio
▲ AgendamentoSalaBO		5,091	9,03	33	/governa-portalrh-negocio
verificaAgendamentoSala	33				
verificaHoraInicio	6				
excluir	5				
atualizaListaDatas	5				
salvar	1				
excluir	1				
buscarTodos	1				
buscar	1				
buscarPorSalaOuDataInicio	1				
countBySala	1				
buscarPorServicoCopa	1				

Figura 2 – Valor calculado pelo Eclipse Metrics para a métrica McCabe

2.2.2 Métricas CK

Sistemas desenvolvidos em Java, ou em outras linguagens orientadas a objetos, possuem uma abordagem diferente na implementação e no desenvolvimento do projeto. Como este tipo de linguagem trata objetos e não blocos estruturados, como C e outras linguagens mais antigas, a abordagem de medição é diferente. Pois, além da análise do código da classe principal do projeto, também é necessário analisar as classes acopladas a essa classe.

Chidamber e Kemerer (1994) formaram um conjunto de seis métricas para definir a complexidade de um sistema orientado a objeto. Mais conhecido como métricas *CK*, é um dos conjuntos mais conhecidos de métricas orientadas a objetos e trata as classes de forma individual analisando o seu posicionamento na hierarquia de classes, a sua coesão e acoplamento. Nesta monografia, serão selecionadas quatro das seis métricas *CK*, definidas a seguir:

- ***LCOM - lack of cohesin in methods***: em português, a métrica de ausência de coesão em métodos, é utilizada para calcular quantos conjuntos de métodos relacionados existem dentro de uma classe, isto é, métodos que compartilham a utilização de algum atributo ou que se referenciam. A ideia é de mostrar que os métodos de uma classe estão coesos se eles utilizam os mesmos atributos dentro dessa classe. Se algum método não utiliza nada da classe, ou utiliza apenas métodos/atributos de outra classe, ele provavelmente está no lugar errado.
- ***WMC - weighted methods per class***: em português, a métrica de métodos ponderados por classe, é responsável por calcular a complexidade de cada classe, sendo que esse calculo é feito a partir da soma das complexidades ciclômáticas dos

métodos dessa classe. O cálculo da complexidade ciclomática já foi exemplificado na seção 2.2.1.

- **CBO - coupling between object classes:** em português, a métrica de acoplamento entre objetos, que é responsável por calcular as conexões de entrada e de saída de uma classe, isto é, para cada classe, são contabilizadas classes que utilizam algum método ou variável dela, como também todas as classes que essa classe referencia. Se uma classe possui um valor alto para a métrica CBO significa que existem classes que são altamente dependentes dela e, portanto, é mais provável que a mudança desta classe afete outras classes do programa.
- **RFC - response for a class:** em português, a métrica de resposta de uma classe, é responsável por calcular o número de métodos que podem ser executados a partir de uma mensagem enviada a um objeto de uma classe. Se uma classe possuir um alto valor de RFC pode ser que tenha um número muito grande de métodos, e/ou pode ser uma classe bastante dependente de outra classe. Um valor alto para esta métrica pode indicar baixa coesão e alto acoplamento.

As duas métricas de Chidamber e Kemerer (1994) que não serão utilizadas nesta monografia são: **DIT - Depth of Inheritance Tree** e **NOC - Number of Children**, em português, profundidade na árvore de herança e número de subclasses, respectivamente. Estas métricas não serão utilizadas, pois para o sistema alvo escolhido os valores encontrados não mostraram ser relevantes. Como a métrica *DIT* representa o número de níveis que uma classe herda métodos e atributos, no sistema alvo todas as classes que foram testadas herdaram métodos e atributos de uma única classe, ou seja, para todas elas o valor encontrado para a métrica foi de um. E para a métrica *NOC*, que conta o número de subclasses diretas de uma classe, o valor encontrado foi de zero para todas as classes, ou seja, nenhuma delas possuía classes que herdavam seus atributos e métodos diretamente.

2.3 Ferramentas de Extração de Métricas

Para o desenvolvimento desta monografia, foram utilizadas ferramentas que auxiliaram na extração das métricas de software. Como a intenção deste trabalho é propor a relação de métricas não convencionais com a ocorrência de defeitos, foi utilizada mais de uma ferramenta de extração. Após uma vasta pesquisa, foram escolhidas três ferramentas: Eclipse Metrics, SonarQube e CKJM.

2.3.1 Eclipse Metrics

A ferramenta *Eclipse Metrics* (METRICS, 2016) é um plug-in que calcula diversas métricas para o seu código durante os ciclos de compilação. Esta ferramenta é uma extensão do ambiente de desenvolvimento Eclipse, que suporta linguagens como Java, C, C++, Python, entre outras. Na Figura [3], observa-se algumas das métricas que esta ferramenta é capaz de extrair.

Para cada métrica, a ferramenta possui o valor máximo aceitável. A partir deste valor, ela exibe as classes e funções do projeto que possuem os valores superiores ao valor permitido para cada métrica. Por exemplo, na Figura [4], pode-se verificar que, para a métrica de complexidade ciclomática, existe uma função cujo o valor calculado para a métrica é superior ao valor permitido.

