DETECÇÃO DE ANOMALIAS DE CÓDIGO USANDO MÉTRICAS DE SOFTWARE

JULIANA PADILHA

DETECÇÃO DE ANOMALIAS DE CÓDIGO USANDO MÉTRICAS DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Eduardo Magno Lages Figueiredo

Belo Horizonte

Março de 2013

Padilha, Juliana.

P123d Detecção de anomalias de código usando métricas de software / Juliana Padilha — Belo Horizonte, 2013.
xv, 159 f.: il.; 29 cm.

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento de Ciência da Computação.

Orientador: Eduardo Magno Lages Figueiredo.

1. Computação - Teses. 2. Engenharia de software – Teses. I.Orientador. II. Título.

CDU 519.6*32 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Detecção de anomalias de código usando métricas de software

JULIANA PADILHA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. EDUARDO MAGNO LAGES FIGUEIREDO - Orientador
Departamento de Ciência da Computação - UFMG

PROF. CLÁUDIO NOGUEIRA SANT'ANNA
Departamento de Ciência da Computação - UFBA

PROF. MARCO TÚLIO DE OLIVEIRA VALENTE
Departamento de Ciência da Computação - UFMG

PROF. PAULO CÉSAR MASIERO
Departamento de Sistemas de Computação - ICMC

Belo Horizonte, 07 de março de 2013.

Dedico esta dissertação a todos que sempre me apoiaram, em especial ao meu namorado, André, meus pais, Milton e Madalena, meu irmão, Cezar.

Agradecimentos

Quem vivenciou a trajetória do curso de mestrado sabe perfeitamente que é uma tarefa árdua. Mas, por outro lado, é uma experiência enriquecedora e de plena superação. A conclusão desta dissertação marca o fim de uma importante etapa da minha vida. Para aqueles que compartilharam comigo desse momento gostaria de agradecê-los.

Preliminarmente, quero agradecer a Deus pelo dom da vida.

Aos meus pais, Milton e Madalena, meu infinito agradecimento por terem acreditado em minha capacidade e principalmente pelo apoio que recebi quando resolvi "jogar" tudo alto para concretizar este sonho.

Ao meu irmão, Milton Cezar, pelas palavras de incentivo e pela presença constante em minha vida.

Ao meu namorado, André, pelo amor, carinho, compreensão, pela ajuda nas matérias do mestrado. Por sempre estar ao meu lado, me fazendo acreditar que eu posso mais. E também, ao Ruimar e a Iracema pelo apoio e orações.

A toda a minha família, amigos, ex-colegas de trabalho, ex-alunos que torceram por mim.

Ao meu orientador, Dr. Eduardo Figueiredo, pela disponibilidade, colaboração, conhecimentos transmitidos e capacidade de estímulo ao longo deste trabalho de mestrado.

Aos professores, Dr. Marco Túlio Valente, Dr. Alessandro Garcia e Dr. Cláudio Sant'Anna pela colaboração em minha pesquisa.

Aos meus amigos de mestrado, principalmente aos amigos do laboratório LLP, pelos momentos divididos juntos. E também, aos alunos de IC pela parceria.

E, por fim, a todos aqueles que por um lapso não mencionei, mas que colaboraram diretamente ou indiretamente para esta dissertação.

Muito obrigada!

"A qualidade é relativa. O que é qualidade para uma pessoa pode ser falta de qualidade para outra."
(G. Weinberge)

Resumo

Métricas são tradicionalmente utilizadas para avaliar a qualidade e manutenibilidade do software, apoiando a identificação de anomalias no código. Recentemente, métricas de interesse foram propostas com este objetivo. Um interesse é algo que se queira tratar como uma unidade conceitual modular, como requisitos funcionais e não funcionais e idiomas de programação. Enquanto as métricas tradicionais quantificam as propriedades de módulos, as métricas de interesse quantificam propriedades de interesse, tais como espalhamento e entrelaçamento. Apesar das métricas de interesse já terem sido utilizadas em estudos experimentais, ainda falta conhecimento empírico quanto à sua eficácia na identificação de anomalias de código. Esta dissertação investiga se métricas de interesse podem fornecer indicadores úteis para a detecção de cinco anomalias de código: *Divergent Change*, *Shotgun Surgery*, *God Class*, *Feature Envy* e *God Method*. Para isso, foram realizados dois estudos experimentais. No primeiro estudo, utilizamos um conjunto de 54 participantes de duas instituições diferentes para detecção de três anomalias em classes em dois sistemas. No segundo estudo, baseado na detecção de anomalias em métodos, utilizamos um conjunto de 47 participantes de duas instituições com o objetivo de detectar casos de duas anomalias em um sistema. Em ambos os estudos, os participantes analisaram métricas tradicionais e de interesse para auxiliar nesta investigação. Os resultados indicaram que as métricas de interesse apoiam os desenvolvedores a detectar alguma destas anomalias de código. Por exemplo, os nossos resultados revelaram que a métrica Número de Interesse por Componente é uma boa indicadora para o *Divergent Change*. No entanto, uma análise conjunta das métricas tradicionais e de interesse muitas vezes é necessária para detectar *God Class* e o *Shotgun Surgery*. Em relação os resultados das anomalias em métodos, eles indicaram que a métrica NCO é uma boa indicadora para o *God Method*. Com base nos resultados destes dois estudos, realizamos a elaboração de método quantitativo apoiado por uma ferramenta para detectar automaticamente anomalias de código.

Palavras-chave: Métricas de Software, Separação de Interesse, Anomalias de Código.

Abstract

Metrics have traditionally have been used to evaluate the maintainability of software programs by supporting identification of symptoms of bad smells. Recently, concern metrics have also been proposed with this purpose. While traditional metrics quantify properties of sotware modules, concern metrics quantify concern properties, such as the scattering and tangling of concerns realized in a program. Despite being increasingly used in experimental studies, although lack empirical knowledge as to their effectiveness in identifying bad smells. This work investigates whether concern metrics may provide useful indicators for detecting five bad smells: Divergent Change, Shotgun Surgery, God Class, Feature Envy and God Method. For this, two experimental studies were performed. In the first study, we used a set of 54 participants from two different institutions for detecting three bad smells in classes in two systems. In the second study, based on detection of bad smells in methods, we used a set of 47 participants from two institutions in order to detect two bad smells in of one system. In both studies, participants analyzed traditional and concern metrics to assist bad smell detection. The results indicated that the concern metrics support developers detect these bad smells. In addition, our results showed that the Number of Concern per Component metric is a good indicator for the Divergent Change. However, elaborated joint analysis of traditional metrics and concern is often necessary to detect God Class and Shotgun Surgery. Regarding the results of bad smells in methods, they indicated that the Number of Concern per Operations metric is a good indicator for the God Method. Based on the results of these two studies, we propose a quantitative method for supporting a automated detection bad smells.

Keywords: Software Metrics, Separations of Concerns, Bad Smell.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

CBO Acoplamento entre Objetos *(Coupling between Objects)*

CDC Espalhamento de Interesses em Classes *(Concern Diffusion over Components)*

CDLOC Espalhamento de Interesses em Linhas de Código *(Concern Diffusions over LOC)*

CDO Espalhamento de Interesses em Operações *(Concern Diffusion over Operations)*

CYCLO Complexidade Ciclomática de McCabe *(McCabe's Cyclomatic Number)*

LCOM Falta de Coesão em Métodos *(Lack of Cohesion in Methods)*

LOC Número de Linhas de Código *(Lines of Code)*

LOCC Linhas de Código do Interesse *(Lines of Concern Code)*

NCC Número de Interesse por Componente *(Number Concerns per Component)*

NCO Número de Interesse por Operações *(Number Concerns per Operations)*

NOA Número de Atributos *(Number of Atributtes)*

NOCA Número de Atributos dos Interesses *(Number Concerns per Atributtes)*

NOCO Número de Operações dos Interesses *(Number Concerns per Operations)*

NOO Número de Operações *(Number of Operations*

PAR Número de Parâmetros *(Number of Parameters)*

WMC Método Ponderado por Classe *(Weighted Method per Class*)

Sumário

Capítulo 1

Introdução

Esforços de manutenção se concentram principalmente na correção de falhas [Eaddy et al., 2008] e na adição de novas funcionalidades ao software [Conejero et al., 2012]. Como consequência, ocorre o declínio de qualidade de software ao invés do controle e melhoria da qualidade do sistema. Métricas de software são mecanismos para avaliar a qualidade [Chidamber & Kemerer, 1994] e identificar anomalias relacionadas ao projeto de software [Marinescu, 2004]. Métricas têm tradicionalmente utilizado conceitos como: tamanho da classe, acoplamento, coesão e complexidade da interface para avaliar o software [Chidamber & Kemerer, 1994; Fenton & Pfleeger, 1996]. Tais métricas tradicionais são vistas como uma solução pragmática para encontrar sintomas de anomalias de código[1] [Fowler, 1999]. Por exemplo, no trabalho de Marinescu [Marinescu, 2004], ele baseou-se em métricas tradicionais para compor estratégia de detecção com o objetivo de encontrar anomalias de código.

1.1 Problema

No geral, a aplicação de métricas de software para a avaliação e melhoria de qualidade de software orientado a objetos são animadoras [Marinescu, 2002]. Entretanto, algumas anomalias são, frequentemente, resultado da baixa separação de interesses e as métricas tradicionais não conseguem facilmente quantificar as propriedades de interesses não modulares. Robillard e Murphy [2007] definem interesse como algo que se queira tratar como uma unidade conceitual modular, incluindo requisitos não funcionais e idiomas de programação. Em muitos casos, o código de um interesse não é encapsulado em um único módulo da linguagem de programação e, em vez disso, ele se espalha e se entrelaça com outros interesses do sistema [Kiczales et al., 1997].

[1] Neste documento estamos usando anomalias de código como tradução de *bad smells.*

Para identificar problemas na modularidade de interesses, métricas de interesse têm sido propostas para quantificar as propriedades dos interesses [Ducasse et al., 2006; Eaddy et al., 2008; Sant'Anna et al., 2008]. Enquanto as métricas tradicionais quantificam as propriedades de módulos, as métricas de interesse quantificam propriedades de interesse, tais como espalhamento e entrelaçamento. Estas métricas foram aplicadas em estudos experimentais com diferentes propósitos, tais como na comparação de programação orientada a aspectos [Kiczales et al., 1997] com programação orientada a objetos [Conejero et al., 2012; Figueiredo et al., 2008; Greenwood et al., 2007] e na identificação de interesses transversais que deveriam ser refatorados [Eaddy et al., 2008; Figueiredo et al., 2012]. Porém, o conhecimento sobre a utilidade das métricas de interesse ainda é limitado para detecção de anomalias de código. O problema que esta dissertação ataca é a identificação de anomalias de código usando não somente métricas tradicionais, mas principalmente métricas de interesse.

1.2 Objetivos

Esta dissertação tem os seguintes objetivos:

- analisar se métricas de interesse são eficazes para detectar anomalias de código;
- averiguar quais métricas tradicionais e de interesse são úteis na identificação de anomalias de código;
- propor um método quantitativo baseando-se na combinação heurística de métricas de software obtidas como úteis para detectar anomalias de código;
- desenvolver uma ferramenta para apoiar o método quantitativo na detecção automática de anomalias.

Para atender os objetivos elencados acima, este trabalho realizou uma série de experimentos com participantes. No primeiro estudo, foi utilizado um conjunto de 54 participantes para detecção de três anomalias em classes. Por outro lado, no segundo estudo, foi utilizado um conjunto de 47 participantes que analisaram métricas tradicionais e de interesse com o objetivo de detectar duas anomalias em métodos.

1.3 Proposta de Solução

O objetivo principal deste trabalho é a elaboração de um método quantitativo, composto por regras heurísticas, para apoiar a identificação automática de anomalias de

código. O método heurístico utilizará métricas de interesse que, quando analisadas em conjunto com as métricas tradicionais, se mostrarem úteis para detecção de anomalias nos estudos realizados. Estes estudos experimentais se dividem em duas partes: (i) detecção de anomalias em classes e (ii) detecção de anomalias em métodos. Ambos os estudos têm como objetivo avaliar a eficácia de métricas de interesse para detectar anomalias de código.

- A primeira parte, anomalias em classes, envolveu 54 participantes os quais foram divididos em três grupos. Cada grupo de participantes analisou um conjunto de métricas diferentes: (i) somente métricas tradicionais, (ii) somente métricas de interesse e (iii) tanto métricas tradicionais como métricas de interesse. Todas as métricas analisadas foram aplicadas ao código fonte de dois sistemas (MobileMedia e Health Watcher) com o objetivo de identificar três anomalias de código [Fowler, 1999; Riel, 1996]: *Divergent Change*, *Shotgun Surgery* e *God Class*.

- A segunda parte, anomalias em métodos, envolveu 47 participantes os quais também foram divididos em três grupos. Cada grupo de participantes analisou um conjunto de métricas diferentes: (i) somente métricas tradicionais, (ii) somente métricas de interesse e (iii) tanto métricas tradicionais como métricas de interesse. Todas as métricas analisadas foram aplicadas ao código fonte de um sistema (MobileMedia) com o objetivo de identificar duas anomalias de código [Fowler, 1999]: *Feature Envy* e *God Method*.

A partir do primeiro estudo, nossos resultados sugerem que métricas de interesse geralmente apoiam a detecção de anomalias. Além disso, a métrica Número de Interesse por Componentes (NCC) se mostrou particularmente eficiente para a detecção de algumas anomalias, como o *Divergent Change*. Outras lições que aprendemos com este estudo são: (i) o uso de muitas métricas não necessariamente requer mais tempo de análise, (ii) métricas tradicionais, geralmente, não detectam com acurácia algumas das anomalias estudadas, (iii) detecção de *God Class* requer uma análise utilizando métricas tradicionais e métricas de interesse e (iv) *Shotgun Surgery* não é tão fácil de ser detectado com métricas de software.

Os resultados do segundo estudo sugerem que a métrica de interesse Número de Interesse por Operações (NCO), geralmente, apoia a detecção da anomalia *God Method* quando combinada a outras métricas de interesse. Outra lição aprendida neste estudo foi que o conjunto de métricas selecionado não foi adequado para capturar as propriedades de *Feature Envy*.

A partir destes dois estudos elaborou-se um método quantitativo composto de regras heurísticas. Para apoiar este método, foi desenvolvida uma ferramenta na linguagem Java que é um *plugin* para a plataforma Eclipse. A versão atual desta ferramenta apoia a detecção automática das seguintes anomalias de código: *Divergent Change*, *Shotgun Surgery*, *God Class* e *God Method*. Ela lista as classes e/ou métodos que são propícias a terem anomalias de código.

1.4 Contribuições

Os resultados parciais desta dissertação foram publicados em um artigo de Workshop:

- Padilha, J. and Figueiredo, E. (2012). Detectando Code Smells com Métricas de Interesse. Workshop de Teses e Dissertações em Engenharia de Software (WTDSoft), 1:15.

Além disso, outros dois artigos - um de periódico e outro de conferência internacional - foram submetidos e aguardam avaliação:

- Padilha, J.; Guimarães, E.; Figueiredo, E.; Garcia, A. and Sant'Anna, C. *Evaluating the Effectiveness of Concern Metrics to Detect Code Smells*.
- Padilha, J.; Figueiredo, E.; Sant'Anna, C. and Garcia, A. *Detecting Smelly Methods with Concern Metrics*.

As principais contribuições deste trabalho são:

1. Identificação de quais métricas de software, tradicionais e de interesse, são úteis para detecção de anomalias de código.
2. Elaboração de um método quantitativo que utiliza métricas tradicionais e métricas de interesse consideradas úteis nos experimentos realizados com participantes para detecção de anomalias de código.
3. Desenvolvimento de uma ferramenta para apoiar o método quantitativo na detecção automática de anomalias.

1.5 Estrutura da Dissertação

Esta dissertação está estruturada da seguinte forma. O Capítulo 2 apresenta e analisa os principais conceitos relacionados a qualidade de software, anomalias de código, métricas e regras heurísticas para detecção de anomalias. O Capítulo 3 relata e analisa os dados dos experimentos realizados para detecção de anomalias em classes. Por outro lado, o Capítulo 4 relata e analisa os dados dos experimentos realizados para detecção de anomalias em métodos. A partir dos estudos realizados, o Capítulo 5 propõe um método quantitativo, composto por métricas tradicionais e por métricas de interesse para apoiar a detecção das anomalias estudadas. Este capítulo também descreve Concern-MeBS, uma ferramenta desenvolvida com o objetivo de apoiar o método quantitativo proposto neste trabalho. Capítulo 6 apresenta as ameaças à validade e os trabalhos relacionados. Por fim, considerações finais são feitas no Capítulo 7 que indica ainda as propostas para trabalhos futuros.

Capítulo 2

Revisão da Literatura

Conforme a engenharia de software amadurece, a medição de software passa a desempenhar um papel cada vez mais importante no entendimento e controle das práticas e produtos do desenvolvimento de software [Kitchenham et al., 1995]. Consequentemente, é essencial que as medidas utilizadas sejam avaliadas. As pessoas responsáveis pela manutenção do software podem utilizar a medição para avalizar se o produto precisa ser melhorado. Portanto, a medição deve representar exatamente os atributos de qualidade que se pretende quantificar [Kitchenham et al., 1995]. Assim, a avaliação deve ser crítica para ter sucesso da medição do software. Este capítulo apresenta uma revisão da literatura sobre medição e qualidade de software. As métricas tradicionais e de interesse utilizadas neste trabalho também são discutidas.

Este capítulo está estruturado da seguinte forma. A Seção 2.1 apresenta os principais conceitos relacionados com a qualidade de software e os problemas que a falta de qualidade podem ocasionar em sistema de software. Nesta seção, apresentamos a definição de cinco anomalias de código usadas futuramente neste documento. Em seguida, apresentamos na Seção 2.2 modelos de qualidade de software. As Seções 2.3 e 2.4 apresentam as métricas de software tradicionais e as métricas de interesse, respectivamente. Em seguida, na Seção 2.5, apresentamos regras heurísticas que consistem na combinações de métricas de software. Por último, as considerações finais deste capítulo são feitas na Seção 2.6.

2.1 Qualidade de Software

A qualidade de software tem se aprimorado significativamente nos últimos anos. A razão para isso é o fato das empresas terem adotado novas técnicas e tecnologias, como o uso de desenvolvimento orientado a objetos e de ferramentas CASE [Sommerville,

2011]. Além disso, tem-se criado uma conscientização da importância do gerenciamento de qualidade de software e da utilização de técnicas de gerenciamento de qualidade provenientes da necessidade de realizar manutenção no software [Sommerville, 2011].

O controle da qualidade do software envolve a monitoração do processo de desenvolvimento de software para assegurar que os procedimentos e os padrões de garantia de qualidade estão sendo seguidos [Pressman, 2011]. Atualmente, a medição é vista como um trabalho pragmático com o objetivo de encontrar o indicador direto para um determinado aspecto da qualidade [Baggen et al., 2010]. No entanto, sem critérios claros, as medições podem impedir comparações úteis com outros projetos [Baggen et al., 2010]. Portanto, métricas têm de ser escolhidas através de uma referência clara e confiável. Elas são experimentalmente avaliadas para medir um atributo de qualidade [Baggen et al., 2010]. A baixa qualidade no software pode levá-lo, a pelo menos dois problemas: a baixa separação de interesses e anomalias de código, os quais são discutidos em detalhes nas próximas seções.

2.1.1 Separação de Interesse

Interesse pode ser definido como algo que se queira tratar como unidade conceitual modular, incluindo requisitos não funcionais e idiomas de programação [Robillard & Murphy, 2007]. O termo separação de interesses foi introduzido por Dijkstra na década de 70 [Dijkstra, 1976] e a importância da separação de interesses é reconhecida desde então.

A separação de interesses direciona decisões de projeto e da implementação de software [Sommerville, 2011]. Classes, métodos e sub-rotinas são alguns dos mecanismos utilizados para realizar a separação de interesses. Contudo, todos esses mecanismos têm problemas em lidar com certos tipos de interesses, os chamados interesses transversais. A programação orientada a aspectos foi proposta para auxiliar a separação dos interesses transversais [Kiczales et al., 1997].

Diferente dos interesses transversais, os interesses centrais de um sistema são os interesses funcionais que se relacionam ao propósito primário do sistema. Os interesses transversais acontecem quando a implementação de um módulo em um sistema implementa simultaneamente vários interesses, como persistência, segurança, entre outros. Interesses transversais possuem duas propriedades principais: espalhamento[1] e entrelaçamento[2]. Espalhamento indica o quão espalhado pelos componentes do sistema o interesse se encontra [Kiczales et al., 1997]. Ou seja, quanto um interesse está espa-

[1]Do inglês *scattering*
[2]Do inglês *tangling*

lhado em diversos componentes de um programa. Entrelaçamento é o quão misturado (ou emanharado) com os outros interesses do sistema um interesse está [Kiczales et al., 1997]. Ou seja, ocorrerá quando um módulo de um sistema inclui código que implementa diferentes interesses do sistema. Estes interesses são mostrados na Figura 2.1 que foi construída com base no sistema MobileMedia. Este sistema foi o utilizado nos estudos exploratórios e é descrito em maior detalhe na Seção 3.2. Os interesses centrais são manipulação de fotos, manipulação de músicas e manipulação de vídeos. Os interesses transversais para este sistema são: segurança e persistência.

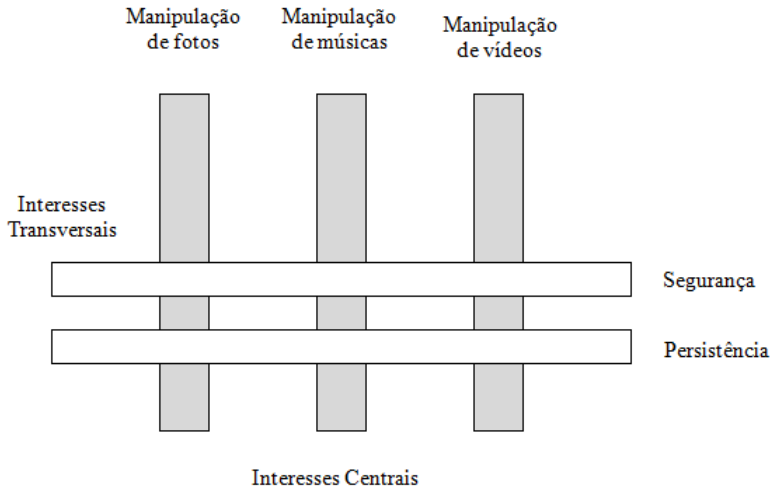

Figura 2.1. Exemplo de Interesses Transversais.

A Figura 2.2 representa um diagrama de classes para ilustrar o espalhamento e o entrelaçamento de dois interesses. Nesta figura, eles são chamados de *Interesse Transversal 1* e *Interesse Transversal 2*. Interesses transversais causam problemas para a manutenção de software [Figueiredo et al., 2008; Eaddy et al., 2008]. A manifestação de interesses transversais em software é frequentemente um indicativo de falhas de modularidade e aumento da instabilidade do software em projetos de software [Figueiredo et al., 2011]. Além disso, interesses transversais podem impactar negativamente pelo menos em um atributo externo de qualidade - ocorrência de defeitos no software [Eaddy et al., 2008].

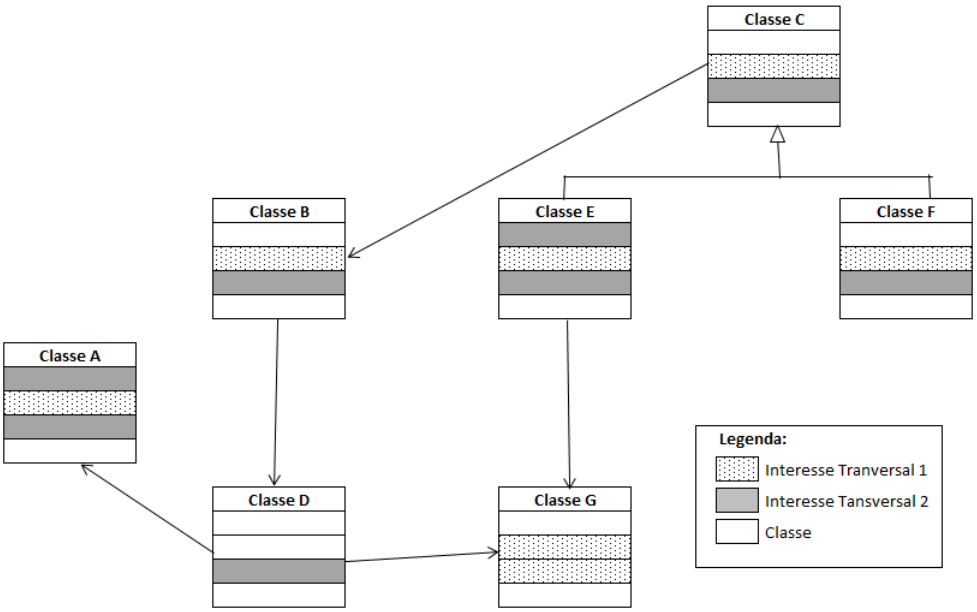

Figura 2.2. Exemplo de Entrelaçamento de Interesses.

2.1.2 Anomalias de Código

Anomalias de código[3] foram propostas por Kent Beck no livro do Fowler [Fowler, 1999] como um meio de diagnosticar sintomas que podem ser indicativos de algo errado no código do sistema. Elas descrevem uma situação em que há indícios de uma falha de projeto. Este trabalho investiga meios para detectar cinco anomalias de código, denominadas: *Divergent Change* [Fowler, 1999], *Shotgun Surgery* [Fowler, 1999], *God Class* [Riel, 1996], *Feature Envy* [Fowler, 1999] e *God Method* [Fowler, 1999]. Estas anomalias de código descritas a seguir foram escolhidas porque recorrentemente aparecem em sistemas de software. Além disso, estudos anteriores relacionaram estas anomalias com a baixa separação dos interesses [Carneiro et al., 2010; Marinescu, 2004].

Divergent Change. Esta anomalia ocorre quando uma classe é frequentemente modificada de maneiras diferentes, por razões diferentes [Fowler, 1999]. Dependendo do número de atribuições de uma dada classe, ela pode sofrer mudanças não relacionadas. O fato de que uma classe sofre diversos tipos de alterações pode estar associadas a um sintoma de entrelaçamento de interesses. Em outras palavras, uma classe que apresenta interesses entrelaçados é susceptível de ser alterada, por diferentes razões.

Shotgun Surgery. Esta anomalia é de alguma forma o oposto da anomalia *Divergent Change*. Uma classe com *Shotgun Surgery* é identificada quando fazemos um tipo de mudança que leva a uma série de pequenas mudanças em muitas classes

[3]Neste documento estamos usando anomalias de código como tradução de *bad smells*.

diferentes [Fowler, 1999]. Na visão de interesse o espalhamento de interesses pode levar a pequenas mudanças em classes que possuem estes interesses.

God Class. Esta anomalia é descrita no livro do Riel [Riel, 1996] como "um objeto que sabe demais ou faz muito". Ela representa uma classe que tem crescido além de toda a lógica para se tornar a classe que faz quase tudo no sistema [Riel, 1996]. Em uma perspectiva diferente, podemos dizer que *God Class* é uma classe que implementa muitos interesses e, por isso, tem muitas responsabilidades [Carneiro et al., 2010]. Portanto, esta classe viola a ideia de que uma classe deve capturar apenas uma abstração chave.

Feature Envy. Esta anomalia refere-se àqueles trechos de código, e.g. métodos, que parecem mais interessantes nas funcionalidades de outra classe do que nas funcionalidades de sua classe. Estes trechos de código acessam diretamente ou através de métodos de acesso uma grande quantidade de dados de outra classe. Isto pode ser um sinal que o trecho de código está mal localizado e que deveria ser transferido para outra classe, ou seja, ele implementa um interesse semelhante ao de outra classe [Fowler, 1999]. Em geral, tenta-se colocar um método ou trecho de código na classe que implementa um interesse semelhante.

God Method. Geralmente, um método é iniciado como um método "normal". Entretanto, mais e mais funcionalidades são adicionadas a ele até ele se tornar fora de controle e difícil de ser mantido e entendido. Assim, o *God Method* tende a centralizar as funcionalidades de uma classe da mesma forma que o *God Class* centraliza as funcionalidades de um subsistema inteiro, ou até mesmo de todo um sistema [Fowler, 1999]. Em uma perspectiva diferente, podemos dizer que *God Method* é um método que implementa muitos interesses e, por isso, ele tem muitas responsabilidades.

2.2 Avaliação Baseadas em Métricas de Software

Tem-se observado um grande número de trabalhos na literatura que fazem avaliação usando métricas de software. Uma característica desses trabalhos é a sua diversidade. Schneidewind [Schneidewind, 1992] recomenda um processo de avaliação empírica na qual uma métrica de software é avaliada por mostrar que ela esta associada com alguma outra medida de interesse. Weyuker [Weyuker, 1988] restringe sua discussão para métricas de complexidade e sugere que a avaliação pode ser realizada através da identificação de um conjunto de propriedades desejáveis que as medidas devem possuir. Fenton [Fenton, 1994] e Melton et al. [Melton et al., 1990] sugerem que métricas avaliadas devem obedecer a condição de representação da teoria de medição, de modo que

a compreensão intuitiva de algum atributo seja preservada quando ela é mapeada para um sistema de relação numérica. Estes trabalhos confirmam a afirmação de Chidamber e Kemerer [1994] que métricas de software são os principais mecanismos para avaliar a qualidade. Eles também completam a afirmação de Marinescu [2004] sobre as métricas de software ser vista como uma solução para ajudar na identificação de anomalias de código.

É por isso que, cada vez mais, métricas de software estão desempenhando um papel importante no entendimento, desenvolvimento e manutenção de software. É comum que desenvolvedores realizem a medição de propriedades do código para ter alguma previsão se o projeto tem boa qualidade ou se o código está pronto para ser testado. Muitas ferramentas de desenvolvimento incluem a coleta de métricas e análise de dados para ajudar entender o sistema [Li & Shatnawi, 2007]. Podemos citar como exemplo as ferramentas: Together[4] e Enterprise Architect[5]. Tais métricas quantificam, por exemplo, a complexidade de um módulo, a extensão média dos identificadores em um programa e o número de atributos e operações associados com os objetos em um projeto [Sommerville, 2011].

Em termos gerais, medição é o processo pelo qual números ou símbolos são designados a atributos de entidades do mundo real de forma a descrevê-los de acordo com regras claramente definidas [Fenton & Pfleeger, 1996]. A primeira tarefa de qualquer atividade de medição é identificar as entidades e atributos que se deseja medir [Kitchenham et al., 1995]. Uma entidade é um objeto (como uma pessoa ou um quarto) ou um evento (como uma viagem ou o projeto de desenvolvimento de um software). Um atributo é uma característica ou propriedade de uma entidade, como por exemplo, área de um quarto, o tempo de uma viagem ou o custo de um projeto de desenvolvimento de um software [Sant'Anna, 2004].

Os atributos por sua vez podem ser classificados em: atributos internos e atributos externos. Atributos internos são aqueles atributos que podem ser medidos ao se examinar diretamente o produto, processo ou recurso separadamente do seu comportamento. São exemplos de atributos internos: tamanho, complexidade ou dependência entre módulos [Paula Filho, 2009]. Atributos externos por outro lado, são aqueles que só podem ser medidos em termos de como o processo, produto ou recurso se relaciona com o seu ambiente. Exemplos de atributos externos são: confiabilidade, facilidade de compreensão, manutenibilidade, usabilidade, integridade, eficiência, reusabilidade, portabilidade e interoperabilidade. Geralmente, esses atributos só podem ser medidos quando o código é executado [Paula Filho, 2009].

[4]http://borland.com/products/Together/
[5]http://www.sparxsystems.com/products/ea/index.html

2.2.1 Modelos de Qualidade

É difícil medir os atributos externos de software diretamente [Sommerville, 2011]. Esta observação motivou o desenvolvimento de modelos de qualidade. Estes modelos são geralmente representados em forma de árvore, semelhante ao apresentado na Figura 2.3 [Mccall, 1977; Boehm et al., 1978]. Eles descrevem os relacionamentos pertinentes entre os atributos externos e os atributos internos que os influenciam [Sommerville, 2011].

Na Figura 2.3, os ramos apresentados mais à esquerda representam os atributos externos que se deseja avaliar, como manutenibilidade e confiabilidade. Cada atributo externo é decomposto em atributos internos, como a complexidade ciclomática e tamanho do programa em linhas de código. Portanto, os atributos externos podem ser medidos em termos de atributos internos pelos quais eles são influenciados. Como os atributos internos são fáceis de se medir, métricas de software são propostas para eles [Fenton & Pfleeger, 1996].

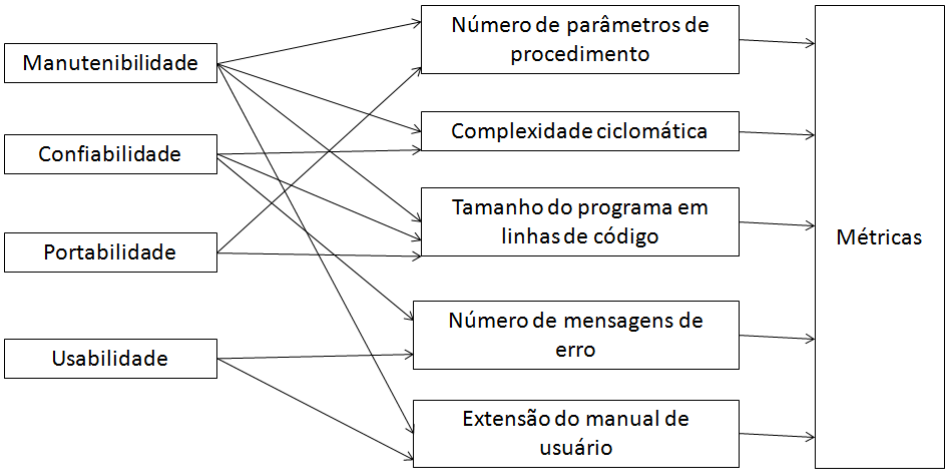

Figura 2.3. Relacionamentos entre Atributos Externos e Internos do Software.

2.2.2 Método Objetivo-Questão-Métrica

A medição de atributos de produto é essencial para o aprimoramento do software [Humphrey & Kellner, 1989]. A dificuldade fundamental da medição é justamente saber o que medir. Basili e Rombach [1988] propuseram um método chamado Objetivo-Questão-Métrica (GQM - *Goal-Question-Metric*). GQM é um método orientado a objetos para a medição de software que apoia a definição *top-down* da medição e a análise *botton-up* dos dados relevantes. As principais vantagens deste método são: (i)

auxiliar na identificação de métricas úteis e relevantes; (ii) apoiar a análise e interpretação dos dados coletados; (iii) propiciar uma avaliação da validade das conclusões tiradas; e (iv) reduzir a resistência quanto a utilização de processos de medição [Gresse & Ruhe, 1996].

Os três elementos principais do método GQM são descritos abaixo [Basili & Rombach, 1988].

1. **Objetivos.** Listar um conjunto de objetivos do processo de medição;
2. **Questões.** Refinar o conjunto de objetivos em perguntas que devem ser respondidas para averiguar se os objetivos foram atingidos;
3. **Métricas.** Averiguar o que necessita ser medido para auxiliar a responder às questões e confirmar se os objetivos da medição do processo foram capazes de responder as questões adequadamente. A escolha da métrica para coletar e avaliar os dados depende do tipo de cada métrica.

Um *objetivo*, pode ser: aumentar a produtividade dos programadores. As *questões* devem estar relacionadas ao objetivo, como: (i) Como o número de linhas de código produzidas por dia podem ser aumentadas? e (ii) Como reduzir o tempo necessário para especificação de requisitos?. Para cada pergunta, as *métricas* relevantes são definidas, como: número total de linhas de código. A vantagem da aplicação do GQM no aprimoramento do processo de medição é que ele separa os *objetivos* das *questões*. Ele enfoca a coleta de dados e sugere que os dados coletados devam ser analisados em função dos objetivos. A Figura 2.4 mostra a estrutura hierárquica gerada pela utilização da abordagem GQM.

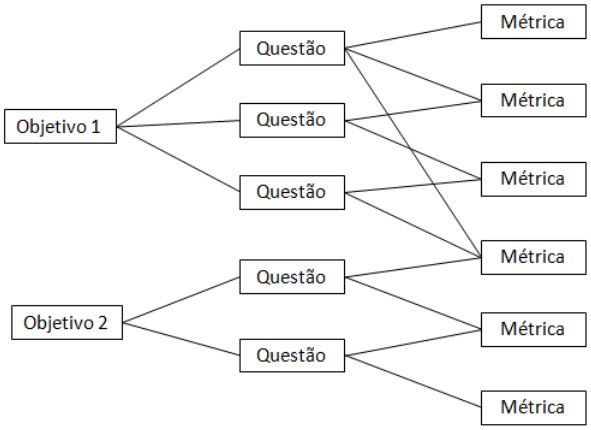

Figura 2.4. Estrutura Hierárquica da Abordagem GQM.

2.3 Métricas Tradicionais

Um conjunto conhecido e utilizado de métricas tradicionais orientados a objetos foi proposto por Chidamber e Kemerer [1994]. Os autores deste conjunto de métricas, que chamamos conjunto CK, afirmam que estas métricas podem auxiliar os usuários no entendimento da complexidade de projetos orientados a objetos, na detecção de anomalias e na previsão de alguns resultados do projeto. Tais métricas também têm sido utilizadas para avaliar atributos de qualidade externa do software, tais como: defeitos de software, testes e esforço de manutenção [Basili et al., 1996; Chidamber & Kemerer, 1994].

Além do conjunto CK, outros conjuntos de métricas tradicionais foram propostos, tais como: conjunto de métricas MOOD [Harrison & Nithi, 1998] e o conjunto Lorez e Kidd [Lorenz & Kidd, 1994]. O conjunto MOOD tem o objetivo de proporcionar indicadores quantitativos para as propriedades dos projetos orientados a objeto através das métricas [Pressman, 2011]. Já o conjunto Lorez e Kidd dividem as métricas baseadas em classes em quatro categorias principais, onde cada uma ocupa uma posição no projeto em nível de componente: tamanho, herança, atributos internos e externos [Pressman, 2011]. As métricas tradicionais que usamos no decorrer deste trabalho são descritas no restante desta seção.

Linhas de Código (LOC). Conta o número de linhas de código de um método ou classe [Fenton & Pfleeger, 1996]. É comum o uso desta métrica para saber o tamanho do código do sistema. A definição de LOC adotada neste trabalho não conta linhas em branco e nem os comentários relativos à documentação. Ressalta-se que estilos diferentes de programação podem influenciar nos resultados obtidos pela aplicação desta métrica. Quanto maior o número de linhas de código, maior será a dificuldade para entender o software, e consequentemente, mais difícil será para localizar as linhas que precisam ser alteradas durante atividades de manutenção do software. Além disso, um código grande torna difícil compreender a implementação das funcionalidades que se deseja reutilizar [Sant'Anna, 2004].

A Figura 2.5 mostra um trecho de código do sistema MobileMedia para exemplificar a contagem da métrica LOC. Este sistema foi utilizado nos experimentos realizados e é descrito em detalhes na Seção 3.2. Desprezando as linhas em branco, o valor da métrica LOC para a classe *VideoAlbumData* é quatorze. Note que separadores da linguagem Java, como parênteses, chaves e ponto-e-vírgula são contados pela métrica LOC.

```
public class VideoAlbumData extends AlbumData{

	public VideoAlbumData() {
		mediaAccessor = new VideoMediaAccessor(this);
	}

	public InputStream getVideoFromRecordStore(String recordStore, String musicName) throws ImageNotFoundException,
PersistenceMechanismException {
		MediaData mediaInfo = null;
		mediaInfo = mediaAccessor.getMediaInfo(musicName);
		int mediaId = mediaInfo.getForeignRecordId();
		String album = mediaInfo.getParentAlbumName();
		byte[] musicData = (mediaAccessor).loadMediaBytesFromRMS(album, mediaId);
		return new ByteArrayInputStream(musicData);

	}
}
```

Figura 2.5. Exemplo da Métrica LOC.

Números de Atributos (NOA). Esta métrica conta o número total de atributos definidos em uma classe [Fenton & Pfleeger, 1996]. Atributos herdados de superclasses não são contados por essa métrica. Quanto maior o número de atributos na classe, mais difícil será para entender o software e também para encontrar os locais que precisam ser modificados durante atividade de manutenção do software [Sant'Anna, 2004]. O argumento é que será mais difícil compreender a implementação das funcionalidades que se deseja reutilizar. Para exemplificar a métrica NOA utilizamos a Figura 2.6, que mostra um diagrama parcial da classe *AbstractController* e suas subclasses do sistema MobileMedia. Nesta figura vimos que a classe *PlayVideoController* possui um atributo, portanto, o valor da métrica NOA para esta classe é um.

Número de Operações (NOO). Esta métrica conta o número total de operações e construtores. Quanto maior o número de operações em uma classe, mais complexa esta classe tende a ser [Li & Henry, 1993]. Isto que dizer que uma classe que tem muitas operações é uma classe que faz muitas coisas. Portanto será mais difícil compreender esta classe para evoluir o software. Considerando a Figura 2.6, contamos que a classe *PlayVideoController* possui quatro operações, portanto, o valor da métrica NOO é igual a quatro.

Número de Parâmetros (PAR). Esta métrica consiste na contagem dos números de parâmetros para uma operação [Fenton & Pfleeger, 1996]. Operações com um grande número de parâmetros geralmente indicam que elas executam processamento complexo. Quanto maior o número de parâmetros para uma operação, mais difícil é a compreensão desta operação e, consequentemente, da classe na qual ela se encontra. Pode-se observar na Figura 2.6 que a operação *handleCommand* da classe *MediaListController* possui um parâmetro - *c*. Portanto, o valor da métrica PAR para esta operação é um. Observando a operação *exchange* da mesma classe verificamos que ela tem três parâmetros - *m*, *p1* e *p2*. Portanto, valor da métrica PAR é três.