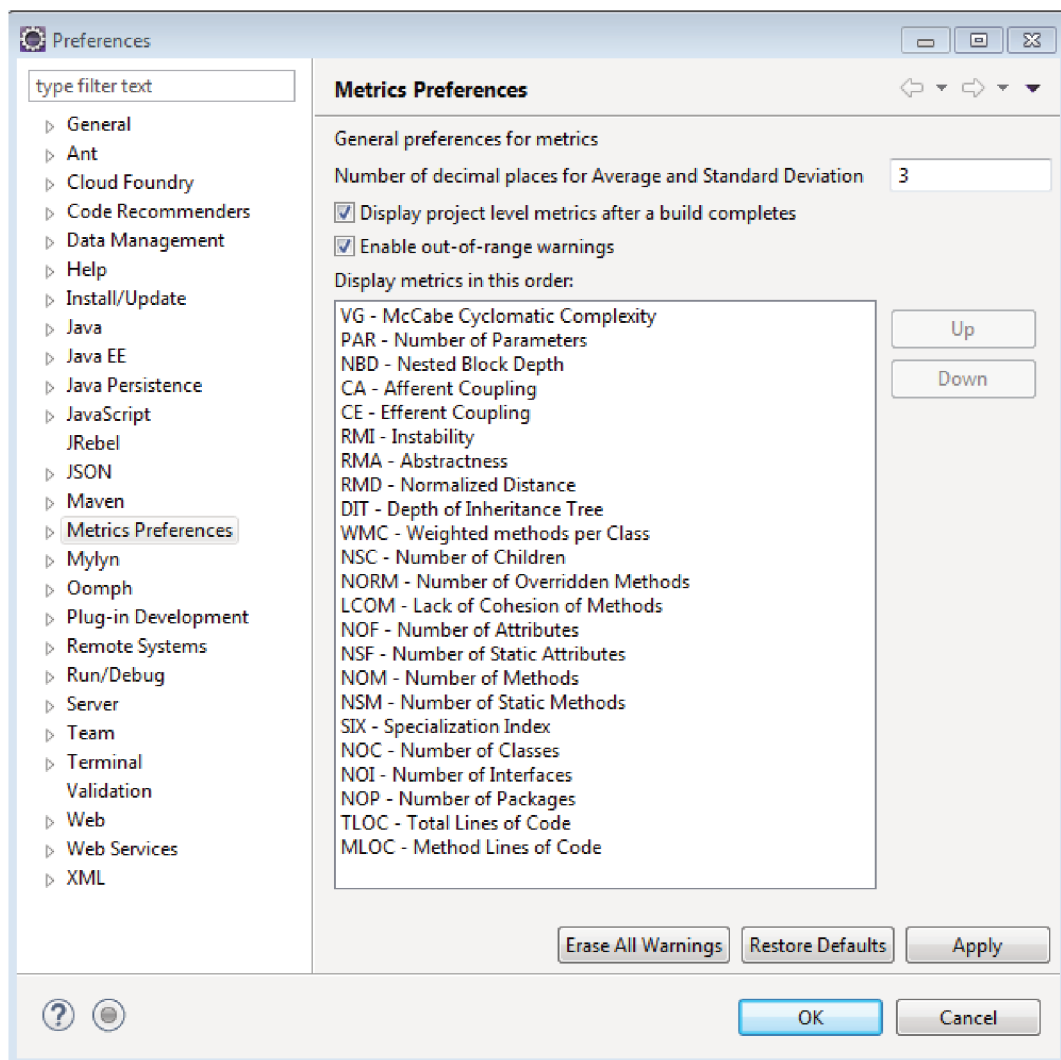
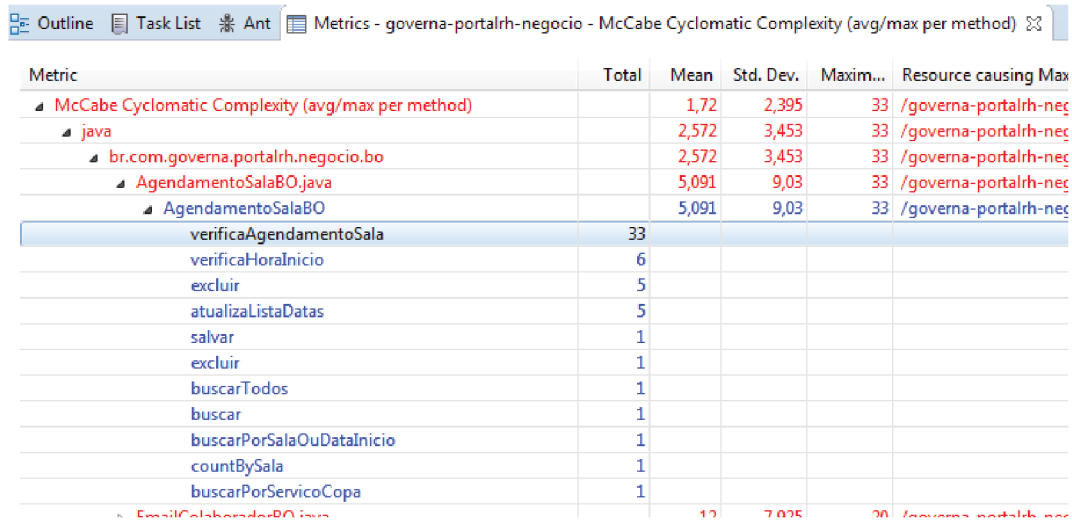


Figura 3 – Conjunto de métricas extraída pelo Eclipse Metrics



The screenshot shows the Eclipse Metrics tool interface. The title bar indicates the project is 'governa-portalrh-negocio' and the metric being calculated is 'McCabe Cyclomatic Complexity (avg/max per method)'. The table below displays the results for various classes and methods.

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Max
▲ McCabe Cyclomatic Complexity (avg/max per method)		1,72	2,395	33	/governa-portalrh-negocio
▲ java		2,572	3,453	33	/governa-portalrh-negocio
▲ br.com.governa.portalrh.negocio.bo		2,572	3,453	33	/governa-portalrh-negocio
▲ AgendamentoSalaBO.java		5,091	9,03	33	/governa-portalrh-negocio
▲ AgendamentoSalaBO		5,091	9,03	33	/governa-portalrh-negocio
verificaAgendamentoSala	33				
verificaHoraInicio	6				
excluir	5				
atualizaListaDatas	5				
salvar	1				
excluir	1				
buscarTodos	1				
buscar	1				
buscarPorSalaOuDataInicio	1				
countBySala	1				
buscarPorServicoCopa	1				
EnviarCalendarioBO.java		12	7,035	20	/governa-portalrh-negocio

Figura 4 – Detalhes de cada métrica exibido pelo Eclipse Metrics

2.3.2 SonarQube

A próxima ferramenta, que também auxiliou no desenvolvimento desta monografia, é o *SonarQube* (SONARQUBE, 2017). Tal ferramenta é uma plataforma de código aberto para a inspeção contínua da qualidade do código. Na Figura [5], pode-se observar a página principal da ferramenta. Nesta tela, são exibidas algumas das principais métricas que a ferramenta calcula, como linha de código e complexidade, e também é exibido a cobertura dos testes unitários junto da sua taxa de sucesso e erros encontrados.

Para calcular a cobertura de testes unitários, esta ferramenta utiliza os testes realizados pelo *JUnit*, já citado anteriormente. A partir dos testes realizados pelo *JUnit*, a ferramenta *SonarQube* mostra a cobertura dos testes. Caso algum defeito seja encontrado, ela apresenta o erro encontrado detalhadamente, além do local exato de onde ocorreu tal defeito.

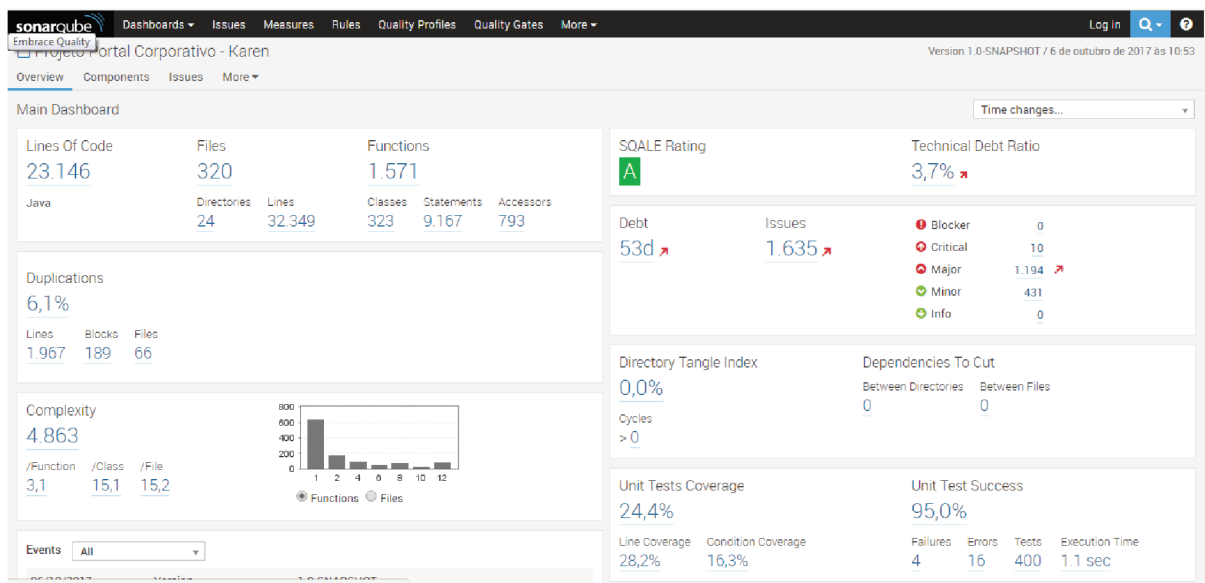


Figura 5 – DashBoard da ferramenta SonarQube

2.3.3 CKJM

Por fim, a ferramenta *CKJM* (SPINELLIS, 2005) calcula as métricas orientadas a objetos propostas por *Chidamber* e *Kemerer*. O cálculo é feito a partir do processamento do *bytecode* de arquivos Java compilados. Para extrair as métricas utilizando esta ferramenta, é preciso utilizar a linha de comando diretamente, pois ela não oferece nenhuma interface com ambientes de desenvolvimento.