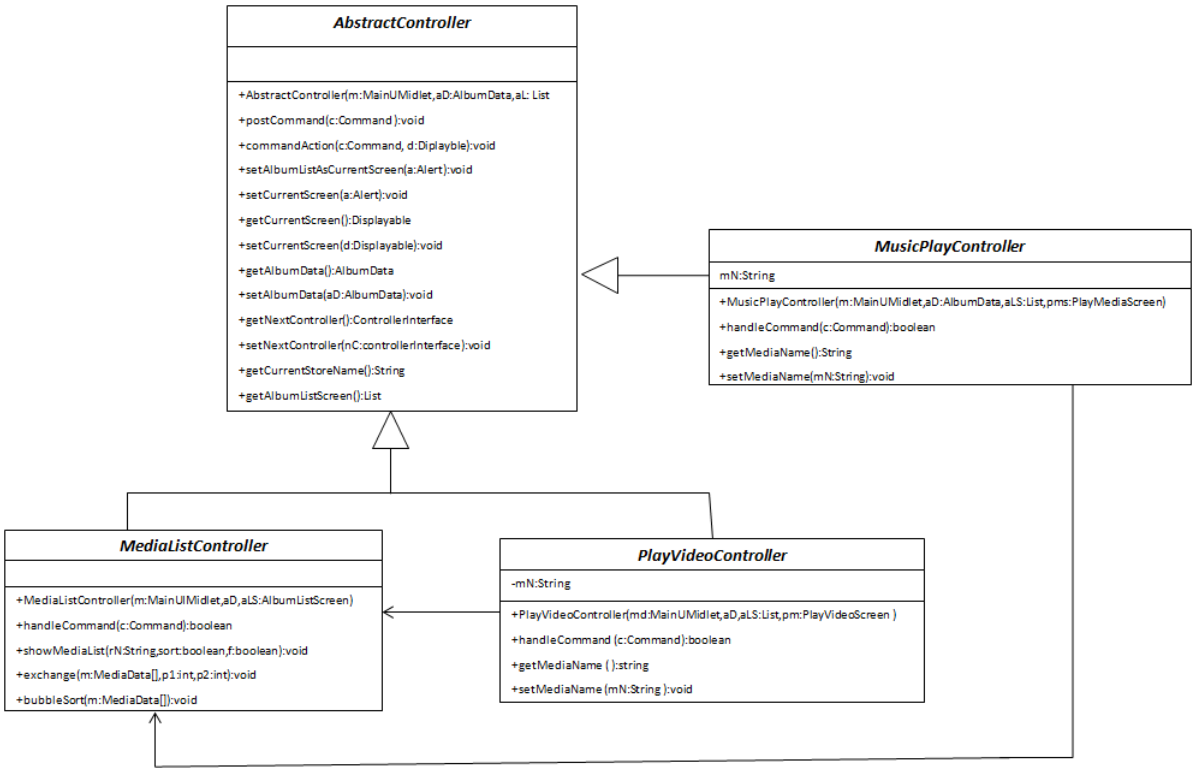

Figura 2.6. Diagrama parcial da classe *AbstractController*

Método Ponderado por Classe (WMC). Esta métrica se relaciona diretamente com a definição de Bunge [Bunge, 1979] sobre complexidade de coisas, uma vez que as operações são propriedades de classes de objetos e a complexidade é determinada pela cardinalidade do seu conjunto de propriedades. Portanto, a complexidade de uma classe é definida em função do número de operações e pesos relativos destas operações. De acordo com Chidamber e Kemerer [1994] se a complexidade de todas as operações é considerada como unidade, então WMC é igual à NOO (Número de Operações). Entretanto, operações podem ser pesadas em função de seus parâmetros [Chidamber & Kemerer, 1994]. Usamos esta última definição de WMC na qual WMC é a soma das operações de uma classe e seus respectivos parâmetros.

Para exemplificar a métrica WMC utilizamos a Figura 2.6. Observando a classe *PlayVideoController*, constatamos que ela possui quatro operações. Para calcular a métrica WMC, as operações devem ser pesadas em função de seus parâmetros. O construtor possui três parâmetros. Além disso, as operações *handleCommand* e *setMediaName* possuem um parâmetro. Portanto, o valor de WMC para esta classe é dez.

Acoplamento entre Objetos(CBO). Esta métrica consiste na contagem do número de classes com as quais uma classe se relaciona [Chidamber & Kemerer, 1994]. CBO está relacionado com a noção de que um objeto é acoplado a outro objeto quando suas operações utilizam operações ou variáveis de instância de outros objetos. Portanto, é um totalizador do número de classes às quais uma determinada classe é cliente. A Figura 2.6 ilustra um exemplo na classe *AbstractController*. Esta classe não está acoplada a nenhuma classe, portanto o valor da métrica CBO é zero. A classe *MusicPlayController* está acoplada as classes *AbstractController* e *MediaListController*, portanto o valor da métrica CBO é dois. Para a classe *PlayVideoController* que está acoplada as classes *MediaListController* e *AbstractController* o valor de CBO também é igual a dois. Por último, a classe *MediaListController* está acoplada apenas a classe *AbstractController* tem o valor de CBO igual a um.

Falta de Coesão em Métodos (LCOM). Esta métrica mede o quanto as operações de uma classe acessam atributos em comum. Quanto maior o número de atributos em comum, maior é a coesão da classe e, portanto, menor é a perda de coesão (LCOM) [Chidamber & Kemerer, 1994]. Esta definição é consistente com os conceitos tradicionais de coesão que medem a inter-relação entre as partes de um software. Se nenhuma das operações de uma classe acessar qualquer variável de instância, elas não têm similaridade. A LCOM esta intimamente ligada às variáveis de instâncias e as operações de uma classe. Portanto, é uma medida dos atributos de uma classe de objetos.

Complexidade Ciclomática de McCabe (CYCLO). Esta métrica foi proposta por McCabe [1976] para quantificar a complexidade de um código através de grafos. A complexidade é obtida através da medição do número de caminhos linearmente independentes. Quanto mais alto o número de CYCLO, mais complexo é o código [McCabe, 1976]. A complexidade ciclomática é derivada a partir de um grafo de fluxo de um código de controle, como mostra a fórmula abaixo [Fenton & Pfleeger, 1996]:

$$CYCLO = E - N + 2 \tag{2.1}$$

Onde:

- E = número de arestas (Ex: estrutura de controle);
- N = número de nodos (Ex: grupo sequencial de declarações contendo apenas uma estrutura de controle);

A medição da complexidade ciclomática, através da métrica CYCLO, mostra que os desenvolvedores são sensíveis ao fato de que os programas com um elevado número de CYCLO (por exemplo, maior que 10) são susceptíveis de serem difíceis de entender. Portanto, têm uma maior probabilidade de conter defeitos. O número de CYCLO também indica o número de casos de testes que teriam de ser realizadas para executar todos os percursos dentro de um programa [Fenton & Pfleeger, 1996].

2.4 Métricas de Interesse

As métricas de interesse foram definidas para capturar propriedades de modularidade associadas com a realização de interesses em artefatos de software [Sant'Anna et al., 2008]. Seu objetivo é a identificação de anomalias específicas em projeto ou a degeneração causada pela carência de processo de modularização de interesses [Eaddy et al., 2008]. Métricas de interesse necessitam do mapeamento de interesses para elementos de código. Robillard e Murphy [2007] definem interesse como algo que se queira tratar como uma unidade conceitual modular, incluindo requisitos não funcionais e idiomas de programação.

A Figura 2.7 ilustra um mapeamento entre os interesses e os elementos de projeto. O mapeamento consiste em atribuir um interesse para os elementos dos projetos correspondentes à realização. Métricas de interesses quantificam o grau de espalhamento e entrelaçamento de interesses. A seguir, descrevem-se as métricas de interesses usadas neste trabalho. Umas destas métricas foi definida para este trabalho: Número de Interesse por Operações (NCO).

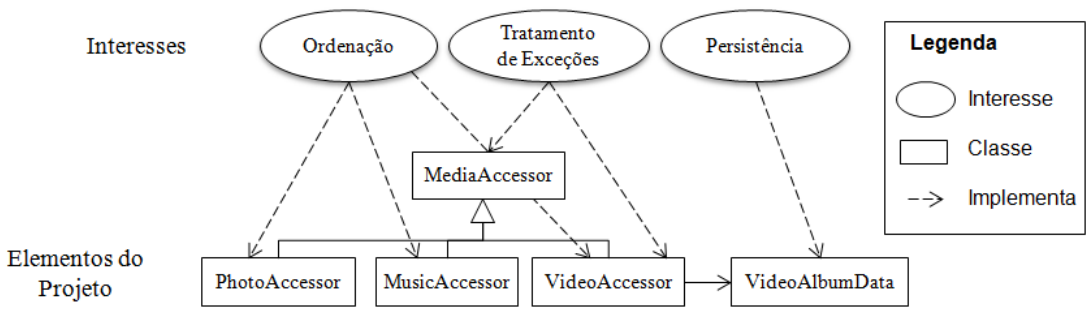

Figura 2.7. Mapeamento de interesses.

Espalhamento de Interesses em Linhas de Código (CDLOC). Foi definida com o objetivo de capturar como o código que implementa um interesse está espalhado pelo código dos outros interesses. CDLOC conta o número de pontos de transição entre o interesse avaliado e os outros interesses do sistema através das linhas de código [Sant'Anna, 2004]. Assim, como as outras métricas de interesse, o uso desta métrica requer que as partes do código do sistema que implementam o interesse avaliado sejam sombreadas. Esse processo de sombreamento divide então o código do sistema em áreas sombreadas e não sombreadas. Os pontos de transição são os pontos do código em que há uma transição de uma área sombreada para uma área não sombreada ou vice-versa. A intuição por trás desta métrica é que os pontos de transição são pontos do programa onde há uma "troca de interesse". Assim, para cada interesse, o código do programa é analisado linha por linha para contar os pontos de transição [Sant'Anna, 2004].

Quanto maior o valor de CDLOC, mais entrelaçado o código do interesse avaliado está com o código de outros interesses e mais difícil será para compreendê-lo. Portanto, o CDLOC indica os pontos não consecutivos do código que serão afetados durante atividades de manutenção. Além disso, quanto maior CDLOC, mais difícil será para reutilizar o código desse interesse [Sant'Anna, 2004]. Para ilustrar esta métrica, a Figura 2.8 mostra o código da classe *AbstractController*. Esta figura indica que existem seis pontos de transição entre o interesse: *Chain of Responsability*, que pertence ao conjunto de padrões de projeto Gang-of-Four(GoF) [Gamma et al., 1994] e outros interesses da classe (código em branco). Assim, nesta classe o valor da métrica CDLOC para o interesse avaliado é seis.

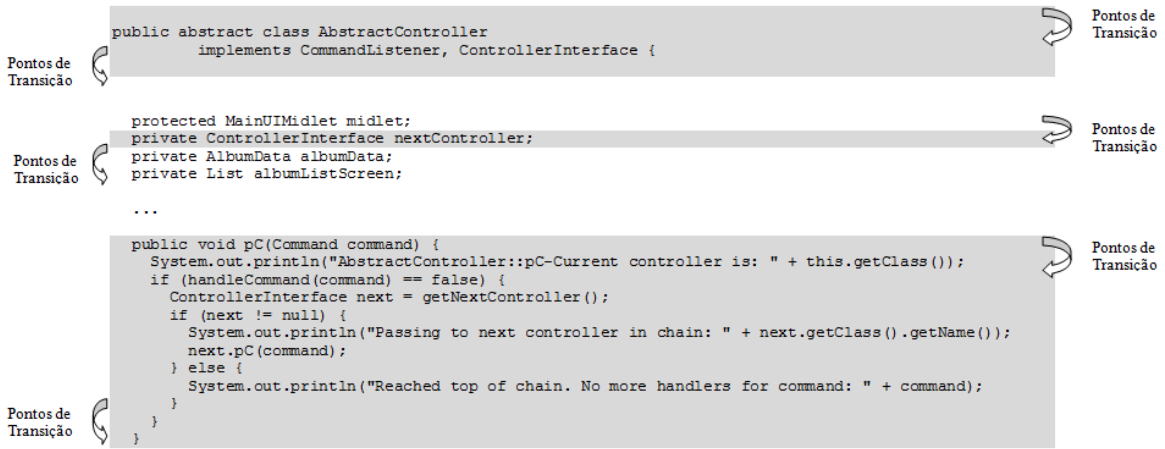

Figura 2.8. Exemplo da Métrica CDLOC na Classe *Abstract Controller*.

Linhas de Código do Interesse (LOCC). Conta o número de linhas de código que implementam um interesse em cada classe. Ou seja, ela conta as linhas de código que foram sombreadas para um interesse [Eaddy et al., 2008]. A definição de LOCC adotada neste trabalho não conta linhas em branco e nem os comentários relativos à documentação [Fenton & Pfleeger, 1996]. Pode-se observar na Figura 2.8 o sombreamento relativo ao interesse *Chain of Responsability* no código da classe *AbstractController*. Esta figura indica que o valor da métrica LOCC para o interesse avaliado é dezoito.

Espalhamento de Interesses em Classes (CDC). Esta métrica conta o número de classes que contribuem para a implementação do interesse avaliado [Sant'Anna, 2004]. Além disso, ela conta também às classes que fazem referência as classes principais do interesse. Ou seja, conta o número de classes que acessam as classes principais usando-as em declaração de atributos, parâmetros de operações, tipos de retorno ou variáveis locais, ou chamam suas operações. Essa métrica mede o grau com o qual um interesse está espalhado pelas classes do projeto do software. Quanto menos espalhado estiver o interesse, mais fácil será para entendê-lo. Quanto menos espalhado estiver o interesse, menos classes terão que ser alteradas em atividades de manutenção, e menos classes terão que ser compreendidas e estendidas em atividades de reutilização [Sant'Anna, 2004].

A Figura 2.9 mostra um diagrama parcial da classe *Abstract Controller* e três subclasses do sistema MobileMedia. Este diagrama mostra as classes que contribuem para implementação de um dos interesses do sistema do padrão do projeto *Chain of Responsability* [Hannemann & Kiczales, 2002]. O código da classe abstrata *AbstractController* e demais classes que implementam o interesse está sombreado em cinza. Este diagrama mostra quatro classes, todas estão sombreadas indicando onde se encontra o interesse *Chain of Responsability*. Considerando apenas as classes que aparecem no diagrama da Figura 2.9, o valor para a métrica CDC para o interesse analisado é quatro.

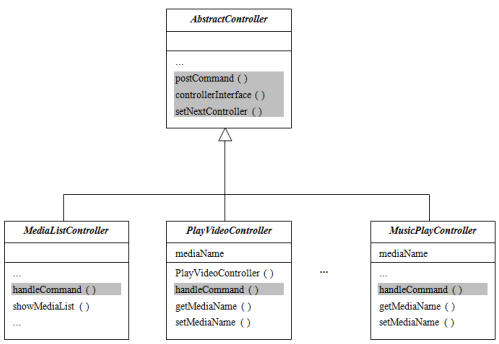

Figura 2.9. Diagrama Parcial da Classe *AbstractController*.

Espalhamento de Interesses em Operações (CDO). Conta o número de operações cujo propósito principal é contribuir para a implementação do interesse avaliado [Sant'Anna, 2004]. Esta métrica conta também o número de operações que acessam qualquer operação ou classe principal. Ressalta que, construtores e operações abstratas também são contados. Quanto mais operações forem afetadas pelo interesse, mais difícil será para entendê-lo, reutilizá-lo e mais operações serão modificadas durante atividades de manutenção [Sant'Anna, 2004].

Pelo código da Figura 2.9, deve-se contar as operações que implementam o interesse *Chain of Responsability*. Nesta figura, as operações já estão sombreadas e, portanto o valor da métrica CDO é igual a seis. Ou seja, seis operações realizam o interesse avaliado: *postCommand*, *controllerInterface* e *setNextController* na classe *AbstractController* e três implementações da operação *handleCommand* nas classes *MediaListController*, *PlayVideoController* e *MusicPlayController*.

Número de Operações dos Interesses (NOCO). Conta o número de operações que implementam um interesse na classe [Figueiredo et al., 2012]. Observando o sombreamento da classe *AbstractController* na Figura 2.9, constatamos que, três operações desta classe implementam o interesse do padrão *Chain of Responsability*. Sendo assim, o valor da métrica NOCO é três para o interesse avaliado.

Número de Atributos dos Interesses (NOCA). Conta o número de atributos que implementam um interesse na classe [Figueiredo et al., 2012]. Atributos são variáveis de instância, de classes ou constantes. A Figura 2.10 mostra um diagrama parcial da classe *MediaAccessor* e de suas subclasses do sistema MobileMedia (Seção 3.2). Este diagrama mostra as classes que contribuem para implementação de dois interesses, *Persistência* e *Tratamento de Exceção* do sistema. A contagem de NOCA é feita através da contagem dos atributos sombreados para o interesse analisado. Considerando o sombreamento na classe *MediaAccessor*, todos os cinco atributos desta classe estão sombreados para o interesse *Persistência*. Portanto, o valor da métrica NOCA é cinco para este interesse nesta classe.

Número de Interesse por Componente (NCC). Conta o número de interesses em cada classe ou interface. Seu objetivo é apoiar desenvolvedores sobre a observância ao grau de entrelaçamento no código [Figueiredo et al., 2012]. A Figura 2.10 mostra as classes que contribuem para implementação de dois interesses, *Persistência* e *Tratamento de Exceção* do sistema. Analisando a classe *MediaAccessor*, observamos que ela implementa os dois interesses marcados no diagrama. Portanto, o valor da métrica NCC para esta classe é dois. Em relação às subclasses, *ImageMediaAccessor* e *MusicMediaAccessor*, observamos que elas também implementam os dois interesses analisados. Portanto, o valor de NCC para cada uma destas subclasses é igual a dois.

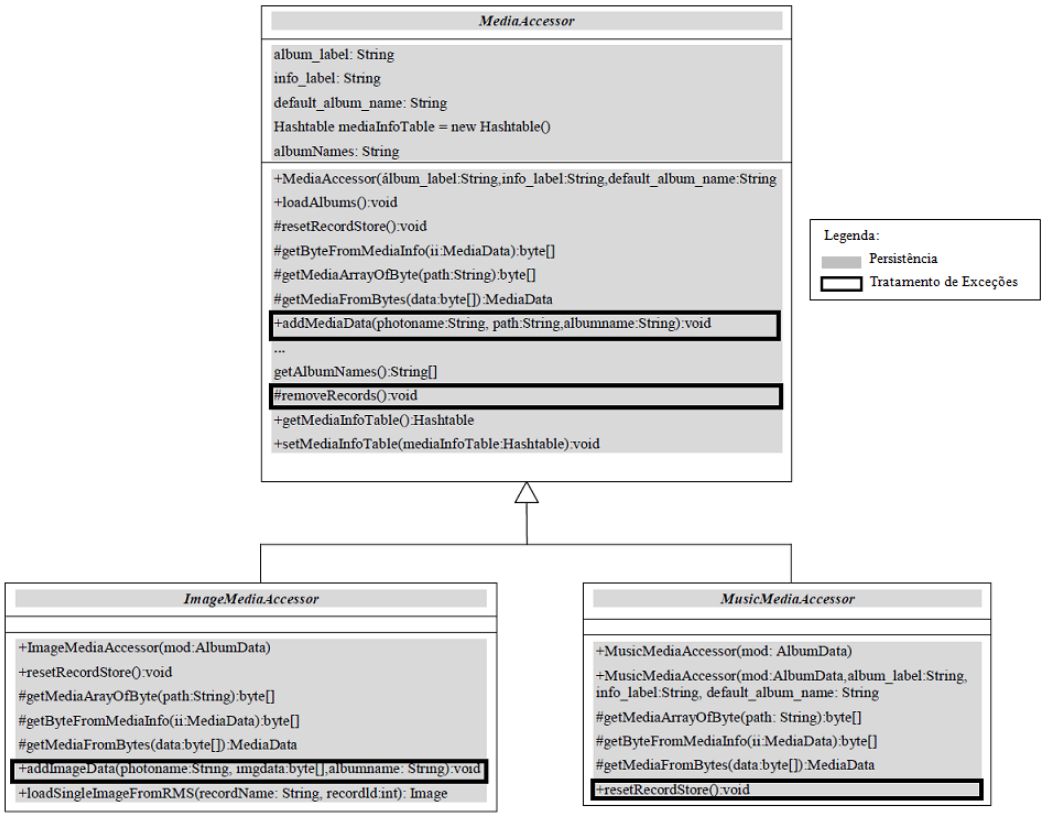

Figura 2.10. Diagrama Parcial da Classe *MediaAccessor*.

Número de Interesse por Operações (NCO). Esta métrica foi proposta neste trabalho. Ela conta o número de interesses implementados por operação nas classes analisadas. O objetivo desta métrica é apoiar desenvolvedores sobre a observância ao grau de entrelaçamento das operações de uma classe. A Figura 2.10 mostra a classe *MediaAccessor* que possui duas operações, *addMediaData* e *removeRecords*, que contribuem para a implementação de dois interesses, *Persistência* e *Tratamento de Exceção*. Portanto, o valor da métrica NCO para estas operações é dois. Analisando a classe *ImageMediaAccessor*, observamos que ela implementa os dois interesses na operação *addImageData*, sendo dois o valor da métrica NCO para esta operação.

2.5 Regras Heurísticas

Avaliar a qualidade e a melhoria de programas orientados a objetos é uma tarefa difícil. As métricas de software são poderosos mecanismos para avaliar e controlar a qualidade do projeto [Chidamber & Kemerer, 1994]. Entretanto, as métricas também apresentam alguns problemas. É difícil fornecer uma interpretação relevante para a medição do

software devido à baixa granularidade das métricas e à quantidade de dados gerados [Marinescu, 2004; Lanza & Marinescu, 2006].

Outra dificuldade reside na lacuna que existe entre como a qualidade é percebida e como ela é expressa fora do sistema de software. A interpretação das métricas isoladamente, geralmente, não é suficiente [Marinescu, 2004]. Na maioria dos casos em que as métricas são usadas individualmente, elas não fornecem informações suficientes sobre a causa de um problema. O valor de uma métrica pode indicar que há uma anomalia no código, mas deixa o desenvolvedor sem indício sobre qual é a causa final da anomalia. Consequentemente, isso tem um impacto negativo na relevância dos resultados da medição [Marinescu, 2004; Lanza & Marinescu, 2006].

A fim de superar as limitações das métricas de software, Marinescu [2004] e Sahraoui [2000] propuseram abordagens para facilitar a interpretação das métricas, fornecendo definições precisas das métricas e modelos de interpretação que podem ser aplicados na avaliação de projetos. Em suma, elas propuseram um mecanismo para analisar o código fonte de um sistema utilizando regras baseada em métricas, também denominada heurística. Marinescu [2004] chamou este mecanismo de *estratégia de detecção*. Enquanto, Sahraoui [2000] chama-o de heurísticas. Nesta dissertação, nós a chamamos de *regras heurísticas* ou simplesmente *heurísticas*.

As regras heurísticas buscam ajudar os desenvolvedores a detectar e a localizar problemas no sistema. Uma regra heurística é composta de uma condição lógica, baseada em métricas, as quais detectam elementos de um projeto com anomalias de código. Usando as regras heurísticas, um desenvolvedor pode diretamente localizar classes ou métodos potencialmente afetados por anomalias de código, como *God Class* [Riel, 1996] e *God Method* [Fowler, 1999].

O uso de métricas em regras heurísticas baseia-se em mecanismos de filtragem e composição [Marinescu, 2004]. O mecanismo de filtragem reduz o conjunto de dados inicial, de modo que somente os valores que apresentam uma propriedade especial são mantidos. O propósito é detectar os elementos de um projeto que têm propriedades especiais que são capturadas por métricas. Assim, para definir um filtro de dados é necessário definir os valores para os limites inferiores e superiores do subconjunto filtrado [Marinescu, 2004; Lanza & Marinescu, 2006].

Os limites do subconjunto são definidos com base no tipo de filtro de dados. Os filtros podem ser agrupados em duas categorias principais: Filtros Absolutos e Filtros Relativos [Marinescu, 2004]. *Filtros Absolutos* podem ser parametrizado com um valor numérico representando o limiar; por exemplo: HigherThan(20) e LowThan(6). *Filtros Relativos* delimitam os dados filtrados e são definidos por um parâmetro que especifica o número de entidades a serem recuperados, em vez de especificar o valor máximo

(ou mínimo) permitido no conjunto de resultados. Assim, os valores no conjunto de resultados serão em relação ao conjunto original de dados, por exemplo: TopValues(10) e BottomValues(5%) [Marinescu, 2004].

Como consequência, em adição ao mecanismo de filtragem que apoia a interpretação individual dos resultados das métricas, um segundo mecanismo correlaciona a interpretação de vários conjuntos de resultados. O mecanismo de composição baseia-se na composição de um conjunto de operadores que agrupa métricas diferentes em uma regra composta [Marinescu, 2004; Lanza & Marinescu, 2006]. Os operadores lógicos usados em regras heurísticas são: AND e OR.

Para ilustrar o que é uma regra heurística, apresentamos aqui uma das regras que foi proposta por Marinescu [2002]. Esta regra pretende detectar *God Class* [Riel, 1996]. A regra heurística proposta por Marinescu para esta anomalia é baseada em três métricas tradicionais: (i) ATFD que mede o acoplamento, (ii) WMC que mede a complexidade e (iii) TCC que mede a coesão de uma classe. Esta regra é definida e apresentada como segue [Marinescu, 2002]:

$$GodClass := ((ATFD, TopValues(20\%))and(ATFD, HigherThan(4)))$$

$$and((WMC, HigherThan(20))or(TCC, LoweThan(0.33))$$

A métrica ATFD[6], Acesso aos Dados Estrangeiros, conta o número de acesso que a classe faz aos atributos de outras classes, sendo diretamente ou acesso através dos métodos de acesso [Marinescu, 2002, 2004; Lanza & Marinescu, 2006]. WMC, Método Ponderado por Classe, é a soma ponderada da complexidade de todos os métodos em uma classe [Chidamber & Kemerer, 1994]. TCC[7], Coesão de Classe Estreita, é definida como um número relativo de métodos diretamente conectados. Dois métodos estão diretamente conectados se eles acessam uma instância comum das variáveis da classe [Marinescu, 2002, 2004; Lanza & Marinescu, 2006]. A regra heurística para God Class diz que a classe não deve ter valores superiores a 20% para ATFD e não ter ATFD mais alto que 4. Diz também que, WMC não deve ser um valor mais alto que 20 ou TCC com um valor mais baixo que 0.33. Estes critérios representam a ocorrência da anomalia *God Class*.

Todas as regras heurísticas encontradas na literatura [Marinescu, 2004; Lanza & Marinescu, 2006; Sahraoui et al., 2000] foram baseadas em métricas tradicionais. Isto é, uma característica comum destas heurísticas é que elas são restritas as propriedades

[6]Do inglês Access To Foreign Data
[7]Do inglês Tight Class Cohesion

modulares que são explicitamente definidas na linguagem de programação orientada a objetos, tais como classes e métodos. Então, diante da limitação destas regras heurísticas, serão propostas no Capítulo 5 novas regras heurísticas que envolveram métricas de interesse.

2.6 Considerações Finais

Avaliar a qualidade de um software não é uma tarefa fácil. Ainda mais se esta avaliação for feita na base do senso comum, o que não propicia uma avaliação objetiva do software. Portanto, é necessária a existência de metodologias que controlem a qualidade. A forma mais utilizada tem sido as métricas de software, pois permitem realizar uma avaliação quantitativa e objetiva do software.

A tese desta dissertação é que as métricas de interesse e também as regras heurísticas, que combinam métricas de interesse com as métricas tradicionais sejam mecanismos que ajudam na identificação de anomalias de código; que foram relatadas neste capítulo. Neste contexto, surge-se a necessidade de avaliar se as métricas de software, em especial as métricas de interesse, são eficazes para detectar anomalias de código. Diante desta necessidade, apresentarem nos Capítulos 3 e 4 dois experimentos realizados com participantes para detectar anomalias em classes e em métodos, respectivamente. No Capítulo 5, apresentaremos as regras heurísticas compostas por métricas obtidas como úteis nos experimentos com os participantes.

Capítulo 3

Detecção de Anomalias em Classes

Métricas são utilizadas para avaliar a qualidade e manutenibilidade do software, apoiando a identificação de anomalias de código. Apesar das métricas de interesse já terem sido utilizadas em estudos experimentais [Conejero et al., 2012; Figueiredo et al., 2008; Greenwood et al., 2007; Eaddy et al., 2008; Figueiredo et al., 2012], ainda falta conhecimento empírico quanto à sua eficácia na identificação destas anomalias.

Este capítulo apresenta um estudo para avaliar a acurácia das métricas de interesses na detecção de anomalias em classes. Nosso estudo utilizou as métricas tradicionais como ponto de comparação. Portanto, nós realizamos uma análise comparativa entre as métricas tradicionais e as de interesse a fim de identificar se esta última apoia a detecção de três anomalias de classes: *Divergent Change*, *Shotgun Surgery* e *God Class*.

3.1 Questões de Pesquisa

O objetivo deste estudo é verificar se e quais métricas de interesse são adequadas para detectar anomalias em classes. Portanto, uma questão geral de pesquisa que visa averiguar tal objetivo foi definida da seguinte forma.

RQ. As métricas de interesse apoiam a detecção de anomalias em classes?

Realizamos uma investigação com participantes para analisar a utilidade das métricas de interesse, a fim de responder à questão de pesquisa RQ. As questões específicas a seguir foram derivadas a partir da RQ descrita anteriormente.

- *RQ1. Qual a acurácia das métricas de interesse comparadas com as métricas tradicionais na detecção de anomalias em classes?*

- *RQ2. A experiência dos desenvolvedores impacta na acurácia para detectar as anomalias em classes?*
- *RQ3. Um conjunto composto por muitas métricas pode fazer com que a tarefa de identificação das anomalias em classes seja mais demorada?*
- *RQ4. Existe uma métrica específica que detecta com acurácia cada anomalia em classe?*
- *RQ5. Existe uma combinação de métricas que detecta com acurácia anomalias em classes?*

3.2 Sistemas Utilizados

O estudo apresentado neste capítulo envolveu dois sistemas: Health Watcher [Greenwood et al., 2007] e MobileMedia [Figueiredo et al., 2008]. Estes sistemas foram selecionados porque (i) eles foram usados anteriormente em outros estudos relacionados com manutenibilidade [Conejero et al., 2012; Ferrari et al., 2010; Figueiredo et al., 2008; Greenwood et al., 2007] e (ii) temos acesso a desenvolvedores e especialistas nestes sistemas. Portanto, nós fomos capazes de recuperar a lista de referência das anomalias de código para cada sistema analisado. Uma breve descrição das funcionalidades do Health Waltcher e do MobileMedia e seus principais interesses é apresentada nas subseções a seguir.

3.2.1 Health Watcher

Health Watcher é um sistema Web de informação que suporta o registro de reclamações em um sistema de saúde pública [Greenwood et al., 2007]. Neste sistema, as reclamações são registradas, atualizadas e consultadas através de um cliente Web. O seu projeto está estruturado com o objetivo de desacoplar diferentes partes do sistema de forma a tornar mais fácil modificar seus módulos separadamente. Portanto, a parte de interface do usuário é desacoplada das outras partes que implementam a lógica de negócio que, por sua vez, são também desacopladas do gerenciamento da base de dados do sistema. O Health Watcher tem cerca de 4 KLOC. A Figura 3.1 ilustra um diagrama de classes simplificado que apresenta as classes e interfaces principais do Health Watcher.

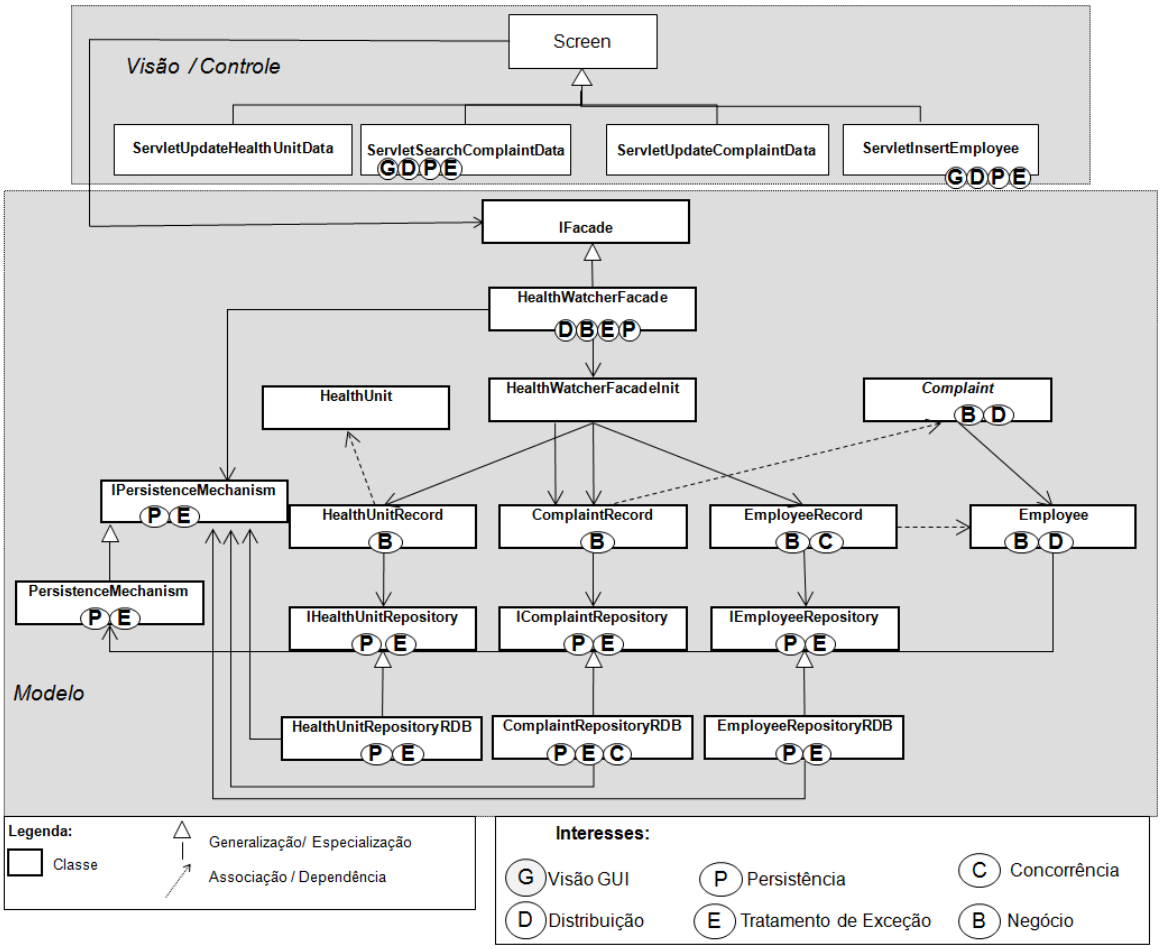

Figura 3.1. Diagrama Parcial do Projeto Health Watcher.

A Tabela 3.1 lista algumas classes do sistema agrupando-as pela sua responsabilidade. A interface do usuário é representada pelas classes *Servlet*, tais como *ServletInsertEmployee* e *ServletComplaintData*. O acesso aos serviços do Health Watcher é feito através da interface *IFacade*, que é implementada pela classe *HealthWatcherFacade*. Um dos requisitos do Health Watcher é permitir que vários clientes pudessem acessar o sistema em um mesmo tempo. As classes *HealthWatcherFacade* e *HealthWatcherFacadeInit* implementam a distribuição das requisições dos clientes. A classe *HealthWatcherFacadeInit* funciona como um portal de acesso as classes de negócio, tais como *ComplaintRecord* e *EmployeeRecord*. As classes *Record* acessam os dados utilizando as interfaces, *Repository*, as quais desacoplam a lógica de negócio do tipo de dados específico que está sendo gerenciado. As classes *RepositoryRDB* têm o propósito de implementar um repositório para a base de dados relacional, como por exemplo as classes *HealthUnitRepositoryRDB* e *EmployeeRepositoryRDB*.

Tabela 3.1. Responsabilidades das Classes do Healh Watcher

Classes	O que elas fazem
HWServlet ServletInsertEmployee ServletSearchComplaintData ServletUpdateComplaintData ServletUpdateHealthUnitData	Estas classes implementam as telas de interface com o usuário.
IFacade HealthWatcherFacade HealthWatcherFacadeInit	Estas classes provêm uma interface simples para acesso aos serviços do sistema.
HealthUnit Employee Complaint	Estas classes representam os conceitos básicos da lógica de négocio do domínio do sistema.
HealthUnitRecord EmployeeRecord ComplaintRecord	Estas classes representam um agrupamento de objetos de uma classe básica. Por exemplo, EmployeeRecord agrupa os objetos Employee.
HealthUnitRepositoryRDB EmployeeRepositoryRDB ComplaintRepositoryRDB	Estas classes contém métodos para manipulação de objetos de persistência. O código destas classes dependem de uma API específica para acessar alguma plataforma de persistência, de forma que quaisquer mudanças nesta plataforma irá impactar diretamente nestas classes.
IHealthUnitRepository IEmployeeeRepository IComplaintRepository	Estas interfaces desacoplam as classes Record das classes RepositoryRDB. Desta foram, mudanças no código de acesso a dados não tem impacto no código de negócio
PersistenceMechanism	Esta classe implementa serviços tais como conectar e desconectar uma base de dados, mecanismos de transação e controle de concorrência no banco de dados.

O Health Watcher implementa vários interesses típicos de um sistema Web. Seis destes interesses que são usados neste estudo estão anotados no diagrama da Figura 3.1. Uma breve descrição destes interesses é apresentada a seguir.

1. **Negócio.** Este interesse define os elementos e regras de negócio que atendem a estrutura, controlam ou influenciam o negócio do sistema.

2. **Concorrência.** Este interesse fornece um controle para evitar a inconsistência nas informações manipuladas pelo sistema.

3. **Distribuição.** Este interesse é responsável pela externalização dos serviços do sistema no servidor e apoiar a distribuição aos clientes.

4. **Tratamento de Exceção.** Este interesse suporta a recuperação de erros. Muitas linguagens de programação, como Java, têm suporte explícito para levantamento e tratamento de exceções. O escopo deste interesse começa com uma cláusula de marcação (*try*) e termina com uma cláusula de tratamento (*catch*).

5. **Persistência.** Este interesse provê a habilidade de reter dados entre execuções do sistema. Em outras palavras, é responsável por salvar e recuperar as informações manipuladas pelo sistema.

6. **Visão GUI**. Este interesse fornece uma interface Web para o sistema. Ele permite a interação do usuário através de elementos gráficos como ícones e telas.

3.2.2 MobileMedia

Este estudo utilizou a 7ª versão do sistema MobileMedia[1]. Este sistema é uma linha de produtos de software (LPS) [Figueiredo et al., 2008] que inclui funcionalidades, como manipulação de fotos, músicas e vídeos, em dispositivos móveis, tais como celulares e smartphones. Ele é um sistema de código livre com aproximadamente 3,5 KLOC. A Figura 3.2 mostra algumas das principais classes do MobileMedia em um diagrama UML simplificado. Este diagrama é na verdade uma versão parcial do projeto desta aplicação. No MobileMedia, as funcionalidades são acessadas através da tela e teclado do dispositivo móvel. O projeto da aplicação é estruturado de tal forma a desacoplar suas diferentes partes. Assim, a parte responsável pela interface com o usuário (telas) está desacoplada da parte que implementa os controladores de funcionalidades. Esta última também se encontra desacoplada das classes do modelo. O objetivo deste desacoplamento é tornar alterações mais fáceis e independentes em cada uma das partes da aplicação.

A Tabela 3.2 resume as principais responsabilidades associadas às classes do MobileMedia. A interação do usuário com o MobileMedia é feita por classes *Screens*, tais como *MediaListScreen* e *PhotoViewScreen* (Figura 3.2). O acesso aos serviços da aplicação é feito através de controladores que estendem uma classe abstrata chamada *AbstractController*. Um dos requisitos do MobileMedia é permitir o uso de senhas de segurança em álbuns de mídia. Portanto, os usuários podem definir senhas individuais para cada álbum da aplicação. *MediaData* funciona como uma entidade do tipo mídia

[1]http://sourceforge.net/projects/mobilemedia/

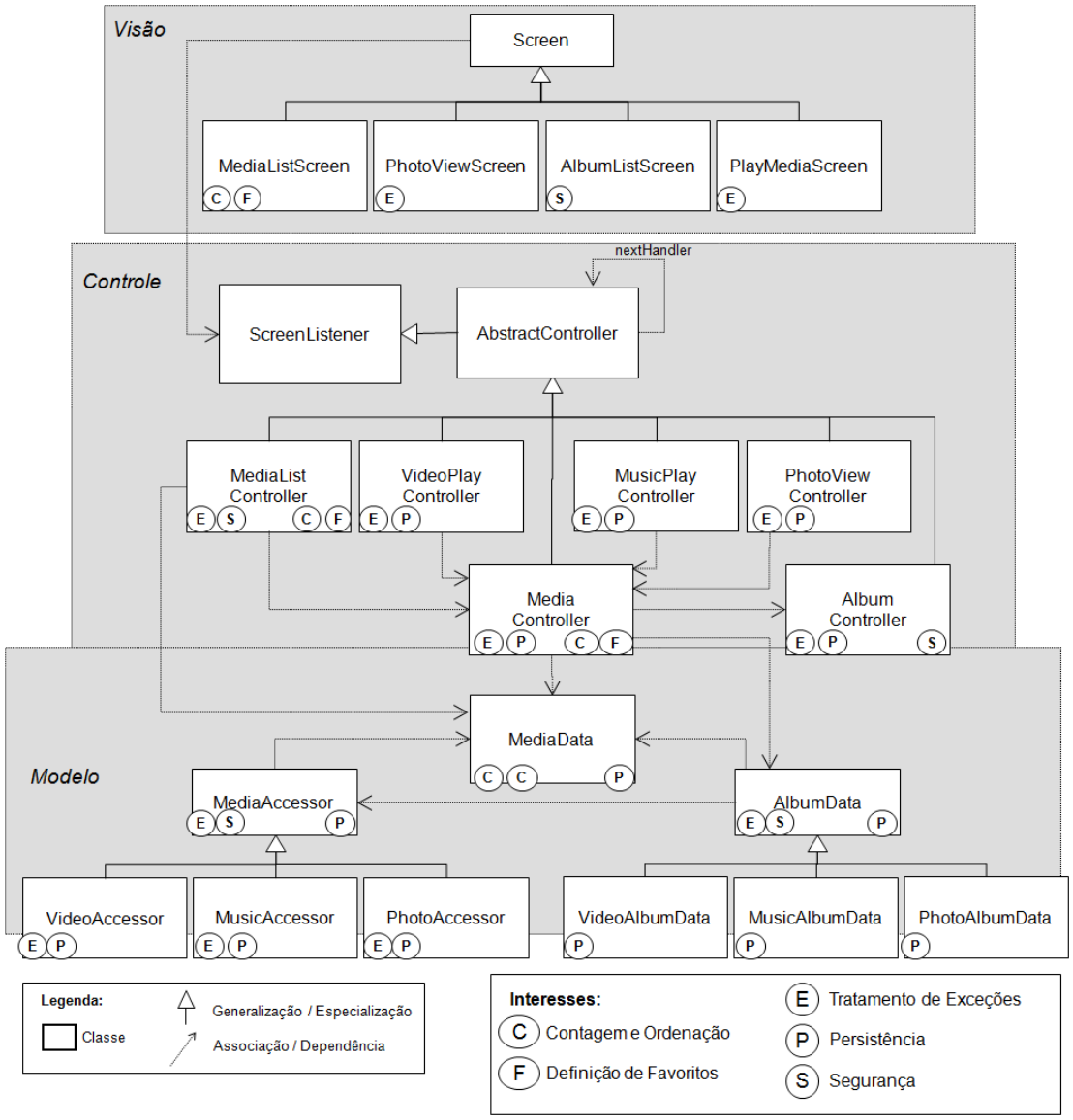

Figura 3.2. Diagrama Parcial do Projeto MobileMedia.

que pode ser tanto música quanto vídeo ou foto (existem classes específicas para os tipos de mídia que não são apresentados na Figura 3.2). As classes *Accessor* (*VideoAccesor*, *MusicAccessor* e *PhotoAccessor*) persistem as mídias diretamente no dispositivo de armazenamento (como cartões de memória). Por outro lado, as classes *AlbumData* (*VideoAlbumData*, *MusicAlbumData* e *PhotoAlbumData*) oferecem uma fachada para acessos aos dados. Assim, estas classes desacoplam a lógica de negócio dos tipos específicos de dados armazenados.