Para realizar a análise, deve-se executar, pela linha de comando, o java com a *flag -jar*, fornecendo como argumento a localização do arquivo *CKJM.jar*. Em seguida, especifica-se como argumentos os arquivos de classes que será analisada. A fim de exemplificar como foi utilizada esta ferramenta, a Figura 6 ilustra que, para cada classe analisada, o programa retorna um conjunto de seis números, onde cada um representa uma métrica proposta por *Chidamber* e *Kemerer*.

```
C:\Users\karen>java -jar C:/Users/karen/Desktop/ckjm-1.9/ckjm-1.9/build/ckjm-1.9.jar D:/Projetos/TCC/Fonte/trunk/governa-portalrh-all/governa-portalrh-negocio/target/classes/br/com/governa/portalrh/negocio/bo/FeriasExtratoB0.class
```

Figura 6 – Linha de comando de como usar a ferramenta CKJM

2.4 Trabalhos Correlatos

Nesta seção são apresentados os principais trabalhos que influenciaram na dissertação desta monografia. Para o melhor entendimento dos trabalhos e de suas influências,

eles estão organizados cronologicamente de acordo com a Figura 7. A partir desta linha do tempo, observa-se o título de cada trabalho selecionado e seu ano de publicação. A intenção de organizá-los desta maneira foi de compreender o impacto que um trabalho causa nos trabalhos posteriores, e observar o avanço que ocorreu ao decorrer do tempo.

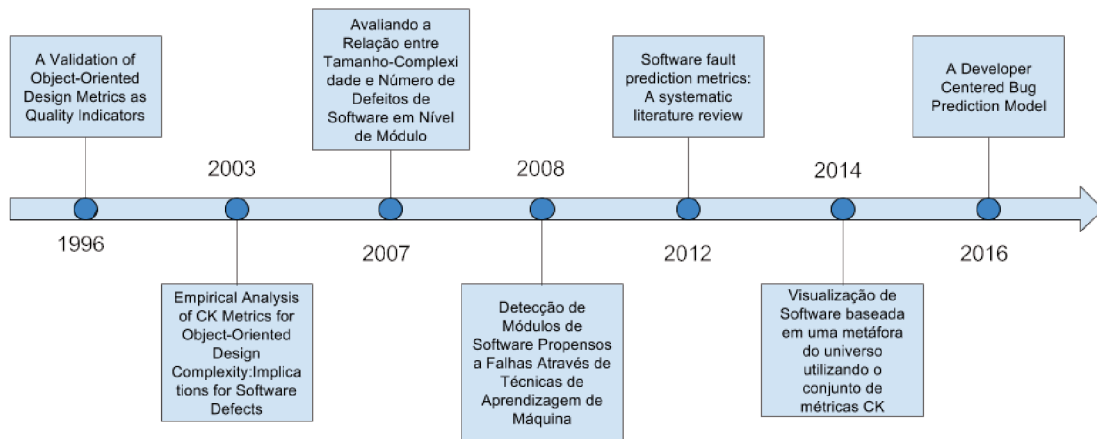


Figura 7 – Linha do tempo dos artigos relacionados

O trabalho proposto por Basili et al. (1996), avalia as métricas de *Chidamber* e *Kemerer* (CHIDAMBER; KEMERER, 1994) como preditoras de classes propensas a falhas. O autor se baseia na necessidade de diminuir o esforço efetuado pelas equipes de teste para identificar e verificar todos os módulos de um sistema, além de identificar onde se encontram as falhas. Por isso, as métricas de software se tornam essenciais. Elas são fontes de informações para tomada de decisões, o que torna a atividade de teste de software mais eficiente, pois ao identificar módulos propensos a falhas, a atividade de teste de software pode concentrar seus esforços em testar e verificar esses módulos.

Para realizar os testes propostos, Basili et al. (1996) utilizou-se das seguintes métricas: Profundidade da Árvore de herança de uma classe (DIT), Número de filhos de uma classe (NOC), Acoplamento entre classes de objetos (CBO), Resposta para uma classe (RCF), Falta de Coesão de Métodos (LCOM). O software que foi utilizado neste artigo foi o *Windows Vista* e, após identificarem quais classes possuíam falhas, foi verificado o quanto essas classes estavam sendo influenciadas por características orientadas à objetos internas e externas. Os resultados mostraram que cinco das seis métricas testadas aparentaram ser boas fontes de predição de classes propensas a falhas.

Alguns anos depois, Subramanyam e Krishnan (2003) desenvolveram uma pesquisa a partir das métricas de projeto baseadas na abordagem orientada à objeto (OO). Os autores afirmaram que as métricas de projeto desempenham um papel importante para ajudar os desenvolvedores a entender os aspectos de design do software e, portanto, melhorar a qualidade do software e a produtividade do desenvolvedor. O estudo mostrou a importância da detecção e remoção de defeitos antes da entrega ao cliente, devido ao

seu potencial papel na influência da satisfação do cliente e as implicações econômicas negativas gerais dos produtos de software com defeito de envio.

Na pesquisa citada acima, [Subramanyam e Krishnan \(2003\)](#) apresentaram evidências em apoio à associação entre um subconjunto de métricas de [Chidamber e Kemerer \(1994\)](#) (CK) e defeitos. O estudo apresentou o efeito das métricas de CK em defeitos após o controle de tamanho de software, e também validou a associação entre um subconjunto de métricas e defeitos CK em dois ambientes usando linguagens de programações distintas, C++ e Java. Os autores também ressaltaram que, diferentemente de outros estudos publicados anteriormente ao deles, na metodologia, foi usado regressão linear ponderada para estudar o efeito de interação de algumas das métricas de CK em defeitos de software.

Para realizarem os testes, o subconjunto de métricas de CK escolhidas foram: peso do método por classe (WMC), CBO e DIT. Também foi utilizada uma métrica de tamanho, a métrica de Linhas de Código (LOC) que se baseia na quantidade de linhas de código que o programa possui. Ao final dos testes, os autores descobriram que, depois de controlar o tamanho do código nas duas linguagens de programação - C++ e Java - algumas medidas do conjunto de métricas de CK explicam a variação nos defeitos significativamente.

Em 2007, [Lucca, Vilela e Jino \(2007\)](#) propuseram um trabalho que apresenta resultados de um experimento com um programa que contém defeitos reais documentados. No referido trabalho, é avaliado a relação entre as métricas de tamanho e de complexidade com a quantidade de defeitos de software. Os resultados mostraram que há maior probabilidade de conter defeitos nos módulos maiores e mais complexos do que nos menores e menos complexos.

[Lucca, Vilela e Jino \(2007\)](#) definiram em seu trabalho, como hipótese principal, que um módulo maior ou mais complexo teria mais chance de conter defeitos do que um módulo menor e menos complexo, e como hipótese secundária foi definido que o tempo necessário para a revelação dos defeitos de um software é menor quando os módulos maiores ou mais complexos são testados primeiro. Para realização dos testes, o experimento utilizou o software *Space*, da Agência Espacial Europeia, na linguagem C.

Após terem definido as hipóteses e o software a ser testado, os autores ([LUCCA; VILELA; JINO, 2007](#)) definiram que além do número de defeitos, também iriam utilizar as métricas de linhas de código (LOC), complexidade ciclomática de *McCabe* e o número de mutantes. A contagem do número de linhas de código foi feita por módulo, a complexidade ciclomática ($V(G)$) foi calculada a partir do número de comandos de decisão do programa mais um, e o número de mutantes de cada módulo foi calculado utilizando-se uma ferramenta chamada *Proteum*.