Cinco interesses refletem decisões importantes do projeto desta aplicação. Contagem e Ordenação, Definição de Favoritos, Tratamento de Exceções, Persistência e Segurança. Classes que de alguma forma implementam estes interesses no projeto do

Tabela 3.2. Responsabilidades das Classes do MobileMedia

Classes	O que elas fazem
Screen MediaListScreen PhotoViewScreen AlbumListScreen PlayMediaScreen	Estas classes implementam as telas de interface com o usuário.
AbstractController MediaListController VideoPlayController MusicPlayController PhotoViewController MediaController AlbumController	Estas classes controlam o acesso as funcionalidades da aplicação. Elas recebem os comandos associados às operações das telas e repassam às classes do modelo.
MediaData	Esta classe representa os tipos básicos de mídias (vídeo, música e foto). Em outras palavras, é a principal classe do domínio da aplicação.
AlbumData VideoAlbumData MusicAlbumData PhotoAlbumData	Estas classes representam abstrações de dados gravados em dispositivos de armazenamento. Por exemplo, as classes controladoras não sabem como os dados são armazenados, pois conhecem apenas os álbuns de mídia (abstrações).

MobileMedia estão marcadas no diagrama da Figura 3.2. Por exemplo, classes que implementam Segurança são identificados pela letra 'S' no diagrama. Estes interesses foram usados neste estudo e são descritos a seguir.

1. **Contagem e Ordenação.** Este interesse permite contar o número de vezes que uma determinada mídia foi visualizada pelo usuário. Além disso, o usuário pode ordenar a lista de mídia de um álbum pelo número de visualizações.

2. **Definição de Favoritos.** Este interesse permite que o usuário defina uma determinada mídia como favorita. Assim, ele tem a opção de visualizar somente as mídias favoritas de um álbum. Fotos, vídeos e músicas podem ser marcados como favoritos.

3. **Tratamento de Exceções.** Este interesse se caracteriza por um mecanismo para tratar ocorrências que alteram o fluxo normal de execução. Muitas linguagens de programação, como Java, têm suporte explícito para levantamento e

tratamento de exceções. O escopo deste interesse começa com uma cláusula de marcação (*try*) e termina com uma cláusula de tratamento (*catch*).

4. **Persistência.** Este interesse se refere à habilidade da aplicação em reter dados entre execuções. Em outras palavras, ela é responsável por salvar e recuperar as informações manipuladas pela aplicação. Na implementação de persistência do MobileMedia, exceções específicas de persistência trafegam por outras classes da aplicação.

5. **Segurança.** Este interesse permite que senhas sejam associadas a álbuns de mídia. Assim, o usuário somente pode efetuar algumas operações, como visualizar e apagar álbuns protegidos, se a senha for fornecida.

3.3 Lista de Referência de Anomalias em Classes

Antes de conduzir o estudo, nós realizamos uma análise sistemática no código do Health Watcher e do MobileMedia para determinar quais classes são afetadas por anomalias relevantes (Seção 2.1.2). Nós também contamos com a ajuda de dois especialistas no desenvolvimento, manutenção ou avaliação de cada sistema. Nosso objetivo foi detectar classes propícias a terem cada anomalias estudada em ambos os sistemas.

Cada especialista em um sistema foi instruído a trabalhar individualmente e a utilizar sua própria estratégia para detectar anomalias nas classes dos sistemas. Como resultado, diversas estratégias diferentes foram utilizadas. Por exemplo, um especialista focou na inspeção de código utilizando uma análise mais comum. Seguindo um caminho diferente, outro especialista utilizou um conjunto complementar de estratégias de detecção automática [Marinescu, 2004] para identificar classes candidatas a terem anomalias de código. Os conjuntos de classes - obtidos por cada especialista - como possíveis candidatas a terem anomalias de código, não eram exatamente os mesmos, embora tivessem muitas classes em comum (cerca de 80% para o Health Watcher e 75% para o MobileMedia). A fim de alcançar um consenso, foi promovida discussões entre os especialistas do mesmo sistema.

O resultado da discussão foi registrado como uma decisão conjunta. Portanto, as classes na lista de referência para cada anomalia, apresentados nas Tabelas 3.3 e 3.4, foram aprovadas por pelo menos dois especialistas e checadas posteriormente pela autora desta dissertação. Esta lista de referência é utilizada para quantificar o número de acertos e erros dos participantes conforme será explicado na Seção 3.6.1.

Tabela 3.3. Lista de Referência de Anomalias em Classes para o Health Watcher

Classes com Divergent Change		
HealthWatcherFacade	IFacade	HealthUnitRecord
ServletInsetEmployee	ServletUpdateHealthUnitData	HealthWatcherFacadeInit
ServletSearchComplaintData	ComplaintRecord	IPersistenceMechanism
ServletUpdateComplaintData	EmployeeRecord	PersistenceMechanism
Classes com Shotgun Surgery		
IHealthUnitRepository	HealthUnitRepositoryRDB	IPersistenceMechanism
IEmployeeRepository	EmployeeRepositoryRDB	PersistenceMechanism
IComplaintRepository	ComplaintRecordRDB	
Classes com God Class		
HealthWatcherFacade	HealthWatcherFacadeInit	PersistenceMechanism

Tabela 3.4. Lista de Referência de Anomalias em Classes para o MobileMedia

Classes com Divergent Change		
ImageMediaAccessor	MediaController	MediaListController
MediaAccessor		
Classes com Shotgun Surgery		
ControllerInterface	MediaAccessor	ScreenSingleton

3.4 Histórico dos Participantes

Esta seção discute o histórico dos participantes em relação às seguintes aptidões: Diagrama de Classes, Linguagem Java, Experiência de Trabalho e Medição de Software. Este estudo envolveu um conjunto de 54 participantes de duas instituições distintas: UFMG (Brasil) e Lancaster (UK), mapeados conforme o Anexo C. Os participantes da primeira instituição foram 11 jovens profissionais de TI que cursavam um curso de especialização em Engenharia de Software, 4 estudantes de pós-graduação e 12 estudantes de graduação. Os participantes da segunda instituição foram 14 estudantes de pós-graduação e 13 estudantes de graduação. Nós organizamos os participantes em grupos, de tal forma que, cada um trabalhou com um conjunto de métricas: métricas tradicionais, métricas de interesse ou métricas híbridas (conjunto de métricas que é composto tanto de métricas tradicionais como também de métricas de interesse). Os participantes foram agrupados tentando-se equilibrá-los pelo histórico de conhecimento. O estudo foi realizado utilizando os sistemas Health Watcher e MobileMedia, ambos os projetos são orientados a objetos com implementação Java.

Antes de começar a rodar o experimento, nós aplicamos um questionário de histórico (Anexo A) para termos uma prévia do conhecimento de cada participante. Embora os participantes tenham sido convidados a responder o questionário, eles não eram obrigados a responder para a realização do experimento. A Tabela 3.5 resume o conhecimento que os participantes afirmaram ter no questionário de histórico. Eles responderam as perguntas relacionadas a: (i) Diagrama de Classes, (ii) Programação em Java, (iii) Experiência de Trabalho e (iv) Métricas de software. Os participantes foram nomeados de S1 até S54 para manter a identificação anônima. Alguns participantes não responderam o questionário, os quais aparecem na última coluna (Não Responderam) da Tabela 3.5. As outras colunas mostram o conhecimento que os participantes afirmaram ter em uma habilidade particular. Há participantes que não aparecem em uma linha, pois eles disseram não ter experiência nesse tópico em particular. Por exemplo, com relação à experiência de trabalho dos participantes para detectar *Divergent Change*: os participantes S1 até S3 (e outros) não responderam o questionário; os participantes S4, S6, S9, S19, S21, S22, S23 e S24 afirmaram que eles não têm experiência de trabalho.

Tabela 3.5. Histórico dos Participantes que Detectaram Anomalias em Classes

Anomalia Divergent Change					
Métricas		**Tradicional**	**Interesse**	**Híbrida**	**Não Responderam**
Aptidão	Diagrama de Classes	S4 - S6	S9 - S11	S14 - S24	S1, S2, S3, S7, S8, S12, S13, S18
	Java	S4 - S6	S9 - S11	S14 - S24	
	Experiência de Trabalho	S5	S10,S11	S14 - S17, S20	
	Medição	S4 - S6	S9 - S11	S14 - S24	
Anomalia Shotgun Surgery					
		Tradicional	**Interesse**	**Híbrida**	**Não Responderam**
Aptidão	Diagrama de Classes	S27 - S29	S31 - S32	S34 - S37, S39 - S44	S25, S26, S30, S33, S38
	Java	S27 - S29	S31 - S32	S34 - S37, S39 - S45	
	Experiência de Trabalho	S28	S32, S32	S34 - S37, S40	
	Medição	S27 - S29	S31 - S32	S34 - S37, S39 - S44	
Anomalia God Class					
		Tradicional	**Interesse**	**Híbrida**	**Não Responderam**
Aptidão	Diagrama de Classes	S45, S46	S48 - S50	S51- S54	-
	Java	S45, S46	S48 - S50	S51- S54	
	Experiência de Trabalho	S45	S48	S51	
	Medição	S45, S46	S49 - S50	S51 -S54	

Podemos observar pela Tabela 3.5 que todos os participantes que responderam o questionário, exceto o S47, têm pelo menos conhecimento básico em Diagrama de Classes e Programação em Java. Na verdade, pedimos aos participantes para indicar seu nível de conhecimento, escolhendo uma das seguintes opções: (i) nenhum, (ii)

pouco, (iii) moderado e (iv) alta experiência (vide Anexo A). Em geral, excluindo 13 participantes que não responderam ao questionário de histórico, nós observamos que (i) cerca de 50% dos participantes têm de moderado a alto conhecimento em Diagrama de Classes e Programação em Java, (ii) 66% dos participantes têm de moderado a alto conhecimento pelo menos em um tópico, e (iii) apenas 16% dos participantes têm baixo conhecimento em todos os tópicos. Portanto, podemos concluir que, em geral, todos os participantes têm o mínimo de conhecimento básico necessário para executar as tarefas experimentais. Eles foram distribuídos de forma equilibrada entre os grupos de métricas.

Os participantes foram organizados como seguem: (i) 24 participantes detectaram *Divergent Change*, (ii) 20 participantes detectaram *Shotgun Surgery*, e (iii) 10 participantes detectaram *God Class*. Health Watcher foi usado por participantes de Lancaster para detectar todas as três anomalias de código, enquanto o MobileMedia foi usado por participantes da UFMG para detectar *Divergent Change* e *Shotgun Surgery*. Para maiores detalhes sobre a distribuição dos participantes consulte os Anexos D, E e F.

3.5 Tarefas Experimentais

O estudo foi precedido por uma sessão de treinamento de aproximadamente 30 minutos para permitir que os participantes se familiarizassem com as métricas avaliadas e com as anomalias de código. Após a sessão de treinamento, cada participante recebeu um documento contendo: (i) uma breve descrição e uma visão parcial do projeto do sistema - por meio de um diagrama de classe e (ii) uma descrição dos interesses envolvidos no respectivo sistema analisado. O documento também descreve as etapas e as diretrizes que os participantes deveriam seguir as perguntas que eles deveriam responder e as informações que deveriam registrar. O protocolo do experimento está incluído no Anexo B.

Além disso, nós fornecemos aos participantes os resultados das métricas no respectivo sistema em análise. A fim de identificar as classes com anomalias, pedimos aos participantes para raciocinar sobre as métricas e identificar quais delas (sozinha ou combinada com outras métricas) fornecem indicadores relevantes para uma determinada anomalia em classe. Nós também pedimos a cada participante para informar e explicar quais métricas foram úteis e quais não foram úteis para detectar a anomalia.

Cada grupo de participantes (grupo tradicional, grupo de interesse, ou grupo híbrido)[2] teve acesso somente aos resultados das métricas que foram atribuídos a eles. Os participantes não tiveram acesso ao código do sistema. Pedimos a cada grupo de participantes para realizar as seguintes etapas: (i) ler a descrição do sistema analisado, (ii) identificar as classes com maior probabilidade de terem a anomalia em análise, e (iii) listar as métricas úteis e não-úteis para aquela anomalia. Cada grupo de participantes registrou o seu tempo ao concluir as tarefas experimentais.

3.6 Resultados

Esta seção apresenta os resultados dos nossos experimentos. A Seção 3.6.1 introduz duas métricas, Cobertura e Precisão, usadas para suportar a análise dos resultados. As Seções 3.6.2 até 3.6.4 reportam os resultados por anomalia.

3.6.1 Quantificando Cobertura e Precisão

Três métricas, Verdadeiro Positivo (VP), Falso Positivo (FP) e Falso Negativo (FN), foram coletadas a partir de dados fornecidos pelos participantes. Verdadeiro Positivo e Falso Positivo quantificam o número de anomalias de código identificadas corretamente ou erroneamente por um participante. Falso Negativo, por outro lado, quantifica o número de anomalias de código que o participante não conseguiu detectar em relação à lista de referência. Com base nessas métricas, quantificamos a Cobertura e a Precisão para apoiar a nossa análise. *Cobertura (C)* mede a fração de classes relevantes listadas pelo participante. As classes relevantes são classes que aparecem na lista de referência (VP + FN). *Precisão (P)* quantifica as classes corretamente listadas como anomalia (VP), pelo total de classes que um participante listou (VP + FP).

$$C = \frac{VP}{VP+FN} \tag{3.1}$$

$$P = \frac{VP}{VP+FP} \tag{3.2}$$

[2]Para simplificar a leitura do texto, deste ponto em diante usamos: (i) *grupo tradicional* significando grupo de participantes que analisaram métricas tradicionais; (ii) *grupo de interesse* significando grupo de participantes que analisaram métricas de interesse; (iii) *grupo híbrido* significando grupo de participantes que analisaram tanto métricas tradicionais quanto métricas de interesse

Nós focamos nossa discussão principalmente na Cobertura, pois ela é a medida de abrangência. Ou seja, Cobertura alta significa que o participante foi capaz de identificar mais anomalias no sistema. Alta Precisão, por outro lado, significa que um participante indicou mais anomalias relevantes (VP) do que irrelevantes (FP). Para detecção de anomalias, o alto número de anomalias não listadas (falsos negativos) é pior que o alto número de anomalias listadas incorretamente (falsos positivos), pois a inspeção manual, que é inevitável, tende a revelar os falsos positivos.

3.6.2 Detecção de *Divergent Change*

A Tabela 3.6 apresenta os resultados para a identificação de *Divergent Change*. As linhas nesta tabela apresentam 3 partes dos dados: Cobertura (C), Precisão (P) e o Tempo (T) em minutos utilizados pelos participantes para completar suas tarefas. No total, 24 participantes tiveram que identificar a anomalia *Divergent Change* no sistema. Observamos a partir dos dados desta tabela que os participantes do grupo de interesse e do grupo híbrido obtiveram melhores resultados que o grupo tradicional. A média do grupo de interesse foi 62% de Cobertura. Dois dos cinco participantes deste grupo identificaram todas as anomalias (100% de Cobertura). Por outro lado, o melhor valor alcançado por um participante que utilizou apenas métricas tradicionais foi 33% de Cobertura. Os resultados dos participantes no grupo híbrido variaram entre 0% a 8% de Cobertura (S13 e S19) até a Cobertura de 100% (S16). Estes resultados revelam que, mesmo quando analisadas isoladamente, métricas de interesse são meios eficazes para detecção de *Divergent Change*.

A Tabela 3.6 também sugere que a detecção de *Divergent Change* com métricas tradicionais parece mais difícil comparada às métricas de interesse. A explicação pode ser que esta anomalia está intimamente relacionada com a baixa separação de interesses. Isto é, muitas vezes *Divergent Change* ocorre quando vários interesses estão entrelaçados em um módulo [Carneiro et al., 2010]. Portanto, este módulo é susceptível de ser modificado por razões distintas. Focando nos participantes que usaram as métricas de interesse (grupo de interesse e híbrido), é interessante notar que 10 dos 18 participantes em ambos os grupos tiveram pelo menos 50% de Cobertura. A Precisão também foi maior que 50% para 8 participantes.

Tabela 3.6. Resultado para Divergent Change

Métricas	**Tradicional**						**Interesse**					**Híbrida**												
Grupos	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24
C(%)	17	17	17	33	25	25	100	100	33	25	50	75	8	25	50	100	25	50	0	50	25	50	25	50
P(%)	67	50	40	50	17	25	63	100	100	25	29	100	50	75	25	67	33	40	0	67	17	40	17	50
T(min)	15	15	40	38	41	36	26	29	29	15	33	40	31	23	36	27	39	24	11	18	19	13	13	12

Tabela 3.7. Resultado do Shotgun Surgery

Métricas	**Tradicional**					**Interesse**			**Híbrida**										
Grupos	S25	S26	S27	S28	S29	S30	S31	S32	S33	S34	S35	S36	S37	S38	S39	S40	S41	S42	S43	S44
C(%)	13	13	0	67	33	75	25	33	13	50	67	33	33	33	0	0	33	0	0	0
P(%)	25	33	0	25	25	35	40	25	25	80	6	33	25	33	0	0	20	0	0	0
T(min)	6	10	27	12	14	13	28	14	35	14	19	15	4	10	14	9	21	3	7	5

Tabela 3.8. Resultado do God Class

Métricas	**Tradicional**			**Interesse**			**Híbrida**			
Grupos	S45	S46	S47	S48	S49	S50	S51	S52	S53	S54
C(%)	33	33	67	100	67	100	33	100	100	100
P(%)	33	33	67	75	100	75	50	100	60	75
T(min)	18	25	27	37	66	43	22	53	51	35

3.6.3 Detecção de *Shotgun Surgery*

A Tabela 3.7 apresenta os resultados para detecção da anomalia *Shotgun Surgery*. Esta tabela segue a mesma estrutura da Tabela 3.6. Os dados da Tabela 3.7 mostraram que nenhum grupo de participantes se destacou com bons resultados. De fato, apenas um participante em cada grupo alcançou mais de 60% de Cobertura: S28 obteve 67% na análise de métricas tradicionais, S30 obteve 75% no grupo de interesse e S35 obteve 67% de Cobertura na análise das métricas híbridas.

O grupo de interesse teve uma performance um pouco melhor que os participantes dos outros grupos. Este grupo teve 44% de Cobertura, em média. No entanto, as baixas taxas de acertos para quase todos os participantes sugere que as métricas utilizadas neste estudo não podem indicar precisamente classes com *Shotgun Surgery*.

Além de uma Cobertura baixa, quase todos os participantes (exceção do S34) também tiveram baixas taxas de Precisão. De fato, mais da metade das classes que os participantes indicaram como *Shotgun Surgery* eram incorretas de acordo com a lista de referências, independentemente das métricas utilizadas. Curiosamente, os participantes que foram alocados para detectar *Shotgun Surgery*, em geral, gastaram menos tempo em suas tarefas do que os participantes designados a detectar outras anomalias. Ou seja, embora os participantes não tiveram muito sucesso na detecção do *Shotgun Surgery*, eles não levaram muito tempo para concluir suas tarefas. Esse resultado pode indicar que, se os desenvolvedores não têm meios adequados para detectar uma determinada anomalia, eles desistem de suas tarefas rapidamente.

3.6.4 Detecção de *God Class*

A Tabela 3.8 apresenta os resultados do *God Class*. Apenas 10 participantes foram alocados para a detecção desta anomalia de classe. Os dados apresentados nesta tabela sugerem que as métricas tradicionais, quando usadas isoladamente não oferecem meios adequados para detectar *God Class*. Dois participantes (S45 e S46) do grupo tradicional alcançaram 33% de Cobertura e Precisão. Esse desempenho é muito pior comparado com os resultados alcançados pelos grupos de interesse e híbridos.

Dois dos três participantes no grupo de interesse e três dos quatro participantes no grupo híbrido obtiveram 100% de Cobertura. A exceção são os participantes S49 e S51 que obtiveram 67% e 33% de Cobertura, respectivamente. Além disso, o participante S52 do grupo híbrido obteve valor total tanto de Cobertura quanto de Precisão. Portanto, uma análise conjunta das métricas de interesse e das tradicionais parece ter sucesso na detecção desta anomalia.

3.7 Análise e Discussão

Esta seção pretende responder a questão de pesquisa geral e as subquestões específicas definidas na Seção 3.1. Nós focamos nos resultados mais interessantes, mas os dados completos do estudo podem ser encontrados nos Anexos D, E e F.

3.7.1 Comparação das Métricas Tradicionais e de Interesse

O principal objetivo deste primeiro estudo é avaliar a eficácia das métricas de interesse para detectar anomalias em classes. Portanto, esta seção tem como objetivo responder à seguinte questão de pesquisa especificada na Seção 3.1.

RQ1. Qual a acurácia das métricas de interesse comparadas com as métricas tradicionais na detecção de anomalias em classes?

Baseando-se nos dados e discussões da Seção 3.6, foi observado que o uso de métricas de interesse foi consistemente melhor quando: (i) utilizada isoladamente no caso do *Divergent Change* e (ii) utilizada em conjunto com as métricas tradicionais para o caso do *God Class*. As métricas de interesse selecionadas não apresentaram bons resultados para *Shotgun Surgery*. Na verdade, a superioridade da utilização sozinha ou combinada das métricas de interesse variou de acordo com o tipo de anomalias. Estes resultados indicam que a acurácia do conjunto de métricas é largamente dependente da adequação de cada métrica para quantificar uma propriedade explicitamente mencionada na definição de anomalias. Por exemplo, *God Class* é caracterizada pela "grande quantidade de métodos na classe" [Fowler, 1999] e "a realização de múltiplas responsabilidades" [Riel, 1996]. Enquanto a primeira propriedade é diretamente quantificada por métricas tradicionais, a segunda é mais bem capturada por métricas de interesse. Isso provavelmente explica o porquê das métricas híbridas se saíram melhores para a detecção de *God Class*.

3.7.2 Histórico dos Participantes

Esta seção analisa como o histórico dos participantes pode impactar nos resultados deste estudo. Em outras palavras, nós pretendemos responder a seguinte questão de pesquisa levantada na Seção 3.1.

RQ2. A experiência dos desenvolvedores impacta na acurácia para detectar as anomalias em classes?

Como discutido na Seção 3.4, todos os participantes possuem o mínimo de conhecimento básico nos tópicos considerados relevantes para o desenvolvimento de software e participação neste estudo, denominados: Diagrama de Classes UML e Programação Java. Por isso, decidimos fazer esta análise exclusivamente em relação à experiência de trabalho dos participantes. A experiência de trabalho variou muito entre os participantes, desde nenhuma experiência em desenvolvimento de software para mais de 3 anos de trabalho na indústria de desenvolvimento de software (Anexos D, E e F). Nesta análise, foram excluídos os participantes que não responderam ao questionário de histórico.

A fim de simplificar a análise, nós dividimos os participantes em duas categorias de acordo com a sua experiência de trabalho: (i) *alguma experiência* para identificar os participantes que trabalharam por pelo menos 6 meses em indústria de desenvolvimento de software e (ii) *sem experiência* indica os participantes que nunca trabalharam ou trabalharam por menos de 6 meses. Em seguida, calculamos a média de Cobertura e Precisão obtida por participantes de cada categoria, como apresentada na Figura 3.3.

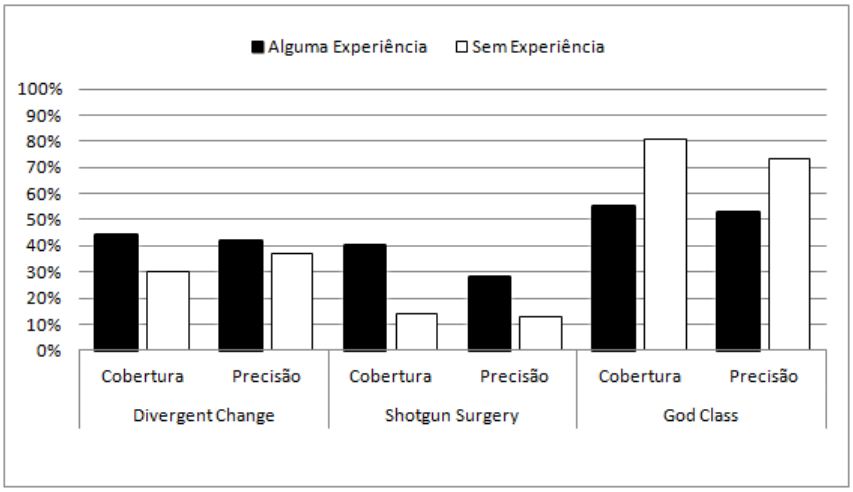

Figura 3.3. Experiência de Trabalho dos Participantes.

A Figura 3.3 mostra os valores médios de Cobertura e de Precisão agrupados pelas duas categorias de experiência de trabalho discutidas acima. Foram considerados os dados de ambos os sistemas, Health Watcher e MobileMedia, organizados pelas três anomalias: *Divergent Change*, *Shotgun Surgery* e *God Class*. Dados para o *Divergent Change* e *Shotgun Surgery* suportam o senso comum de que os desenvolvedores mais experientes são capazes de detectar mais anomalias. Isto é, os valores médios de ambos, para Cobertura e Precisão, foram mais elevados para os participantes mais experientes.

É interessante notar que, no entanto, os resultados para *God Class* (Figura 3.3) não coincidem com o senso comum. Ou seja, participantes sem experiência de trabalho

em desenvolvimento de software se saíram melhor que os participantes com alguma experiência. Uma possível explicação para este resultado é que *God Class* é a anomalia mais simples de entender, em comparação com as outras anomalias de código. Por isso, não se exige experiência avançada em desenvolvimento de software. Além disso, apenas três participantes (S45, S48 e S51) que detectaram *God Class* relataram ter experiência de trabalho. Outros 7 participantes alegaram que eles não têm experiência de trabalho. Portanto, os resultados desta análise podem ter sido impactados por outra variável não controlada neste estudo. Assim, mais estudos são necessários para permiti-nos tirar conclusões sobre como a experiência de trabalho pode ter impactado na detecção do *God Class*.

3.7.3 Muitas Métricas, Análise mais Demorada?

Esta seção faz uma análise do tempo gasto pelos participantes para detectar anomalias em classe. Para verificar a questão a seguir, nós analisamos o tempo que foi gasto para detectar as anomalias em classe e a respectiva correlação com os valores de Cobertura. O objetivo é responder à seguinte questão de pesquisa.

RQ3. Um conjunto composto por muitas métricas pode fazer com que a tarefa de identificação das anomalias de classe seja mais demorada?

Divergent Change. A Figura 3.4 mostra os dados de Cobertura (eixo x) e o tempo gasto (eixo y) pelos participantes para detectar a anomalia *Divergent Change*. Observamos nesta figura que as métricas tradicionais tiveram o pior resultado para detectar esta anomalia. Os participantes que usaram estas métricas geralmente gastaram longo tempo (exceto um) e obtiveram baixos valores de Cobertura. Em média, os participantes do grupo tradicional gastaram 31 minutos. Em comparação, participantes que usaram as métricas de interesse gastaram em média 26 minutos e os participantes que usaram métricas híbridas gastaram 24 minutos em média. A Figura 3.4 também destaca que 4 dos 5 participantes do grupo de interesse gastaram quase o mesmo tempo (cerca de 30 minutos) para detectar *Divergent Change*. Por outro lado, o tempo gasto pelos participantes do grupo híbrido variou bastante. Este resultado pode ser explicado pelo elevado número de métricas disponíveis para este grupo, as quais os permitiram usar várias estratégias de detecção diferentes.

Shotgun Surgery. A Figura 3.5 mostra os resultados da eficiência de tempo e Cobertura para o *Shotgun Surgery*. Verificamos que os participantes do grupo de interesse gastaram mais tempo (18 minutos em média) que os outros participantes para executarem suas tarefas. Os grupos tradicional e híbrido obtiveram resultados

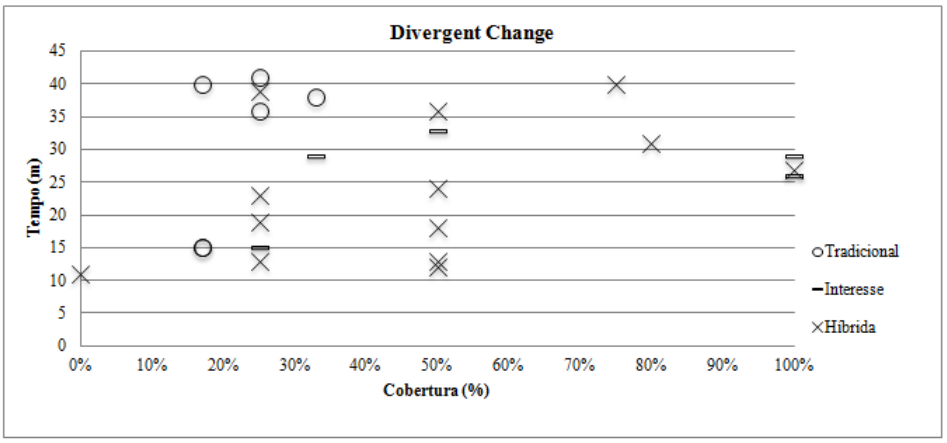

Figura 3.4. Tempo Eficiente para Divergente Change.

semelhantes, em torno de 13 e 14 minutos em média. O que é interessante, é que seis participantes no grupo híbrido desempenharam suas tarefas muito rapidamente; gastaram menos 10 minutos para a detecção de *Shotgun Surgery*. Infelizmente, a conclusão mais rápida da tarefa não implica na obtenção de bons resultados. Na realidade, os participantes que concluíram suas tarefas mais rapidamente tendem a pontuar pior em termos de Cobertura como pode ser observado na Figura 3.5.

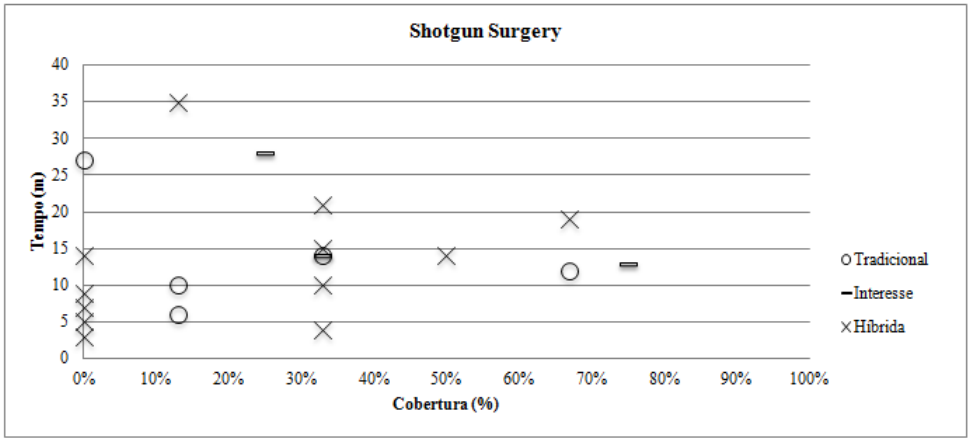

Figura 3.5. Tempo Eficiente para Shotgun Surgery.

God Class. A Figura 3.6 apresenta os dados de Cobertura e tempo gasto pelos participantes para detectar o *God Class*. Podemos observar nesta figura que os participantes do grupo de métricas de interesse gastaram mais tempo para detectar *God Class* que os outros participantes. Em média, os participantes do grupo de interesse gastaram 49 minutos, contra 40 minutos no grupo híbrido e 23 no tradicional. Semelhante ao *Shotgun Surgery*, a Figura 3.6 deixa claro que para o *God Class* quanto

maior o tempo de análise, melhor são os resultados alcançados em termos de Cobertura. No entanto, observou-se que a detecção de *God Class* exige uma análise cuidadosa de muitas métricas, como indicada pela superioridade do grupo híbrido em relação a Cobertura.

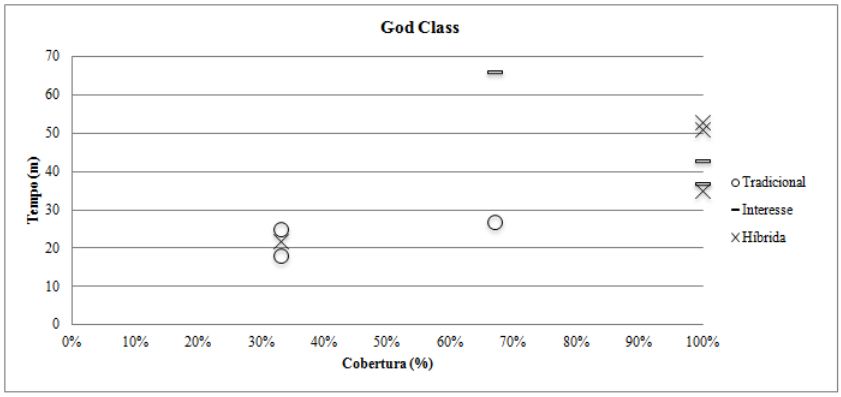

Figura 3.6. Tempo Eficiente para God Class.

Em resumo, o senso comum é que quando os participantes utilizam um maior conjunto de métricas, como as métricas híbridas, propensas há gastar mais tempo para detectar uma anomalia de classe. Este não foi sempre o caso para as anomalias analisadas. Alguém pode argumentar que as métricas de interesse teriam o tempo mais eficiente por que: (i) o conjunto inclui apenas quatro métricas e (ii) as suas definições capturam as propriedades dos interesses que podem estar melhor associadas com a definição da anomalia. No entanto, depois de uma análise geral dos dados acima mencionados, foi observado que as métricas híbridas apresentaram resultados um pouco melhor em relação ao tempo gasto para duas anomalias: *Divergent Change* e *Shotgun Surgery*. O grupo tradicional gastou menos tempo para a detecção do *God Class*. A razão para o tempo bom e eficiência das métricas tradicionais para o *God Class* pode ser porque os participantes estão mais familiarizados com elas. Além disso, a definição mais simples de *God Class* facilita a relação com métricas tradicionais.

3.7.4 Métricas Específicas

Esta seção está voltada para a questão de pesquisa a seguir enunciada na Seção 3.1. Como explicado na Seção 3.5, os participantes relataram as métricas consideradas úteis para cada anomalia estudada. Com base em suas respostas, analisamos nesta seção as métricas que foram reportadas como úteis por, pelo menos, três participantes.

RQ4. Existe uma métrica específica que detecta com acurácia cada anomalia em classe?

A Tabela 3.9 mostra as métricas que pelo menos três participantes relataram ter utilizado para identificar o *Divergent Change*. Neste caso, a nossa análise também se restringiu as métricas utilizadas por participantes que atingiram Cobertura maior que 30%. Ambas as métricas: Número de Interesse por Componente (NCC) e Falta de Coesão em Métodos (LCOM), foram consideradas úteis para detectar *Divergent Change* por 11 participantes. A última linha da Tabela 3.9 mostra que os participantes que consideram essas métricas úteis alcançaram 60% e 34% de Cobertura em média, respectivamente. Além disso, a métrica Espalhamento de Interesse em Componentes (CDC) foi considerada útil por três participantes.

É interessante observar que os participantes que consideraram as métricas de interesse NCC e CDC úteis para detectar *Divergent Change* obtiveram os melhores resultados em termos de Cobertura. Em particular, NCC parece ser a métrica mais eficaz (dentre as analisadas) para detectar *Divergent Change*. Por exemplo, S7, S8, S12 e S16 usaram NCC - sozinha ou combinada com outras métricas - e obtiveram 100% de Cobertura. Observou-se também que os participantes que indicaram NCC como não sendo útil conseguiram menos de 25% de Cobertura, como é o caso de S13 e S19 (vide Anexo D).

Tabela 3.9. Métricas Consideradas Úteis para o Divergent Change

Métricas	**NCC**	**LCOM**	**WMC**	**CDC**	**LOC**
Participantes que usaram estas métricas	S7, S8, S9, S11 S12, S14, S15 S16, S20, S23 S24	S1, S2, S4, S6 S12, S13, S14 S15, S17, S22 S24	S2, S4, S6, S14 S18	S8, S10, S23	S2, S17, S20
Média Cobertura (%)	60	34	30	50	31

A Tabela 3.10 apresenta as métricas consideradas úteis por pelo menos três participantes para detectar *Shotgun Surgery*. A métrica Acoplamento entre Objetos (CBO) foi considerada útil por 11 participantes. No entanto, esses participantes alcançaram apenas 16% de Cobertura em média. Por outro lado, cinco participantes indicaram a métrica Espalhamento de Interesse em Componentes (CDC) como sendo útil e obtiveram 23% de Cobertura em média. Além disso, a métrica Número de Interesse por Componente (NCC) foi considerada útil por 4 participantes que atingiram 42% de Cobertura em média.

Se considerarmos a Cobertura atingida pelos participantes, as métricas as métricas de interesse CDC e NCC parecem ser mais úteis que a métrica tradicional CBO para identificar o *Shotgun Surgery*. No entanto, uma análise combinada das Tabelas

3.7 e 3.10 não nos permitiu concluir que essas métricas (e nenhuma outra) sejam realmente boas para detectar *Shotgun Surgery* devido a baixa taxa de Cobertura dos participantes.

Tabela 3.10. Métricas Consideradas Úteis para o Shotgun Surgery

Métricas	**CBO**	**CDC**	**NCC**
Participantes que usaram estas métricas	S25, S26, S27 S28, S29, S37, S39, S40, S42 S43, S44	S30, S31, S33, S40, S43	S32, S35, S36, S37
Média Cobertura (%)	16	23	42

A Tabela 3.11 mostra para o *God Class*, as métricas consideradas úteis por pelo menos três participantes e com média de Cobertura superior a 60%. As métricas Acoplamento entre Objetos (CBO) e a Falta de Coesão em Métodos (LCOM) foram consideradas úteis para detectar *God Class* por quatro participantes. Os participantes que usaram estas métricas alcançaram cerca de 67% de Cobertura, em média. Por outro lado, outras três métricas também foram consideradas úteis e alcançaram valores acima de 85% de Cobertura. São elas: Métodos Ponderados por Classe (WMC), Linhas de Código (LOC) e Espalhamento de Interesse em Linhas de Código (CDLOC). Estes resultados sugerem que as métricas de tamanho, tais como a LOC e a WMC, e a métrica de interesse CDLOC são boas indicadoras de *God Class*.

Tabela 3.11. Métricas Consideradas Úteis para o God Class

Métricas	**CBO**	**LCOM**	**WMC**	**LOC**	**CDLOC**	**CDO**
Participantes que usaram estas métricas	S46, S47, S51, S54	S45, S51, S53, S54	S47, S52, S53	S52, S53, S54	S48, S49, S53	S49, S50, S51
Média Cobertura (%)	67	67	89	100	89	66

3.7.5 Combinações de Métricas

Nesta seção, analisamos as possíveis combinações de métricas que podem ser úteis para detectar anomalias específicas e responder a questão de pesquisa abaixo. Estas combinações serão exploradas no Capítulo 5 para a elaboração de regras heurísticas. Para identificarmos boas combinações, contamos com a análise de métricas que foram úteis para detectar cada anomalia de classe discutidas na Seção 3.7.4. Para determinar quais métricas foram utilizadas em conjunto para detectar anomalias de código discutida na

Seção 3.7.4, foi realizada a análise dos participantes que usaram as mesmas métricas e tiveram alto percentual em termos de Cobertura.

RQ5. Existe uma combinação de métricas que detecta com acurácia anomalias em classes?

Divergent Change. Nesta análise, só consideramos os participantes que atingiram mais de 30% de Cobertura. Por exemplo, observamos que as métricas NCC e a LCOM foram utilizadas em conjunto pelos participantes S12 e S15. Estes participantes alcançaram 75% e 50% de Cobertura, respectivamente. Curiosamente, enquanto o S12 teve 100% de Precisão, S15 teve apenas 25%. A razão pode ser que, o participante S15 usou um valor limite para as métricas mais flexíveis e, assim, obteve mais classes candidatas a *Divergent Change* que são na verdade os falsos positivos. Portanto, S12 teve uma taxa de Precisão mais elevada.

Shotgun Surgery. Como a maioria dos participantes apresentou um fraco desempenho para detectar *Shotgun Surgery*, consideramos nesta análise todos os participantes que obtiveram pelo menos 16% de Cobertura. Observou-se que as métricas individuais, como NCC, CDC e CBO, parecem ser um pouco melhor do que as outras (Seção 3.7.4). Uma combinação dessas métricas, por exemplo, NCC e CBO, foram utilizadas em conjunto pelo participante S37 que atingiu 33% de Cobertura. Na verdade, todos os participantes que usaram a métrica NCC sozinha ou em combinação com outras métricas, pontuaram mais do que 30% de Cobertura. Este é o caso dos participantes S32 (33%), S35 (76%), S36 (33%) e S37 (33%).

God Class. Na análise desta anomalia, nós consideramos os participantes que atingiram mais de 66% de Cobertura. Observamos alguns casos de métricas que foram usadas juntas. Por exemplo, a combinação CBO com LCOM parece ser a melhor combinação de métricas. Além disso, a combinação WMC com LOC foi usada por S47 e S53 e se mostrou eficiente para ambos participantes, que tiveram mais 67% de Cobertura. Outro caso foi a combinação CDO com CDLOC utilizada pelo participante S49. Este participante usou ambas as métricas e obteve um resultado de 67% de Cobertura. Porém, os participantes S48 e S50 utilizaram estas métricas (CDLOC e CDO) individualmente e mesmo assim tiveram bons resultados - 100% de Cobertura.

3.7.6 Outras Lições Aprendidas

Esta seção discute algumas lições adicionais aprendidas pela análise dos resultados deste estudo. Primeiro, ela faz uma análise dos dados por anomalias e em seguida, discute os resultados com base nos sistemas utilizados.