Para realizar os testes foi definido como critério de avaliação o número de módulos

a serem testados até que todos os defeitos conhecidos do software fossem revelados. Se o teste dos módulos do programa numa certa ordem de tamanho ou complexidade permitisse revelar 100% dos defeitos conhecidos antes do que o teste dos mesmos módulos em ordem aleatória, então a hipótese seria considerada aceita. Para o teste da hipótese principal, foi calculado o número de módulos que provavelmente teriam que ser testados até que o último defeito conhecido fosse revelado numa ordem aleatória, e esse número foi comparado com o número de módulos que seriam necessários testar até encontrar o último defeito conhecido na ordem de cada uma das métricas de tamanho e complexidade utilizadas. Os dados coletados mostraram que, para o programa em questão, a hipótese secundária, de que há maior probabilidade de conter defeitos nos módulos maiores e mais complexos do que nos menores e menos complexos.

Com base nas informações recolhidas no estudo destes trabalhos, essa monografia também visa encontrar uma relação entre as métricas de softwares e defeitos documentados. Os dois primeiros trabalhos estudados investigaram essa relação usando as métricas CK, enquanto o trabalho proposto por Lucca ([LUCCA; VILELA; JINO, 2007](#)) utilizou métricas convencionais, como LOC e complexidade ciclomática. Nesses trabalhos estudados, foi mostrado que existe uma relação entre determinadas métricas de software com a ocorrência de defeitos.

A principal diferença entre os trabalhos estudados com relação a esta monografia, é que neste estudo é usado um sistema comercial de uma empresa local. Com relação a metodologia, o trabalho que mais se assemelha a essa monografia é o trabalho proposto por [Lucca, Vilela e Jino \(2007\)](#). Contudo, usaremos outro conjunto de métricas, mesclando as métricas convencionais, como LOC e complexidade ciclomática, com as métricas CK. Assim, será verificada a importância da utilização de métricas de software para detecção de erros em escala diferente dos demais trabalhos estudados nessa secção.

3 Desenvolvimento

Neste capítulo são apresentados os passos que foram seguidos para o desenvolvimento deste trabalho, desde a escolha do programa a ser utilizado até a análise dos resultados encontrados. Inicialmente, é explicada toda a metodologia utilizada para o desenvolvimento, e em seguida os resultados e suas devidas análises.

3.1 Metodologia

O objetivo inicial desta monografia é replicar o trabalho feito por [Lucca, Vilela e Jino \(2007\)](#), de modo a seguir os passos que os autores descreveram ao decorrer do trabalho, além de verificar se os resultados obtidos por eles são consistentes usando sistema. Os autores mostraram que duas das três métricas utilizadas provaram ter alguma relação com a ocorrência de defeitos. Desta forma, para este trabalho, essas mesmas métricas serão utilizadas. Além disso, com a finalidade de encontrar alguma métrica mais eficiente, também serão utilizadas novas métricas.

Logo, seguindo o trabalho citado anteriormente, a seguinte sequência de passos foi adotada para esta monografia:

1. Escolha do Programa
2. Realização dos testes de software
3. Extração das métricas
4. Escolha das Métricas de Software
5. Análise dos resultados

Como já citado anteriormente, o experimento realizado neste trabalho utilizou um sistema desenvolvido por uma empresa privada, em Java, usado para auxiliar no gerenciamento interno, chamado Portal Corporativo (Figura [8]). O software possui diversas funcionalidades, sendo que as principais delas são: cadastro de funcionários, cadastro de equipes, controle de férias, controle de folgas, controle de uso das salas e reuniões, controle do uso dos veículos da empresa, controle das despesas geradas pelos veículos, controle de viagens, etc. As Figuras 9, 10 e 11 apresentam algumas dessas funcionalidades. O sistema possui 320 classes que estão divididas em 5 módulos, que totalizam 23.146 linhas de código.

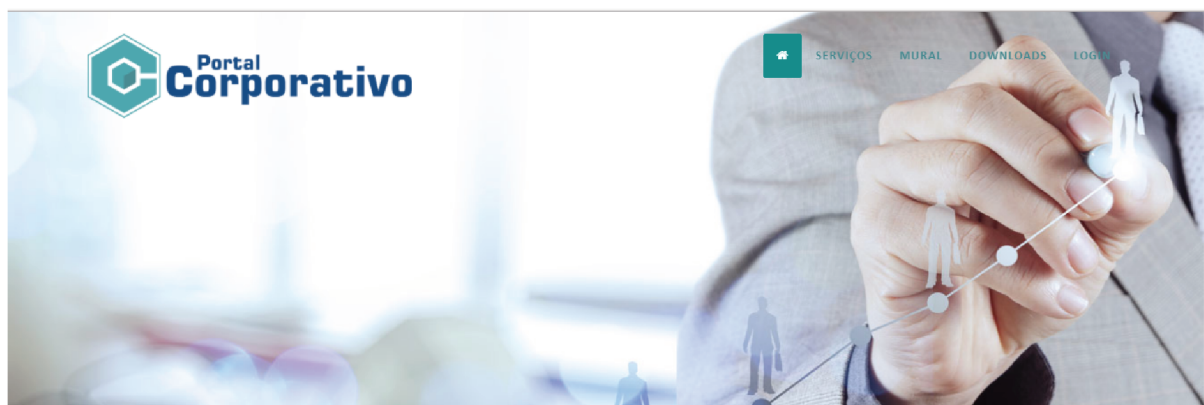


Figura 8 – Página principal Portal Corporativo

Dentre as principais funcionalidades do sistema, as Figuras 9, 10 e 11 são utilizadas para o melhor entendimento do funcionamento e objetivo do sistema. Assim que o colaborador faz *login* no portal, ele é redirecionado para a página inicial do sistema, Figura [9]. Um dos objetivos do sistema é auxiliar no agendamento de veículos da empresa, de modo a registrar em qual cidade o colaborador irá viajar, os dias da viagem e o veículo que ele utilizará. Na Figura [10], pode-se observar uma página com o resumo desses agendamentos. Outro objetivo do sistema é registrar folgas, férias e afastamentos dos colaboradores, como exibido na Figura [11].