Divergent Change. Como argumentado na Seção 3.6.2, as métricas de interesse são significativamente melhores para detectar *Divergent Change*. No entanto, uma análise individual por participante não indica que as métricas de interesse são sempre consistentemente melhor que as métricas híbridas. Por exemplo, o participante S10 do grupo de interesse teve a pior performance em termos de Cobertura (25%) e Precisão (25%). No entanto, a superioridade global dos participantes na análise das métricas de interesse apenas se torna mais aparente quando levamos em conta a média de Cobertura discutido nas Seções 3.7.1 e 3.7.4. Além disso, observou-se que os dois melhores participantes do grupo híbrido (S12 e S16) informaram que eles usaram a métrica de interesse NCC. Outra observação interessante é que os melhores participantes do grupo de interesse e do grupo híbrido foram os que consistentemente usaram pelo menos uma ou duas métricas de coesão para captar as diferentes dimensões da coesão: ou uma dimensão sintática de coesão através da LCOM ou uma dimensão semântica de coesão via NCC [Silva et al., 2012]. Portanto, podemos concluir que a coesão é um bom indicador de *Divergent Change*.

Shotgun Surgery. A Seção 3.6.3 discutiu a dificuldade de detectar classes com *Shotgun Surgery* com as métricas analisadas. Esta dificuldade está provavelmente relacionada com os "sintomas globais" associados com esta anomalia, em oposição ao "sintoma local" do *Divergent Change*. *Shotgun Surgery* está relacionado com as manifestações de mudanças em vários módulos, que são provocadas por alterações em um único módulo. Por outro lado, a análise de *Divergent Change* é focada principalmente em um único módulo. Na verdade, a coesão, uma propriedade local de um módulo, foi a propriedade analisada por participantes na detecção de *Divergent Change*. Por outro lado, os participantes que detectaram *Shotgun Surgery* foram principalmente direcionados por medidas que quantificam propriedades globais, tais como o espalhamento de interesse (CDC e CDO) e acoplamento (CBO).

God Class. O uso de ambas as métricas híbridas ou de interesse apresentaram uma alta Precisão para a detecção do *God Class*. A média da Cobertura foi superior a 80%, em ambos os casos (Seção 3.6.4). Além disso, a diferença média entre o grupo híbrido e de interesse foi de apenas 6% de Cobertura, em favor das métricas de interesse. Foi realizada uma análise individual e mais detalhada dos participantes que atingiram a maior Precisão com o uso das métricas híbridas. Durante esta análise, observou-se que os melhores desempenhos neste caso foram obtidos por participantes (ex: S52, S53 e S54) que usaram principalmente as métricas tradicionais. Por exemplo, S52 no grupo híbrido atingiu 100% de Cobertura e 100% de Precisão e reportou ter usado LOC, NOM, NOA e WMC. Por outro lado, nenhum dos participantes que utilizaram apenas as métricas tradicionais obteve melhor desempenho que os do grupo híbrido e

de interesse. Este resultado sugere que as métricas tradicionais podem ser indicadoras úteis de classes com *God Class* desde que as métricas de interesse complemente-as.

Além disso, uma análise global de todos os resultados revela que *God Class* é muito mais fácil de ser detectado que as outras duas anomalias de classe. Nossas observações dos resultados revelam que a natureza evolutiva nas definições do *Divergent Change* e do *Shotgun Surgery* faz as ocorrências destas anomalias mais difíceis de serem detectadas. As definições destas anomalias referem-se ao histórico de mudança dos módulos individuais ou múltiplos, o que não ocorre na definição do *God Class*. Portanto, ao detectar *Divergent Change* e *Shotgun Surgery*, os desenvolvedores precisam fazer alguma previsão sobre a probabilidade de certas propriedades dos módulos e interesses para induzirem mudanças no futuro. Os resultados apresentados anteriormente para o *Divergent Change* e *Shotgun Surgery* indicam que o uso de métricas de interesse foi mais útil para apoiar as análises de predição de mudanças.

Sistemas Utilizados. Esta parte analisa se as diferentes características dos sistemas impactaram sobre o desempenho dos participantes. Nós restringimos nossa análise a duas anomalias, *Divergent Change* e *Shotgun Surgery*, pois *God Class* não foi analisado no MobileMedia (Seção 3.2). A Figura 3.7 apresenta os valores médios de Cobertura e de Precisão para os 44 participantes que identificaram *Divergent Change* e *Shotgun Surgery*. Esta figura mostra que os participantes que detectaram as anomalias no Health Watcher sempre obtiveram melhores resultados que os participantes que trabalharam com o MobileMedia. No entanto, a diferença entre os dois sistemas, em termos de Cobertura não são relevantes. Ou seja, a diferença média em termos de Cobertura é inferior a 4% para o *Divergent Change* e de 1% para o *Shotgun Surgery*. Isto sugere que as taxas de Cobertura na detecção destas anomalias não sofrem influência do sistema analisado.

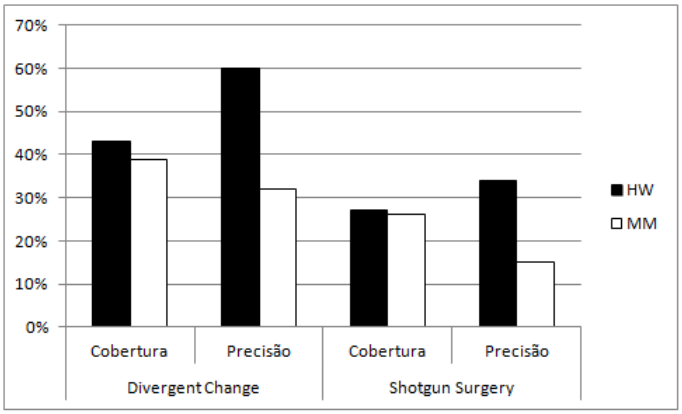

Figura 3.7. Cobertura e Precisão por Sistemas.

3.8 Considerações Finais

Avaliar a manutenibilidade do software é largamente dependente da disponibilidade de métricas que detectem com acurácia as anomalias de código. O estudo apresentado neste capítulo analisou a eficácia das métricas de interesse para detectar anomalias em classes. Os resultados obtidos revelaram que as métricas de interesse são claramente úteis para detectar a anomalia *Divergent Change*.

Investigamos também quais métricas específicas são mais adequadas para detectar cada uma das três anomalias de código analisadas. Em geral, os resultados indicaram que a acurácia de cada conjunto de métricas é largamente dependente da adequação de cada métrica para quantificar uma propriedade explicitamente mencionada na definição da anomalia. Além disso, observou-se que a métrica, Número de Interesse por Componente (NCC), foi eficiente para detectar o *Divergent Change*, mesmo quando usada sozinha. Esta métrica parece quantificar a dimensão de coesão do módulo que não é capturada por outras métricas [Silva et al., 2012].

Em resumo, as métricas obtidas como úteis neste estudo foram: (i) NCC e LCOM para o *Divergent Change*; (ii) CBO, CDC, NCC para o *Shotgun Surgery*; e (iii) CBO, LCOM, WMC e LOC para o *God Class*. As combinações de métricas úteis obtidas neste mesmo estudo foram:(i) NCC com LCOM para o *Divergent Change*, (ii) CBO com CDC e NCC para o *Shotgun Surgery* e (iii) CBO com LCOM ou WMC com LOC para o *God Class*. Este estudo representa um primeiro passo para a avaliação das métricas de interesse para detectar anomalias de código. Primeiro, realizamos um estudo com as anomalias em classes e no próximo capítulo apresentamos um estudo com as anomalias em métodos com o mesmo propósito do primeiro estudo.

Capítulo 4

Detecção de Anomalias em Métodos

Métricas são utilizadas para avaliar a qualidade e manutenibilidade do software apoiando a identificação de anomalias em métodos. Apesar das métricas de interesse já terem sido utilizadas em estudos experimentais [Conejero et al., 2012; Figueiredo et al., 2008; Greenwood et al., 2007; Eaddy et al., 2008; Figueiredo et al., 2012], ainda falta conhecimento empírico quanto à sua eficácia na identificação destas anomalias. Este capítulo apresenta um estudo que avalia a utilidade de métricas de interesse para detecção de anomalias em métodos. Nosso estudo utilizou as métricas tradicionais como base de estudo. Portanto, nós realizamos uma análise comparativa entre as métricas tradicionais e de interesse para identificar se esta última apoia a detecção de duas específicas anomalias: *Feature Envy* e *God Method*.

4.1 Questões de Pesquisa

O objetivo deste estudo é verificar se as métricas de interesse são adequadas para detectar anomalias em métodos. Portanto, uma questão geral de pesquisa que visa averiguar se isto é verdade é definida a seguir.

RQ. As métricas de interesse apoiam a detecção de anomalias em métodos?

Realizamos uma investigação com participantes para analisar a utilidade das métricas de interesse, a fim de responder à questão de pesquisa RQ. Questões específicas foram derivadas a partir da RQ descrita anteriormente:

- *RQ1. Qual a acurácia das métricas de interesse comparadas com as métricas tradicionais na detecção de anomalias em métodos?*
- *RQ2. A experiência dos desenvolvedores impacta na acurácia para detectar as anomalias em método?*
- *RQ3. Um conjunto composto por muitas métricas pode fazer com que a tarefa de identificação das anomalias em método seja mais demorada?*
- *RQ4. Existe uma métrica específica que detecta com acurácia cada anomalia em método?*
- *RQ5. Existe uma combinação de métricas que detecta com acurácia anomalias em métodos?*

4.2 Sistema Utilizado e Lista de Referência de Anomalias em Métodos

Sistema Usado. O estudo envolveu o sistema MobileMedia [Figueiredo et al., 2008] descrito na Seção 3.2.2. Este sistema foi selecionado, pois (i) foi utilizado anteriormente em outros estudos relacionados com manutenibilidade [Conejero et al., 2012; Ferrari et al., 2010; Figueiredo et al., 2008; Greenwood et al., 2007] e (ii) temos acesso a desenvolvedores e especialistas deste sistema. Diante disso, nós fomos capazes de ter acesso à lista de referência das anomalias para este sistema. Uma breve descrição das funcionalidades do MobileMedia e seus principais interesses podem ser encontrada na Seção 3.2.2.

Lista de Referência. Antes de conduzir o estudo, nós realizamos uma análise sistemática no código do MobileMedia pretendendo determinar quais métodos foram afetadas por anomalias relevantes (Seção 2.1.2). Nós também contamos com dois participantes especialistas em desenvolvimento, manutenção ou avaliação de sistemas. O objetivo é detectar métodos propícios a terem as anomalias em estudo. Cada especialista realizou o mesmo procedimento descrito na Seção 3.3. Discutimos com os especialistas cada método potencialmente afetado por anomalia. O resultado desta discussão foi registrado como uma decisão conjunta. Portanto, os métodos na lista de referência de cada anomalia apresentados na Tabela 4.1 foram aprovadas por nós em comum acordo com os especialistas.

Tabela 4.1. Lista de Referência de Anomalias em Métodos para o MobileMedia

Métodos com Feature Envy	
AlbumController.resetMediaData	MediaListController.bubbleSort
BaseController.init	MediaListController.exchange
Métodos com God Method	
MainUIMidlet.startApp	MediaListController.showMediaList
AlbumController.handleCommand	MusicPlayController.handleCommand
MediaController.handleCommand	PhotoViewController.handleCommand
MediaController.showImage	

4.3 Histórico dos Participantes

Este estudo envolveu um conjunto de 47 participantes de duas instituições distintas: UFMG (Brasil) e UFBA (Brasil). Os participantes da primeira instituição foram 15 alunos do curso de pós-graduação e 22 do curso de graduação. Os participantes da segunda instituição foram 10, sendo que 9 eram estudantes de pós-graduação e 1 era estudante de graduação. Os participantes foram organizados para trabalharem individualmente, de tal forma que cada um trabalhou com um conjunto de métricas: métricas tradicionais, métricas de interesse ou métricas híbridas (conjunto de métricas que é composto tanto de métricas tradicionais como por métricas de interesse). Os participantes foram agrupados tentando-se equilibrá-los pelo histórico de conhecimento. O estudo foi realizado utilizando apenas o sistema MobileMedia, que é um projeto orientado a objetos com implementação Java.

Antes de começar a rodar o experimento, nós aplicamos um questionário de histórico (Anexo A) para termos uma prévia do conhecimento de cada participante. A Tabela 4.2 resume o conhecimento que os participantes afirmaram ter no questionário de histórico. Eles responderam as perguntas relacionadas a: (i) Diagrama de Classes, (ii) Programação em Java, (iii) Experiência de Trabalho e (iv) Métricas de software. As três últimas colunas da Tabela 4.2 mostram o conhecimento que os participantes afirmaram ter em uma habilidade particular. Há participantes que não aparecem em uma linha porque eles não têm experiência nesse tópico em particular. Por exemplo, os seguintes participantes não têm experiência de trabalho: S1, S21, S23, S32, S36, S43, S44 e S45.

Tabela 4.2. Histórico dos Participantes que Detectaram Anomalias em Métodos

Métricas		Tradicional	Interesse	Híbrida
Aptidão	Diagrama de Classes	S1-S16	S17-S32	S33-S47
	Programação Java	S1-S16	S17-S32	S33-S47
	Experiência de Trabalho	S2-S16	S17-S20, S22, S24-S31	S33-S35, S37-S42, S46, S47
	Medição	S1-S16	S17-S32	S33-S47

Podemos observar na Tabela 4.2 que todos os participantes têm pelo menos um conhecimento básico em Diagrama de Classes, Programação Java e em Medição. Na verdade, pedimos aos participantes para indicar seu nível de conhecimento, escolhendo uma das seguintes opções: (i) nenhum, (ii) pouco, (iii) moderado e (iv) alta experiência (vide Anexo A). No entanto, depois de analisar os resultados, observamos que o nível de conhecimento não tem um grande impacto sobre as principais conclusões. Portanto, resolvemos não controlar esta variável no estudo. Nós apenas usamos esses dados para fazer uma distribuição balanceada dos participantes entre os grupos de métricas.

Todos os 47 participantes identificaram as duas anomalias em estudo, porém, com objetivo de facilitar a nossa discussão nas Seções 4.4 e 4.5, onde discutimos a acurácia em relação a identificação destas anomalias resolvemos organizá-los da seguinte forma: (i) do S1 ao S47 são os que fizeram a análise do *Feature Envy*, e (ii) do S48 a S94 referem-se aos participantes que fizeram a análise do *God Method*. Para maiores detalhes sobre a distribuição dos participantes consulte os Anexos C, G e H. A aplicação do experimento para anomalias em métodos foi realizada da mesma forma que foi feita a aplicação do experimento para as anomalias em classes, conforme explicado detalhadamente na Seção 3.5.

4.4 Resultados

Esta seção apresenta os resultados dos nossos experimentos. A Seção 4.4.1 introduz duas métricas: Cobertura e Precisão para suportar a análise dos resultados. Seções 4.4.2 e 4.4.3 reportam os resultados por anomalias.

4.4.1 Quantificando Cobertura e Precisão

Utilizamos três métricas para suportar a análise dos dados fornecidos pelos participantes: Verdadeiro Positivo (VP), Falso Positivo (FP) e Falso Negativo (FN) explicadas na Seção 3.6.1. Com base nessas métricas, quantificamos a Cobertura e a Precisão como apresentamos a seguir.

$$C = \frac{VP}{VP+FN} \tag{4.1}$$

$$P = \frac{VP}{VP+FP} \tag{4.2}$$

4.4.2 Detecção de *Feature Envy*

A Tabela 4.3 apresenta os resultados para a identificação de *Feature Envy*. As linhas nesta Tabela apresentam 3 partes dos dados: Cobertura (C), Precisão (P) e o Tempo (T) em minutos utilizado pelos participantes para completar suas tarefas. No total, 47 participantes tiveram que identificar a anomalia *Feature Envy* no sistema. Os dados da Tabela 4.3 mostraram que nenhum grupo de participantes se destacou com bons resultados.

De fato, apenas um participante, S4, alcançou 50% de Cobertura e 10% de Precisão. Na verdade, apenas 6 participantes - de 47 - foram capazes de identificar corretamente pelo menos uma instância desta anomalia, ou seja, apenas 6 participantes atingiram uma Cobertura maior que 0%. Estes participantes usaram 4 métricas tradicionais e apenas 2 de interesse. As baixas taxas de acertos para quase todos os participantes sugere que as métricas utilizadas neste estudo não podem indicar precisamente métodos com *Feature Envy*. É importante notar que estas métricas são recorrentemente utilizadas em estudos de modularidade [Eaddy et al., 2008; Figueiredo et al., 2008; Garcia et al., 2005; Marinescu, 2004].

Tabela 4.3. Resultado para o Feature Envy

Feature Envy																
	Tradicional															
Participante	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16
R(%)	0%	0%	0%	50%	0%	0%	0%	0%	0%	25%	0%	0%	25%	0%	0%	25%
P (%)	0%	0%	0%	10%	0%	0%	0%	0%	0%	33%	0%	0%	8%	0%	0%	50%
T (m)	42	15	39	48	53	10	11	21	11	7	19	26	19	14	24	26

Feature Envy																
	Interesse															
Participante	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S32
R(%)	0%	0%	0%	25%	0%	0%	0%	0%	0%	0%	25%	0%	0%	0%	0%	0%
P (%)	0%	0%	0%	11%	0%	0%	0%	0%	0%	0%	25%	0%	0%	0%	0%	0%
T (m)	26	30	29	36	31	10	8	9	12	6	20	16	22	30	24	17

Feature Envy															
	Híbrida														
Participante	S33	S34	S35	S36	S37	S38	S39	S40	S41	S42	S43	S44	S45	S46	S47
R(%)	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
P (%)	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
T (m)	24	40	30	25	29	15	22	20	11	8	16	43	38	29	19

4.4.3 Detecção de *God Method*

A Tabela 4.4 apresenta os resultados para detecção da anomalia *God Method*. Esta tabela segue a mesma estrutura da Tabela 4.3. Observando os dados desta tabela, constatamos que os participantes do grupo tradicional e híbrido obtiveram melhores resultados em termo de Cobertura que o grupo de interesse. Este resultado sugere que as métricas de interesse quando utilizadas isoladamente não oferecem meios adequados para detectar *God Method*. Apenas um participante (S72) do grupo de interesse obteve mais 50% de Cobertura. Esse desempenho é muito pior do que aquele que os grupos tradicionais e híbridos alcançaram, em média, eles marcaram 65% e 55% de Cobertura, respectivamente. Na verdade, este resultado não é uma surpresa, já que a definição do *God Method* diz explicitamente sobre o tamanho e coesão - atributos facilmente capturados por métricas tradicionais.

4.5 Análise e Discussão

Esta seção pretende responder a questão de pesquisa geral e as subquestões específicas definidas na Seção 3.1. Nós focamos nos resultados mais interessantes, mas os dados completos podem ser encontrados nos Anexos G e H.

4.5.1 Comparação das Métricas Tradicionais e de Interesse

O principal objetivo deste primeiro estudo é avaliar a eficácia das métricas de interesse para detectar anomalias de código. Portanto, esta seção tem como objetivo responder à seguinte questão de pesquisa especificada na Seção 4.1.

RQ1. Qual a acurácia das métricas de interesse comparadas com as métricas tradicionais na detecção de anomalias em métodos?

Baseando-se nos dados e discussão da Seção 4.4, foi observado que o uso de métricas de interesse apenas foi bom quando utilizada em conjunto com as métricas tradicionais para o caso do *God Method*. As métricas selecionadas não apresentaram bons resultados para *Feature Envy*. Estes resultados indicam que a acurácia do conjunto de métricas é largamente dependente da adequação de cada métrica para quantificar uma propriedade explicitamente mencionada na definição de anomalias.

Tabela 4.4. Resultado para o God Method

God Method																
	Tradicional															
Participante	S48	S49	S50	S51	S52	S53	S54	S55	S56	S57	S58	S59	S60	S61	S62	S63
R(%)	71%	57%	71%	71%	57%	71%	0%	86%	71%	71%	57%	86%	71%	71%	57%	71%
P(%)	100%	67%	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%	71%	100%	57%	71%
T(m)	11	13	9	13	14	7	10	6	10	15	7	15	7	8	12	7

God Method																
	Interesse															
Participante	S64	S65	S66	S67	S68	S69	S70	S71	S72	S73	S74	S75	S76	S77	S78	S79
R(%)	14%	0%	0%	29%	0%	43%	29%	14%	57%	14%	43%	29%	0%	43%	29%	0%
P(%)	100%	0%	0%	33%	0%	60%	50%	25%	100%	25%	100%	40%	0%	75%	0%	0%
T(m)	15	0	23	24	14	4	9	5	8	4	5	10	12	16	9	15

God Method															
	Híbrida														
Participante	S80	S81	S82	S83	S84	S85	S86	S87	S88	S89	S90	S91	S92	S93	S94
R(%)	71%	43%	57%	57%	86%	29%	43%	14%	57%	86%	57%	71%	43%	57%	57%
P(%)	100%	38%	100%	100%	100%	50%	100%	33%	100%	100%	100%	100%	38%	100%	100%
T(m)	13	15	8	20	0	5	6	3	12	14	7	8	9	11	14

4.5.2 Histórico dos Participantes

Esta seção analisa como o histórico dos participantes pode impactar nos resultados deste estudo. Em outras palavras, nós pretendemos responder a seguinte questão de pesquisa levantada na Seção 4.1.

RQ2. A experiência dos desenvolvedores impacta na acurácia para detectar as anomalias em métodos?

Como discutido na Seção 4.3, todos os participantes possuem o mínimo de conhecimento básico nos tópicos considerados relevantes para o desenvolvimento de software, denominados: Diagrama de Classes UML e Programação Java. Por isso, decidimos fazer esta análise exclusivamente em relação à experiência de trabalho dos participantes. A fim de simplificar a análise, nós dividimos os participantes em duas categorias de acordo com a sua experiência de trabalho: (i) *alguma experiência* para identificar os participantes que trabalharam por pelo menos 6 meses em indústria de desenvolvimento de software e (ii) *sem experiência* indica os participantes que nunca trabalharam ou trabalharam por menos de 6 meses. Em seguida, calculamos a média de Cobertura e Precisão obtida por participantes de cada categoria para a anomalia *God Method*, como apresentada na Figura 4.1. Esta análise se restringe ao *God Method* porque os valores de Cobertura para o *Feature Envy* foram predominantemente nulos (0%).

A Figura 4.1 mostra os valores médios de Cobertura e Precisão agrupados pelas duas categorias de trabalho discutidas anteriormente. Foram considerados apenas os dados da anomalia *God Method*. É interessante notar que, no entanto, os resultados para esta anomalia não coincide com o senso comum. Ou seja, participantes sem experiência em desenvolvimento de software se saíram melhor que os participantes com alguma experiência. Este resultado se assemelha ao observado para a anomalia *God Class* (Seção 3.7.2). Uma possível explicação para este resultado é que o *God Method* é a anomalia mais simples de entender, em comparação a outras anomalias de código. Por isso, não necessita de experiência avançada em desenvolvimento de software. Outra possível explicação é que o critério adotado por um participante experiente pode ser mais rígido do que o de um participante inexperiente. O sistema utilizado neste estudo (MobileMedia) é um sistema pequeno, então, um participante muito experiente pode estar acostumado com sistemas muito grandes, portanto, com métodos maiores que os do MobileMedia. Assim, eles foram induzidos ao erro na hora de identificar as instâncias de *God Method* em um sistema de menor escala.

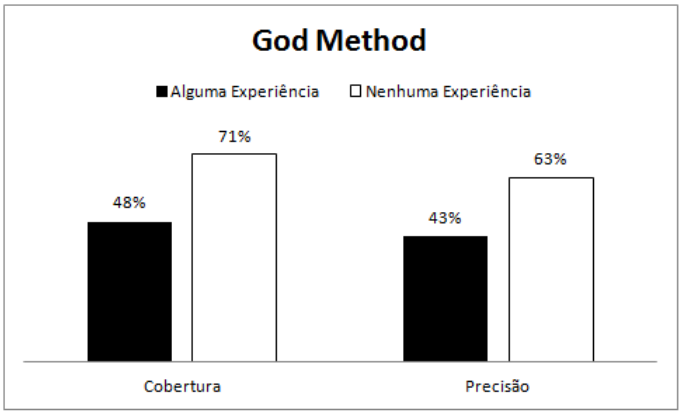

Figura 4.1. Experiência de Trabalho dos Participantes.

4.5.3 Muitas Métricas, Análise mais Demorada?

Esta seção faz uma análise do tempo gasto pelos participantes para detectar anomalias em métodos. Para verificar a questão a seguir, nós analisamos o tempo que foi gasto para detectar as anomalias em métodos e a respectiva correlação com os valores de Cobertura.

RQ3. Um conjunto composto por muitas métricas pode fazer com que a tarefa de identificação das anomalias de métodos seja mais demorada?

Feature Envy. A Figura 4.2 mostra os dados de Cobertura (eixo x) e tempo gasto (eixo y) pelos participantes para detectar a anomalia *Feature Envy*. Observa-se nesta figura que os participantes do grupo de interesse consumiram o menor tempo (20 minutos em média) para executar suas tarefas em relação aos outros grupos. Os grupos tradicionais e híbridos apresentaram resultados semelhantes em torno de 25 minutos em média. Uma possível explicação para este resultado é que o conjunto de métricas de interesse apenas incluem seis métricas contra oito métricas do conjunto de métricas tradicionais. Portanto, com menos dados para analisar, os participantes poderiam ter terminado a sua tarefa mais rápida. Este resultado poderia confirmar o senso comum de que quando os participantes usam um conjunto maior de métricas, tais como as métricas híbridas, eles estão propensos há gastar mais tempo para detectar uma anomalia de código. No entanto, isto não foi sempre o caso para as anomalias analisadas, conforme discutimos a seguir para o *God Method*.

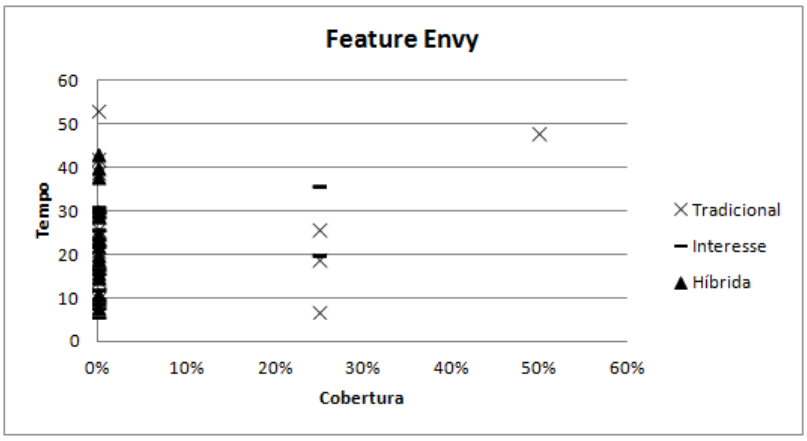

Figura 4.2. Tempo Eficiente para o Feature Envy.

God Method. A Figura 4.3 apresenta os dados da eficiência de tempo e Cobertura para o *God Method*. Podemos observar nesta figura que o conjunto de métricas de interesse não apresentou tempo eficiente para esta anomalia. De fato, os participantes que usaram estas métricas normalmente demoraram um pouco mais do que os outros participantes para executar suas tarefas. Em média, os participantes do grupo de interesse gastaram 11 minutos, contra 10 minutos em ambos os grupos tradicionais e híbridos. Uma análise cuidadosa da Figura 4.3 sugere que, em geral para o *God Method*, quanto mais longa a análise, melhores resultados os participantes alcançaram em termos de Cobertura. Este resultado pode ser confirmado pelo fato de que a maioria dos participantes que gastaram mais de 10 minutos para analisar os dados teve mais de 50% de Cobertura. Portanto, confirma-se que a anomalia de método *God Method* exige uma análise mais cuidadosa de muitas métricas, como indicada pela superioridade dos grupos tradicionais e híbridos.

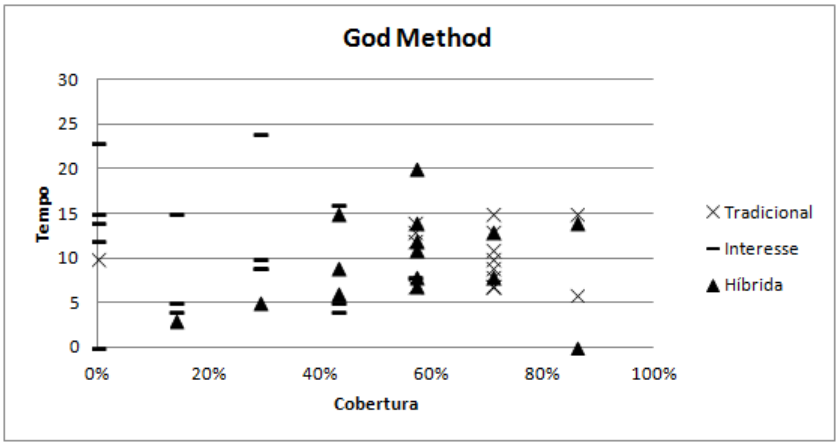

Figura 4.3. Tempo Eficiente para o God Method.

4.5.4 Métricas Específicas

Esta seção está voltada para a questão de pesquisa a seguir. Como explicado na Seção 3.5, os participantes relataram as métricas que foram consideradas úteis para cada anomalia estudada. Com base em suas respostas, analisamos nesta seção as métricas que foram consideradas úteis pelos participantes.

RQ4. Existe uma métrica específica que detecta com acurácia cada anomalia em método?

A Tabela 4.5 mostra as métricas que os participantes relataram ter utilizado para o *Feature Envy*. Como nesta anomalia a taxa de Cobertura foi muito baixa, consideramos as métricas que foram citadas por pelos dois participantes. Para a análise do *Feature Envy*, apenas quatro métricas foram consideradas úteis. As linhas desta tabela mostram o número de participantes que usaram cada métrica e a média de Cobertura destes participantes. Observa-se nesta tabela que a métrica PAR foi considerada útil por 3 participantes e eles alcançaram 33% de Cobertura em média. Por outro lado, as métricas NOA, LOCC e LCOM foram indicadas como úteis por dois participantes cada. Quem utilizou a métrica NOA obteve 38% de Cobertura em média. No entanto, quem utilizou as métricas LOCC e LCOM obtiveram apenas 25% de Cobertura em média. Estes resultados confirmaram que nenhuma métrica analisada parece realmente útil para detectar *Feature Envy*.

Tabela 4.5. Métricas Consideradas Úteis para o Feature Envy

Métricas	**PAR**	**NOA**	**LOCC**	**LCOM**
Participante que usaram estas métricas	S4, S13 S16	S4, S13	S20, S27	S10, S16
Média Cobertura (%)	33%	38%	25%	25%

Também foram analisadas as métricas que foram consideradas úteis para detectar *God Method* por pelo menos cinco participantes. Restringimos as nossas análises as métricas com a média de Cobertura superior a 40%. O total de Cobertura e frequência de utilização de cada uma destas métricas é apresentada na Tabela 4.6. As métricas consideradas mais úteis são as métricas tradicionais LOC e CYCLO. Elas também foram as métricas que apresentaram a maior taxa de Cobertura: 64% e 63%, respectivamente. Outra métrica com média de Cobertura de 63% foi PAR, que tem o objetivo de quantificar o número de parâmetros nos métodos. Esta métrica foi utilizada por 9 participantes.

Tabela 4.6. Métricas Consideradas Úteis para o God Method

Métricas	LOC	CYCLO	NCO	PAR	NCC	CDLOC
Participantes que usaram estas métricas	S48 - S63 S80, S82 - S86, S88 - S90, S92 - S94	S48 - S53, S55, S57, S58, S60, S63, S82 - S85, S90, S92, S94	S64, S67, S69 - S71, S73, S74, S77, S78, S80 - S82, S91, S92, S94	S52, S56, S59, S60, S80, S82, S87, S89, S94	S67, S69, S71, S75, S81, S89	S69, S70, S72, S77, S81, S83
Média Cobertura (%)	64%	63%	40%	63%	41%	46%

Três métricas de interesse foram consideradas úteis por 5 ou mais participantes. A métrica NCO destina-se a calcular o número de interesses em operações. Ela foi a métrica de interesse mais utilizada pelos participantes - 15 no total. Esta métrica de interesse foi a que apresentou a maior taxa de Cobertura - 40%, em média. Este resultado parece ser intuitivo. Pois, o método é o local de medida para esta métrica e esta anomalia de código é problema estrutural no nível do método. No entanto, outras duas métricas de interesse relacionadas ao nível de classe também foram consideradas úteis. Elas são as seguintes métricas: NCC (Cobertura de 41%) e CDLOC (Cobertura de 46%).

4.5.5 Combinações de Métricas

Nesta seção, analisamos as possíveis combinações de métricas que podem ser úteis para detectar anomalias específicas e responder a questão de pesquisa abaixo. Para tal, contamos com a análise prévia de métricas que foram úteis para detectar cada anomalia em método. Com o objetivo de determinar quais métricas foram utilizadas em conjunto para detectar anomalias em métodos, realizamos a análise dos participantes que usaram as mesmas métricas e tiveram alto percentual em termos de Cobertura.

RQ5. Existe uma combinação de métricas que detecta com acurácia anomalias em métodos?

Feature Envy. Como a maioria dos participantes tiveram um desempenho fraco para detectar *Feature Envy*, consideramos nesta análise apenas os participantes que identificaram corretamente pelo menos uma instância da anomalia. Observou-se que métricas individuais como, PAR, LOCC e LCOM foram usadas por alguns participantes. A combinação destas métricas, por exemplo, PAR e LCOM foram utilizadas pelo participante, S16, que atingiu 25% de Cobertura. Por outro lado, o participante, S4, obteve 50% de Cobertura - o melhor resultado para o *Feature Envy* - usando PAR combinado com NOA. Mas, como só tivemos apenas um participante para cada combi-

nação de métrica, concluímos que não foi possível obter uma combinação de métricas para esta anomalia.

God Method. A análise desta anomalia foi filtrada pelos participantes que obtiveram mais de 40% de Cobertura. Observamos alguns casos de métricas que foram utilizadas com sucesso em conjunto. Por exemplo, a combinação de LOC com CYCLO foi utilizada por 17 participantes que obtiveram mais de 40% de Cobertura. Obtivemos outras combinações de métricas apontadas pelos participantes onde a taxa de Cobertura foi maior que 70%, são elas: (i) LOC, NOO e PAR usada por S59, (ii) LOC, PAR e NCC usada por S89, (iii) NOCO e NCO usada por S91 e (iv) LOC, PAR, NOO e NCO usada por S80.

4.6 Considerações Finais

O estudo apresentado neste capítulo analisou a eficácia das métricas de interesse para detectar anomalias em métodos. Os resultados obtidos revelaram que as métricas de interesse podem ser úteis para apoiar a detecção da anomalia *God Method*. No entanto, nenhuma das métricas tradicionais e de interesse avaliadas pode ser considerada útil para a detecção de *Feature Envy*.

Também foram investigadas quais métricas específicas são mais adequadas para a detecção de anomalias em métodos. Em geral, os resultados indicaram que a acurácia de cada conjunto de métricas é largamente dependente da adequação de cada métrica para quantificar uma propriedade explicitamente mencionada na definição da anomalia. Em particular, observou-se que três métricas de interesse (NCO, NCC e CDLOC) podem ser capazes de ajudar a detectar o *God Method* quando usadas em conjunto com outras métricas tradicionais.

Este estudo representa um primeiro passo para a avaliação de métricas de interesse para detectar anomalias de código em métodos. Além disso, os resultados deste estudo foram fundamentais para a identificação de métricas (ou conjunto de métricas) para a elaboração das regras heurísticas propostas no Capítulo 5.

Capítulo 5

Regras Heurísticas baseada em Interesse

Avaliar a qualidade e a melhoria de programas orientados a objetos não é uma tarefa fácil. As métricas de software são poderosos mecanismos para avaliar e controlar a qualidade do software [Chidamber & Kemerer, 1994]. Porém, a interpretação das métricas isoladamente não é suficiente [Marinescu, 2004]. Pois, na maioria dos casos em que as métricas são usadas individualmente, elas não fornecem informações suficientes sobre a causa do problema. O valor de uma métrica pode indicar que há uma anomalia no código, mas deixa o desenvolvedor sem indícios sobre qual é a anomalia. Consequentemente, isso tem um impacto negativo na relevância dos resultados da medição [Marinescu, 2004; Lanza & Marinescu, 2006]. Portanto, para superar as limitações das métricas de software, Marinescu [2004] propôs uma abordagem para aumentar a importância da interpretação das métricas. Ele propôs um mecanismo para analisar o código fonte de programas utilizando regras baseadas em métricas compostas. Marinescu [2004] chamou este mecanismo de *estratégia de detecção*. Nesta dissertação, utilizamos o termo *regras heurísticas* ou simplesmente *heurísticas*.

Neste capítulo, propomos um método quantitativo composto por métricas tradicionais e métricas de interesse para apoiar a detecção das anomalias estudadas nesta dissertação. A Seção 5.1 apresenta a explicação da notação utilizada para representar as regras heurísticas. As Seções 5.2 e 5.3 apresentam as regras heurísticas que combinam métricas tradicionais e de interesse para detectar anomalias de código em classes e métodos, respectivamente. Por último, na Seção 5.4 apresentamos ConcernMeBS, uma ferramenta desenvolvida para automatizar as heurísticas propostas neste capítulo.

5.1 Notação gráfica para Regras Heurísticas

Lanza e Marinescu [2006] utilizaram em seu livro uma notação gráfica para representar a *estratégia de detecção* proposta por Marinescu [2004] discutidas na Seção 2.5. Como esta detecção é baseada em operadores lógicos AND (E) e OR (OU), Lanza e Marinescu [2006] decidiram utilizar a notação gráfica de circuitos elétricos para representar uma *estratégia de detecção* ao invés de fórmulas matemáticas.

A Figura 5.1 apresenta a estrutura geral de uma regra heurística usando esta notação. Nesta figura, as entradas são feitas através da condição de filtragem *simples* representada através de um retângulo arredondado e acinzentado. Cada elemento de filtragem é composto por uma descrição informal e um compartimento branco (caixa) com a expressão lógica de filtragem. A expressão é formada por uma métrica seguida pelo operador de filtragem ($>$, $<$, $\geq$ ou $\leq$) e do valor limite para a métrica. As entradas são combinadas por operadores lógicos (E e OU) resultando na saída que é a anomalia de código.

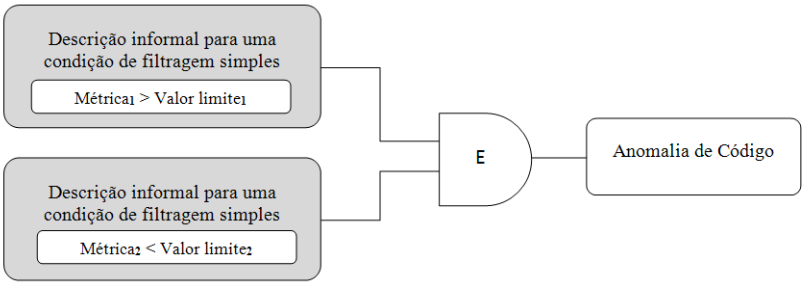

Figura 5.1. Notação Gráfica Representada por Portas Lógicas.

5.2 Anomalias de Código em Classes

As heurísticas apresentadas nesta seção são resultados da análise dos dados coletados nos experimentos realizados com participantes e descritos no Capítulo 3. As regras heurísticas serão apresentadas seguindo a seguinte estrutura: (i) Motivação, descreve uma breve descrição das anomalias em classes; (ii) Métricas Selecionadas, relata e descreve quais métricas foram obtidas como úteis nos experimentos realizados com os participantes; (iii) Heurística Primária, mostra a regra heurística que foi composta após a análise das métricas reportadas como úteis pelos participantes; e (iv) Heurística Secundária, apresenta a regra heurística alternativa sugerida neste trabalho que é composta por algumas métricas que tem potencial para detecção das anomalias, mas que

não foram percebidas nos experimentos pelos os participantes. Portanto, a heurística secundária é resultante do melhoramento da heurística primária.

5.2.1 Divergent Change

Motivação. Esta anomalia ocorre quando uma classe é frequentemente modificada de maneiras diferentes, por razões diferentes [Fowler, 1999]. Dependendo do número de atribuições de uma dada classe, ela pode sofrer diversos tipos de alterações que podem estar associadas a um sintoma de entrelaçamento de interesses. Em outras palavras, uma classe que apresenta interesses entrelaçados é susceptível de ser alterada, por diferentes razões.

Métricas Selecionadas. As métricas NCC e LCOM foram obtidas como úteis nos experimentos realizados. Tivemos um total de 24 participantes para detectar a anomalia *Divergent Change*. Destes participantes, 11 consideraram útil a métrica NCC e também 11 participantes relataram como útil a métrica LCOM. CDO_C foi selecionada com base na definição da anomalia *Divergent Change*. As métricas utilizadas para compor a heurística primária e secundária para anomalia *Divergent Change* são descritas a seguir.

1. Número de Interesse por Componente (NCC) [Figueiredo et al., 2012]: Conta o número de interesse em cada classe ou interface.

2. Falta de Coesão em Métodos (LCOM) [Chidamber & Kemerer, 1994]: Mede o quanto as operações de uma classe acessam atributos em comum. Quanto maior o número de atributos em comum, maior é a coesão da classe e, portanto, menor é a perda de coesão.

3. Espalhamento de Interesse em Operações (CDO_C) [Sant'Anna, 2004]: Conta o número de operações que implementam um interesse. Este "$_C$" indica que os valores desta métrica são agrupados por classe.

Heurística Primária. Esta heurística foi composta após análise das métricas reportadas como úteis pelos participantes, conforme explicado na Seção 3.7.5. Para anomalia *Divergent Change*, a combinação das métricas indicadas pelos participantes foi a NCC com LCOM. Portanto, a Figura 5.2 mostra a regra heurística obtida no experimento para esta anomalia. Esta heurística combina duas métricas NCC e LCOM Pois, se uma classe possui a medida de LCOM alta isto indica que há uma individualidade entre as operações. Os valores desta métrica são filtrados com os valores da métrica NCC que mede quanto interesses uma classe possui. Nesta figura também são

apresentados os valores limites para cada métrica, que foram elaborados com base na medição das métricas do sistema MobileMedia. Os melhores valores para estes limites devem ser ajustados de acordo com o tamanho do sistema.

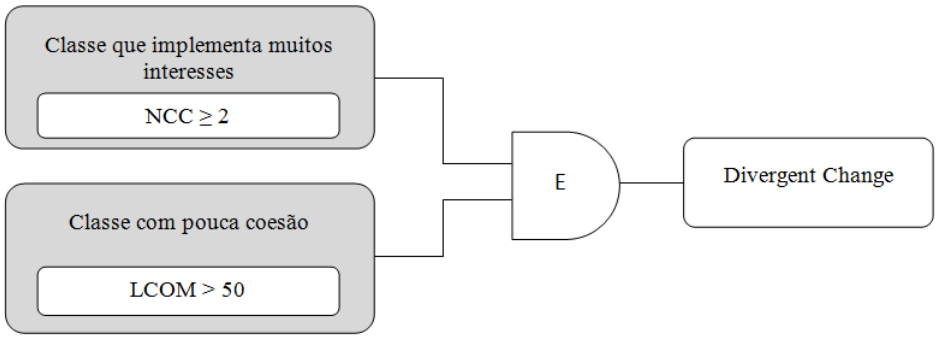

Figura 5.2. Regra Heurística Primária para Anomalia *Divergent Change*.

Heurística Secundária. A heurística secundária foi criada como uma possível heurística alternativa. Sua criação foi motivada por perceber-se que algumas combinações de métricas tem potencial para detectar a anomalia investigada, porém não foram percebidas pelos participantes nos experimentos. Diante disso, criou-se a heurística secundária para o *Divergent Change* utilizando a combinação das métricas NCC com CDO_C, conforme é ilustrado na 5.3. A métrica CDO_C foi selecionada porque conta o número de operações que implementam um interesse. Quanto mais operações forem afetadas pelo interesse, mais operações são modificadas durante a atividade de manutenção. Os valores desta métrica são filtrados com os valores da métrica NCC, pois ela mede quantos interesses uma classe possui. Uma classe que implementa muitos interesses distintos é um indício que a classe tem a anomalia *Divergent Change*.

Na Figura 5.3 são apresentados os valores limites para cada métrica que foram elaborados com base em um sistema pequeno: MobileMedia. Portanto, estes limites devem ser ajustados de acordo com o tamanho do sistema. Para alguns domínios ou sistemas específicos, é esperado que a heurística secundária possa revelar mais anomalias que a heurística primária. Uma avaliação desta heurística deve ser realizada para comprovar-se a sua eficácia.

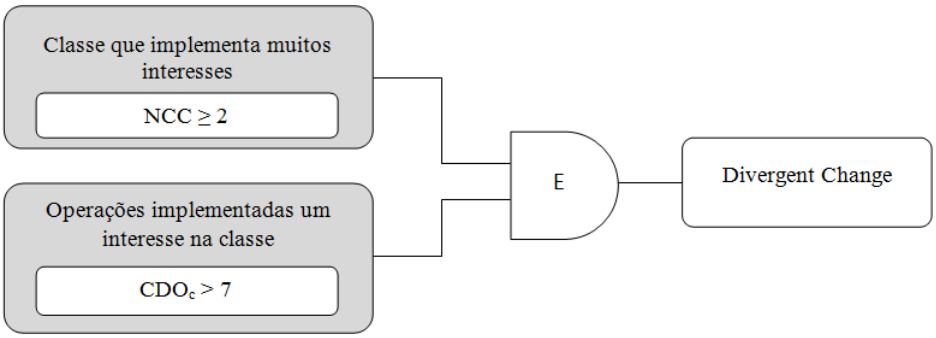

Figura 5.3. Regra Heurística Secundária para Anomalia *Divergent Change*.

5.2.2 Shotgun Surgery

Motivação. Esta anomalia é de alguma forma o oposto da anomalia *Divergent Change*. Uma classe com *Shotgun Surgery* é identificada quando fazemos um tipo de mudança em uma classe e esta se propaga para uma série de pequenas mudanças em muitas classes diferentes [Fowler, 1999]. Na visão de interesse, o espalhamento de interesses pode levar a pequenas mudanças em classes que possuem o interesse a ser modificado.

Métricas Selecionadas. As métricas CBO, CDC e NCC foram obtidas como úteis nos experimentos realizados. Tivemos um total de 19 participantes que trabalharam na detecção da anomalia *Shotgun Surgery*. Destes participantes, 11 relataram que utilizaram a métrica CBO, 5 utilizaram a métrica CDC e 4 utilizaram a métrica NCC. CDLOC foi selecionada com base na definição da anomalia *Shotgun Surgery*.As métricas utilizadas para compor a heurística primária e secundária para anomalia *Shotgun Surgery* são descritas a seguir. Informações mais detalhadas sobre estas métricas podem ser encontradas no Capítulo 3 ou nos trabalhos citados.

1. Acoplamento entre Objetos (CBO) [Chidamber & Kemerer, 1994]: Esta métrica consiste na contagem do número de classes com as quais uma classe se relaciona.

2. Espalhamento de Interesse em Componentes (CDC) [Sant'Anna, 2004]: Conta o número de classes e interfaces que implementam um interesse.

3. Número de Interesse por Componente (NCC)[Figueiredo et al., 2012]: Conta o número de interesse em cada classe ou interface.

4. Espalhamento de Interesse em Operações ($CDLOC_C$)[Sant'Anna, 2004]: Conta o número de pontos de transição entre o interesse avaliado e os outros interesses do sistema através das linhas de código. Este "$_C$" indica que os valores desta métrica são agregados por classe.

Heurística Primária. Esta heurística foi composta após análise das métricas reportadas como úteis pelos participantes. A Figura 5.4 mostra a heurística para a anomalia *Shotgun Surgery*. Esta heurística combina três métricas: CDC, CBO e NCC. A métrica CDC irá contar o número de classes que implementam um interesse, consequentemente filtrando as classes que estão com interesses muito espalhados. Em seguida, usamos a métrica CBO para filtrar quais são as classes que estão relacionadas a outras. E por último, o filtramos o valores de CBO pela métrica NCC, que retorna as classes que implementam muitos interesses. Esta regra heurística permite-nos obter a anomalia *Shotgun Surgery*. Nesta figura são apresentados os valores limites para cada métrica. Eles foram elaborados baseando-se no sistema MobileMedia. Portanto, salientamos que estes valores devem ser ajustados de acordo com o tamanho do sistema.

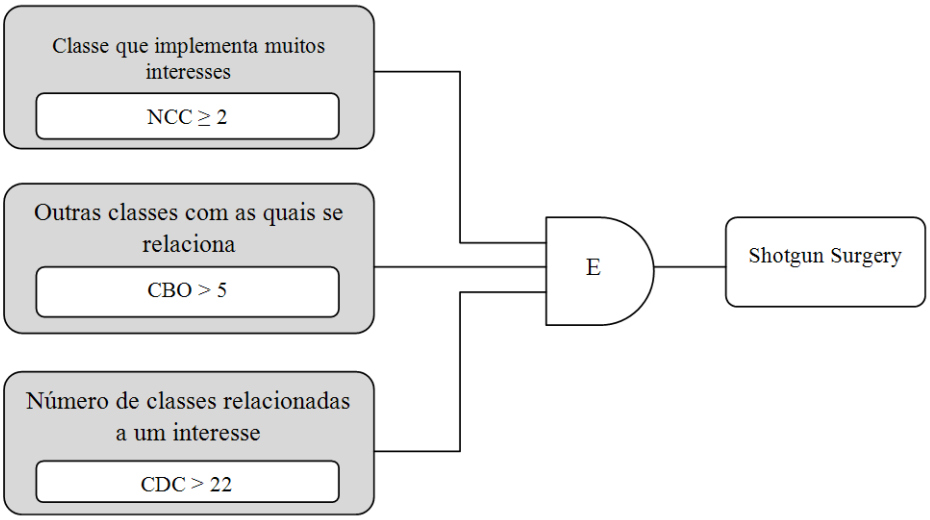

Figura 5.4. Regra Heurística Primária para Anomalia *Shotgun Surgery*.

Heurística Secundária. Esta heurística foi criada como uma possível heurística alternativa para o *Shotgun Surgery* utilizando a combinação das métricas NCC com CDLOC_C. A métrica CDLOC_C foi escolhida pois captura como o código que implementa um interesse está espalhado pelo código dos outros interesses, ou seja, entrelaçado. Esta mistura de interesses pode causar a propagação de mudanças para outras classes. Em seguida, utilizamos a métrica NCC como filtro, pois ela mede quantos interesses uma classe possui. Uma classe que implementa muitos interesses distintos pode ser propícia a propagar mudanças para outras classes por diferentes razões (interesses). A filtragem da combinação destas duas métricas retorna a anomalia *Shotgun Surgey*. A Figura 5.5 ilustra a composição da segunda regra heurística para o *Shotgun Surgery*. Nela são apresentados os valores limites para cada métrica.

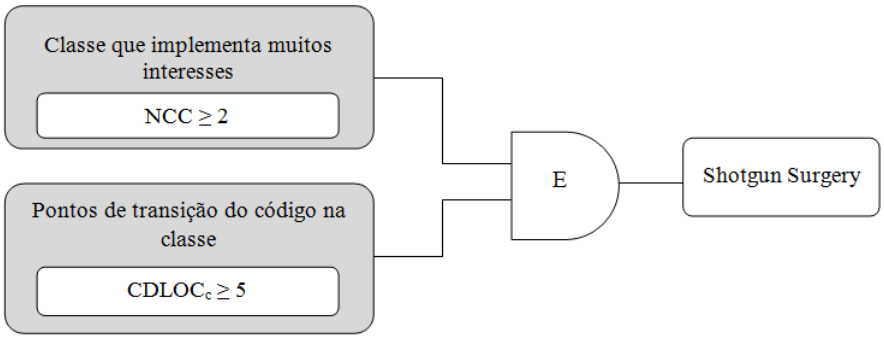

Figura 5.5. Regra Heurística Secundária para Anomalia *Shotgun Surgery*.

5.2.3 God Class

Motivação. Uma classe com esta anomalia é caracterizada por ter muitas funcionalidades e pela tendência de atrair mais e mais responsabilidades [Riel, 1996]. Em uma perspectiva diferente, podemos dizer que *God Class* é uma classe que implementa muitos interesses e, por isso, tem muitas responsabilidades [Carneiro et al., 2010].

Métricas Selecionadas. As métricas CBO, LCOM, WMC, LOC e CDO foram as obtidas como úteis nos experimentos realizados. Tivemos um total de 9 participantes para detectar a anomalia *God Class*. Destes participantes, 4 indicaram a métrica CBO com útil. A métrica LCOM também foi indicada como útil por 4 participantes. As métricas WMC, LOC e CDO foram indicadas como úteis por 3 participantes cada. A métrica NCC foi escolhida com base na definição da anomalia *God Class*. As métricas utilizadas para compor a heurística primária e secundária para esta anomalia são descritas a seguir. Informações mais detalhadas sobre estas métricas podem ser encontradas no Capítulo 3 ou nos trabalhos citados.

1. Acoplamento entre Objetos (CBO) [Chidamber & Kemerer, 1994]: É a contagem do número de classes com as quais estão acopladas.

2. Linhas de Código (LOC): [Fenton & Pfleeger, 1996]: É a contagem do número de linhas de código de uma classe.

3. Falta de Coesão em Métodos (LCOM) [Chidamber & Kemerer, 1994]: Mede quantos métodos de uma classe acessam atributos em comum.

4. Método Ponderado por Classe (WMC) [Chidamber & Kemerer, 1994]: A complexidade de uma classe é definida em função ao número de métodos e pesos relativos a este métodos obtidos pela quantidade de parâmetros.

5. Espalhamento de Interesse em Operações (CDO_C) [Sant'Anna, 2004]: Conta o número de operações que implementam um interesse. Este "$_C$" indica que os valores desta métrica são agrupados por classes.

6. Número de Interesse por Componente (NCC) [Figueiredo et al., 2012]: Conta o número de interesse em cada classe ou interface.

Heurística Primária. Esta heurística foi composta após análise das métricas reportadas como úteis pelos participantes. A Figura 5.6 mostra a heurística primária para a anomalia *God Class*. Esta heurística combina dois conjuntos de métricas: (i) composto pelas métricas LOC, LCOM e CBO ou (ii) composto pelas métricas WMC e CDO_C. No primeiro conjunto de métricas, a métrica LOC é utilizada para restringir às classes que tem muitas linhas de código. Em seguida, utilizamos a métrica LCOM para capturar as classes com pouca coesão. Por último, filtramos os valores de LCOM pela métrica CBO para filtrar quais são as classes que estão relacionadas a outras. No segundo conjunto de métricas utilizamos a métrica CDO_C para capturar o número de operações que implementam um interesse. Filtramos o resultado desta métrica pela métrica WMC que retorna as classes com complexidade alta. Esta regra heurística permite-nos obter a anomalia *God Class*.

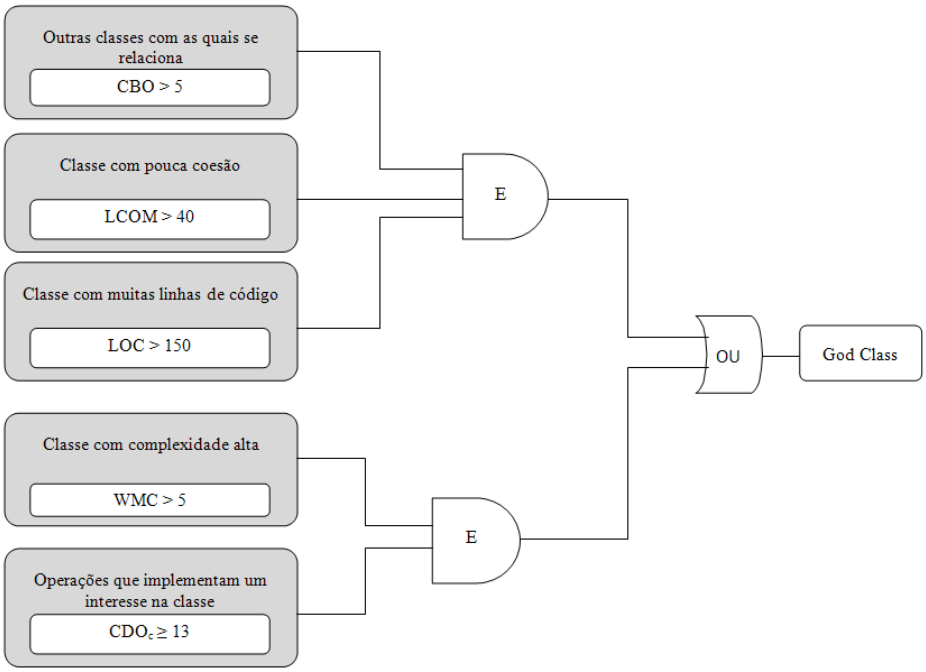

Figura 5.6. Regra Heurística Primária para Anomalia *God Class*.

Heurística Secundária. Esta heurística foi criada como uma possível heurística alternativa para o *God Class*. Para esta heurística utilizamos a combinação das métricas WMC com CDO_C e com NCC, conforme ilustrado na Figura 5.7. A métrica WMC foi selecionada, pois uma classe com complexidade alta é um indício de que ela está centralizando muitas funcionalidades. Por isso utilizamos o valor retornado pela WMC para filtrar a métrica CDO_C, pois ela conta o número de operações que implementam um interesse na classe. Quanto mais operações forem implementadas por um interesse, mais responsabilidades esta classe está propícia a ter. Por último, restringimos os valores da métrica CDO_C através da métrica NCC, pois mede quantos interesses uma classe possui. Uma classe que implementa muitos interesses distintos pode indicar que ela agrega muitas responsabilidades.

Na Figura 5.7 são apresentados os valores limites para cada métrica, que foram elaborados com base no sistema MobileMedia. Portanto, estes valores devem ser ajustados de acordo com o tamanho do sistema. Assim, para alguns domínios ou sistemas específicos, é esperado que a heurística secundária possa revelar mais anomalias que a heurística primária. Uma avaliação desta heurística deve ser realizada para comprovar-se a eficácia.

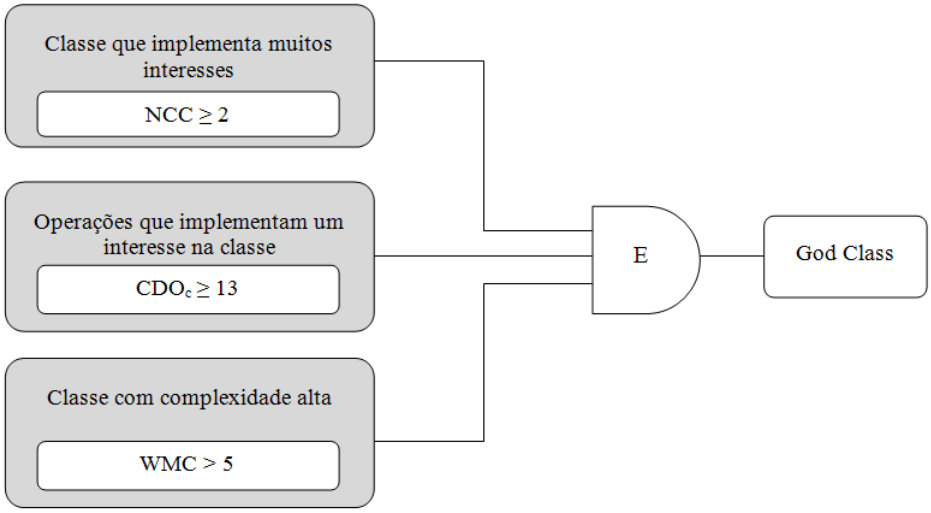

Figura 5.7. Regra Heurística Secundária para Anomalia *God Class*.

5.3 Anomalias de Código em Métodos

As heurísticas apresentadas nesta seção são resultados da análise dos dados coletados nos experimentos realizados com participantes e descritos no Capítulo 4. As regras heurísticas serão apresentadas seguindo a seguinte estrutura: (i) Motivação, descreve

uma breve descrição das anomalias em métodos; (ii) Métricas Selecionadas, relata e descreve quais métricas foram obtidas como úteis nos experimentos realizados com os participantes; (iii) Heurística Primária, mostra a regra heurística que foi composta após a análise das métricas reportadas como úteis pelos participantes; e (iv) Heurística Secundária, apresenta a regra heurística alternativa sugerida neste trabalho. Esta heurística é composta por algumas métricas quem tem um potencial de detecção das anomalias, mas que podem não ter sido percebidas pelos participantes do experimento.

5.3.1 Feature Envy

Motivação. Esta anomalia de código refere-se àqueles métodos que parecem mais interessantes nas funcionalidades de outra classe do que nas funcionalidades de sua classe. Estes métodos acessam diretamente ou através de métodos de acesso a grande quantidade de dados de outra classe. Isto pode ser um sinal que o método está mal localizado e que deveria ser transferido para outra classe. Ou seja, este método implementa funcionalidade semelhante à outra classe. Em geral, tenta-se colocar um método na classe que implementa funcionalidades semelhantes [Fowler, 1999].

Métricas Selecionadas. O conjunto de métricas é selecionado com base nos experimentos realizados com 47 participantes para a anomalia *Feature Envy*. Entretanto, nenhuma das métricas (tradicionais ou de interesse) avaliadas podem ser consideradas úteis para a detecção desta anomalia nos experimentos realizados.

Heurística Primária. Os participantes listaram algumas métricas como: PAR, NOA, LOCC e LCOM. Porém, as médias de taxas de Cobertura e Precisão foram muito baixas. O melhor resultado de combinação de métricas para o *Feature Envy* foi alcançado apenas por um participante (50% de Cobertura) que utilizou a combinação das métricas PAR e NOA. Mas, como tivemos apenas um participante com uma taxa de Cobertura satisfatória, concluímos que os conjuntos de métricas fornecido não eram adequados para capturar as propriedades desta anomalia. Sendo assim, não foi possível criar a heurística primária.

Heurística Secundária. Não foi possível compor uma heurística secundária, pois reconhecemos que o conjunto de métricas que investigamos não foi o adequado para detectar o *Feature Envy*. Um novo conjunto de métricas deve ser estudado e analisado ou mesmo proposto para identificar esta anomalia em trabalhos futuros.

5.3.2 God Method

Motivação. Geralmente, um método é iniciado com um método "normal". Entretanto, mais e mais funcionalidades são adicionadas a ele, até ele se tornar fora de controle e difícil de ser mantido e entendido. Assim, o *God Method* tende a centralizar as funcionalidades de uma classe da mesma forma que o *God Class* centraliza as funcionalidades de um subsistema inteiro, ou até mesmo de todo um sistema [Fowler, 1999]. Em uma perspectiva diferente, podemos dizer que o *God Method* é um método que implementa muitos interesses e, por isso, ele tem muitas responsabilidades.

Métricas Selecionadas. As métricas LOC, CYCLO e NCO foram as obtidas como úteis nos experimentos realizados. Tivemos um total de 47 participantes para a anomalia *God Method*. Destes participantes, 27 indicaram a métrica LOC como útil, 18 indicaram a métrica CYCLO e 15 a métrica NCO. A métrica LOCC foi selecionada com base na definição da anomalia *God Method*. As métricas utilizadas para compor a heurística primária e secundária para esta anomalia são descritas a seguir. Informações mais detalhadas sobre estas métricas podem ser encontrados no Capítulo 4 ou nos trabalhos citados.

1. Linhas de Código (LOC_m) [Fenton & Pfleeger, 1996]: Conta o número de linhas de código de um método, excluindo linhas em branco e comentários. Este "m" indica que os valores desta métrica são agrupados por métodos.

2. Complexidade Ciclomática de McCabe (CYCLO) [Fenton & Pfleeger, 1996]: Conta o número de desvios em um método. Cada vez que um desvio ocorre (IF, FOR, WHILE) essa métrica é incrementado por um.

3. Número de Interesse por Operações (NCO): Conta o número de interesses em cada operação ou construtor de uma classe.

4. Linhas de Código do Interesse ($LOCC_m$) [Eaddy et al., 2008]: Conta o número de linhas de código que implementam um interesse. Este "m" indica que os valores desta métrica são agrupados por métodos.

Heurística Primária. Esta heurística foi composta após análise das métricas reportadas como úteis pelos participantes. A Figura 5.8 mostra a heurística para a anomalia *God Method*. A métrica LOC_m conta o número de linhas de código de um método, portanto ela retornará apenas os métodos que tenham o valor maior que o valor limite estabelecido, que no caso desta heurística é 30. A CYCLO foi utilizada para filtrar as operações que possuem complexidade alta devido aos desvios (IF, FOR,

WHILE). Por fim, utilizamos a métrica NCO para restringir ao número de interesses existentes em cada operação retornando assim o *God Method*. Nesta heurística são apresentados os valores limites para cada métrica formulados com base no sistema MobileMedia.

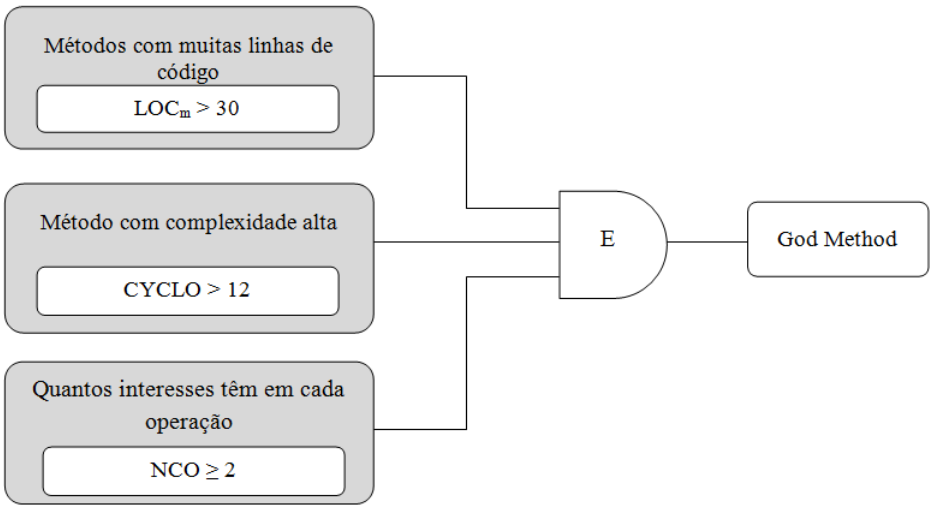

Figura 5.8. Regra Heurística Primária para Anomalia God Method.

Heurística Secundária. A heurística secundária foi criada como uma possível heurística alternativa. Diante disso, criou-se a heurística secundária para o *God Method* utilizando a combinação das métricas LOCC$_m$ com CYCLO e com NCO. A Figura 5.9 ilustra a composição da segunda heurística para o *God Method*. A métrica LOCC$_m$ foi selecionada, pois conta quantas linhas de código implementam um interesse. Um método que implementa um interesse muito grande está sujeito a ser um *God Method*. Em seguida, utilizamos a métrica CYCLO para filtrar os valores retornados pela LOCC$_m$, pois calcula a complexidade dos métodos. Métodos com muitos desvios (IF, FOR e WHILE) pode ser indício de método que centraliza muitas funcionalidades. Por último, restringimos os valores de CYCLO à métrica NCO, pois ela calcula o número de interesses existentes em cada operação. Um método que implementa muitos interesses distintos pode indicar que é um método propício a ter esta anomalia.

Na heurística da Figura 5.9 é apresentado os valores limites para cada métrica que foram elaborados com base no sistema MobileMedia. Por isso, estes valores limites devem ser ajustados de acordo com o tamanho do sistema. Assim, para alguns domínios ou sistema específicos, é esperado que a heurística secundária possa revelar mais anomalias que a heurística primária. Uma avaliação desta heurística deve ser realizada para comprovar-se a sua eficácia.

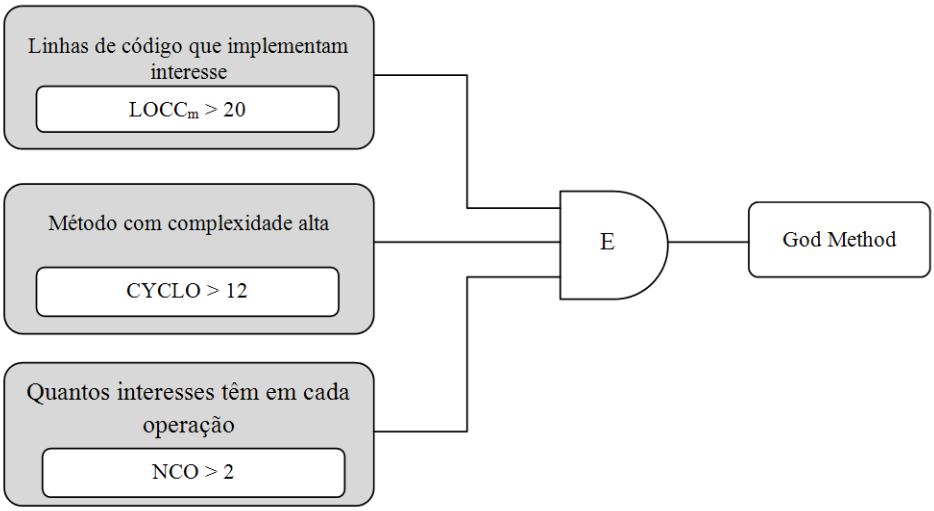

Figura 5.9. Regra Heurística Secundária para Anomalia God Method.

5.4 Ferramenta

Atualmente, existem muitas ferramentas (Together[1] e CodePro[2]) disponíveis no mercado que realizam a coleta de métricas em software e detectam anomalias de código. Entretanto, tais ferramentas não foram desenvolvidas utilizando métricas que já foram avaliadas em experimentos com participantes. Diante disso, foi desenvolvida a ferramenta ConcernMeBS que utiliza um conjunto de métricas tradicionais e de interesse obtidas como úteis nos experimentos realizados.

A detecção de anomalias de código é uma tarefa tediosa sem o suporte de uma ferramenta. Esta seção descreve ConcernMeBS, uma ferramenta que detecta anomalias de código em programas Java. Porém, é uma detecção parcialmente automática, pois depende do mapeamento de interesses que é feito manualmente. Esta ferramenta foi implementada como um plugin do Eclipse[3]. Ela automatiza as regras heurísticas apresentadas nas Seções 5.2 e 5.3 para detectar as anomalias *Divergent Change*, *Shotgun Surgery*, *God Class* e *God Method*. A Seção 5.4.1 apresenta as principais funcionalidades desta ferramenta. A Seção 5.4.2 mostra algumas telas que estão relacionadas à interface com o usuário.

[1]http://borland.com/products/Together/
[2]http://loose.upt.ro/reengineering/research/codepro
[3]Eclipse Project: http://www.eclipse.org/

5.4.1 Implementação

ConcernMeBS tem cerca de 5 KLOC. Ele depende de três plugins: ConcernMapper [Robillard & Weigand-Warr, 2005], ConcernMorph [Figueiredo et al., 2009] e Metrics Plugin[4]. ConcernMapper é utilizado pelo usuário para mapear os interesses existentes em elementos sintáticos, como as classes, métodos e atributos. ConcernMorph possui um Coletor de Métricas que é responsável por computar as métricas de interesse. E por último, o Metrics Plugin é utilizado para coletar os valores das métricas tradicionais. Para facilitar a observância da falta de coesão de uma classe realizamos uma pequena modificação na métrica LCOM desta ferramenta, cuja seu valor variava entre 0 e 1. Foi feita a multiplicação do valor desta métrica por 100 e o arredondamento do resultado.

ConcernMeBS, incluiu três novas métricas além das disponíveis em ConcernMorph e Metrics Plugin: duas de interesse e uma tradicional. As métricas de interesse são: Número de Interesse por Componentes (NCC) e Número de Interesse por Operações (NCO). A métrica tradicional incluída foi Acoplamento entre Objetos (CBO). Além disso, foi implementado um módulo para definir as regras heurísticas. As regras heurísticas combinam números relacionados às métricas como os valores limites que são ajustáveis de acordo com a preferência do usuário. O objetivo é determinar automaticamente se há instâncias de anomalias de código no projeto analisado. A versão atual do ConcernMeBS apoia a detecção de quatro anomalias (*Divergent Change*, *Shotgun Surgery*, *God Class* e *God Method*), mas este número pode ser facilmente aumentado usando pontos de extensão da ferramenta.

5.4.2 Interfaces de Usuário

Para ilustrar a interação do usuário com a ferramenta ConcernMeBS foi utilizado o sistema MobileMedia. A Figura 5.10 mostra a tela da identificação das seguintes anomalias de código: *Divergent Change*, *Shotgun Surgery*, *God Class* e *God Method*. A ferramenta além de mostrar as anomalias existentes no sistema, também indica em qual classe/método e em qual arquivo ela se encontra. Um duplo clipe sobre a anomalia escolhida exibe os valores que compõem a heurística utilizada para detectar esta anomalia.

Esta ferramenta utiliza-se de uma página de preferência para mostrar ao usuário as regras heurísticas e seus valores limites ajustáveis de acordo com a preferência do usuário. A Figura 5.11 ilustra esta página de preferência. Os valores limites que aparecem para cada métrica são os valores sugeridos para o MobileMedia. Portanto,

[4]Metrics Plugin: http://metrics.sourceforge.net

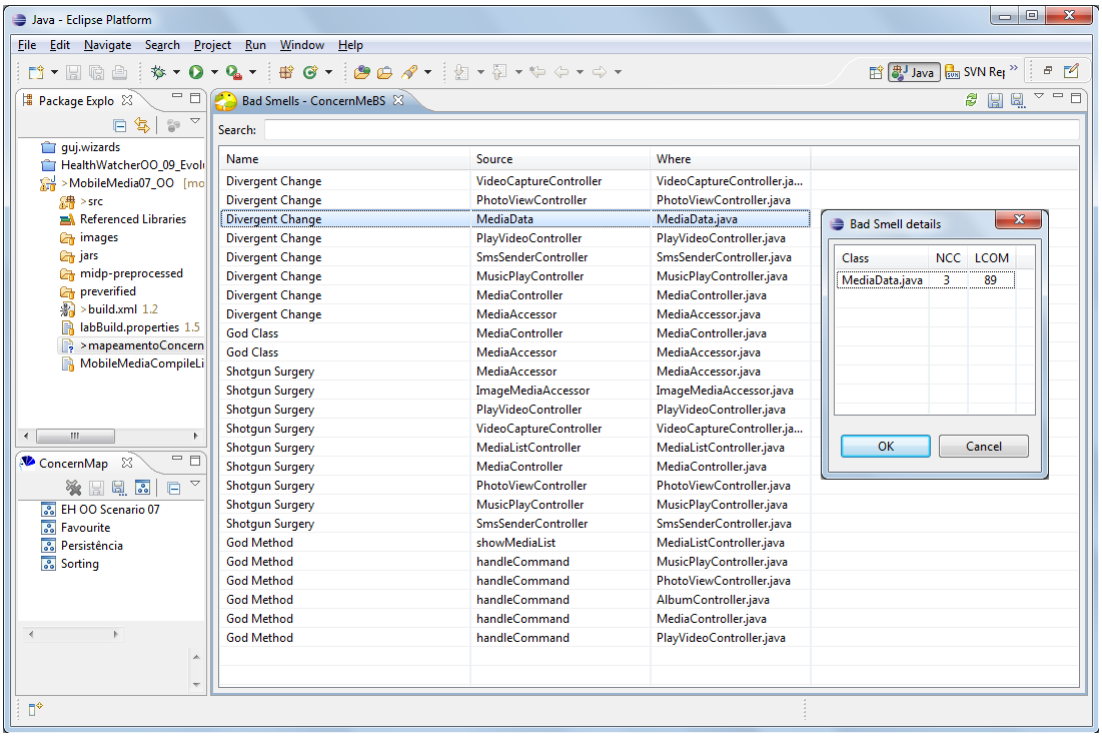

Figura 5.10. Tela Principal do ConcernMeBS.

para sistemas maiores é necessário ajustar estes valores para um melhor desempenho desta ferramenta. Na verdade, o usuário da ferramenta pode experimentar vários valores limites para um sistema de forma interativa para filtrar o número de candidatas a anomalias. Por exemplo, se um valor limite resultar em um número muito grande de anomalias candidatas, basta aumentar o filtro. O Anexo I apresenta os passos para a instalação do ConcernMeBS.

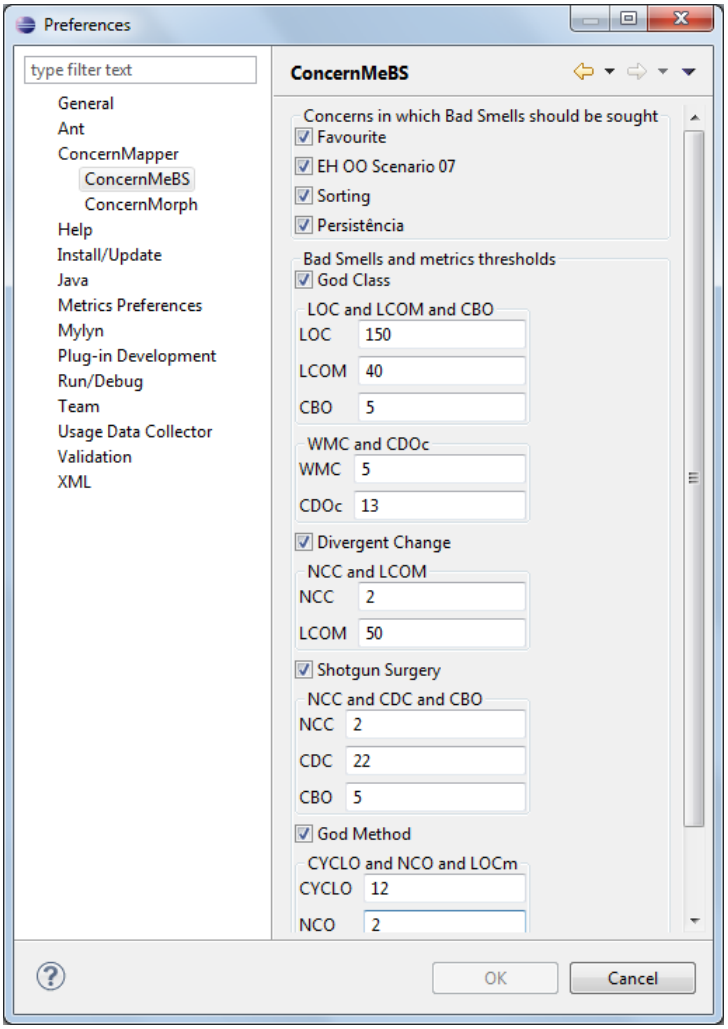

Figura 5.11. Tela da Página de Preferência.

5.5 Considerações Finais

As regras heurísticas que foram apresentadas neste capítulo foram obtidas em experimentos realizados com participantes. Nesses experimentos coletamos as métricas que foram utilizadas por eles, analisamos e formamos essas heurísticas. Foram propostas duas regras heurísticas (i) Heurística Primária: foi elaborada através das métricas reportadas como úteis pelos participantes e (ii) Heurística Secundária: foi a contribuição deste trabalho. Para exemplificar estas heurísticas utilizamos uma notação gráfica de circuitos elétricos para tornar mais clara à compreensão.

As combinações de métricas formadas para a Heurística Secundárias são as seguintes: (i)NCC e CDO_C para o *Divergent Change*, (ii) NCC e $CDLOC_C$ para o *Shotgun Surgery*, (iii) NCC e CDO_C e WMC para o *God Class* e (iv) $LOCC_m$ e CYCLO e NCO para o *God Method*. Acredita-se que estas regras possibilitam detectar anomalias de

código. Pois, foram elaboradas com base na definição de cada anomalia específica. Este capítulo também apresentou uma ferramenta - ConcernMeBS - para apoiar a detecção de anomalias de código. Esta ferramenta foi implementada como um *plugin* do Eclipse.

Capítulo 6

Ameaças à Validade e Trabalhos Relacionados

Quando um experimento é projetado inadequadamente, pode levar o pesquisador a obter uma conclusão errônea sobre a hipótese que está sendo estudada. Diante disso, apresentamos na Seção 6.1 uma lista dos tipos de ameaças à validade que nosso estudo exploratório está propício a ter. Em seguida, a Seção 6.2 apresenta os trabalhos existentes na literatura relacionados a este trabalho de mestrado. Por último, na Seção 6.3 são apresentadas as considerações finais deste capítulo.

6.1 Ameaças à Validade

A questão fundamental a respeito dos resultados do experimento é o quanto válidos eles são. Os resultados devem ser válidos para a população da qual o conjunto de participantes foi envolvido. É interessante também, generalizar os resultados para uma população mais ampla. Os resultados possuem a validade adequada se são válidos para a população, para a qual tendem ser generalizados.

Existe uma lista de ameaças à validade dos resultados do experimento: a validade de conclusão, a validade interna, a validade de construção e a validade externa [Wohlin et al., 2012]. Nem todas as ameaças à validade são aplicáveis a todos os experimentos, mas esta lista pode ser vista com uma lista de verificação.

Validade de Conclusão. Preocupa-se com as questões que afetam a capacidade de se chegar a conclusão correta sobre as relações entre o tratamento e o resultado de um experimento [Wohlin et al., 2012]. Durante a avaliação da validade de conclusão é necessário considerar conceitos como: a correta análise e interpretação estatística do

resultado (por exemplo, teste estatístico versus tamanho da amostra), confiabilidade das medidas e confiabilidade da implementação dos tratamentos.

A ameaça de conclusão pode vir a ocorrer em nosso estudo exploratório. Pois, apesar de termos uma amostra significativa para a análise estatística, não realizamos um teste estatístico rigoroso. Apenas usamos duas métricas, denominadas: Cobertura e Precisão para realizar a análise dos dados.

Validade Interna. Esta ameaça define se o relacionamento observado entre o tratamento e o resultado é causal, e não é resultado da influência de outro fator (não controlado ou medido) [Wohlin et al., 2012]. Algumas ameaças a este tipo de validade são descritas a seguir:

- História. Em um experimento, diferentes tratamentos podem ser aplicados ao mesmo objeto em tempos diferentes. Então, existe um risco que a história afete os resultados do experimento, desde a circunstâncias que não são as mesma em ambas as ocasiões [Wohlin et al., 2012].

- Maturação. Os participantes podem tornar-se mais capazes ou ficarem desmotivados com o passar do tempo [Wohlin et al., 2012]. Como por exemplo, quando os participantes são afetados negativamente (cansado ou chateado) durante o experimento, ou positivamente (aprendem) durante o treinamento do experimento.

- Testagem. Se o teste é repetido, os participantes podem responder de formas de diferentes desde que eles saibam como o teste é conduzido. Se não precisa de uma familiarização com o teste, é importante que o resultado dos testes não tenham um retorno para os participantes [Wohlin et al., 2012].

- Instrumentação. Este efeito é causado por artefatos usados para executar experimentos, tais como formulários de coletas de dados, documentos para serem inspecionados em uma inspeção de experimento [Wohlin et al., 2012].

- Seleção. Dependendo de como os participantes são selecionados, o efeito da seleção pode variar e influenciar nos resultados. Portanto, os participantes devem ser selecionados de maneira aleatória ou os grupos divididos de maneira igualitária tanto no aspecto quantitativo quanto qualitativo [Wohlin et al., 2012].

- Estatística Regressiva. Esta ameaça ocorre quando os participantes são classificados dentro de grupos experimentais baseando-se nas previsões do experimento ou do estudo de caso [Wohlin et al., 2012].

Um fator de variação do experimento é a forma de como é feito o mapeamento dos interesses. Assim, a acurácia das métricas de interesse poderia depender da precisão do mapeamento entre cada interesse e os elementos de código. Felizmente, nós observamos em um estudo anterior [Figueiredo et al., 2011] que, com exceção da métrica Espalhamento de Interesses em Linhas de Código (CDLOC), o processo de mapeamento não afeta significativamente as métricas de interesses avaliadas neste trabalho. Além disso, a fim de suavizar esta ameaça, contamos com mapeamentos de interesses produzidos pelos próprios desenvolvedores dos sistemas utilizados no estudo.

Validade de Construção. Esta ameaça considera os relacionamentos entre a teoria e a observação, ou seja, se o tratamento reflete a causa bem e o resultado reflete o efeito bem [Wohlin et al., 2012]. As ameaças mais comuns a este tipo de validade estão relacionadas com:

- Projeto do experimento. Esta ameaça cobrem questões de validade que estão relacionados com a concepção da experiência e de modo a refletir a construção a ser estudada [Wohlin et al., 2012].

- Ameaças sociais para a construção da validade. Esta ameaça está preocupada com questões relacionadas ao comportamento dos participantes e dos experimentadores. Eles podem, com base no fato de que eles fazem parte de uma experiência, agirem de maneira diferente o que dá resultados falsos ao experimento [Wohlin et al., 2012].

A ameaça de construção pode ocorrer em nosso experimento, apesar de termos discutido diversas vezes o projeto do experimento. Para minimizar as ameaças sociais, fizemos o experimento em quatro instituições em dois países.

Validade Externa. São condições que limitam nossa capacidade de generalizar os resultados do nosso experimento para a prática industrial [Wohlin et al., 2012]. Existem três tipos de interações com o tratamento: participantes, lugares e tempo:

- Interação de seleção e tratamento. Este é um efeito de ter uma amostra de participantes, não representativa da população que queremos generalizar [Wohlin et al., 2012]. Um exemplo desta ameaça é selecionar apenas os programadores em um experimento de inspeção, quando os programadores, bem como testadores e engenheiros de sistemas em geral já participam de inspeções.

- Interação de configuração e tratamento. Este efeito não tem configuração de experimento ou material representativo, como por exemplo, práticas industriais [Wohlin et al., 2012]. Um exemplo é conduzir um experimento de brincadeira.