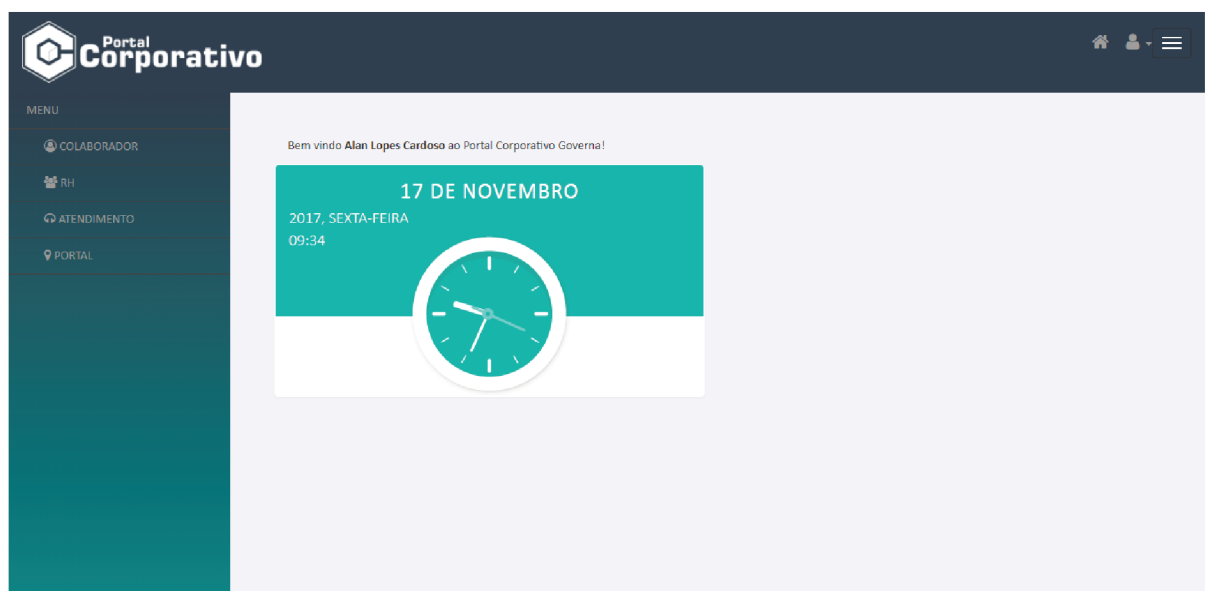


Figura 9 – Página inicial após a realização do login

	Motorista	Veiculo	Inicio	Fim	Status
	Eduardo David Hipolito	OPJ - 5766	01/03/2017	04/03/2017	Encerrado
	Gustavo Hipolito Alves Naves	HMX - 2263	01/03/2017	06/03/2017	Encerrado
	Oscar Vinicius Florindo Naves	HIG - 9699	02/03/2017	06/03/2017	Encerrado
	Thiago Santos Dutra	PYM-4770	01/03/2017	08/03/2017	Encerrado
	José Odélio Ferreira França	HMV - 5905	02/03/2017	08/03/2017	Encerrado
	Petronio Primo Naves	HGA - 3217	02/03/2017	04/03/2017	Encerrado
	Donizetti Sebastiao de Jesus	HMV - 5901	02/03/2017	04/03/2017	Encerrado
	Donizetti Sebastiao de Jesus	HMV - 5905	03/03/2017	04/03/2017	Encerrado
	Luiz Antonio Alcantara	PYJ-1427	03/03/2017	06/03/2017	Encerrado

Figura 10 – Página destinada a realização dos agendamentos de veículo

	Motivo	Data inicio	Data fim
	Licença Maternidade	03/10/2016	30/01/2017

Total de 1

← Primeiro 1 Último → 25

Figura 11 – Página destinada ao cadastro de afastamento dos colaboradores

Primeiramente, para que este sistema fosse utilizado, foi necessário conhecer e mapear seus defeitos. Para a extração dos defeitos, foram realizados dois tipos de testes: os testes unitários (teste de caixa-branca) e os testes funcionais (teste de caixa-preta). Os testes unitários foram desenvolvidos utilizando o *JUnit* que, como já mencionado na seção 2.1, é um *framework* em Java. Já para os testes funcionais, foi utilizado a ferramenta *Selenium*, citado na seção 2.1, no qual é possível realizar os testes a partir de uma extensão instalada no navegador. Para a realização dos testes não é necessário abranger todo código

do sistema, é preciso verificar as partes do código onde são feitas as operações, transições e demais regras de negócio, onde os erros são comumente encontrados.

Para o sistema alvo deste trabalho, apenas dois módulos foram usados para a realização dos testes, pois é onde se encontra a regra de negócio e demais operações, totalizando 89 classes. Os demais módulos do sistema são destinados a interfaces, classes de testes e especificação das entidades. Após a realização dos testes pelo *JUnit* e pelo *Selenium*, foram encontrados e mapeados 20 defeitos, localizados em 13 classes.

Após a realização dos testes, foram extraídas as métricas de softwares, para que assim fosse feita as análises necessárias. Para isso, nesta monografia foi adotado dois métodos de extração, sendo eles: uma extensão da plataforma *Eclipse*, denominada *Eclipse Metrics*, que consegue extrair métricas como Linhas de Código e Complexidade Ciclométrica, e uma ferramenta chamada *CKJM Extended*, que tem como foco extrair as métricas de orientação à objeto propostas por [Chidamber e Kemerer \(1994\)](#).

Para definir a hipótese central, analisou-se a relação de algumas métricas com a ocorrência de defeitos de software para mostrar que as partes do código que se destacarem mais em relação às métricas definidas, ou seja, os locais onde os valores encontrados estavam acima da média, teriam mais chances de conter defeitos do que nas demais partes do código. Após as análises realizadas acerca dos trabalhos relacionados, notam-se que as métricas de Tamanho de Código e Complexidade Ciclométrica foram utilizadas pela maioria dos autores para apresentar essa relação. Tais métricas também foram utilizadas no trabalho que inspirou esta monografia, o trabalho de [Lucca, Vilela e Jino \(2007\)](#).

Contudo, a intenção deste experimento é mesclar as métricas tradicionais com as métricas de orientação à objeto propostas por [Chidamber e Kemerer \(1994\)](#). Assim, alguns trabalhos ([SUBRAMANYAM; KRISHNAN, 2003](#)) ([BASILI et al., 1996](#)) que utilizaram métricas de orientação à objeto foram analisados e as três métricas a seguir foram adotadas para análises de correlação entre defeitos e métricas de software, sendo elas: *WMC*, *CBO*, *DIT*. Neste experimento, também serão usadas as métricas de *LOC* e *V(G)*, já utilizadas em experimentos anteriores ([LUCCA; VILELA; JINO, 2007](#)) ([BASILI et al., 1996](#)). Além das três métricas de [Chidamber e Kemerer \(1994\)](#) que foram escolhidas, analisou-se também a métrica de RFC, para verificar qual o motivo dela não ter sido usada nos trabalhos estudados.

Finalmente, com todos os defeitos mapeados e todas as métricas extraídas, realiza-se as análises necessárias para a conclusão deste experimento. Foram feitas diversas análises de correlações entre as métricas extraídas a fim de encontrar alguma relação entre elas e os defeitos encontrados. Na próxima seção, os resultados encontrados nas análises feitas serão apresentados. Na sequência, será efetuada uma investigação de como estes resultados influenciaram ou não na ocorrência de defeitos.

3.2 Resultados

Após todo o processo de extrações de defeitos e métricas, foi concluído que o sistema alvo possui 20 defeitos encontrados em 13 das 89 classes analisadas. Tal fato refere-se a uma probabilidade de 0,224 de uma dessas classes conter um defeito. Esta probabilidade aumenta quando realizamos esta análise separadamente para cada métrica, considerando apenas as classes cujos os valores de cada uma das métricas estavam acima da média.

Tabela 1 – Tabela de probabilidade de defeito por métrica

	WMC	CBO	RFC	LCOM	LOC	V(G)
Classes acima da média	30	17	17	23	27	26
Classes acima da média com defeitos	7	7	6	6	6	7
Probabilidade	0,23	0,41	0,35	0,26	0,23	0,27

Na Tabela [1], pode-se verificar que, para cada métrica analisada, o valor da probabilidade aumenta em relação a probabilidade geral de uma classe conter defeito. Para a métrica WMC, 30 das 89 classes estavam acima da média, ou seja, 33,7%. Ao considerar apenas esse subconjunto de classes, a probabilidade de uma classe conter defeito sobe para 23%. Já para a métrica CBO, 17 classes estavam acima da média, um total de 19% de classes. Para esse subconjunto, a probabilidade de uma classe conter defeito sobe para 41%.

Analisando a métrica RFC, verifica-se 17 classes acima da média, também 19%. Desta forma, a probabilidade de conter tal defeito é de 35%. Para a métrica LCOM, tem-se 23 classes acima da média, ou 25%. Para esse subconjunto, há uma probabilidade de 26% de uma classe conter defeito. Já a métrica de tamanho de código (LOC), 27 classes estavam acima da média, ou 30%. Deste subconjunto, a probabilidade de uma classe conter defeito fica em 23%. Por fim, para a métrica de complexidade (V(G)), 26 classes estavam acima da média, ou 29%. Deste último subconjunto, a probabilidade de uma classe conter defeito é de 27%.

Os valores apresentados acima comprovam que classes com os valores das métricas acima da média tem a probabilidade maior de conter defeitos em relação à probabilidade geral de uma classe conter defeito. Para todas as métricas, houve aumento da probabilidade de uma classe conter defeito, o que é validado ao considerar subconjuntos formados apenas pelas classes cujos valores da métrica estavam acima de suas médias. Na Tabela [2], observa-se a probabilidade de uma classe conter defeito para cada métrica em relação a probabilidade geral que foi apresentada no começo desta sessão.

Tabela 2 – Tabela de Probabilidade de classes com defeitos

Classes analisadas	Probabilidade	%
Classes com defeitos	0,224	100%
Classes com LOC acima da média	0,23	103%
Classes com WMC acima da média	0,23	103%
Classes com LCOM acima da média	0,26	116%
Classes com V(G) acima da média	0,27	120%
Classes com RFC acima da média	0,35	156%
Classes com CBO acima da média	0,41	187%

A partir da Tabela [2], percebe-se que as classes que possuem os valores das métricas acima da média têm uma probabilidade entre 3% à 87% maior de possuir um defeito em relação a probabilidade geral de uma classe possuir defeito sem considerar métrica alguma.

Como já ressaltado nos capítulos anteriores, foi proposto utilizar neste estudo, além das métricas convencionais de tamanho e complexidade, métricas diferentes do trabalho proposto por [Lucca, Vilela e Jino \(2007\)](#). Com este número maior de métricas analisadas, obtém-se maiores informações sobre a relação entre métrica e defeito. As métricas utilizadas foram propostas por [Chidamber e Kemerer \(1994\)](#), e analisando a Tabela [2], percebe-se que estas métricas possuíram um impacto maior na probabilidade de uma classe conter defeito em relação à probabilidade das métricas convencionais.

Tabela 3 – Tabela de média e desvio padrão dos valores das métricas de classes com defeitos (A) e classes sem defeitos (B)

WMC			CBO		
	A	B		A	B
Média	18,4	13,8	Média	15,6	12
Desvio Padrão	8,21	5,78	Desvio Padrão	2,99	3,5
LCOM			LOC		
	A	B		A	B
Média	0,32	0,2	Média	133	93
Desvio Padrão	0,25	0,1	Desvio Padrão	49	35
RFC			V(G)		
	A	B		A	B
Média	49,9	36,1	Média	28	22
Desvio Padrão	21,5	14,6	Desvio Padrão	15,8	13,3

No intuito de extrair mais informações sobre a relação entre métrica e defeito, foram analisadas separadamente as classes que tiveram defeitos mapeados (na tabela denominamos como o grupo de classes A) e classes que não tiveram defeitos mapeados (denominadas como o grupo de classes B na tabela). Por exemplo, temos 13 classes com defeitos mapeados e 72 classes sem defeitos mapeados. Considerando a métrica *WMC*,

para as 13 classes com defeitos o valor médio da métrica foi de 18,4, e para as 72 classes sem defeitos o valor médio da métrica foi de 13,8.

Para cada métrica, tais classes foram divididas em dois grupos com o objetivo de calcular o valor médio e o desvio padrão em cada grupo. Salienta-se que tal processo foi efetuado para as seis métricas escolhidas e os dados coletados são apresentados na Tabela [3].

A partir da tabela de médias e desvios padrões, é possível analisar que para todas as seis métricas, as classes que tiveram defeitos relatados possuem uma média maior em relação as classes sem defeitos. A fim de comparar os valores coletados nos dois grupos de classes, foi calculado o coeficiente de variação de cada métrica. Dessa forma, pode-se analisar se os valores encontrados são homogêneos ou heterogêneos de acordo com o resultados obtidos. O cálculo do coeficiente de variação (CV) é realizado da seguinte maneira:

$$CV = (DesvioPadrão/Média) * 100\%$$

A primeira métrica analisada foi a *WMC*. Para as classes com defeitos relatados, tal métrica apresentou um CV de 44%, enquanto que, para as classes sem defeitos, o CV obtido foi de 41%. A partir destes resultados, observa-se que o grupo com as classes sem defeitos é mais homogêneo em relação ao grupo das classes com defeitos. Tal fato significa que os valores extraídos para a métrica em questão variaram mais nas classes com defeitos em comparação com as classes sem defeitos.

Com relação a segunda métrica analisada, *CBO*, o grupo com as classes com defeitos apresentou um CV de 19%. Já o grupo das classes sem defeitos apresentou um CV de 29%. Neste caso, observa-se que o grupo de classes com defeitos é consideravelmente mais homogêneo do que o grupo de classes sem defeitos. Além disso, pode-se verificar que a métrica *CBO* possui coeficientes de variação significativamente menores do que os conferidos para a métrica *WMC*.

A terceira métrica analisada foi a *LCOM*, que apresentou um CV de 78% para classes com defeitos e um CV de 50% para classes sem defeitos. Para ambos os grupos, o coeficiente de variação encontrado foi bastante elevado, o que permite concluir que os dois grupos são heterogêneos. Isto é, os valores extraídos para a referida métrica não mostraram ser homogêneos para ambos.

Na sequência, a métrica *LOC* apresentou um CV de 36% para o grupo de classes com defeitos e um CV de 37% para o grupo de classes sem defeitos. Observamos que, as taxa obtidas de CV dos dois grupos são semelhantes e altas, o que permite observar que os valores extraídos para cada grupo variaram numa proporção muito próxima. Ademais, como a homogeneidade está diretamente relacionada ao valor do desvio padrão quando comparado com a média, verifica-se que os valores encontrados para os dois grupos são

altos, ou seja, são heterogêneos, pois o CV de ambos foi alto, assim como o CV encontrado anteriormente para a métrica *LCOM*.

Para a métrica *RFC*, calculamos um CV de 43% para o grupo de classes com defeitos e 40% para o grupo de classes sem defeitos. Os coeficientes extraídos para esta métrica se assemelha aos coeficientes extraídos para a métrica *WMC*, pois o grupo com as classes sem defeitos é mais homogêneo em relação ao grupo das classes com defeitos em ambas as métricas.

Finalmente, para a métrica $V(G)$, calculamos um CV de 56% para o grupo de classes com defeitos e 57% para o grupo de classes sem defeitos. Apesar dos coeficientes terem valores parecidos, o que confirma o fato da variação dentro dos dois grupos se assemelhar, os valores encontrados são relativamente altos. Tal fato permite concluir que os dois grupos são heterogêneos, ou seja, os valores extraídos para a métrica $V(G)$ não mostraram ser homogêneos para nenhum dos grupos.

A partir das análises efetuadas acerca de cada métrica, ressalta-se que as métricas *LCOM*, *LOC* e $V(G)$ possuem valores para o CV significativamente altos. Isto é, não é possível inferir se estas métricas influenciam ou não na ocorrência de defeitos, pois, por possuírem coeficientes de variação altos, os grupos de classes são heterogêneos. Tal fato permite constatar que os valores para estas métricas não estão diretamente relacionadas com a ocorrência de defeito.

As demais métricas analisadas apresentaram valores para o CV mais baixos e, consequentemente, mostraram ser mais homogêneas que as classes citadas acima. Isto é, para os valores obtidos pode-se inferir que estas métricas influenciam de alguma maneira na ocorrência de defeitos. Dentre estas métricas, os resultados encontrados nas Tabelas [2] e [3] indicam que *CBO* e *RFC* variaram menos e ambas são as que aumentam mais a probabilidade de ocorrência de defeito.

Para concluir as análises, foi verificada a possibilidade de encontrar a existência de alguma relação entre as métricas escolhidas. Por exemplo, se classes maiores tendem a ser mais complexas, ou se classes com o *RFC* alto tendem a ter o *CBO* alto também. Para tanto, optou-se por usar o coeficiente de correlação de *Pearson*, de modo a compreender se as métricas estão relacionadas entre si e qual seria a força dessa relação entre elas. As análises foram realizadas sobre os valores encontrados para os coeficientes de correlação, apresentados na Tabela 4.