- Interação de história e tratamento. Este efeito ocorre quando o experimento é conduzido em um dia ou hora especial, podendo influenciar nos resultados [Wohlin et al., 2012]. Como por exemplo, um questionário sobre segurança de um sistema é conduzido dias depois de uma grande falha na segurança de um software, pessoas tendem a responder de forma diferentemente do que responderiam dias antes.

Quando as ameaças à validade externa são reduzidas, tornam o ambiente experimental o mais realista possível. Por outro lado, a realidade não é homogênea. O mais importante é caracterizar e relatar as características do ambiente, como a experiência pessoal, ferramentas, métodos, a fim de avaliar a aplicabilidade em um contexto específico [Wohlin et al., 2012]. Diante disso, tentamos minimizar estas ameaças trabalhando com participantes randômicos de vários grupos. Trabalhamos com programadores novatos e mais experientes, como também, alunos de graduação e de pós-graduação.

6.2 Trabalhos Relacionados

As métricas têm sido historicamente utilizadas para detectar anomalias de código [Lanza & Marinescu, 2006; Marinescu, 2004]. Marinescu [Marinescu, 2004] propôs a utilização de estratégias compostas de métricas tradicionais para a detecção de anomalias de código. Ele observou que várias métricas são necessárias para capturar todos os fatores que definem anomalias. A avaliação da sua abordagem de detecção indica uma acurácia de cerca de 60% para a maior parte das anomalias de código. Ele baseou-se em várias métricas tradicionais também utilizadas neste estudo, mas não usou métricas de interesses.

Vários estudos têm utilizado as métricas tradicionais e de interesses para avaliar atributos de software, tais como manutenibilidade [Sant'Anna et al., 2003], instabilidade [Figueiredo et al., 2008; Greenwood et al., 2007] e propensão a erros [Eaddy et al., 2008; Ferrari et al., 2010]. Alguns desses estudos [Figueiredo et al., 2008; Greenwood et al., 2007] dependem de métricas de interesse para apoiar a comparação das decomposições orientadas a aspectos [Kiczales et al., 1997] e orientadas a objetos. Ao contrário deste trabalho de mestrado, esses estudos assumem implicitamente que as métricas de interesses são indicadoras confiáveis do atributo de qualidade. Esta pesquisa, por outro lado, destina-se a verificar se as métricas de interesses podem ser adequadas para detectar anomalias.

Eaddy e seus colegas [Eaddy et al., 2008] realizaram três experimentos para avaliar a utilidade de métricas de interesse na identificação de módulos propícios a erros. O experimento deles utilizou seis métricas de interesses, duas delas foram usadas em

nosso experimento, denominadas Espalhamento de Interesses em Componentes (CDC) e Espalhamento de Interesses em Operações (CDO). Eles encontraram uma moderada a forte correlação entre as métricas de interesses e defeitos nos módulos para todos os três experimentos. O objetivo de nosso estudo é diferente, devido ao fato de que não estamos focados na análise de erros propícios. Nosso trabalho complementa e amplia descobertas de Eaddy.

Em uma iniciativa preliminar, foi realizado um estudo piloto [Sant'Anna et al., 2008] abordando um subconjunto das metas estabelecidas para este trabalho de dissertação. O trabalho de Sant'Anna [2008] envolveu 30 participantes que detectaram as anomalias *Divergent Change* e *Shotgun Surgery* no sistema Health Watcher. Este trabalho de mestrado amplia o trabalho anterior, realizando novas experiências, onde se analisam os dados de dois sistemas (Health Watcher e MobileMedia) envolvendo 54 participantes. Além disso, também incluímos *God Class* na atualização da lista de anomalias. Além das anomalias de classes, realizamos também um estudo que envolveu 47 participantes para detectar anomalias em métodos (*Feature Envy* e *God Method*) utilizando-se de um sistema (MobileMedia). Os resultados obtidos neste trabalho relatam várias novas descobertas sobre a acurácia das métricas de interesses em relação ao trabalho anterior de Sant'Anna [Sant'Anna et al., 2008].

6.3 Considerações Finais

As conclusões obtidas aqui são restritas às métricas envolvidas, anomalias e sistemas utilizados. Após uma análise das ameaças à validade apresentadas anteriormente, constatamos que os nossos estudos experimentais estão propícios a terem todas as ameaças. Pois, apesar de termos uma amostra significativa para análise, não realizamos um teste estatístico rigoroso dos dados coletados nos experimentos.

Outro fator de variação do experimento é que apesar de ter havido uma discussão sobre o projeto e aplicação do experimento em instituições e países distintos, mesmo assim ele está sujeito a sofrer ameaças. Reconhecemos estas ameaças, mas elas são típicas de estudos como o nosso. Entretanto, o nosso estudo preenche a lacuna na literatura relatando a análise original sobre o uso de métricas de interesse para a detecção de anomalias. Além disso, esta dissertação descreveu a estrutura experimental do presente estudo que pode ser usada em outras aplicações experimentais.

Capítulo 7

Conclusões e Trabalhos Futuros

A avaliação da manutenibilidade do software é o ponto crítico na Engenharia de Software, pois ela é largamente dependente da disponibilidade de métricas que detectem com acurácia as anomalias de código. Diversos estudos empíricos [Conejero et al., 2012; Figueiredo et al., 2008; Garcia et al., 2005; Greenwood et al., 2007] usam a eficácia das métricas de interesse para avaliar a qualidade e manutenibilidade de software. Entretanto, estes estudos não avaliam diretamente a acurácia das métricas de interesse para detecção destas anomalias. Por isso, nesta dissertação, foi feita uma avaliação das métricas de interesse com 101 participantes para este fim. Neste capítulo apresentamos as conclusões obtidas nesta dissertação (Seção 7.1) e os trabalhos a serem realizados em futuros desdobramentos desta pesquisa (Seção 7.2).

7.1 Conclusões

Dois estudos experimentais com propósitos semelhantes foram realizados no contexto deste trabalho de mestrado. O primeiro estudo envolveu anomalias em classes, onde os resultados obtidos revelaram que as métricas de interesse são claramente úteis para detectar a anomalia *Divergent Change*. As métricas de interesse também se mostraram boas quando combinadas, com as métricas tradicionais para detectar as anomalias *God Class* e *Shotgun Surgery*. No segundo estudo, que envolveu anomalias em métodos, os resultados obtidos revelaram que as métricas de interesse, em especial a Número de Interesses por Operações (NCO), é útil para detectar a anomalia *God Method* quando combinada com métricas tradicionais. No entanto, nenhuma das métricas tradicionais e de interesse avaliadas foram consideradas úteis para detectar *Feature Envy*. Em geral, os resultados indicaram que a acurácia de cada conjunto de métricas é largamente dependente da adequação de cada métrica, para assim quantificar uma propriedade

explicitamente mencionada na definição da anomalia.

Com o resultado destes dois estudos foi feito uma análise destas métricas para compor as regras heurísticas. Assim, as heurísticas foram compostas por métricas reportadas como úteis pelos participantes. Já que uma métrica sozinha não é eficaz na captura de uma anomalia, isto requer combinações de métricas. Com o propósito de automatizar a detecção de anomalias de código desenvolveu-se uma ferramenta, denominada ConcernMeBS, que implementa as heurísticas para apoiar a detecção de quatro anomalias *Divergent Change*, *Shotgun Surgery*, *God Class* e *God Method*.

Dentro deste contexto, as contribuições deste trabalho de mestrado foram as seguintes:

1. Foram empiricamente observados e documentados dois estudos que mostram que métricas de interesse podem ser eficazes para detectar algumas anomalias de código, como: *Divergent Change*, *God Class*, *Shotgun Surgery* e *God Method*.
2. Foi proposta uma nova métrica, Número de Interesse por Operações (NCO), para auxiliar a medição de propriedades de interesses em métodos.
3. Foi proposto um método quantitativo pela composição de regras heurísticas para apoiar a detecção das anomalias de código.
4. Foi desenvolvido uma ferramenta para apoiar a detecção automática das anomalias estudadas.

A realização dos estudos experimentais serviu como uma primeira avaliação da eficácia das métricas de interesse para detectar anomalias de código. Apesar de não poderem ser generalizados, os resultados dos dois estudos experimentais serviram como indicação de que as métricas de interesse são boas para identificar algumas anomalias de código. Porém, existe uma carência destas métricas para capturar todas as propriedades das anomalias de código classificadas pela literatura.

7.2 Trabalhos Futuros

O trabalho apresentado nesta dissertação pode ser continuado, entre outras coisas, com a realização das seguintes atividades:

- Realização de um experimento envolvendo teste estatístico rigoroso. Atualmente, utilizamos apenas duas métricas, Cobertura e Precisão, suportadas por estatística descritivas para realizar a análise dos dados. A generalização dos resultados usando estatística indutiva requer novas reaplicações do estudo.

- Selecionar outros conjuntos de métricas de software para identificar as anomalias. Em particular, a anomalia *Feature Envy* não pode ser adequadamente detectada pelas métricas utilizadas neste trabalho.
- Realização de outros estudos experimentais envolvendo não apenas métricas, mas também regras heurísticas para avaliar a sua eficiência na detecção das anomalias de código.
- Definição de novas métricas de interesse para apoiar identificação de anomalias de código que não podem ser facilmente detectadas pelas métricas existentes.
- Melhoria da ferramenta ConcernMeBS. Por exemplo, pela inclusão de outras métricas e melhoria da interface.

Referências Bibliográficas

Baggen, R.; Schill, K. & Visser, J. (2010). Standardized code quality benchmarking for improving software maintainability. Em *4th International Workshop on Software Quality and Maintainability, March 15, 2010, Madrid, Spain.*

Basili, V. R.; Briand, L. C. & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22:751–761.

Basili, V. R. & Rombach, H. D. (1988). The tame project: Towards improvement-oriented software environments. *IEEE Trans. Software Eng.*, 14(6):758–773.

Boehm, B. W.; Brown, J. R.; Kaspar, H.; Lipow, M.; Macleod, G. J. & Merrit, M. J. (1978). *Characteristics of Software Quality. Vol. 1.* TRW series of software technology. Elsevie, Amsterdam, Lausanne, New York.

Bunge, M. (1979). Treatise on basic philosophv: Ontology. Em *I: The World of Systems.*

Carneiro, G. d. F.; Silva, M.; Mara, L.; Figueiredo, E.; Sant Anna, C.; Garcia, A. & Mendonca, M. (2010). Identifying code smells with multiple concern views. Em *Proceedings of the 2010 Brazilian Symposium on Software Engineering*, SBES 10, pp. 128–137, Washington, DC, USA. IEEE Computer Society.

Chidamber, S. R. & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20:476–493.

Conejero, J. M.; Figueiredo, E.; Garcia, A.; Hernandez, J. & Jurado, E. (2012). On the relationship of concern metrics and requirements maintainability. *Inf. Softw. Technol.*, 54(2):212–238.

Dijkstra, E. W. (1976). *A Discipline of Programming.* Prentice-Hall.

Ducasse, S.; Girba, T. & Kuhn, A. (2006). Distribution map. Em *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, ICSM 06, pp. 203–212, Washington, DC, USA. IEEE Computer Society.

Eaddy, M.; Zimmermann, T.; Sherwood, K. D.; Garg, V.; Murphy, G. C.; Nagappan, N. & Aho, A. V. (2008). Do crosscutting concerns cause defects? *IEEE Trans. Softw. Eng.*, 34:497–515.

Fenton, N. (1994). Software measurement: A necessary scientific basis. *IEEE Trans. Softw. Eng.*, 20(3):199–206.

Fenton, N. E. & Pfleeger, S. L. (1996). *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 2nd edição.

Ferrari, F.; Burrows, R.; Lemos, O.; Garcia, A.; Figueiredo, E.; Cacho, N.; Lopes, F.; Temudo, N.; Silva, L.; Soares, S.; Rashid, A.; Masiero, P.; Batista, T. & Maldonado, J. (2010). An exploratory study of fault-proneness in evolving aspect-oriented programs. Em *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE 10, pp. 65–74, New York, NY, USA. ACM.

Figueiredo, E.; Cacho, N.; Sant Anna, C.; Monteiro, M.; Kulesza, U.; Garcia, A.; Soares, S.; Ferrari, F.; Khan, S.; Castor Filho, F. & Dantas, F. (2008). Evolving software product lines with aspects: an empirical study on design stability. Em *Proceedings of the 30th international conference on Software engineering*, ICSE 08, pp. 261--270, New York, NY, USA. ACM.

Figueiredo, E.; Garcia, A.; Maia, M.; Ferreira, G.; Nunes, C. & Whittle, J. (2011). On the impact of crosscutting concern projection on code measurement. Em *Proceedings of the tenth international conference on Aspect-oriented software development*, AOSD 11, pp. 81–92, New York, NY, USA. ACM.

Figueiredo, E.; Sant'Anna, C.; Garcia, A. & de Lucena, C. J. P. (2012). Applying and evaluating concern-sensitive design heuristics. *Journal of Systems and Software*, 85(2):227–243.

Figueiredo, E.; Whittle, J. & Garcia, A. (2009). Concernmorph: metrics-based detection of crosscutting patterns. Em *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE 09, pp. 299–300, New York, NY, USA. ACM.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA.

Gamma, E.; Helm, R.; Johnson, R. E. & Vlissides, J. M. (1994). Design patterns: Elements of reusable object-oriented software. Addison-Wesley.

Garcia, A.; Sant'Anna, C.; Figueiredo, E.; Kulesza, U.; Lucena, C. & von Staa, A. (2005). Modularizing design patterns with aspects: a quantitative study. Em *Proceedings of the 4th international conference on Aspect-oriented software development*, AOSD 05, pp. 3--14, New York, NY, USA. ACM.

Greenwood, P.; Bartolomei, T. T.; Figueiredo, E.; Dosea, M.; Garcia, A. F.; Cacho, N.; Sant'Anna, C.; Soares, S.; Borba, P.; Kulesza, U. & Rashid, A. (2007). On the impact of aspectual decompositions on design stability: An empirical study. Em Ernst, E., editor, *ECOOP*, volume 4609 of *Lecture Notes in Computer Science*, pp. 176–200. Springer.

Gresse, C., R. D. & Ruhe, G. (1996). A practical approach for building gqm-based measurement programs–lessons learned from three industrial case studies. Em *Proceedings of the 10th Brasilian Symposium on Software Engineering.*

Hannemann, J. & Kiczales, G. (2002). Design pattern implementation in java and aspectj. *SIGPLAN Not.*, 37(11):161–173.

Harrison, R., C. S. J. & Nithi, R. V. (1998). An evaluation of the mood set of object-oriented software metrics. *IEEE Trans. Softw. Eng.*, 24(6):491–496.

Humphrey, W. S. & Kellner, M. I. (1989). Software process modeling: principles of entity process models. Em *Proceedings of the 11th international conference on Software engineering*, ICSE 89, pp. 331–342, New York, NY, USA. ACM.

Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; marc Loingtier, J. & Irwin, J. (1997). Aspect-oriented programming. Em *ECOOP*. SpringerVerlag.

Kitchenham, B.; Pfleeger, S. L. & Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929–944.

Lanza, M. & Marinescu, R. (2006). *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Li, W. & Henry, S. (1993). Maintenance metrics for the object-oriented paradigm. Em *Proc. IEEE Symp. Software Metrics*, pp. 52–60.

Li, W. & Shatnawi, R. (2007). An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. *J. Syst. Softw.*, 80:1120–1128.

Lorenz, M. & Kidd, J. (1994). *Object-Oriented Software Metrics*. Prentice Hall.

Marinescu, R. (2002). Measurement and quality in object-oriented design.

Marinescu, R. (2004). Detection strategies: Metrics-based rules for detecting design flaws. Em *In Proc. IEEE International Conference on Software Maintenance*.

McCabe, T. J. (1976). A complexity measure. Em *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, Los Alamitos, CA, USA. IEEE Computer Society Press.

Mccall, J. A.; Richards, P. K. W. G. F. (1977). *Factors in Software Quality. Vol 1.* Technical Report ADA049014, GENERAL ELECTRIC CO SUNNYVALE CALIF. Concepts and Definitions of Software Quality.

Melton, A. C.; Gustafson, D. A.; Bieman, J. M. & Baker, A. L. (1990). A mathematical perspective for software measures research. *Software Engineering Journal*, 5:246–254.

Paula Filho, W. d. P. (2009). *Engenharia de Software:fundamentos, métodos e padrões*. Rio de Janeiro, Rio de Janeiro, 3 edição.

Pressman, R. S. (2011). *Engenharia de Software: uma abordagem profissional*. McGraw-Hill, Rio de Janeiro, 7 edição.

Riel, A. J. (1996). *Object-Oriented Design Heuristics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edição.

Robillard, M. P. & Murphy, G. C. (2007). Representing concerns in source code. *ACM Trans. Softw. Eng. Methodol.*, 16(1).

Robillard, M. P. & Weigand-Warr, F. (2005). Concernmapper: simple view-based separation of scattered concerns. Em *Proceedings of the 2005 OOPSLA workshop on Eclipse technology exchange*, eclipse 05, pp. 65–69, New York, NY, USA. ACM.

Sahraoui, H. A.; Godin, R. & Miceli, T. (2000). Can metrics help to bridge the gap between the improvement of oo design quality and its automation? Em *Proceedings of the International Conference on Software Maintenance*, ICSM 00, pp. 154 – 162, Washington, DC, USA. IEEE Computer Society.

Sant'Anna, C.; Garcia, A.; Chavez, C.; Lucena, C. & v. von Staa, A. (2003). On the reuse and maintenance of aspect-oriented software: An assessment framework. Em *In Proc. Of the Brazilian Symposium on Soft. Engineering*.

Sant'Anna, C.; Garcia, A. & Lucena, C. (2008). Evaluating the efficacy of concern-driven metrics: A comparative study. Em *In Proceedings of the 2nd Workshop on Assessment of Contemporary Modularization Techniques*.

Sant'Anna, C. N. (2004). Manutenibilidade e reusabilidade de software orientado a aspectos: Um framework de avaliação.

Schneidewind, N. F. (1992). Methodology for validating software metrics. *IEEE Trans. Softw. Eng.*, 18(5):410–422.

Silva, B. C. d.; Sant Anna, C.; Chavez, C. & Garcia, A. (2012). Concern-based cohesion: Unveiling a hidden dimension of cohesion measurement. Em *ICPC*, pp. 103–112.

Sommerville, I. (2011). *Engenharia de Software*. Pearson Addison-Wesley, 9ª edição.

Weyuker, E. J. (1988). Evaluating software complexity measures. *IEEE Trans. Softw. Eng.*, 14(9):1357–1365.

Wohlin, C.; Runeson, P.; Host, M.; Ohlsson, M. C.; Regnell, B. & Wesslen, A. (2012). *Experimentation in software engineering: an introduction*. Springer, Norwell, MA, USA.

Anexo A

Questionário de Histórico

Este anexo apresenta o questionário de histórico. Ele foi aplicado aos participantes para medirmos o nível de conhecimento de cada um deles.

Questionário

1. Nome:

2. Data de Nascimento:

3. Sobre sua experiência em *projeto de software e programação*.

a. Marque os cursos que você já fez, ou então se tais conteúdos foram dados em conjunto com algum dos cursos que você fez

[ ]	Sistemas de banco de dados	[ ]	Programação Java
[ ]	Programação orientada a objetos	[ ]	Projeto de software
[ ]	Tecnologia Web	[ ]	Gerência de projetos de sistemas
[ ]	Sistemas distribuídos	[ ]	Tópicos avançados de banco de dados
[ ]	Análise e projeto de software com UML	[ ]	Sistemas tolerantes a falhas

b. Você já trabalhou (ou está trabalhando) em empresa de desenvolvimento de sistemas? Se sim, por quanto tempo?

( ) Não, nunca trabalhei em empresa de desenvolvimento.
( ) Trabalhei até 1 ano.
( ) Trabalhei entre 1 e 3 anos.
( ) Trabalhei mais de 3 anos.

4. Como você classifica seu conhecimento em relação aos seguintes tópicos?

a. Diagrama de classes

( ) Experiente ( ) Pouco
( ) Moderado ( ) Nenhum

b. Linguagem de programação Java

( ) Experiente ( ) Pouco
( ) Moderado ( ) Nenhum

c. Medição de software

( ) Experiente ( ) Pouco
( ) Moderado ( ) Nenhum

Anexo B

Passos do Experimento

Este apêndice apresenta em detalhes os passos dos experimentos em anomalias de código, que foi aplicado aos participantes para os seguintes sistemas:

- MobileMedia para o experimento de Anomalias em Classes
- Health Watcher para o experimento de Anomalias em Classes
- MobileMedia para o experimento de Anomalias em Métodos

EXPERIMENTO EM CLASSES

Nomes: _____

_____

Atividades

Passo 1) Ouvir a explicação do exercício.
Passo 2) Somente após o término da explicação do exercício, comece a seguir os passos descritos a seguir.
Passo 3) Comece pela leitura do texto abaixo que descreve o software MobileMedia.

IMPORTANTE

Antes de começar a leitura, favor indicar que horas são agora:

____:____ ⇦

O MobileMedia – Descrição Textual

O MobileMedia (MM) é uma aplicação que inclui funcionalidades, como manipulação de fotos, músicas e vídeos, em dispositivos móveis (aparelhos celulares, smartphones, etc.). A Figura 1 mostra algumas das principais classes do MobileMedia em um diagrama UML. Este diagrama é na verdade uma versão simplificada do projeto desta aplicação. No MM, as funcionalidades são acessadas através da tela e teclado do dispositivo móvel. O projeto da aplicação é estruturado de tal forma a desacoplar suas diferentes partes. Assim, a parte responsável pela interface com o usuário (telas) está desacoplada da parte que implementa os controladores de funcionalidades. Esta última também se encontra desacoplada das classes do modelo. O objetivo deste desacoplamento é tornar alterações mais fáceis e independentes em cada uma das partes da aplicação.

A interação do usuário com o MobileMedia é feita por classes Screens, tais como MediaListScreen e PhotoViewScreen (Figura 1). O acesso aos serviços da aplicação é feito através de controladores que estendem uma classe abstrata chamada AbstractController. Uma dos requisitos do MobileMedia é permitir o uso de senhas de segurança em álbuns de mídia. Portanto, os usuários podem definir senhas individuais para cada álbum da aplicação. MediaData funciona como uma entidade do tipo mídia que pode ser tanto música quanto vídeos ou fotos (existem classes específicas para os tipos de mídia que não são apresentados na Figura 1). As classes Accessor (VideoAccesor, MusicAccessor e PhotoAccessor) persistem as mídias diretamente no dispositivo de armazenamento (como cartões de memória). Por outro lado, as classes AlbumData (VideoAlbumData, MusicAlbumData e PhotoAlbumData) oferecem uma fachada para acessos aos dados. Assim, estas classes desacoplam a lógica de negócio dos tipos específicos de dados armazenados. A Tabela 1 resume as principais responsabilidades associadas às classes do MM. Ademais, cinco requisitos refletem decisões importantes do projeto desta aplicação. Estes requisitos são: *Contagem e Ordenação*, *Definição de Favoritos*, *Tratamento de Exceções*, *Persistência e Segurança*. A Tabela 2 descreve estes requisitos. Classes que de alguma forma implementam estes requisitos no projeto do MM estão marcadas no diagrama da Figura 1. Por exemplo, classes que implementam Segurança são identificados pela letra ‘S’ no diagrama.

O Projeto MobileMedia

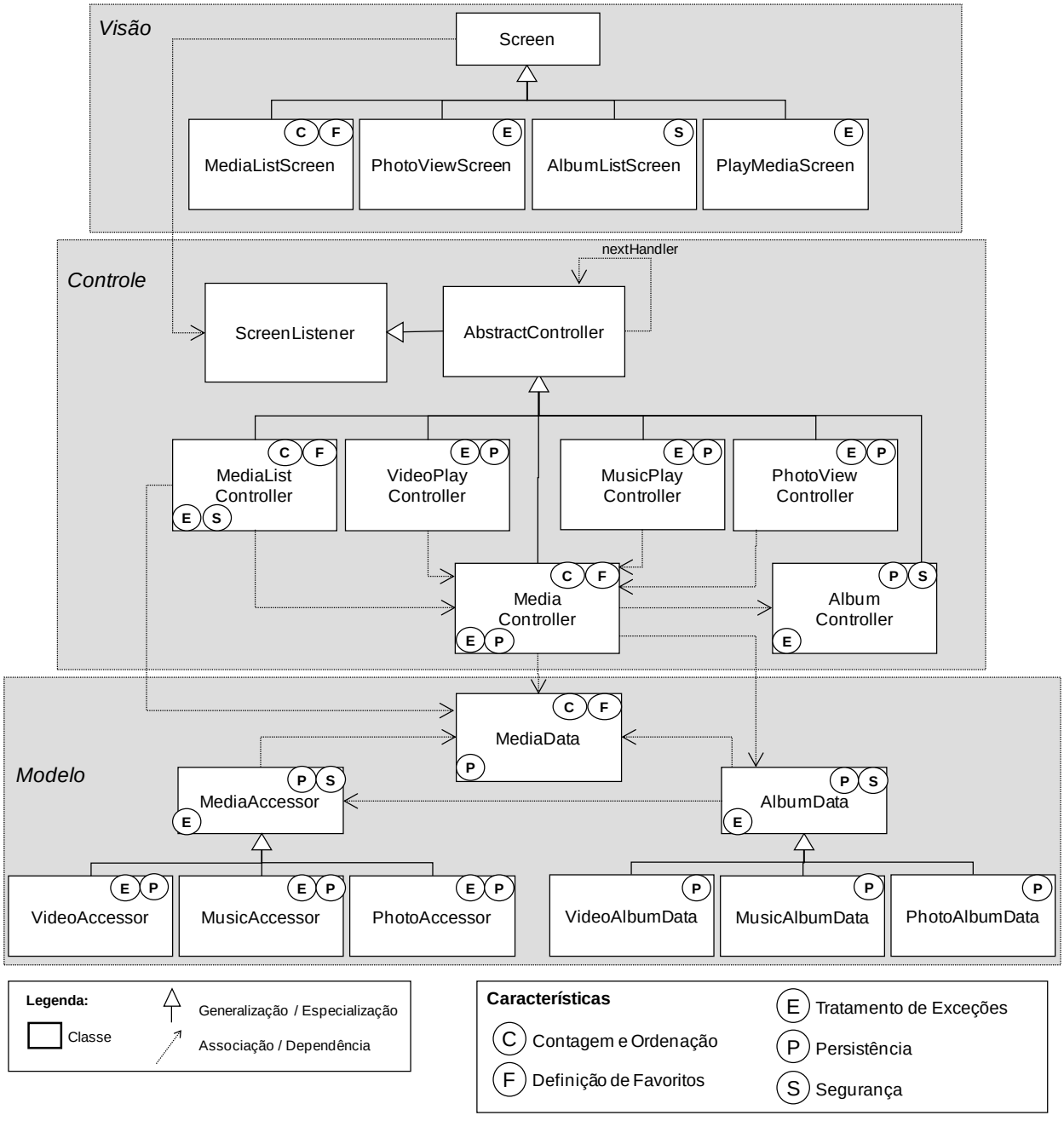

Figura 1. Visão de projeto parcial do MobileMedia

Tabela 1. Responsabilidade das classes do MobileMedia

Classes	O que elas fazem
Screen MediaListScreen PhotoViewScreen AlbumListScreen PlayMediaScreen	Estas classes implementam as telas de interface com o usuário.
AbstractController MediaListController VideoPlayController MusicPlayController PhotoViewController MediaController AlbumController	Estas classes controlam o acesso as funcionalidades da aplicação. Elas recebem os comandos associados às operações das telas e repassam às classes ao modelo.
MediaData	Esta classe representa os tipos básicos mídias (vídeo, música e foto). Em outras palavras, é a principal classe do domínio da aplicação.
AlbumData VideoAlbumData MusicAlbumData PhotoAlbumData	Estas classes represemtam abstrações de dados gravados em dispositivos de armazenamento. Por exemplo, as classes controladoras não sabem como os dados são armazenados, pois conhecem apenas os álbuns de mídia (abstrações).
MediaAccessor VideoAccessor MusicAccessor PhotoAccessor	Estas classes implementam efetivamente os serviços de gravação e recuperação de dados nos dispositivos de armazenamento. Nesta implementação do MobileMedia, os dados são persistidos na própria memória do dispositivo móvel.

Tabela 2. Alguns requisitos do MobileMedia

Requisitos	O que significa
Contagem e Ordenação	Este requisito permite contar o número de vezes que uma determinada mídia foi visualizada pelo usuário. Além disso, o usuário pode ordenar a lista de mídia de um álbum pelo número de visualizações.
Definição de Favoritos	Este requisito permite que o usuário defina uma determinada mídia como favorita. Assim, ele tem a opção de visualizar somente as mídias favoritas de um álbum.
Tratamento de Exceções	Este requisito se caracteriza por um mecanismo para tratar ocorrências que alteram o fluxo normal de execução. Muitas linguagens de programação, como Java, têm suporte explicito para levantamento e tratamento de exceções. O escopo deste requisito começa com uma cláusula de marcação (*try*) e termina com uma cláusula de tratamento (*catch*). Assim, **não** está relacionado ao requisito: (1) inicialização de uma exceção (comandos *throw*) e (2) sinalização de uma exceção para outros módulos (cláusulas *throws* em assinaturas de métodos).
Persistência	Este requisito se refere à habilidade da aplicação em reter dados entre execuções. Em outras palavras, ela é responsável por salvar e recuperar as informações manipuladas pela aplicação. Na implementação de persistência do MobileMedia, exceções específicas de persistência trafegam por outras classes da aplicação.
Segurança	Este requisito permite que senhas sejam associadas a álbuns de mídia. Assim, o usuário somente pode efetuar algumas operações, como visualizar e apagar álbuns protegidos, se a senha for fornecida.

IMPORTANTE

Indique abaixo que horas são agora: ____:____ ⇦

Passo 4) Os projetistas do sistema MobileMedia precisam de sua ajuda para identificar **quais classes** que contêm o Bad Smell "**Divergent Change**". Observe a descrição do Bad Smell e as métricas que lhes foram fornecidas. Lembre-se que é importante raciocinar sobre as métricas e identificar quais delas (uma, algumas ou todas) são indicadoras relevantes com base na descrição Bad Smell.

Passo 5) Responda às seguintes perguntas:

a) Quais são as classes com maior probabilidade de ter o Bad Smell "**Divergent Change**"? Você deve ordenar sua lista de classes; aquelas com maior probabilidade devem vir primeiro.

b) Explique quais métricas que você usou para detectar o Bad Smell "**Divergent Change**", e como elas foram úteis para identificar as classes mencionadas na pergunta anterior. Quais métricas não foram úteis?

IMPORTANTE

Indique abaixo que horas são agora: **____:____** ⇦

Passo 6) O objetivo agora é detectar a presença do Bad Smell "**Shotgun Surgery**". Então, identifique **quais classes** quando alteradas são mais propensas a propagar as alterações para outras classes. Comece a ler a descrição do Bad Smell e das métricas. Lembre-se que é importante raciocionar sobre as métricas e identificar quais delas (uma, algumas ou todas) são indicadores relevantes com base na descrição do Bad Smell.

Passo 7) Responda às seguintes perguntas:

a) Quais são as classes que (quando alteradas) tem a maior probabilidade de propagação de mudanças para outras classes? Você deve ordenar sua lista de classes; aquelas com maior probabilidade devem vir primeiro.

b) Explique quais as métricas que você usou para detectar o Bad Smell "**Shotgun Surgery**" e como elas foram úteis para identificar as classes mencionadas na pergunta anterior. Quais não foram úteis?

IMPORTANTE

Quando você terminar de responder esta pergunta, indique abaixo que horas são agora: **____:____** ⇦

EXPERIMENT IN CLASS

Group Members
1. _____
2. _____

Experimental Steps

Step 1) The group start to follow the guidelines and answer the questions. The answers are provided by the **group** (just one form per group).

Step 2) Start to read the description of the Health Watcher system provided below.
IMPORTANT: before you *start* to read the description, indicate here what time it is now: **_:_ _pm**

Health Watcher System – Design

The Health Watcher (HW) system allows a citizen to register complaints to the health public system. Figure 1 shows a diagram representing some of the classes and interfaces of the Health Watcher system. This diagram is actually a simplification of the HW design. In the HW system, complaints are registered, updated and queried through a Web client. The system is structured with the goal of decoupling different parts of the system in order to make it easy to change them separately. Therefore, the user interface part of the system is decoupled from the part which implements the business logic which, in turn, is also decoupled from the database management.

The user interface is represented by the Servlet classes, such as ServletInsertEmployee and ServletSearchComplaintData (Figure 1). Accesses to the HW services are made through the IFacade interface, which is implemented by the HealthWatcherFacade. One of the Health Watcher's requirements is to allow several customers to access the system at the same time. Therefore, a client-server approach is used to distribute part of the execution, which is implemented by the classes HealthWatcherFacade and the HealthWatcherFacadeInit. The HealthWatcherFacadeInit works as a portal to access the business collections, such as ComplaintRecord and EmployeeRecord. Records access the data also using interfaces, like IComplaintRepository, which decouple the business logic from the specific type of data management in use. Figure 1 shows the classes that serve the purpose of implementing a repository for a relational database: HealthUnitRepositotyRDB, EmployeeRepositoryRDB and ComplainRepositoryRDB. Table 1 summarizes the main responsibilities associated with each class in the HW system. Table 2 describes the main design concerns.

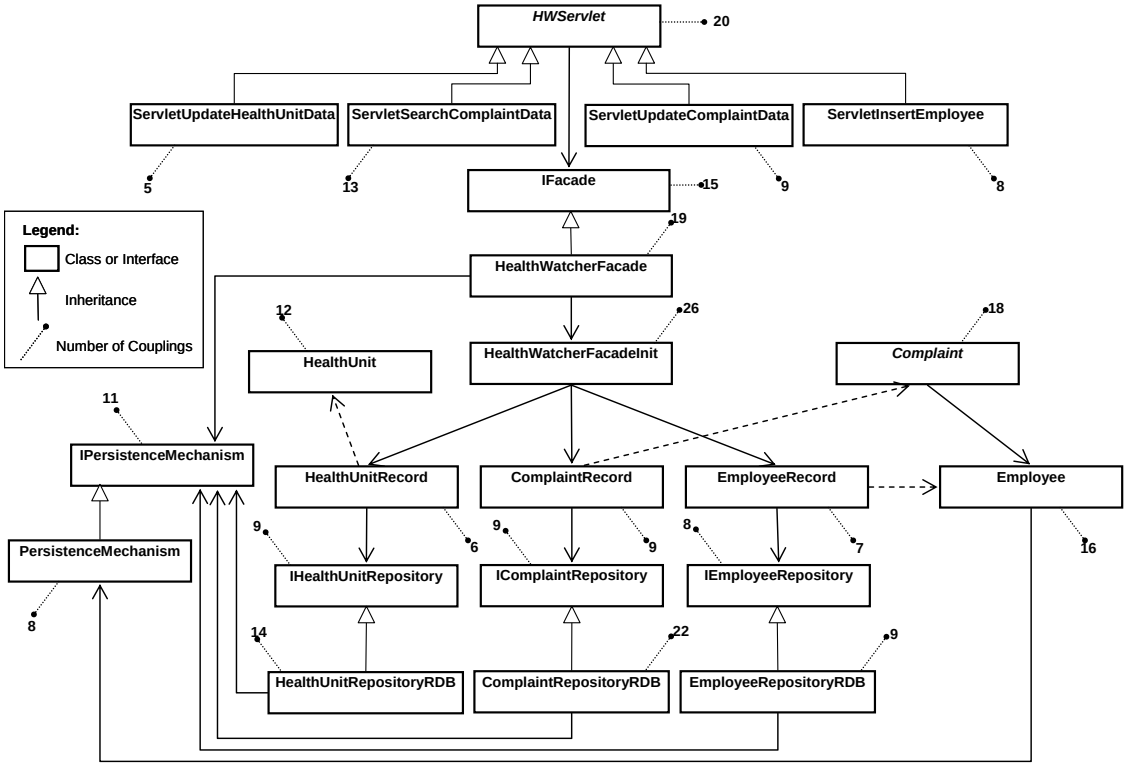

Figure 1. A Partial View of the HW Design

Table 1. Class Responsibilities

Classes	**What they do**
HWServlet ServletInsertEmployee, ServletSearchComplaintData, ServletUpdateComplaintData, ServletUpdateHealthUnitData	These classes implement the user interface of the system.
IFacade, HealthWatcherFacade, HealthWatcherFacadeInit	These classes provide a simple interface to all services of the system.
HeathUnit, Employee, Complaint	These classes represent business basic concepts of the Health Watcher system domain.
HealthUnitRecord, EmployeeRecord, ComplainRecord	These classes represent a grouping of objects from a basic class. For instance, EmployeeRecord groups objects of Employee.
HealthUnitRepositotyRDB, EmployeeRepositoryRDB, ComplainRepositoryRDB	These classes contain methods for manipulating persistent objects of the business basic classes. The code of these classes depends on a specific API for accessing some persistence platform.
IHealthUnitRepositoty, IEmployeeRepository, IComplainRepository	These interfaces establish a decoupled relationship between the “Record” classes and “RepositoryRDB” classes. The goal is to make the business code more independent from the data access code and from a specific persistence platform.
PersistenceMechanism	This class implements services such as connecting to and disconnecting from the database, transaction mechanism, concurrency mechanism.

Table 2. Main Design Concerns

Concern	Description
Business	It defines the business elements and rules.
Concurrency	It provides a control for avoiding inconsistency in the information manipulated by the system.
Distribution	It is responsible for externalizing the system services at the server side and supporting their distribution to the clients.
Exception Handling	Ii supports error recovery.
Persistence	It is responsible for storing the information manipulated by the system.
View (GUI)	It provides a Web interface for the system.

Step 3) The designers of the Health Watcher system need your help to identify **which classes** contain the bad smell "Divergent Change". Start to read the description of the bad smell and the measures. Remember that it is important to reason about the metrics and identify which of them (one, some, or all) are relevant indicators based on the bad smell description.

IMPORTANT: before you *start* to read the text and measures, indicate here what time it is now: **_:_ _pm**

Step 4) Your **group** should now answer the following questions:

a) Which are the classes with the highest probability of having the bad smell "Divergent Change"? You should rank your list of classes; the ones with highest probability should come first.

IMPORTANT: when you *finish* to answer this question, indicate here what time it is now: **_:_ _pm**

b) Explain which metrics you have used for detecting the bad smell, and how they were useful to identify the classes mentioned in the previous question. Which ones were not useful at all?

Step 5) The goal now is to detect the presence of "Shotgun Surgery" bad smells. So identify **which classes** when changed are more likely to propagate changes to other classes. Start to read the description of the bad smell and the measures. Remember that it is important to reason about the metrics and identify which of them (one, some, or all) are relevant indicators based on the bad smell description.

IMPORTANT: before you *start* to read the text and measures, indicate here what time it is now: **_:_ _pm**

Step 6) Your **group** should now answer the following questions:

a) Which are the classes (when changed) that have the highest probability of propagating changes to other classes? You should rank your list of classes; the ones with highest probability should come first.

IMPORTANT: when you *finish* to answer this question, indicate here what time it is now: **_:_ _pm**

b) Explain which metrics you have used for detecting the bad smell, and how they were useful to identify the classes mentioned in the previous question. Which ones were not useful at all?

Step 7) Answer the questionnaire about the experiment design issues.

EXPERIMENTO EM MÉTODOS

Nome: _____

Atividades

Passo 1) Ouvir a explicação do exercício.

Passo 2) Somente após o término da explicação do exercício, comece a seguir os passos descritos a seguir.

Passo 3) Comece pela leitura do texto abaixo que descreve o software MobileMedia.

IMPORTANTE

Antes de começar a leitura, favor indicar que horas são agora: ____:____ ⇦

O MobileMedia – Descrição Textual

O MobileMedia (MM) é uma aplicação que inclui funcionalidades, como manipulação de fotos, músicas e vídeos, em dispositivos móveis (aparelhos celulares, smartphones, etc.). A Figura 1 mostra algumas das principais classes do MobileMedia em um diagrama UML. Este diagrama é na verdade uma versão simplificada do projeto desta aplicação. No MM, as funcionalidades são acessadas através da tela e teclado do dispositivo móvel. O projeto da aplicação é estruturado de tal forma a desacoplar suas diferentes partes. Assim, a parte responsável pela interface com o usuário (telas) está desacoplada da parte que implementa os controladores de funcionalidades. Esta última também se encontra desacoplada das classes do modelo. O objetivo deste desacoplamento é tornar alterações mais fáceis e independentes em cada uma das partes da aplicação.

A interação do usuário com o MobileMedia é feita por classes Screens, tais como MediaListScreen e PhotoViewScreen (Figura 1). O acesso aos serviços da aplicação é feito através de controladores que estendem uma classe abstrata chamada AbstractController. MediaData funciona como uma entidade do tipo mídia que pode ser tanto música quanto vídeos ou fotos (existem classes específicas para os tipos de mídia que não são apresentados na Figura 1). As classes Accessor (VideoAccesor, MusicAccessor e PhotoAccessor) persistem as mídias diretamente no dispositivo de armazenamento (como cartões de memória). Por outro lado, as classes AlbumData (VideoAlbumData, MusicAlbumData e PhotoAlbumData) oferecem uma fachada para acessos aos dados. Assim, estas classes desacoplam a lógica de negócio dos tipos específicos de dados armazenados. A Tabela 1 resume as principais responsabilidades associadas às classes do MM. Além disso, quatro interesses refletem decisões importantes do projeto desta aplicação. Estes interesses são: *Contagem e Ordenação*, *Tratamento de Exceções*, *Definição de Favoritos* e *Persistência*. A Tabela 2 descreve estes interesses. Classes que de alguma forma implementam estes interesses no projeto do MM estão marcadas no diagrama da Figura 1. Por exemplo, classes que implementam *Persistência* são identificadas pela letra *'P'* no diagrama.

O Projeto MobileMedia

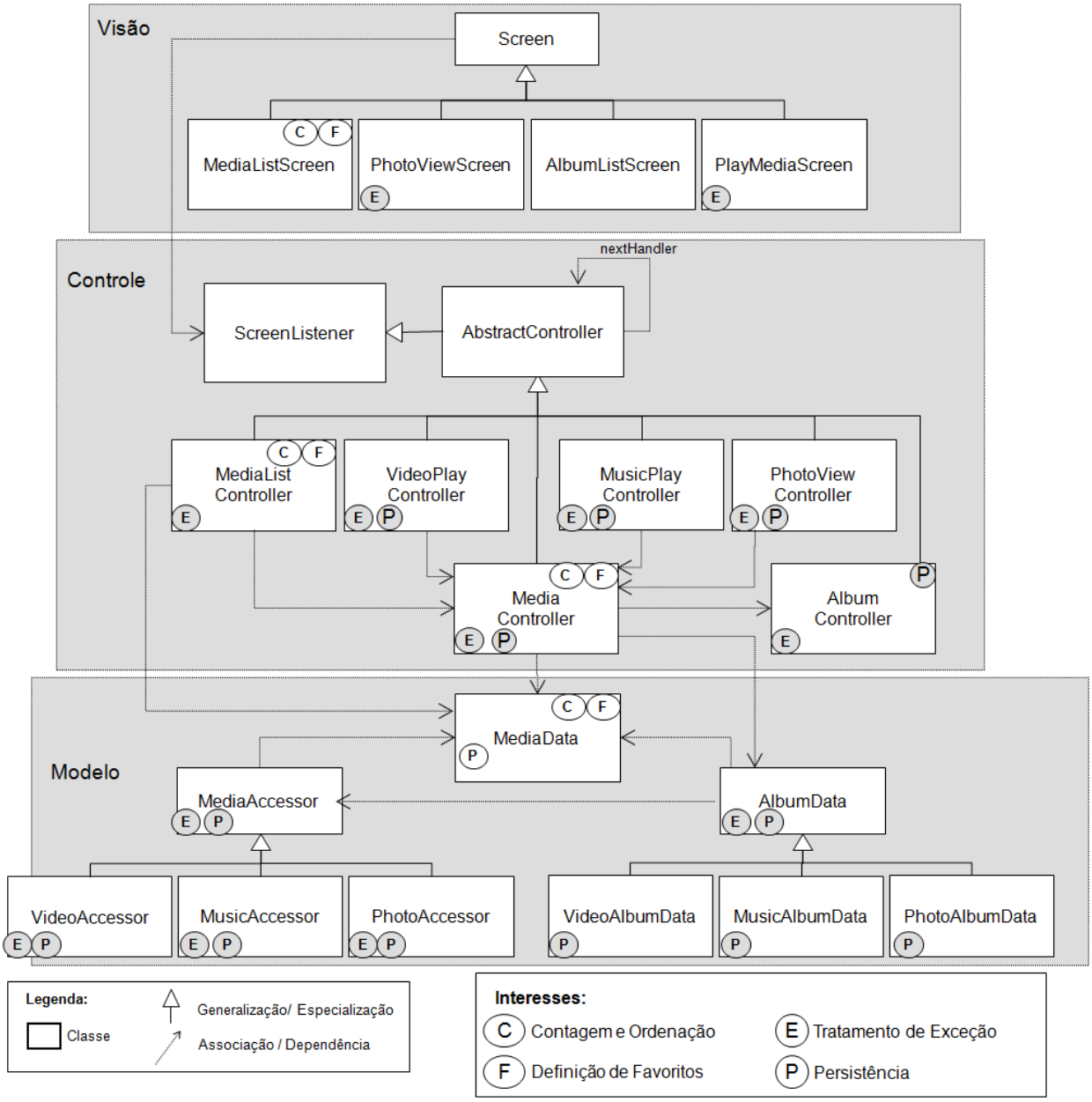

Figura 1. Visão de Projeto Parcial do MobileMedia

Tabela 1. Responsabilidade das Classes do MobileMedia

Classes	O que elas fazem
Screen MediaListScreen PhotoViewScreen AlbumListScreen PlayMediaScreen	Estas classes implementam as telas de interface com o usuário.
AbstractController MediaListController VideoPlayController MusicPlayController PhotoViewController MediaController AlbumController	Estas classes controlam o acesso as funcionalidades da aplicação. Elas recebem os comandos associados às operações das telas e repassam às classes ao modelo.
MediaData	Esta classe representa os tipos básicos mídias (vídeo, música e foto). Em outras palavras, é a principal classe do domínio da aplicação.
AlbumData VideoAlbumData MusicAlbumData PhotoAlbumData	Estas classes represemtam abstrações de dados gravados em dispositivos de armazenamento. Por exemplo, as classes controladoras não sabem como os dados são armazenados, pois conhecem apenas as abstrações.
MediaAccessor VideoAccessor MusicAccessor PhotoAccessor	Estas classes implementam efetivamente os serviços de gravação e recuperação de dados nos dispositivos de armazenamento. Nesta implementação do MobileMedia, os dados são persistidos na própria memória do dispositivo móvel.

Tabela 2. Alguns Interesses do MobileMedia

Interesses	O que significa
Contagem e Ordenação	Este interesse permite contar o número de vezes que uma determinada mídia foi visualizada pelo usuário. Além disso, o usuário pode ordenar a lista de mídia de um álbum pelo número de visualizações.
Definição de Favoritos	Este interesse permite que o usuário defina uma determinada mídia como favorita. Assim, ele tem a opção de visualizar somente as mídias favoritas de um álbum.
Tratamento de Exceções	Este interesse se caracteriza por um mecanismo para tratar ocorrências que alteram o fluxo normal de execução. Muitas linguagens de programação, como Java, têm suporte explicito para levantamento e tratamento de exceções. O escopo deste requisito começa com uma cláusula de marcação (*try*) e termina com uma cláusula de tratamento (*catch*).
Persistência	Este interesse se refere à habilidade da aplicação em reter dados entre execuções. Em outras palavras, ela é responsável por salvar e recuperar as informações manipuladas pela aplicação. Na implementação de persistência do MobileMedia, exceções específicas de persistência trafegam por outras classes da aplicação.

IMPORTANTE

Antes de começar o Passo 4, favor indicar que horas são agora: **____:____** ⇦

Passo 4) Os projetistas do sistema MobileMedia precisam de sua ajuda para identificar **quais métodos** que contêm o c*ode smell* "**Feature Envy**". Observe a descrição deste *code smell* e das métricas que lhes foram fornecidas. Lembre-se que é importante raciocinar sobre as métricas e identificar quais delas (uma, algumas ou todas) são indicadoras relevantes com base na descrição do *code smell*.

Exemplo de como deve ser identificado um método: AlbumController.gotToPreviousScreen

classe **método**

Passo 5) **Responda às seguintes perguntas:**

a) Quais são os métodos com maior probabilidade de ter o *code smell* "**Feature Envy**"? Você deve ordenar sua lista de métodos; aqueles com maior probabilidade devem vir primeiro.

b) Liste quais métricas você utilizou para detectar o *Code Smell* "**Feature Envy**" e como elas foram úteis para identificar as classes mencionadas na pergunta anterior. Quais métricas não foram úteis?

IMPORTANTE

Antes de começar o Passo 6, favor indicar que horas são agora: **____:____** ⇦

Passo 6) O objetivo agora é detectar a presença do c*ode smell* "**God Method**". Então, identifique **quais métodos** que contêm este c*ode smell*. Observe a descrição do **God Method** e das métricas que lhes foram fornecidas. Lembre-se que é importante raciocinar sobre as métricas e identificar quais delas (uma, algumas ou todas) são indicadoras relevantes com base na descrição do c*ode smell*.

Exemplo de como deve ser identificado um método: AlbumController.gotToPreviousScreen

classe **método**

Passo 7) Responda às seguintes perguntas:

a) Quais são os métodos com maior probabilidade de ter o c*ode smell* "**God Method**"? Você deve ordenar sua lista de métodos; aqueles com maior probabilidade devem vir primeiro.

b) Liste quais métricas você usou para detectar o *code smell* "**God Method**" e como elas foram úteis para identificar as classes mencionadas na pergunta anterior. Quais métricas não foram úteis?

IMPORTANTE

Quando terminar as atividades, favor indicar que horas são agora: **____:____** ⇦

Anexo C

Mapeamento dos Participantes

Este anexo apresenta o mapeamento dos participantes.

Todos os participantes foram organizados como seguem: (i) 24 participantes detectaram *Divergent Change*, (ii) 20 participantes detectaram *Shotgun Surgery* e (iii) 10 participantes detectaram *God Class*. Health Watcher foi usado por participantes de Lancaster para detectar todas as três anomalias de código, enquanto o MobileMedia foi usado por participantes da UFMG para detectar *Divergent Change* e *Shotgun Surgery*.

Já os 47 participantes que identificaram as duas anomalias em método foram organizados da seguinte forma: (i) do S1 ao S47 são os que fizeram a análise do *Feature Envy* e (ii) do S48 a S94 referem-se aos participantes que fizeram a análise do *God Method*.

Mapeamento dos Participantes

(Experimento para Anomalias em Classes)

	Grau de Estudo	Replicação	Quantidade	Métricas	Anomalia 1	Anomalia 2	DC	GC	SS
Lancs - 2007	Pos-Graduação	1	1	Tradicional	**DC**	**SS**	S1		S25
	Pos-Graduação	1	2	Tradicional	**DC**	**SS**	S2		S26
	Pos-Graduação	1	3	Interesse	**DC**	**SS**	S7		S30
	Pos-Graduação	1	4	Híbrida	**DC**	**SS**	S12		S33
	Graduação	2	1	Tradicional	**DC**	-	S3		
	Graduação	2	2	Interesse	**DC**	-	S8		
	Graduação	2	3	Híbrida	**DC**	-	S13		
Lancaster - 2008	Pós-Graduação	3	1	Tradicional	**DC**	**SS**	S4		S27
	Pós-Graduação	3	2	Interesse	**DC**	**SS**	S9		S31
	Pós-Graduação	3	3	Híbrida	**DC**	**SS**	S14		S34
	Graduação	4	1	Tradicional	**GC**	-		45	
	Graduação	4	2	Interesse	**GC**	-		48	
	Graduação	4	3	Híbrida	**GC**	-		51	
	Graduação	5	1	Tradicional	**GC**	-		46	
	Graduação	5	4	Tradicional	**GC**	-		47	
	Graduação	5	2	Interesse	**GC**	-		49	
	Graduação	5	5	Interesse	**GC**	-		50	
	Graduação	5	3	Híbrida	**GC**	-		52	
	Graduação	5	6	Híbrida	**GC**	-		53	
	Graduação	5	7	Híbrida	**GC**	-		54	
UFMG - 2011	Profissionais TI	6	1	Tradicional	**DC**	**SS**	S5		S28
	Profissionais TI	6	2	Tradicional	**DC**	**SS**	S6		S29
	Profissionais TI	6	3	Interesse	**DC**	**SS**	S10		S32
	Profissionais TI	6	4	Interesse	**DC**		S11		**Não Fez**
	Profissionais TI	6	5	Híbrida	**DC**	**SS**	S15		S35
	Profissionais TI	6	6	Híbrida	**DC**	**SS**	S16		S36
UFMG - 2012	Pos-Graduação	7	1	Híbrida	**DC**	**SS**	S17		S37
	Pós-Graduação	7	2	Híbrida	**DC**	**SS**	S18		S38
	Graduação	8	1	Híbrida	**DC**	**SS**	S19		S39
	Graduação	8	2	Híbrida	**DC**	**SS**	S20		S40
	Graduação	8	3	Híbrida	**DC**	**SS**	S21		S41
	Graduação	8	4	Híbrida	**DC**	**SS**	S22		S42
	Graduação	8	5	Híbrida	**DC**	**SS**	S23		S43
	Graduação	8	6	Híbrida	**DC**	**SS**	S24		S44
							#24	#10	#20

Mapeamento dos Participantes
(Experimento para Anomalias em Métodos)

	Grau de Estudo	Replicação	Métrica	Feature Envy	God Method	
UFMG - 2012	**Pós-Graduação**	1	**Tradicional**	S1	S48	15 participantes
		1		S2	S49	
		1		S3	S50	
		1		S4	S51	
		1		S5	S52	
		1	**Interesse**	S17	S64	
		1		S18	S65	
		1		S19	S66	
		1		S20	S67	
		1		S21	S68	
		1	**Híbrida**	S33	S80	
		1		S34	S81	
		1		S35	S82	
		1		S36	S83	
		1		S37	S84	
	Graduação	2	**Tradicional**	S6	S53	22 participantes
		2		S7	S54	
		2		S8	S55	
		2		S9	S56	
		2		S10	S57	
		2		S11	S58	
		2		S12	S59	
		2		S13	S60	
		2	**Interesse**	S22	S69	
		2		S23	S70	
		2		S24	S71	
		2		S25	S72	
		2		S26	S73	
		2		S27	S74	
		2		S28	S75	
		2		S29	S76	
		2	**Híbrida**	S38	S85	
		2		S39	S86	
		2		S40	S87	
		2		S41	S88	
		2		S42	S89	
		2		S43	S90	
UFBA - 2013	**Pós-Graduação**	3	**Tradicional**	S14	S61	10 participantes
		3		S15	S62	
		3		S16	S63	
		3	**Interesse**	S30	S77	
		3		S31	S78	
		3		S32	S79	
		3	**Híbrida**	S44	S91	
		3		S45	S92	
		3		S46	S93	
		3		S47	S94	
				#47	#47	

Anexo D

Resultado da Aplicação Experimento para Divergent Change

Este anexo apresenta os dados obtidos com a aplicação do experimento com os participantes na identificação do *Divergent Change*.
Esta anomalia foi aplicada para os dois sistemas: Health Watcher e MobileMedia.

Divergent Change: 1ª Reaplicação do Experimento (Lancaster)

(Experimento aplicado a 4 estudantes de Pós-Gradução)

Participantes	**S1**	**S2**	**S7**	**S12**
	Métricas Tradicionais	**Métricas Tradicionais**	**Métricas de Interesse**	**Métricas Híbridas**
Histórico	Não Respondeu	Não Respondeu	Não Respondeu	Não Respondeu
Sistema Usado	**Health Watcher**			
Tempo (min)	6	5	5	9
Anomalia de código	**Divergent Change**			
Tempo (min)	9	10	21	31
Classes Identificadas	Complaint (falso +) HealthWatcherFacadeInit (ok) PersistenceMechanism (ok)	Complaint (falso +) Employee (falso +) HealthWatcherFacadeInit (ok) PersistenceMechanism (ok)	HealthWatcherFacade (ok) ServletInsetEmployee (ok) ServletSearchComplaintData (ok) ServletUpdateComplaintData (ok) IFacade (ok) ServletUpdateHealthUnitData (ok) ComplaintRecord (ok) EmployeeRecord (ok) HealthUnitRecord (ok) HealthWatcherFacadeInit(ok) IPersistenceMechanism (ok) PersistenceMechanism (ok) ComplaintRepositoryRDB (falso +) Employee (falso +) EmployeeRepositoryRDB (falso +) HealthUnitRepositoryRDB (falso +) IComplaintRepository (falso +) IEmployeeRepository (falso +) IHealthUnitRepository (falso +)	ServletInsetEmployee (ok) ServletSearchComplaintData (ok) ServletUpdateComplaintData (ok) ServletUpdateHealthUnitData (ok) PersistenceMechanism (ok) HealthWatcherFacadeInit (ok) ComplaintRecord (ok) EmployeeRecord (ok) HealthUnitRecord (ok)
Número de Acertos	2	2	12	9
Classes do Oráculo	12	12	12	12
Falsos Negativos	10	10	0	3
Cobertura	17%	17%	100%	75%
Número de Falsos Positivos	1	2	7	0
Total de Classes Identificadas	3	4	19	9
Precisão	67%	50%	63%	100%
Métricas Úteis	- Perda de Coesão em Métodos (LCOM)	- Perda de Coesão em Métodos (LCOM) - Número de Atributos (NOA) - Número de Métodos (NOM) - Linhas de Código (LOC) - Método Ponderado por Classe (WMC)	- Número de Interesses por Classes (NCC)	- Número de Interesses por Classes (NCC) - Falta de Coesão em Métodos (LCOM)
Métricas Não Úteis	- Acoplamento entre Objetos (CBO) - Linhas de Código (LOC) - Método Ponderado por Classe (WMC)			- Método Ponderado por Classe (WMC)

Divergent Change: 2ª Reaplicação do Experimento (Lancaster)

(Experimento aplicado a 6 estudantes de graduação)

Participantes	**S3**	**S8**	**S13**
	Métricas Tradicionais	**Métricas de Interesse**	**Métricas Híbridas**
Histórico	Não Responderam	Não Responderam	Não Responderam
Sistema Usado	**Health Watcher**		
Tempo (min)	5	4	3
Anomalias de Código	**Divergent Change**		
Tempo (min)	35	25	28
Classes Identificadas	HealthWatcherFacadeInit (ok) Complait (falso +) Complaint Repository (falso +) HealthWatcherFacade (ok) HWServlet (falso +)	HealthWatcherFacade (ok) IFacade (ok) ServletSearchComplaint (ok) ServletUpdateComplaint (ok) ServletInsetEmployee (ok) ServletUpdateHealthUnit (ok) HealthWatcherFacadeInit (ok) IPersistenceMechanism (ok) PersistenceMechanism (ok) ComplaintRecord (ok) EmployeeRecord (ok) HealthUnitRecord (ok)	Complaint (falso +) HealthWatcherFacadeInit (ok)
Número de Acertos	2	12	1
Classes do Oráculo	12	12	12
Falsos Negativos	10	0	11
Cobertura	17%	100%	8%
Número de Falsos Positivos	3	0	1
Total de Classes Identificadas	5	12	2
Precisão	40%	100%	50%
Métricas Úteis	- Acoplamento entre Objetos (CBO)	- Número de Interesses por Classe (NCC) - Espalhamento de Interesses em Classes (CDC)	- Falta de Coesão em Métodos (LCOM) - Acoplamento entre Objetos (CBO)
Métricas Não Úteis	- Linhas de Código (LOC) - Número de Atributos (NOA) - Método Ponderado por Classe (WMC) - Número de Métodos (NOM)	- Espalhamento de Interesses em Operações (CDO) - Espalhamento de Interesses em Linhas de Código (CDLOC)	- Linhas de Código (LOC) - Número de Atributos (NOA) - Número de Interesses por Classe (NCC) - Espalhamento de Interesses em Operações (CDO)

Divergent Change: 3ª Reaplicação do Experimento (Lancaster)

(Experimento aplicado a 3 estudantes de Pós-Graduação)

Participantes	S4	S9	S14
	Métricas Tradicionais	**Métricas de Interesse**	**Métricas Híbridas**
HISTÓRICO			
Experiência de Trabalho	Apenas Acadêmica	Apenas Acadêmica	Trabalhou para a Universidade
Diagrama de Classe	Pouco	Médio	Alto
Programação Java	Pouco	Alto	Alto
Medição de Software	Pouco	Alto	Baixo
Sistema Usado	**Health Watcher**		
Tempo (min)	10	7	2
Anomalias de Código	**Divergent Change**		
Tempo (min)	28	22	21
Classes Identificadas	HealthWatcherFacadeInit (ok) ComplainRepositoryRDB (falso +) IFacade (ok) PersistenceMechanism (ok) HeathUnit (falso +) HealthWatcherFacade (ok) HealthUnitRepositotyRDB (falso +) Employee (falso +)	HealthWatcherFacadeInit (ok) IFacade (ok) HealthWatcherFacade (ok) ServletSearchComplaintData (ok)	PersistenceMechanism (ok) HealthWatcherFacadeInit (ok) HealthWatcherFacade (ok) ComplainRepositoryRDB (falso +)
Número de Acertos	4	4	3
Classes do Oráculo	12	12	12
Falsos Negativos	8	8	9
Cobertura	33%	33%	25%
Número de Falsos Positivos	4	0	1
Total de Classes Identificadas	8	4	4
Precisão	50%	100%	75%
Métricas Úteis	- Acoplamento entre Objetos (CBO) - Falta de Coesão em Métodos (LCOM) - Método Ponderado por Classe (WMC)	- Número de Interesse por Classe (NCC) - Espalhamento de Interesses em Linhas de Código (CDLOC) - Espalhamento de Interesses em Operações (CDO)	- Falta de Coesão em Métodos (LCOM) - Número de Interesses por Classe (NCC) - Número de Atributos (NOA) - Número de Métodos (NOM) - Métodos Ponderados por Classe (WMC)
Métricas Não Úteis	- Número de Atributos (NOA)	- Espalhamento de Interesses em Classes (CDC)	

Divergent Change: 4ª Reaplicação do Experimento (UFMG)

(Experimento aplicado a 12 Profissionais de TI)

Participantes		**S5**		**S6**		**S10**		**S11**		**S15**		**S16**	
		Métricas Tradicionais		**Métricas Tradicionais**		**Métricas Interesse**		**Métricas Interesse**		**Métricas Híbridas**		**Métricas Híbridas**	
Pessoas		Pessoa 1		Pessoa 3		Pessoa 5		Pessoa 7		Pessoa 9		Pessoa 11	
		Pessoa 2		Pessoa 4		Pessoa 6		Pessoa 8		Pessoa 10		Pessoa 12	
Background													
Cursos	Sistema de Banco Dados	X	X	X	X	X	X	X	X	X	X	X	
	Programação OO	X	X	X	X	X	X	X	X	X	X	X	
	UML Design	X	X	X	X			X	X		X	X	
	Programação JAva	X	X	X	X	X	X	X	X	X	X	X	
	Projeto SW			**X**	X	X			X			X	
	Banco de Dados Avançado	X	X	X	X	X			X		X	X	
Experiência em Trabalho		> 3	> 3	0.5-1	1 - 3	0.5-1	> 3	> 3	> 3	0.5-1	> 3	> 3	> 3
Diagrama de Classe		Pouco	Médio	Médio	Médio	Médio	Alto	Alto	Médio	Médio	Pouco	Alto	Pouco
Programação Java		Pouco	Médio	Médio	Pouco	Pouco	Alto	Alto	Médio	Médio	Pouco	Médio	Médio
Métrica de Software		Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Alto	Pouco
Sistema Usado		**MobileMedia**											
Tempo (min)		8		10		5		7		4		12	
Anomalia de Código		**Divergent Change**											
Tempo (min)		33		26		10		26		32		15	
Classes Identificadas		MediaController **(Ok)** MainUIMidl.(F+) PhotoViewC.(F+) MusicPlayC. (F+) PlayVideoC. (F+) VideoCaptureControl. (F+)		MediaData (F+) MediaAcessor **(Ok)** SelectMediaControler (F+) SmsMessag.(F+)		AbstractCont.(F+) ScreenSingle.(F+) MediaAcessor **(Ok)** AlbumData (F+)		MediaController **(Ok)** MediaListController **(Ok)** MediaListScreen (F+) AlbumData(F+) MediaUtil (F+) MediaMusicAcessor (F+) MediaMusicUtil (F+)		MediaController **(Ok)** MainUIMid. (F+) PhotoViewC.(F+) PlayVideoC. (F+) MusicPlayC. (F+) VideoCaptureControl. (F+) MediaListSc. (F+) MediaListContr.**(Ok)**		MediaControlle r **(Ok)** MediaListContr. **(Ok)** AlbumData (F+) MediaAcessor **(Ok)** AlbumControl. **(Ok)** MediaData (F+)	
Número de Acertos		1		1		1		2		2		4	
Classes do Oráculo		4		4		4		4		4		4	
Falsos Negativos		3		3		3		2		2		0	
Cobertura		25%		25%		25%		50%		50%		100%	
Falsos Positivos		5		3		3		5		6		2	
Total de Classes Identificadas		6		4		4		7		8		6	
Precisão		17%		25%		25%		29%		25%		67%	
Métricas Úteis		- Acoplamento entre Objetos (CBO)		- Perda de Coesão em Métodos (LCOM) - Acoplamento entre Objetos (CBO) - Método Ponderado por Classe (WMC)		- Espalhamento de Interesse em Classes (CDC)		- Número de Interesse por Componente (NCC)		- Número de Interesse por Componte (NCC) -Perda de Coesão em Métodos (LCOM)		- Número de Interesses por Componente (NCC)	

Divergent Change: 5ª Reaplicação do Experimento (UFMG)

(Experimento aplicado a 4 estudantes de Pós-Graduação)

Participantes		S17		S18	
		Métricas Híbridas		**Métricas Híbridas**	
Pessoas		Pessoa 1		Pessoa 3	
		Pessoa 2		Pessoa 4	
Cursos	Programação OO	x	x	Não Respondeu	x
	Projeto UML	x	x	Não Respondeu	x
	Programação Java	x	x	Não Respondeu	x
	Projeto de Software	x	x	Não Respondeu	Não Respondeu
Experiência de Trabalho		> 3 anos	> 3 anos	Não Respondeu	Não Respondeu
Histórico	Sistema de Banco de Dados	Alta	Alta	Não Respondeu	Não Respondeu
	Programação OO	Alta	Alta	Não Respondeu	Não Respondeu
	Projeto UML	Médio	Médio	Não Respondeu	Não Respondeu
	Programação Java	Alta	Alta	Não Respondeu	Não Respondeu
	Projeto de Software	Médio	Médio	Não Respondeu	Não Respondeu
Sistema Usado		**MobileMedia**			
Anomalia de Código		**Divergent Change**			
Tempo (min)		39		24	
Classes Identificadas		Mediacontroller **(OK)** MainUIMidlet (Falso +) PhotoViewController (Falso +)		MediaAcessor **(OK)** MediaController **(OK)** MediaData (Falso +) AbstractController (Falso +) SelectMediaController (Falso +)	
Número de Acertos		1		2	
Classes de Oráculo		4		4	
Falsos Negativos		3		2	
Cobertura		25%		50%	
Número de Falsos Positivos		2		3	
Total de Classes Identificadas		3		5	
Precisão		33%		40%	
Métricas Úteis		- Perda de Coesão em Métodos (LCOM) - Linhas de Código (LOC)		- Acomplamento entre Objetos (CBO) - Métodos Ponderados por Classe (WMC)	
Métricas Não Úteis		- Número de Atributos (NOA)			

Divergent Change: 6ª Reaplicação do Experimento (UFMG)

(Experimento aplicado a 12 estudantes de Graduação)

Participantes		**S19**		**S20**		**S21**		**S22**		**S23**		**S24**	
		Métricas Híbridas		**Métricas Híbridas**		**Métricas Híbridas**		**Métricas Híbridas**		**Métricas Híbridas**		**Métricas Híbridas**	
Pessoas		Pessoa 1		Pessoa 3		Pessoa 5		Pessoa 7		Pessoa 9		Pessoa 11	
		Pessoa 2		Pessoa 4		Pessoa 6		Pessoa 8		Pessoa 10		Pessoa 12	
Cursos	Programação OO	x	x	x	x	x	x	x	x	x	x	x	x
	Projeto UML	x	x			x		x	x	x	x	x	x
	Programação Java	x	x	x	x	x	x	x	x	x	x	x	x
	Projeto SW	x	x		x			x	x		x	x	x
Experiência de Trabalho		Nenhum	Nenhum	6 meses	1 to 2 anos	Nenhum	Nenhum	1 ano	1 ano	Nenhum	Nenhum	Nenhum	Nenhum
Histórico	Diagrama de Classe	Pouco	Pouco	Médio	Pouco	Médio	Pouco	Pouco	Pouco	Pouco	Pouco	Nenhum	Nenhum
	Programação Java	Pouco	Pouco	Médio	Nenhum	Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Nenhum	Nenhum
	Medição de Software	Médio	Pouco	Pouco	Pouco	Médio	Médio	Médio	Pouco	Pouco	Pouco	Pouco	Pouco
Sistema Usado		**MobileMedia**											
Anomalia		**Divergent Change**											
Tempo (min)		11		18		19		13		13		12	
Classes Identificadas		AlbumListScreen (F+) NetworkScreen (F+) VideoCaptureController (F+)		MediaController **(OK)** MediaAccessor **(OK)** AlbumData (F+)		MediaController **(OK)** MainUIMidlet (F+) PhotoViewController (F+) MusicPlayController (F+) PlayPhotoController (F+) VideoCaptureController (F+)		MediaAccessor **(OK)** MediaData (F+) AbstractController (F+) MediaController **(OK)** SelectMediaController (F+)		MediaController **(OK)** MainUIMidlet (F+) PhotoViewController (F+) MusicPlayController (F+) PlayVideoController (F+) VideoCaptureController (F+)		MediaController **(OK)** MainUIMidlet (F+) MediaAccessor **(OK)** MediaData (F+)	
Número de Acertos		0		2		1		2		1		2	
Classes do Oráculo		4		4		4		4		4		4	
Falso Negativo		4		2		3		2		3		2	
Cobertura		0%		50%		25%		50%		25%		50%	
Falso Positivo		3		1		5		3		5		2	
Total de classes identificadas		3		3		6		5		6		4	
Precisão		0%		67%		17%		40%		17%		50%	
Métricas Úteis		- Número de Atributos (NOA) - Número de Métodos (NOM)		- Número de Interesses por Componente (NCC) - Número de Métodos (NOM) - Linhas de Código (LOC)		- Acoplamento de Objetos (CBO)		- Perda de Coesão em Métodos (LCOM)		- Acoplameto de Objetos (CBO) - Espalhamento de Interesses em Componetes (CDC) - Número de Interesse por Componente (NCC)		- Acoplamento de Objetos (CBO) - Perda de Coesão em Métodos (LCOM) - Número de Interesse em Componente (NCC)	

Anexo E

Resultado da Aplicação Experimento para Shotgun Surgery

Este anexo apresenta os dados obtidos com a aplicação do experimento com os participantes na identificação do *Shotgun Surgery*.
Esta anomalia foi aplicada para os dois sistemas: Health Watcher e MobileMedia.

Shotgun Surgery: 1ª Reaplicação do Experimento (Lancaster)

(Experimento aplicado 4 participantes de Pós-Graduação)

Participantes	**S25**	**S26**	**S30**	**S33**
	Métricas Tradicionais	**Métricas Tradicionais**	**Métricas de Interesse**	**Métricas de Interesse**
Histórico	Não Respondeu	Não Respondeu	Não Respondeu	Não Respondeu
Sistema Usado	**Health Watcher**			
Anomalia de Código	**Shotgun Surgery**			
Tempo (min)	6	10	13	35
Classes Identificadas	HealthWatcherFacadeInit (F+) ComplaintRepositoryRDB **(ok)** HealthWatcherFacade (F+) HWServlet (F+)	HealthWatcherFacadeInit (F+) Complaint (F+) HealthWatcherRepositoryRDB **(ok)**	IHealthUnitRepository **(ok)** IEmployeeRepository **(ok)** IComplaintRepository **(ok)** HealthUnitRepositoryRDB **(ok)** EmployeeRepositoryRDB **(ok)** ComplaintRepositoryRDB **(ok)** HealthUnitRecord (F+) EmployeeRecord (F+) ComplaintRecord (F+) HealthUnit (F+) Employee (F+) Complaint (F+) HWServlet (F+) ServletInsetEmployee (F+) ServletSearchComplaintData (F+) ServletUpdateComplaintData (F+) ServletUpdateHealthUnitData (F+)	IFacade (F+) HealthWatcherFacadeInit (F+) HealthWatcherFacade (F+) PersistenceMechanism **(ok)**
Número de Acertos	1	1	6	1
Classes do Oráculo	8	8	8	8
Falso Negativo	7	7	2	7
Cobertura	13%	13%	75%	13%
Falsos Positivos	3	2	11	3
Total de Classes Identificadas	4	3	17	4
Precisão	25%	33%	35%	25%
Métricas Úteis	CBO	CBO NOA NOM	CDC	CDC
Métricas Não Úteis	LCOM NOA NOM WMC			

Shotgun Surgery: 2ª Reaplicação do Experimento (Lancaster)

(Experimento aplicado a 3 participantes de Pós-Graduação)

Participantes	**S27**	**S31**	**S34**
	Métricas Tradicionais	**Métricas de Interesse**	**Métricas Híbridas**
Histórico			
Experiência	Apenas Acadêmica	Apenas Acadêmica	Experiência de Trabalho
Diagrama de Classe	Baixa	Média	Alta
Java	Baixa	Alta	Média
Métricas	Baixa	Alta	Baixa
Sistema Usado	**Health Watcher**		
Tempo (min)	8	3	6
Anomalia de Código	**Shotgun Surgery**		
Tempo (min)	19	25	8
Classes Identificadas	HealthWatcherFacadeInit (F+) ComplaintRecord (F+) EmployeeRecord (F+) HealthUnitRecord (F+)	IPersistenceMechanism **(ok)** PersistenceMechanism **(ok)** IFacade (F+) HealthWatcherFacade (F+) Complaint (F+)	PersistenceMechanism **(ok)** ComplaintRecord (F+) ComplaintRepositoryRDB **(ok)** HealthUnitRepositotyRDB **(ok)** EmployeeRepositoryRDB **(ok)**
Acertos	0	2	4
Falso Negativo	8	6	4
Cobertura	0%	25%	50%
Falso Positivo	4	3	1
Total de Classes Identificadas	4	5	5
Precisão	0%	40%	80%
Métricas Úteis	CBO	CDO	
Métricas Não Úteis		CDC CDLOC	

Shotgun Surgery: 3ª Reaplicação do Experimento (UFMG)

(Experimento aplicado a 11 participantes Profissionais de TI)

*Participantes (Pessoa 7 e Pessoa 8) não fizeram o experimento Shotgun Surgery

Participantes		S28		S29		S32		S35		S36	
Métricas		**Métricas Tradicionais**		**Métricas Tradicionais**		**Métricas Interesse**		**Métricas Híbridas**		**Métricas Híbridas**	
Pessoas		Pessoa 1		Pessoa 3		Pessoa 5		Pessoa 9		Pessoa 11	
		Pessoa 2		Pessoa 4		Pessoa 6		Pessoa 10		Pessoa 12	
Sexo		M	M	**M**	M	F	M	M	M	M	M
Idade		52	38	25	31	30	25	25	52	25	25
Cursos	Banco de Dados	X	X	X	X	X	X	X	X	X	
	Programação OO	X	X	X	X	X	X	X	X	X	
	Projeto UML	X	X	X	X				X	X	
	Programação Java	X	X	X	X	X	X	X	X	X	
	Projeto de Software			X	X	X				X	
	Banco de Dados Avançado	X	X	X	X	X			X	X	
Experiência de Trabalho		> 3	> 3	0.5-1	01/mar	0.5-1	> 3	0.5-1	> 3	> 3	> 3
Diagrama de Classe (Aptidão)		Pouco	Médio	Médio	Médio	Médio	Alto	Médio	Pouco	Alto	Pouco
Java (Aptidão)		Pouco	Médio	Médio	Pouco	Pouco	Alto	Médio	Pouco	Médio	Médio
Medição de Software		Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Alto	Pouco
Sistema Usado		**MobileMedia**									
Anomalia de Código		**Shotgun Surgery**									
Tempo (min)		12		14		15		19		15	
Classes Identificadas		MediaAcessor **(Ok)** MediaData (F+) MediaContr. (F+) Abstract Co. (F+) SelectedMediaContr. (F+) SmsMessagi.(F+) ScreenSingleton **(Ok)** AddMediaToAlbum (F+)		MediaAcessor **(Ok)** AbstractCon. (F+) MediaController (F+) SelectMediaController (F+)		MediaController(F+) MediaData (F+) MediaAcessor **(Ok)** AlbumData (F+)		MediaAcessor **(Ok)** MediaData (F+) AbstractController (F+) MediaContro.(F+) PhotoViewS.(F+) PlayMediaSc.(F+) VideoPlayC.(F+) MusicPlayController (F+) PhotoViewController (F+) VideoAcess. (F+) MusicAcess. (F+) PhotoAcess. (F+) Albumcontroller(F+) AlbumData (F+) VideoAlbumData (F+) MusicAlbumData (F+)		MediaAcessor **(Ok)** MediaController (F+) MediaData (F+)	
Acertos		2		1		1		1		1	
Classes do Oráculo		3		3		3		3		3	
Falsos Negativos		1		2		2		2		2	
Cobertura		67%		33%		33%		33%		33%	
Falsos Positivos		6		3		3		15		2	
Total de classes Identificadas		8		4		4		16		3	
Precisão		25%		25%		25%		6%		33%	
Métricas Úteis		CBO		CBO		NCC		CDO CDC		CBO NCC	
Métricas Não Úteis		LCOM LOC NOA NOM WMC		LCOM WMC NOM NOA LOC		CDO CDLOC CDA CDC		CDA CDLOC CBO LOC NOA NOM WMC NCC LCOM		CDC CDO CDA CDLOC LOC NOA NOM WMC LCOM	

Shotgun Surgery: 4ª Reaplicação do Experimento (UFMG)

(Experimento aplicado a 4 participantes de Pós-Graduação)

Participantes		**S37**		**S38**	
Métricas		**Métricas Híbridas**		**Métricas Híbridas**	
Pessoas		Pessoa 1		Pessoa 3	
		Pessoa 2		Pessoa 4	
Cursos	Programação OO	x	x	Não Respondeu	x
	Projeto UML	x	x	Não Respondeu	x
	Programação JAVA	x	x	Não Respondeu	x
	Projeto de Software	x	x	Não Respondeu	Não Respondeu
Experiência de Trabalho		> 3 anos	> 3 anos	Não Respondeu	Não Respondeu
Histórico	Diagrama Classe	Pouco	Alto	Não Respondeu	Não Respondeu
	Java	Pouco	Alto	Não Respondeu	Não Respondeu
	Modularidade	Nenhum	Médio	Não Respondeu	Não Respondeu
	Evolução do Sw	Pouco	Alto	Não Respondeu	Não Respondeu
	Projeto de SW	Médio	Médio	Não Respondeu	Não Respondeu
Sistema Usado		**MobileMedia**			
Anomalia de Código		**Shotgun Surgery**			
Tempo (min)		4		10	
Classes Identificadas		MediaAcessor **(OK)** MediaData (Falso +) Mediacontroller (Falso +) AbstractController (Falso +)		MediaController (Falso +) AlbumData (Falso +) MediaAcessor **(OK)**	
Acertos		1		1	
Classes do Oráculo		3		3	
Falsos Negativos		2		2	
Cobertura		33%		33%	
Falso Positivos		3		2	
Total de classes Identificadas		4		3	
Precisão		25%		33%	
Métricas Úteis		CBC NCC			
Métricas Não Úteis					

Shotgun Surgery: 5° Reaplicação do Experimento (UFMG)

(Experimento aplicado para 12 estudantes de Graduação)

Participantes		**S39**		**S40**		**S41**		**S42**		**S43**		**S44**	
Métricas		**Métricas Híbridas**		**Métricas Híbridas**		**Métricas Híbridas**		**Métricas Híbridas**		**Métricas Híbridas**		**Métricas Híbridas**	
Pessoas		Pessoa 1		Pessoa 3		Pessoa 5		Pessoa 7		Pessoa 9		Pessoa 11	
		Pessoa 2		Pessoa 4		Pessoa 6		Pessoa 8		Pessoa 10		Pessoa 12	
Cursos	Programação OO	x	x	x	x	x	x	x	x	x	x	x	x
	Projeto UML	x	x			x		x	x	x	x	x	x
	Programação Java	x	x	x	x	x	x	x	x	x	x	x	x
	Projeto de Software	x	x		x			x	x		x	x	x
Experiência de Trabalho		Nenhum	Nenhum	6 meses	1 a 2 anos	Nenhum	Nenhum	1 ano	1 ano	Nenhum	Nenhum	Nenhum	Nenhum
Histórico	Modularidade	Pouco	Pouco	Pouco	Médio	Pouco	Médio	Pouco	Pouco	Pouco	Pouco	Pouco	Nenhum
	Evolução Software	Pouco	Pouco	Pouco	Médio	Nenhum	Pouco	Pouco	Pouco	Pouco	Pouco	Pouco	Nenhum
	Projeto de Software	Médio	Médio	Pouco	Pouco	Pouco	Médio	Médio	Médio	Pouco	Pouco	Pouco	Pouco
Sistema Usado		**MobileMedia**											
Anomalias de Código		**Shotgun Surgery**											
Tempo (min)		14		9		21		3		7		5	
Classes Identificadas		MediaController (F+) MainUIMidlet (F+) AbstractMediaController (F+) SelectMediaController (F+)		MediaController (F+) MediaListController (F+) SelectMediaController (F+) AbstractController (F+)		MediaController (F+) MediaData (F+) AbstractController (F+) MediaAccessor **(OK)** AlbumData (F+)		MediaController (F+) MainUIMidlet (F+) PhotoViewController (F+)		MediaController (F+) MainUIMidlet (F+) PhotoViewController (F+) MusicPlayController (F+) PlayVideoController (F+) VideoCaptureController (F+)		MediaController (F+) MainUIMidlet (F+) PlayVideoController (F+)	
Acertos		0		0		1		0		0		0	
Classes do Oráculo		3		3		3		3		3		3	
Falso Negativo		3		3		2		3		3		3	
Cobertura		0%		0%		33%		0%		0%		0%	
Falso Positivo		4		4		4		3		6		3	
Total de Classes Identificadas		4		4		5		3		6		3	
Precisão		0%		0%		20%		0%		0%		0%	
Métricas Úteis		CBO		CBO LCOM CDC CDO				CBO		CBO CDC		CBO	
Métricas Não Úteis		LCOM NCC LOC WMC		N/A						NCC LCOM			

Anexo F

Resultado da Aplicação Experimento para God Class

Este anexo apresenta os dados obtidos com a aplicação do experimento com os participantes na identificação do *God Class*.
Esta anomalia foi aplicada para o sistema Health Watcher.