Com o objetivo de calcular os coeficientes de correlação, foi utilizado o conceito proposto por *Pearson*. De acordo com [Figueiredo e Alexandre \(\)](#), o coeficiente de correlação de *Pearson* é uma medida de associação linear entre variáveis. O conceito principal desta correlação é entender que duas variáveis se associam quando elas guardam semelhanças na distribuição dos seus valores, ou seja, o aumento ou decremento de uma unidade

em uma das variáveis gera o mesmo impacto nas demais variáveis analisadas.

O coeficiente de correlação de *Pearson* varia entre -1 e 1. Ressalta-se que o sinal indica a direção positiva, caso os valores analisados são diretamente relacionados, ou negativa, caso os valores são inversamente relacionados, e o valor indica a força da relação entre as variáveis. Isto é, quanto mais perto de 1 (independente do sinal) maior é o grau de dependência estatística linear entre as variáveis. Por outro lado, quanto mais próximo de zero, menor é a força dessa relação. Desta forma, é possível realizar a seguinte interpretação do coeficiente:

- 0.9 a 1.0 positivo ou negativo indica uma correlação muito forte;
- 0.7 a 0.9 positivo ou negativo indica uma correlação forte;
- 0.5 a 0.7 positivo ou negativo indica uma correlação moderada;
- 0.3 a 0.5 positivo ou negativo indica uma correlação fraca;
- 0 a 0.3 positivo ou negativo indica uma correlação desprezível.

Assim, após feita as análises de relação entre métrica e defeito, foi realizada uma Tabela [4] de correlação entre as métricas. Com estes valores tabulados, foi possível analisar qual conjunto de métricas estão diretamente relacionados entre si, além de verificar se essa relação pode influenciar na ocorrência de defeitos.

Tabela 4 – Tabela de Correlações entre as métricas

	WMC	CBO	RFC	LCOM	LOC	V(G)
WMC	1	0,72	0,86	0,62	0,96	0,94
CBO	0,72	1	0,81	0,65	0,73	0,59
RFC	0,86	0,81	1	0,64	0,94	0,81
LCOM	0,62	0,65	0,64	1	0,57	0,45
LOC	0,96	0,73	0,94	0,57	1	0,95
V(G)	0,94	0,5	0,81	0,45	0,95	1

A partir dos valores obtidos da Tabela [4], e da interpretação dada sobre o coeficiente de *Pearson*, observa-se que a maioria das relações entre as métricas analisadas são fortes, ou seja, acima de 0,7. Pode-se observar também que existem relações com valores acima de 0,9, ou seja, muito forte. Esses valores mostram que, por exemplo, quanto maior for uma classe, ou maior o valor da métrica *LOC*, mais complexa ela se torna, o que implica no valor das métricas *V(G)* e *WMC*, e também implica no aumento da métrica *RFC*, ou seja, aumenta o número de métodos que podem ser executados a partir de uma mensagem enviada para a classe. Com isso, pode-se entender que, a métrica *LOC* pode não estar diretamente relacionada com a ocorrência de defeitos, como analisado na Tabela

[2], mas ela está diretamente relacionada com as demais métricas, pois os valores dos coeficientes de *Pearson* encontrados foram muito altos. A mesma análise pode ser feita para a métrica $V(G)$, ela pode não implicar na ocorrência de defeito diretamente, considerando o sistema testado, mas ela está diretamente relacionada as demais métricas que tiveram um maior desempenho em relação a ocorrência de defeitos.

3.3 Comparação dos Resultados

O presente trabalho, assim como o trabalho proposto por [Lucca, Vilela e Jino \(2007\)](#), realizou uma investigação acerca da relação entre métricas e o número de defeitos. Os dados coletados pelos autores do trabalho que inspirou esta monografia apresentaram que há maior probabilidade de ocorrência de defeitos em módulos maiores ou mais complexos. O conceito de módulo utilizado por eles é semelhante ao conceito de classe utilizado neste trabalho. Os referidos autores descobriram que a probabilidade de ocorrência de defeito em um módulo com o tamanho (LOC) acima da média chega a ser 67,5% maior do que em módulos com o valor da métrica abaixo da média. Além disso, quando analisaram pela complexidade ($V(G)$), encontraram uma probabilidade 78,8% maior em módulos com o valor da métrica acima da média.

Além de métricas de tamanho e complexidade, neste experimento foram utilizadas outras métricas para realizar as análises. Como já citado ao decorrer do trabalho, foi utilizado o conjunto das métricas orientadas a objetos, propostas por [Chidamber e Kemerer \(1994\)](#). Ao comparar os resultados obtidos com os coletados no trabalho do [Lucca](#), nota-se que as métricas utilizadas por eles tiveram um impacto maior na probabilidade de uma classe possuir defeito em relação ao impacto que essas mesmas métricas obtiveram em cima do sistema alvo desta monografia. Por outro lado, as demais métricas utilizadas demonstraram ter um impacto maior que as métricas tradicionais tiveram no trabalho anteriormente citado.

Observa-se que para a métrica de tamanho, a probabilidade de um módulo, ou classe, acima da média ter um defeito é apenas 3% maior em relação à probabilidade geral calculada, enquanto que para a métrica de complexidade essa probabilidade chegou a 20%. Tal fato permite verificar que, em relação ao ([LUCCA; VILELA; JINO, 2007](#)), as porcentagens que foram encontradas são substancialmente mais baixas. Porém, ao analisar as métricas CK, constatou-se probabilidades de 56% para a métrica RFC e 87% para a métrica CBO . Neste último caso, pode-se notar que a probabilidade encontrada para uma métrica CK é 10% maior que a probabilidade obtida por [Lucca](#) para a métrica LOC .

Vale a pena salientar que, nas análises realizadas utilizando o coeficiente de *Pearson*, encontrou-se que as métricas LOC e $V(G)$, apesar de não obterem um resultado bom em relação as demais classes quando foram relacionadas aos defeitos encontrados (como

observado na Tabela [2]), mostraram ter uma relação muito forte com as demais métricas, como exibido na Tabela [4].

Finalmente, observa-se que, além de ter uma probabilidade alta, a métrica *CBO* destacou-se em todas as análises que foram feitas. Isto é, o objetivo deste trabalho referente a avaliação de métricas distintas do trabalho de Lucca, Vilela e Jino (2007) de forma a identificar alguma que poderia ter um resultado melhor do que os autores do referido trabalho conseguiram obter, foi atingido.

Uma das razões para a diferença encontrada entre os valores das métricas *LOC* e *V(G)* em relação ao trabalho de Lucca, Vilela e Jino (2007) pode estar relacionada a linguagem de programação em que os dois sistemas alvos estão implementados, *C* e *Java*. Ou seja, a escolha das métricas pode ser determinante de acordo com a linguagem em que o sistema testado for desenvolvido, pois nem sempre as métricas básicas, *LOC* e *V(G)*, terão bons resultados. E o mesmo pode ser dito para as métricas CK, pois nenhum outro trabalho foi encontrado onde o sistema alvo testado estivesse escrito em outra linguagem, a não ser *Java*.

4 Conclusão

Finalmente, neste capítulo será apresentado uma síntese desta monografia, desde uma avaliação do trabalho proposto até os resultados alcançados. Foi feito um breve resumo sobre os passos que foram seguidos ao decorrer do experimento. Também será destacada a importância deste estudo e serão demonstradas sugestões para trabalhos futuros.

Primeiramente, foi decidido utilizar um software de uma empresa privada, denominado Portal Corporativo [8], para a extração das métricas. Após a escolha do sistema base, duas ferramentas foram utilizadas para a extração das métricas, sendo elas: Eclipse Metrics (METRICS, 2016) e CKJM (SPINELLIS, 2005). Com os dados coletados, uma série de análises foi realizada. A principal proposta do trabalho era de mostrar que classes que possuem os valores das métricas acima da média, tem a probabilidade maior de possuir defeito em relação as demais classes.

Para os dois conjuntos de métricas escolhidos, foi encontrado que, para as métricas básicas analisadas (LOC e $V(G)$) as probabilidades encontradas não foram tão significativas como no trabalho de Lucca, Vilela e Jino (2007). Para a métrica LOC , foi analisado que, a probabilidade de uma classe com o valor da métrica acima da média conter defeito, é apenas 3% maior em relação as demais classes. E para a métrica $V(G)$ a probabilidade chegou a ser 20% maior para classes com o valor da métrica acima da média. Enquanto que no trabalho de Lucca as probabilidades encontradas foram de 67,5% e 78,8%, respectivamente.

Contudo, o intuito deste trabalho também era de encontrar novas métricas que poderiam ser mais eficientes do que as métricas básicas. Por isso, as métricas CK foram analisadas. As probabilidades de uma classe com o valor de uma métrica CK acima da média conter defeito variou entre 3% e 87%. A métrica com a probabilidade mais alta encontrada foi a CBO , cujo valor é maior que as duas probabilidades encontradas no trabalho do Lucca.

Além de analisar as probabilidades, também foram feitas averiguações acerca da relação entre a presença de defeito e o valor encontrado para cada métrica. Para essa análise, a métrica CBO foi a que mais se destacou, ou seja, foi a métrica cujo os grupos de classes foram mais homogêneos. Nesta mesma análise verificou-se que três, das seis métricas analisadas, possuíam coeficientes de variação muito altos, ou seja, os grupos de classes analisados eram heterogêneos. Tais métricas foram: $LCOM$, $V(G)$ e LOC .

Apesar de demonstrarem não possuir nenhuma relação direta com a ocorrência de defeitos, as métricas básicas, LOC e $V(G)$, demonstraram ser as métricas com as relações

mais fortes com as demais métricas analisadas, como exibido na Tabela [4]. Com isso, pode-se concluir que, mesmo não influenciando diretamente a ocorrência de defeitos no sistema analisado, estas métricas merecem ser levadas em consideração em estudos sobre a relação entre métricas de software e defeitos.

O objetivo deste trabalho era identificar quais métricas influenciam ou não na presença de defeitos, e como isso facilitaria não só a realização dos testes de software, mas também os desenvolvedores, que poderiam utilizar as métricas para descobrir em tempo de codificação a presença de algum possível defeito que só seria relatado mais tarde nos testes, economizando tempo para os testes. Pois, ao verificar que uma dada métrica tem uma relação direta com a ocorrência de defeitos, seria realista iniciar a atividade de teste pelas classes que apresentarem o valor da métrica acima da média, por exemplo. Assim, os defeitos poderiam ser encontrados mais rapidamente. Porém, como foi observado, a escolha do conjunto de métricas a ser analisado depende da linguagem em que o sistema está sendo desenvolvido, pois observou-se comportamentos diferentes para a mesma métrica analisada em sistemas implementados em linguagens diferentes, *Java*, no sistema alvo desta monografia, e *C*, no sistema alvo do trabalho desenvolvido por [Lucca, Vilela e Jino \(2007\)](#).

Ao final deste trabalho observou-se que a relação entre as métricas pode ser analisada mais profundamente, e também como essa relação pode influenciar na presença de defeitos. Pois, como foi observado, apesar das métricas *LOC* e *V(G)* possuírem um desempenho baixo para predição de defeito no sistema alvo, elas mostraram ter uma forte relação com as demais métricas, ou seja, de alguma forma elas influenciaram no valor das demais métricas. Este tópico poderia ser mais aprofundado em um trabalho futuro, para investigar se o forte relacionamento entre *LOC* e *V(G)* se repete em outros sistemas e como usar tal informação para priorização de testes.

Referências

- BASIL, V. R. et al. A Validation of Object-Oriented Design Metrics as Quality Indicators. v. 22, n. 10, 1996. Citado 4 vezes nas páginas 8, 9, 18 e 24.
- CHIDAMBER, S. R.; KEMERER, C. F. A Metrics Suite for Object Oriented Design. v. 20, n. 6, p. 476–493, 1994. Citado 10 vezes nas páginas 3, 9, 11, 13, 14, 18, 19, 24, 26 e 30.
- CHOWDHURY, I.; ZULKERNINE, M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, Elsevier, v. 57, n. 3, p. 294–313, 2011. Citado na página 9.
- EUGÊNIO, M.; BEZERRA, R. Detecção de Módulos de Software Propensos a Falhas Através de Técnicas de Aprendizagem de Máquina Detecção de Módulos de Software Propensos a Falhas Através de Técnicas de Aprendizagem de Máquina. 2008. Citado na página 10.
- FIGUEIREDO, D.; ALEXANDRE, J. Desvendando os mistérios do coeficiente de correlação de pearson (r). Accessed: 2017-10-01. Citado na página 28.
- GRAHAM, D. et al. *Foundations of Software Testing: ISTQB Certification*. [S.l.]: Intl Thomson Business Pr, 2008. ISBN 9781844803552, 9781844809899. Citado na página 10.
- JULIANO, R. C. Visualização de software baseada em uma metáfora do universo utilizando o conjunto de métricas ck. 2014. Citado na página 11.
- JUNIT. 2017. Accessed: 2017-06-25. Disponível em: <<http://junit.org/>>. Citado na página 10.
- LUCCA, W. L. de; VILELA, P. R. d. S.; JINO, M. Avaliando a Relação entre Tamanho-Complexidade e Número de Defeitos de Software em Nível de Módulo. *Anais da VI Jornadas Iberoamericanas de Ingenieria del Software e Ingenieria del Conocimiento*, n. August 2016, p. 333–339, 2007. Citado 11 vezes nas páginas 8, 9, 19, 20, 21, 24, 26, 30, 31, 32 e 33.
- MCCABE. A complexity measure. *IEEE Transactions on Software Engineering*, v. 2, p. 308–320, 1976. Citado na página 11.
- METRICS. 2016. Accessed: 2017-06-25. Disponível em: <<https://marketplace.eclipse.org/content/eclipse-metrics#group-details>>. Citado 3 vezes nas páginas 12, 15 e 32.
- PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. New York, NY, USA: McGraw-Hill, Inc., 2010. ISBN 0073375977, 9780073375977. Citado 3 vezes nas páginas 8, 10 e 11.
- RADJENOVIC, D. Software fault prediction metrics : A systematic literature review. 2013. Citado na página 8.

SELENIUM. 2017. Accessed: 2017-06-25. Disponível em: <<http://www.seleniumhq.org/>>. Citado na página 11.

SITEMCCABE. 2017. Accessed: 2017-11-30. Disponível em: <http://www2.ic.uff.br/~bianca/engsoft2/index_arquivos/Aula9-EngSoft2.pdf>. Citado 2 vezes nas páginas 4 e 12.

SOMMERVILLE, I. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN 0321313798. Citado 2 vezes nas páginas 8 e 11.

SONARQUBE. 2017. Accessed: 2017-06-25. Disponível em: <<https://www.sonarqube.org/>>. Citado na página 16.

SPINELLIS, D. Tool writing: A forgotten art? *IEEE Software*, v. 22, n. 4, p. 9–11, July/August 2005. ISSN 0740-7459. Disponível em: <<http://www.dmst.aueb.gr/dds/pubs/jrnl/2005-IEEESW-TotT/html/v22n4.html>>. Citado 2 vezes nas páginas 17 e 32.

SUBRAMANYAM, R.; KRISHNAN, M. S. Empirical Analysis of CK Metrics for Object-Oriented Design Complexity : Implications for Software Defects. v. 29, n. 4, p. 297–310, 2003. Citado 4 vezes nas páginas 11, 18, 19 e 24.