God Class: 1ª Reaplicação do Experimento (Lancaster)

(Experimento aplicado a 6 estudantes de graduação)

Participantes		**S45**		**S48**		**S51**	
		Métricas Tradicionais		**Métricas de Interesse**		**Métricas Híbridas**	
Pessoas		Pessoa 1	Pessoa 2	Pessoa 3	Pessoa 4	Pessoa 5	Pessoa 6
Histórico	**Experiência de Trabalho**	Acadêmica	Trabalho	Trabalho	Trabalho	Acadêmica	Trabalho
	Diagrama de Classe	Baixa (acadêmica)	Baixa Nenhuma	Média	Média	Média (acadêmica)	Média (acadêmica)
	Java	Média (acadêmica)	Alta	Alta	Alta	Média (acadêmica)	Alta
	Métricas	Baixa (acadêmica)	Nenhuma Nenhuma	Nenhuma	Nenhuma	Nenhuma	Baixa
Descrição do sistema e Anomalias		8 minutos		12 minutos		6 minutos	
Identificação Anomalia		5 minutos		21 minutos		13 minutos	
Uso das Métricas		5 minutos		4 minutos		3 minutos	
Sistema Usado		**Health Watcher**					
Anomalias de código		**God Class**					
Classes Identificadas		Complaint (F+) HealthWatcherFacadeInit **(ok)** ComplainRepositoryRDB (F+)		HealthWatcherFacadeInit **(ok)** PersistenceMechanism **(ok)** ComplainRepositoryRDB (F+) HealthWatcherFacade **(ok)**		Complaint (F+) HealthWatcherFacadeInit **(ok)**	
Número de Acertos		1		3		1	
Oráculo do God Class		3		3		3	
Falso Negativo		2		0		2	
Cobertura		33%		100%		33%	
Falso Positivo		2		1		1	
Total de Classes Identificadas		3		4		2	
Precisão		33%		75%		50%	
Métricas Úteis		LCOM NOA NOM		CDLOC		CBO LCOM NOM CDC	
Métricas Não Úteis		LOC CBO					

God Class: 2º Reaplicação do Experimento (Lancaster)

(Experimento aplicado a 14 estudantes de graduação)

Participantes		**S46**		**S49**		**S52**	
		Métricas Tradicionais (1)		**Métricas de Interesse (2)**		**Métricas Híbridas (3)**	
Pessoas		Pessoa 1	Pessoa 2	Pessoa 3	Pessoa 4	Pessoa 5	Pessoa 6
Histórico	**Experiência de Trabalho**	Apenas Acadêmica	Apenas Acadêmica	Trabalho	Apenas Acadêmica	Apenas Acadêmica	Apenas Acadêmica
	Diagrama de Classe	Médio	Médio	Médio	Baixo	Médio	Alta
	Java	Médio	Baixo	Alta	Nenhum	Baixo	Alta
	Métricas de Software	Baixo	Baixo	Médio	Nenhum	Médio	Baixo
Descrição do Sistema		3 minutos		6 minutos		41 minutos	
Identificação da anomalia		15 minutos		50 minutos		7 minutos	
Métricas Úteis		7 minutos		10 minutos		5 minutos	
Sistema Usado		**Health Watcher**					
Anomalia de Código		**God Class**					
Classes Identificadas		HealthWatcherFacadeInit **(ok)** Complaint (F+) ComplainRepositoryRDB (F+)		PersistenceMechanism **(ok)** HealthWatcherFacade **(ok)**		PersistenceMechanism **(ok)** HealthWatcherFacade **(ok)** HealthWatcherFacadeInit **(ok)**	
Número de Acertos		1		2		3	
Oráculo do God Class		3		3		3	
Falso Negativo		2		1		0	
Cobertura		33%		67%		100%	
Falso Positivo		2		0		0	
Total Identified Classes		3		2		3	
Precisão		33%		100%		100%	
Métricas Úteis		CBO NOM NOA		CDO CDLOC NCC		LOC NOM NOA WMC	
Métricas Não Úteis		LCOM		CDC		CBO LCOM CDC CDO CDLOC NCC	

Participantes		**S47**		**S50**		**S53**	
		Métricas Tradicionais (4)		**Métricas Híbridas (5)**		**Métricas Híbridas (6)**	
Pessoas		Pessoa 7	Pessoa 8	Pessoa 9	Pessoa 10	Pessoa 11	Pessoa 12
Histórico	**Experiência de Trabalho**	Apenas Acadêmica	Apenas Acadêmica	Apenas Acadêmica	Apenas Acadêmica	Apenas Acadêmica	Apenas Acadêmica
	Diagrama de Classe	Nenhuma	Nenhuma	Baixo	Alto	Médio	Médio
	Java	Nenhuma	Nenhuma	Médio	Alto	Médio	Médio
	Métricas de Software	Nenhuma	Nenhuma	Baixo	Baixo	Baixo	Baixo
Descrição do Sistema		5 minutos		5 minutos		7 minutos	
Identificação da anomalia		5 minutos		17 minutos		34 minutos	
Métricas Úteis		17 minutos		21 minutos		10 minutos	
Sistema Usado		**Health Watcher**					
Anomalia de Código		**God Class**					
Classes Identificadas		ComplainRepositoryRDB (F+) HealthWatcherFacadeInit **(ok)** HealthWatcherFacade **(ok)**		HealthWatcherFacadeInit **(ok)** HealthWatcherFacade **(ok)** PersistenceMechanism **(ok)** ComplainRepositoryRDB (F+)		Complaint (F+) ComplainRepositoryRDB (F+) HealthWatcherFacadeInit **(ok)** PersistenceMechanism **(ok)** HealthWatcherFacade **(ok)**	
Número de Acertos		2		3		3	
Oráculo do God Class		3		3		3	
Falso Negativo		1		0		0	
Cobertura		67%		100%		100%	
Falso Positivo		1		1		2	
Total de Classes Identificadas		3		4		5	
Precisão		67%		75%		60%	
Métricas Úteis		CBO WMC		CDO		CBO LCOM LOC NOM WMC CDLOC NCC	
Métricas Não Úteis		NOA LOC NOM LCOM		CDC CDLOC			

Participantes						**S54**	
						Métricas Híbridas (7)	
Pessoas						Pessoa 13	Pessoa 14
Histórico	**Experiência de Trabalho**					Apenas Acadêmica	Apenas Acadêmica
	Diagrama de Classe					Baixo	Médio
	Java					Alto	Médio
	Métricas de Software					Médio	Médio
Descrição do Sistema						5minutes	
Identificação da anomalia						9 minutes	
Métricas Úteis						21 minutes	
Sistema Usado		**Health Watcher**					
Anomalia de Código		**God Class**					
Classes Identificadas						ComplaintRepository RDB (F+) HealthWatcherFacadeInit **(ok)** HealthWatcherFacade **(ok)** PersistenceMechanism **(ok)**	
Número de Acertos						3	
Oráculo do God Class						3	
Falso Negativo						0	
Cobertura						100%	
Falsos Positivos						1	
Total de classes Identificadas						4	
Precisão						75%	
Métricas Úteis						NCC	
Métricas Não Úteis						LCOM NOM WMC	

Anexo G

Resultado da Aplicação Experimento para Feature Envy

Este anexo apresenta os dados obtidos com a aplicação do experimento com os participantes na identificação do *Feature Envy* no sistema MobileMedia.

Feature Envy: 1ª Replicação do Experimento (UFMG)
(Experimento aplicado a 15 alunos de Pós-Graduação)

		Métricas Tradicionais	Métricas Tradicionais	Métricas Tradicionais	Métricas Tradicionais
Participantes		**S1**	**S2**	**S3**	**S4**
Sexo		M	M	M	M
Idade		28	23	24	28
Cursos	Banco de Dados	x	x	x	x
	Programação OO	x	x	x	x
	Sistemas Distribuídos	x			x
	Projeto UML	x	x	x	x
	Java (SE)	x	x	x	
	Java (ME)	x			
	Banco de Dados Avançado	x		x	x
	Sistemas Tolerantes a Falhas			x	
Experiência		Nenhuma	06 meses a 1 ano	06 meses a 1 ano	> 3 anos
Diagrama de Classe (Habilidade)		Pouca	Alta	Alta	Alta
Java (Habilidade)		Média	Alta	Média	Média
Métrica					
Sistema Usado		**MobileMedia**			
Tempo (minutos)		5	6	8	4
Anomalia de Código		**Feature Envy**			
Tempo (minutos)		42	15	39	48
Métodos Identificados		Nenhuma Método	MediaAcessor.updateMediaInfo (F+) MediaAcessor.loadMediaDataFromRMS (F+)	MediaController.playVideoMedia (F+) MediaController.showImage (F+) MediaController.playMusicMedia (F+) PhotoViewController.getCPVideoScreen (F+) PhotoViewController.setCPVideoScreen (F+) MediaAcessor.createNewAlbum (F+) MediaAcessor.deleteAlbum (F+) MediaAcessor.getAlbumName (F+) MediaAcessor.loadAlbums (F+) MediaAcessor.removeRecords (F+) MediaAcessor.resetRecordStore (F+)	**AlbumController.resetMediaData (ok)** BaseController.goToPreviousScreen (F+) MediaAccessor.addMediaData (F+) MediaAccessor.createNewAlbum (F+) MediaAccessor.deleteAlbum (F+) MediaAccessor.loadAlbums (F+) MediaAccessor.setMediaInfo (F+) MediaAccessor.setMediaInfoTable (F+) MediaAccessor.updateMediaInfo (F+) MediaController.handleCommand (F+) BaseController.handleCommand (F+) MediaListController.mediaListController (F+) **MediaListController.bubleSort (ok)** MediaListController.handleCommand (F+) MusicPlayController.getMediaName (F+) MusicPlayController.musicPlayController (F+) PhotoViewController.photoViewController (F+) MusicPlayController.handleCommand (F+)
Número de Acertos		0	0	0	2
Métodos do Oráculo		4	4	4	4
Falsos Positivos		0	2	11	18
Falsos Negativos		4	4	4	2
Cobertura		**0%**	**0%**	**0%**	**50%**
Precisão		**0%**	**0%**	**0%**	**10%**
Métricas Úteis		Nenhuma métrica foi útil	CBO	WMC CBO NOO LOC	PAR NOA
Métricas Não Úteis		CBO LCOM LOC NOA NOO PAR WMC CYCLO	LCOM LOC NOA NOO PAR WMC CYCLO	LCOM NOA PAR CYCLO	LCOM LOC NOO WMC CYCLO CBO

		Métricas Tradicionais	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse
Participantes		**S5**	**S17**	**S18**	**S19**
Sexo		M	M	M	M
Idade		28	27	29	23
Cursos	Banco de Dados	x	x	x	x
	Programação OO	x	x	x	x
	Sistemas Distribuídos		x	x	
	Projeto UML	x		x	x
	Java (SE)		x	x	x
	Java (ME)				
	Banco de Dados Avançado				
	Sistemas Tolerantes a Falhas				
Experiência		06 meses a 1 ano	1 a 3 anos	Nenhuma	06 meses a 1 ano
Diagrama de Classe (Habilidade)		Pouca	Pouca	Média	Pouca
Java (Habilidade)		Média	Média	Média	Média
Métrica					
Sistema Usado		**MobileMedia**			
Tempo (minutos)		5	7	**5**	9
Anomalia de Código		**Feature Envy**			
Tempo (minutos)		53	26	**30**	29
Métodos Identificados		MediaAccessor.addMediaData (F+) MediaAccessor.addMediaArrayOfBytes (F+) MediaAccessor.MediaAccessor (F+) MediaAccessor.updateMediaInfo (F+) MediaAcessor.deleteSingleMediaFromRMS (F+) MediaAccessor.loadMediaBytesFromRMS (F+) MediaAccessor.setMediaInfo (F+) MainUIMidlet.destroyApp (F+)	MediaController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommand (F+) PhotoViewController.getCpVideoScreen (F+) MediaController.incrementCountViews (F+) MediaController.playVideoMedia (F+) MediaController.showImage (F+) AlbumController.resetMediaData (F+)	Nenhum Método	MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.removeRecords (F+) MediaAccessor.addMediaData (F+) MediaAccessor.createNewAlbum (F+)
Número de Acertos		0	0	0	0
Métodos do Oráculo		4	4	4	4
Falsos Positivos		8	8	0	6
Falsos Negativos		4	4	4	4
Cobertura		**0%**	**0%**	**0%**	**0%**
Precisão		**0%**	**0%**	**0%**	**0%**
Métricas Úteis		CBO NOO PAR LOC	NCO LOCC	NCC NCO	LOCC
Métricas Não Úteis		LCOM NOA WMC CYCLO	NOCA NOCO CDLOC NCC	NOCA NOCO LOCC CDLOC	NOCA NCC NOCO CDLOC NCO

Métrica		Métricas de Interesse	Métricas de Interesse	Métricas Híbridas	Métricas Híbridas
Participantes		**S20**	**S21**	**S33**	**S34**
Sexo		M	M	M	F
Idade		27	23	24	23
Cursos	Banco de Dados	x	x	x	
	Programação OO	x	x	x	
	Sistemas Distribuídos				
	Projeto UML	x		x	x
	Java (SE)	x	x	x	
	Java (ME)	x			
	Banco de Dados Avançado				
	Sistemas Tolerantes a Falhas				
Experiência		06 meses a 1 ano	Nenhuma	06 meses a 1 ano	1 ano to 3 anos
Diagrama de Classe (Habilidade)		Média	Pouca	Pouca	Alta
Java (Habilidade)		Média	Média	Média	Pouca
Métrica					
Sistema Usado					
Tempo (minutos)		10	9	4	10
Anomalia de Código		**Feature Envy**			
Tempo (minutos)		36	31	24	40
Métodos Identificados		MediaController.handleCommand (F+) MusicPlayController.handleCommand (F+) AlbumController.handleCommand (F+) MediaController.incrementCoutViews (F+) **MediaListController.exchange (ok)** MediaAccessor.addMediaData (F+) MediaAccessor.removeRecords (F+) MediaController.playVideoMedia (F+) MediaListController.showMediaList (F+)	MediaController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommand (F+) MediaController.incrementCountsViews (F+) MediaController.playVideoMedia (F+)	MediaAccessor.loadAlbums (F+) MediaAccessor.updateMediaInfo (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.removeRecords (F+) MediaAccessor.addMediaData (F+)	MediaController.handleCommand (F+) MediaListController.showMediaList (F+) MediaAccessor.MediaAccessor (F+) MediaAccessor.addMediaArrayOfBytes (F+) MediaAccessor.addMediaData (F+) MediaAccessor.addMediaData (F+) MediaAccessor.createNewAlbum (F+) MediaAccessor.deleteAlbum (F+) MediaAccessor.deleteSingleMediaFromRMS (F+) MediaAccessor.getAlbumNames (F+) MediaAccessor.getByteFromMediaInfo (F+) MediaAccessor.getMediaArrayOfByte (F+) MediaAccessor.getMediaFromBytes (F+) MediaAccessor.getMediaInfo (F+) MediaAccessor.getMediaInfoTable (F+) MediaAccessor.loadAlbums (F+) MediaAccessor.loadMediaBytesFromRMS (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.removeRecords (F+) MediaAccessor.resetRecordStore (F+) MediaAccessor.setMediaInfo (F+) MediaAccessor.setMediaInfoTable (F+) MediaAccessor.updateMediaInfo (F+)
Número de Acertos		1	0	0	0
Métodos do Oráculo		4	4	4	4
Falsos Positivos		8	5	5	23
Falsos Negativos		3	4	4	4
Cobertura		**25%**	**0%**	**0%**	**0%**
Precisão		**11%**	**0%**	**0%**	**0%**
Métricas Úteis		NCC NCO LOCC	CDLOC NCO	CBO LOC	CBO CDLOC NCC
Métricas Não Úteis		NOCA NOCO CDLOC	LOCC NCC NOCA NOCO	LOCC NOCA NOCO CDLOC NCC NCO LCOM NOA NOO PAR WMC CYCLO	LCOM LOC NOA NOO PAR WMC CYCLO NOCA NOCO LOCC NCO

Métrica		Métricas Híbridas	Métricas Híbridas	Métricas Híbridas
Participantes		**S35**	**S36**	**S37**
Sexo		M	F	M
Idade		21	30	
Cursos	Banco de Dados		x	x
	Programação OO	x	x	x
	Sistemas Distribuídos			
	Projeto UML	x	x	x
	Java (SE)	x	x	x
	Java (ME)			
	Banco de Dados Avançado			x
	Sistemas Tolerantes a Falhas			
Experiência		> 3 anos	Nenhuma	> 3 anos
Diagrama de Classe (Habilidade)		Alta	Média	Média
Java (Habilidade)		Média	Pouca	Alta
Métrica				
Sistema Usado				
Tempo (minutos)		14	12	**10**
Anomalia de Código		**Feature Envy**		
Tempo (minutos)		30	25	**29**
Métodos Identificados		MediaAccessor.MediaAccessor (F+) MediaAccessor.addMediaArrayOfBytes (F+) MediaAccessor.addMediaData (F+) MediaAccessor.addMediaData (F+) MediaAccessor.createNewAlbum (F+) MediaAccessor.deleteAlbum (F+) MediaAccessor.deleteSingleMediaFromRMS (F+) MediaAccessor.getAlbumNames (F+) MediaAccessor.getByteFromMediaInfo (F+) MediaAccessor.getMediaArrayOfByte (F+) MediaAccessor.getMediaFromBytes v MediaAccessor.getMediaInfo (F+) MediaAccessor.getMediaInfoTable (F+) MediaAccessor.loadAlbums (F+) MediaAccessor.loadMediaBytesFromRMS (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.removeRecords (F+) MediaAccessor.resetRecordStore (F+) MediaAccessor.setMediaInfo (F+) MediaAccessor.setMediaInfoTable (F+) MediaAccessor.updateMediaInfo(F+)	MediaController.handleCommand (F+) PhotoViewController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommand (F+)	MediaAccessor.MediaAccessor (F+) BaseController.BaseController (F+) MediaListController.MediaListController (F+) AlbumController.AlbumController (F+)
Número de Acertos		0	0	0
Métodos do Oráculo		4	4	4
Falsos Positivos		21	4	4
Falsos Negativos		4	4	4
Cobertura		**0%**	**0%**	**0%**
Precisão		**0%**	**0%**	**0%**
Métricas Úteis		CBO NOO	CYCLO LOC CDLOC LOCC	CBO LCOM PAR
Métricas Não Úteis		LCOM LOC NOA PAR WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO	CBO LCOM NOA NOO PAR WMC NOCA NOCO NCC NCO	LOC NOA NOO WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO

Feature Envy: 2ª Replicação do Experimento (UFMG)
(Experimento aplicado a 22 alunos de Graduação)

Métrica		**Métricas Tradicionais**	**Métricas Tradicionais**	**Métricas Tradicionais**
Participantes		**S9**	**S10**	**S11**
Sexo		M	M	M
Idade		25	32	30
Cursos	Banco de Dados	x	x	x
	Programação OO	x		x
	Tecnologia Web		x	x
	Sistemas Distribuídos			x
	Projeto UML		x	x
	Programação Java			x
	Arquitetura de Software		x	
	Gerenciamento de Projeto			
	Banco de Dados Avançado			
	Sistemas Tolerantes a Falha			
Experiência		1 a 3 anos	> 3 anos	> 3 anos
Diagrama de Classe(Habilidade)		Pouco	Médio	Médio
Java (Habilidade)		Pouco	Pouco	Médio
Métrica		Pouco	Pouco	Pouco
Sistema usado				
Tempo(Minutos)		9	4	9
Anomalia de Código		**Feature Envy**		
Tempo(Minutos)		11	7	19
Métodos Identificados		MediaAccessor.MediaAccessor (F+) MediaAccessor.addMediaArrayOfBytes (F+) MediaAccessor.addMediaData (F+) MediaAccessor.addMediaData (F+) MediaAccessor.createNewAlbum (F+) MediaAccessor.deleteAlbum (F+) MediaAccessor.deleteSingleMediaFromRMS (F+) MediaAccessor.getAlbumNames (F+) MediaAccessor.getByteFromMediaInfo (F+) MediaAccessor.getMediaArrayOfByte (F+) MediaAccessor.getMediaFromBytes v MediaAccessor.getMediaInfo (F+) MediaAccessor.getMediaInfoTable (F+) MediaAccessor.loadAlbums (F+) MediaAccessor.loadMediaBytesFromRMS (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.removeRecords (F+) MediaAccessor.resetRecordStore (F+) MediaAccessor.setMediaInfo (F+) MediaAccessor.setMediaInfoTable (F+) MediaAccessor.updateMediaInfo(F+) MediaController.mediaController (F+) MediaController.getMedia (F+) MediaController.getScreen (F+) MediaController.getSelectMediaName (F+) MediaController.goToPreviousScreen (F+) MediaController.handleCommand (F+) MediaController.incrementCountViews (F+) MediaController.playVideoMedia (F+) MediaController.setMedia (F+) MediaController.setScreen (F+) MediaController.showImage (F+) MediaController.updateMedia (F+) MediaController.playVideoMedia (F+)	MediaAccessor.setAlbumNames (F+) **AlbumController.resetMediaData (ok)**	nenhuma
Número de Acertos		0	1	0
Métodos do Oráculo		4	4	4
Falsos Positivos		34	2	1
Falsos Negativos		4	3	4
Cobertura		**0%**	**25%**	**0%**
Precisão		**0%**	**33%**	**0%**
Métricas Úteis		CBO	CBO LCOM	LOC LCOM
Métricas Não Úteis		LCOM LOC NOA NOO PAR WMC CYCLO	NOO PAR WMC CYCLO NOA NOO	CBO NOA NOO PAR WMC CYCLO

Métrica		Métricas Tradicionais	Métricas Tradicionais	Métricas de Interesse
Participantes		**S12**	**S13**	**S22**
Sexo		M	M	M
Idade		24	26	27
Cursos	Banco de Dados	x		x
	Programação OO	x		x
	Tecnologia Web			x
	Sistemas Distribuídos			
	Projeto UML	x		x
	Programação Java	x		x
	Arquitetura de Software			x
	Gerenciamento de Projeto			x
	Banco de Dados Avançado			x
	Sistemas Tolerantes a Falha			
Experiência		a 1 ano	a 1 ano	1 a 3 anos
Diagrama de Classe(Habilidade)		Médio	Médio	Alto
Java (Habilidade)		Médio	Médio	Médio
Métrica		Pouco	Pouco	Pouco
Sistema usado				
Tempo(Minutos)		4	6	3
Anomalia de Código		**Feature Envy**		
Tempo(Minutos)		26	19	10
Métodos Identificados		MediaController.handleCommand (F+) MediaAccessor.loadAlbums (F+) MediaController.showImage (F+) MediaController.playMusicMedia (F+) MediaController.removeRecords (F+) MainUIMidlet.startApp (F+) MediaAccesor.updateMediaInfo (F+) MediaController.playVideoMedia (F+) MediaAccessor.loadMediaDataFromRMS(F+) MediaAccessor.addMediaData (F+)	PhotoViewController.photoViewController (F+) MusicPlayController.musicPlayController (F+) MediaListController.mediaListController (F+) **MediaListController.exchange (OK)** MediaListController.showMediaList (F+) MediaController.mediaController (F+) MediaAccessor.mediaAccessor (F+) MediaAccessor.addMediaArrayOfBytes (F+) MediaAccessor.addMediaData (F+) BaseController.baseController (F+) AlbumController.albumController (F+)	MediaController.handleCommand (F+) MediaController.incrementCountViews (F+) MediaController.playVideoMedia (F+) MediaListController.showMediaList (F+) AlbumController.handleCommand (F+)
Número de Acertos		0	1	0
Métodos do Oráculo		4	4	4
Falsos Positivos		10	11	5
Falsos Negativos		4	3	4
Cobertura		**0%**	**25%**	**0%**
Precisão		**0%**	**8%**	**0%**
Métricas Úteis		CBO LCOM LOC	PAR	NOCO CDLOC LOCC NCC
Métricas Não Úteis		NOA NOO PAR WMC CYCLO	LOC CYCLO CBO LCOM NOA NOO WMC CYCLO	NOCO NOCA

Métrica		Métricas de Interesse	Métricas de Interesse	Métricas de Interesse
Participantes		**S23**	**S24**	**S25**
Sexo		M	M	
Idade		23	25	21
Cursos	Banco de Dados	x	x	x
	Programação OO	x	x	x
	Tecnologia Web		x	x
	Sistemas Distribuídos			
	Projeto UML		x	x
	Programação Java	x	x	x
	Arquitetura de Software		x	
	Gerenciamento de Projeto			
	Banco de Dados Avançado			
	Sistemas Tolerantes a Falha			
Experiência		None	> 3 anos	a 3 anos
Diagrama de Classe(Habilidade)		Médio	Alto	Médio
Java (Habilidade)		Médio	Alto	Alto
Métrica		Pouco	Pouco	Pouco
Sistema usado				
Tempo(Minutos)		4	5	6
Anomalia de Código		**Feature Envy**		
Tempo(Minutos)		8	9	12
Métodos Identificados		MediaController.handleCommand (F+) MediaController.incrementCountViews (F+)	MediaController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommand (F+)	MediaController.handleCommand (F+) MediaController.incrementeCountViews (F+) MediaController.playVideoMedia (F+) MediaListController.showMediaList (F+) MediaController.gotoPreviousScreen (F+) MediaAccessor.updateMediaInfo (F+)
Número de Acertos		0	0	0
Métodos do Oráculo		4	4	4
Falsos Positivos		2	3	6
Falsos Negativos		4	4	4
Cobertura		**0%**	**0%**	**0%**
Precisão		**0%**	**0%**	**0%**
Métricas Úteis		LOCC CDLOC NCO	CDLOC	NCO CDLOC
Métricas Não Úteis		NOCA NOCO NCC	NOCA NOCO NCC NCO	NOCA NOCO NCC LOCC

Métrica		**Métricas de Interesse**	**Métricas de Interesse**	**Métricas de Interesse**
Participantes		**S26**	**S27**	**S28**
Sexo				
Idade		21	23	22
Cursos	Banco de Dados	x	x	x
	Programação OO	x	x	x
	Tecnologia Web	x		
	Sistemas Distribuídos			
	Projeto UML		x	x
	Programação Java	x	x	x
	Arquitetura de Software		x	
	Gerenciamento de Projeto			
	Banco de Dados Avançado			
	Sistemas Tolerantes a Falha			
Experiência		> 3 anos	1 a 3 anos	a 1 ano
Diagrama de Classe(Habilidade)		Pouco	Médio	Pouco
Java (Habilidade)		Alto	Pouco	Médio
Métrica		Pouco	Pouco	Pouco
Sistema usado		**MobileMedia**		
Tempo(Minutos)		8	10	6
Anomalias de Código		**Feature Envy**		
Tempo(Minutos)		6	20	16
Métodos Identificados		MediaController.handleCommand (F+) MusicPlayController.handleCommand (F+) PhotoViewController.handleCommand (F+) MediaAccessor.updateMediaInfo (F+)	**AlbumController.resetMediaDat (ok)** MediaAccessor.addMediaData (F+) MediaController.goToPreviousScreen (F+)	MediaController.handleCommand (F+) MusicPlayController.handleCommand (F+) AlbumController.handleCommand (F+)
Número de Acertos		0	1	0
Métodos do Oráculo		4	4	4
Falsos Positivos		4	3	3
Falsos Negativos		4	3	4
Cobertura		**0%**	**25%**	**0%**
Precisão		**0%**	**25%**	**0%**
Métricas Úteis		CDLOC NCO	LOCC	CDLOC NCO
Métricas Não Úteis		NOCA NOCO LOCC NCC	NOCA NOCO CDLOC NCC NCO	NOCA NOCO LOCC NCC

Métrica		Métricas de Interesse	Métricas Hibridas	Métricas Hibridas
Participantes		**S29**	**S38**	**S39**
Sexo				
Idade		22	33	24
Cursos	Banco de Dados	x	x	x
	Programação OO	x		x
	Tecnologia Web	x		
	Sistemas Distribuídos			
	Projeto UML	x	x	x
	Programação Java	x		x
	Arquitetura de Software	x		x
	Gerenciamento de Projeto	x	x	
	Banco de Dados Avançado	x	x	
	Sistemas Tolerantes a Falha			
Experiência		a 1 ano	> 3 anos	a 1 ano
Diagrama de Classe(Habilidade)		Médio	Médio	Médio
Java (Habilidade)		Alto	Pouco	Médio
Métrica		Pouco	Pouco	Pouco
Sistema usado				
Tempo(Minutos)		7	6	6
Anomalia de Código		**Feature Envy**		
Tempo(Minutos)		22	15	22
Métodos Identificados		MediaController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommand (F+) MediaController.showMediaList (F+)	MediaAcessor.loadAlbums (F+) MediaAcessor.updateMediaInfo (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.removeRecords MediaAccessor.addMediaData (F+)	MediaController.handleCommand (F+) MediaAccessor.loadAlbums (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.localMediaBytesFromRMS (F+) MediaAccessor.addMediaData (F+)
Número de Acertos		0	0	0
Métodos do Oráculo		4	4	4
Falsos Positivos		0	5	5
Falsos Negativos		4	4	4
Cobertura		**0%**	**0%**	**0%**
Precisão		**#DIV/0!**	**0%**	**0%**
Métricas Úteis		CDLOC	CBO LOC	CBO LOCC NOCO
Métricas Não Úteis		NOCA NOCO LOCC NCC NCO	LCOM NOA NOO PAR WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO	LCOM LOC NOA NOO PAR WMC CYCLO NOCA CDLOC NCC NCO

Métrica		Métricas Híbridas	Métricas Híbridas	Métricas Híbridas
Participantes		**S40**	**S41**	**S42**
Sexo				
Idade		26	22	22
Cursos	Banco de Dados	x	x	x
	Programação OO	x	x	x
	Tecnologia Web	x		
	Sistemas Distribuídos			
	Projeto UML	x		
	Programação Java	x	x	x
	Arquitetura de Software	x		
	Gerenciamento de Projeto	x		
	Banco de Dados Avançado			
	Sistemas Tolerantes a Falha			
Experiência		1 a 3 anos	a 1 ano	a 1 ano
Diagrama de Classe(Habilidade)		Médio	Médio	Pouco
Java (Habilidade)		Médio	Médio	Pouco
Métrica		Pouco	Pouco	Pouco
Sistema usado				
Tempo(Minutos)		7	7	6
Anomalia de Código		**Feature Envy**		
Tempo(Minutos)		20	11	8
Métodos Identificados		MediaAccessor.handleCommand (F+) MediaController.addMedia (F+) MediaAccessor.createNewAlbum (F+)	MediaAccessor.addAlbum (F+) MediaAccessor.addMediaData (F+) MediaAccessor.updateMediaInfo (F+) MediaController.handleCommand (F+) MediaController.playMusicMedia (F+) MediaController.playVideoMedia (F+) MediaController.incrementCountViews (F+) MediaController.goToPrevieusScrenn (F+)	MediaAccessor.mediaAccessor (F+) MediaAccessor.addMediaArrayOfBytes (F+) MediaAccessor.addMediaData (F+) MediaController.mediaController (F+) MediaAccessor.deleteSingleMediaFromRMS (F+) MediaAccessor.loadMediaBytesFromRMs (F+) MediaAccessor.setMediaInfo (F+) MediaAccessor.updateMediaInfo (F+)
Número de Acertos		0	0	0
Métodos do Oráculo		4	4	4
Falsos Positivos		3	8	8
Falsos Negativos		4	4	4
Cobertura		**0%**	**0%**	**0%**
Precisão		**0%**	**0%**	**0%**
Métricas Úteis		CBO LOC WMC	CBO LOC CYCLO	CBO PAR
Métricas Não Úteis		NOO LCOM NOA CYCLO PAR NOCA NOCO LOCC CDLOC NCC NCO	LCOM NOA NOO PAR WMC NOCA NOCO LOCC CDLOC NCC NCO	LCOM LOC NOA NOO WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO

Métrica		Métricas Híbridas
Participantes		**S43**
Sexo		
Idade		25
Cursos	Banco de Dados	x
	Programação OO	x
	Tecnologia Web	x
	Sistemas Distribuídos	
	Projeto UML	x
	Programação Java	x
	Arquitetura de Software	x
	Gerenciamento de Projeto	x
	Banco de Dados Avançado	
	Sistemas Tolerantes a Falha	
Experiência		nenhuma
Diagrama de Classe(Habilidade)		Pouco
Java (Habilidade)		Pouco
Métrica		Pouco
Sistema usado		
Tempo(Minutos)		6
Anomalia de Código		**Feature Envy**
Tempo(Minutos)		16
Métodos Identificados		MediaAccessor.addMediaData (F+) MediaAccessor.loadMediaBytesFromRMS (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.updateMediaInfo (F+)
Número de Acertos		0
Métodos do Oráculo		4
Falsos Positivos		4
Falsos Negativos		4
Cobertura		**0%**
Precisão		**0%**
Métricas Úteis		CBO PAR NCO
Métricas Não Úteis		WMC NOO LCOM NOA LOC NOO CYCLO NOCA NOCO LOCC CDLOC NCC

Feature Envy: 3º Replicação de Experimento (UFBA)
(Experimento aplicado a 10 alunos de Pós-Graduação)

Métrica		**Métricas Tradicionais**	**Métricas Tradicionais**	**Métricas Tradicionais**
Participantes		**S14**	**S14**	**S16**
Sexo		M	F	M
Idade		29	25	22
Cursos	Banco de Dados	x	x	x
	Programação OO	x	x	x
	Tecnologia Web	x	x	x
	Sistemas Distribuídos	x		
	Projeto UML	x		x
	Programação Java	x	x	x
	Gerenciamento de Projeto	x	x	x
	Sistema de Gerenciamento Projeto	x		x
	Banco de Dados Avançado	x	x	
	Sistema Tolerante Falhas	x		
Experiência de Trabalho		nenhum	Até 1 ano	até 1 ano
Diagrama de Classe		alto	pouco	pouco
Java (Habilidade)		alto	médio	pouco
Métrica		alto	pouco	pouco
Sistema Usado		**MobileMedia**		
Tempo (minutos)		8	13	21
Anomalia de Código		**Feature Envy**		
Tempo (minutos)		14	24	26
Métodos Identificados		MediaController.handleCommand (F+) PhotoViewController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommad (F+) MainUiMidlet.startApp (F+)	MediaController.handleCommand (F+) AlbumController.handleCommand (F+) PhotoViewController.handleCommand (F+) MusicPlayController.handleCommand (F+) MediaAccessor.addMediaData (F+) MediaController.playMusicMedia (F+)	BaseController.goToPreviousScreen (F+) **AlbumController.resetMediaData (ok)**
Número de Acertos		0	0	1
Métodos do Oráculo		4	4	4
Falsos Positivos		5	6	1
Falsos Negativos		4	4	3
Cobertura		**0%**	**0%**	**25%**
Precisão		**0%**	**0%**	**50%**
Métricas Úteis		LOC CYCLO	LOC PAR CYCLO	PAR NOA LCOM
Métricas Não Úteis		CBO LCOM NOA NOO PAR WMC	WMC CBO NOA NOO LCOM	LOC CYCLO CBO NOO WMC

Métrica		**Métricas de Interesse**	**Métricas de Interesse**	**Métricas de Interesse**
Participantes		**S30**	**S31**	**S32**
Sexo		M	M	F
Idade		25	23	24
Cursos	Banco de Dados	x	x	x
	Programação OO	x	x	x
	Tecnologia Web	x		x
	Sistemas Distribuídos		x	
	Projeto UML	x	x	x
	Programação Java	x	x	x
	Gerenciamento de Projeto	x	x	x
	Sistema de Gerenciamento Projeto	x	x	
	Banco de Dados Avançado			
	Sistemas Tolerantes a Falhas			
Experiência de Trabalho		de 1 a 3 anos	1 ano	nenhum
Diagrama de Classe (Habilidade)		médio	médio	alto
Java (Habilidade)		médio	médio	médio
Métrica		pouco	pouco	pouco
Sistema Usado		**MobileMedia**		
Tempo (minutos)		7	11	9
Anomalia de Código		**Feature Envy**		
Tempo (minutos)		30	24	17
Métodos Identificados		MediaController.handleCommand (F+) AlbumController.handleCommand (F+)	MediaController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommand (F+) PhotoViewController.handleCommand (F+)	MediaController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommand (F+)
Número de Acertos		0	0	0
Métodos do Oráculo		4	4	4
Falsos Positivos		18	18	16
Falsos Negativos		4	4	4
Cobertura		**0%**	**0%**	**0%**
Precisão		**0%**	**0%**	**0%**
Métricas Úteis		CDLOC NCO	CDLOC LOCC NCO NCC	CDLOC NCC NCO
Métricas Não Úteis		NCC NOCA NOCO LOCC	NOCA NOCO	NOCA NOCO LOCC

Métrica		**Métricas Híbridas**	**Métricas Híbridas**	**Métricas Híbridas**
Participantes		**S44**	**S45**	**S46**
Sexo		F	M	M
Idade		25	24	41
Cursos	Banco de Dados	x	x	x
	Programação OO	x	x	x
	Tecnologia Web	x		x
	Sistemas Distribuídos		x	x
	Projeto UML	x		x
	Programação Java	x	x	x
	Gerenciamento de Projeto	x		x
	Sistema de Gerenciamento Projeto	x	x	x
	Banco de Dados Avançado			x
	Sistemas Tolerântes a Falhas			x
Experiência de Trabalho		nenhum	nenhum	> 3 anos
Diagrama de Classe (Habilidade)		pouco	médio	médio
Java (Habilidade)		pouco	médio	alto
Métrica		pouco	pouco	médio
Sistema Usado		**MobileMedia**		
Tempo (minutos)		6	9	10
Anomalia de Código		**Feature Envy**		
Tempo (minutos)		43	38	29
Métodos Identificados		MediaController.handleCommand (F+) PhotoViewController.handleCommand (F+) AlbumController.handleCommand (F+) MusicPlayController.handleCommand (F+) BaseController.handleCommand (F+) MediaListController.handleCommand (F+) BaseController.goToPreviousScreen (F+) AlbumController.goToPreviousScreen(F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.addMediaData (F+)	MediaAccessor.addMediaData (F+) MediaAccessor.addMediaArrayOfBytes (F+)	MediaController.handleCommand (F+)
Número de Acertos		0	0	0
Métodos do Oráculo		4	4	4
Falsos Positivos		10	2	1
Falsos Negativos		4	4	4
Cobertura **Precisão**		**0%** **0%**	**0%** **0%**	**0%** **0%**
Métricas Úteis		LOCC LOC CYCLO	CBO PAR	CDLOC
Métricas Não Úteis		CBO WMC NOO LCOM NOA PAR NOCA	WMC LCOM LOC NOA NOO CYCLO NOCA	NOCA NOCO LOCC CDLOC NCC NCO CBO

Anexo H

Resultado da Aplicação Experimento para God Method

Este anexo apresenta os dados obtidos com a aplicação do experimento com os participantes na identificação do *God Method*.
Esta anomalia foi aplicada para o sistema MobileMedia.

1ª Replicação do Experimento (UFMG)
(Experimento aplicado a 15 alunos de Pós-Graduação)

Métrica	Métricas Tradicionais	Métricas Tradicionais	Métricas Tradicionais	Métricas Tradicionais
Participante	**S1**	**S2**	**S3**	**S4**
Tempo (minutos)	11	13	9	13
Métodos Identificados	**MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **MediaListController.showMediaList (ok)**	MediaController.goToPreviousScreen (F+) **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **MediaListController.showMediaList (ok)** MediaAcessor.loadAlbums (F+)	**MediaController.handleCommand (ok)** **MediaListController.showMediaList (ok)** **MusicPlayController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)**	**AlbumController.handleCommand (ok)** **MediaController.handleCommand (ok)** **MediaListController.showMediaList (ok)** **MusicPlayController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)**
Número de Acertos	5	4	5	5
Métodos do Oráculo	7	7	7	7
Falsos Positivos	0	2	0	0
Falsos Negativos	2	3	2	2
Cobertura	**71%**	**57%**	**71%**	**71%**
Precisão	**100%**	**67%**	**100%**	**100%**
Métricas Úteis	LOC CYCLO	LOC CYCLO	LOC CYCLO	LOC CYCLO
Métricas Não Úteis	CBO CBO NOA NOO PAR WMC	CBO LCOM NOA NOO PAR WMC	CBO LCOM NOA NOO PAR WMC	CBO LCOM NOA NOO PAR WMC

Métrica	Métricas Tradicionais	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse
Participante	**S5**	**S17**	**S18**	**S19**
Tempo (minutos)	14	15		23
Métodos Identificados	**MainUIMidlet.startApp (ok)** **PhotoViewController.handleCommand (ok)** **MediaController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)**	**MediaController.handleCommand (ok)**	Nenhum método	MediaAccessor.loadAlbums (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.removeRecords (F+) MediaAccessor.addMediaData (F+) MediaController.incrementCountViews (F+) MediaListController.exchange (F+)
Número de Acertos	4	1	0	0
Métodos do Oráculo	7	7	7	7
Falsos Positivos	0	0	0	6
Falsos Negativos	3	6	7	7
Cobertura	**57%**	**14%**	**0%**	**0%**
Precisão	**100%**	**100%**	**0%**	**0%**
Métricas Úteis	LOC PAR CYCLO	NCO LOCC	LOCC NCC	LOCC CDLOC
Métricas Não Úteis	CBO LCOM NOA NOO WMC	NOCA NOCO CDLOC NCC	NOCA NOCO CDLOC NCC	NOCA NOCO NCC NCO

Métrica	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse
Participante	**S20**	**S21**	**S33**	**S34**
Tempo (minutos)	24	14	13	15
Métodos Identificados	**MediaController.handleCommand (ok)** MediaController.incrementCoutViews (F+) MediaController.playVideoMedia (F+) **MediaListController.showMediaList (ok)** MediaAccessor.addMediaData (F+) MediaAccessor.removeRecords (F+)	MediaController.handleCommand (F+) MediaController.incrementCountsViews (F+) MediaController.playVideoMedia (F+) MediaListController.showMediaList (F+)	**MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **MainUIMidlet.startApp (ok)**	**MediaController.handleCommand (ok)** MediaController.incrementCountViews (F+) MediaController.playVideoMedia (F+) MediaController.goToPreviousScreen (F+) **MediaController.showImage (ok)** MediaController.playMusicMedia (F+) **MediaController.showMediaList (ok)** MediaListController.handleCommand (F+)
Número de Acertos	2	0	5	3
Métodos do Oráculo	7	7	7	7
Falsos Positivos	4	4	0	5
Falsos Negativos	5	7	2	4
Cobertura	**29%**	**0%**	**71%**	**43%**
Precisão	**33%**	**0%**	**100%**	**38%**
Métricas Úteis	NCC NCO LOCC	NCO LOCC	LOC PAR NOO NCO	NCC NCO CDLOC
Métricas Não Úteis	NOCA NOCO CDLOC	NOCA NOCO CDLOC NCC	CBO LCOM NOA WMC CYCLO NOCA NOCO LOCC CDLOC NCC	CBO LCOM LOC NOA PAR WMC CYCLO NOCA NOCO LOCC

Métrica	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse
Participante	**S35**	**S36**	**S37**
Tempo (minutos)	8	20	
Métodos Identificados	**AlbumController.handleCommand (ok)** **MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)**	**MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)**	**MediaController.HandleCommand (ok)** **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **MainUIMidlet.startApp (ok)** **MediaListController.showMediaList (ok)**
Número de Acertos	4	4	6
Métodos do Oráculo	7	7	7
Falsos Positivos	0	0	0
Falsos Negativos	3	3	1
Cobertura	**57%**	**57%**	**86%**
Precisão	**100%**	**100%**	**100%**
Métricas Úteis	LOC PAR CYCLO NCO	CYCLO LOC CDLOC LOCC	LOC CYCLO
Métricas Não Úteis	CBO LCOM NOA NOO WMC NOCA NOCO LOCC CDLOC NCC	CBO LCOM NOA NOO PAR WMC NOCA NOCO NCC NCO	PAR CBO LCOM NOA NOO WMC NOCA NOCO LOCC CDLOC NCC NCO

Métrica	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse
Participante	**S22**	**S23**	**S24**	**S25**
Tempo (minutos)	4	9	5	8
Métodos Identificados	**MediaController.handleCommand (ok)** MediaController.incrementCountViews (F+) MediaControler.playVideoMedia (F+) **MediaListController.showMediaList (ok)** **AlbumController.handleCommand (ok)**	**MediaController.handleCommand (ok)** MediaController.incrementCountViews (F+) MediaController.playVideoMedia (F+) **MediaListController.showMediaList (ok)**	**MediaController.handleCommand (ok)** MediaAccessor.loadAlbums (F+) MediaAccessor.loadMediaDataFromRMS (F+) MediaAccessor.removeRecords (F+)	**MediaController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)**
Número de Acertos	3	2	1	4
Métodos do Oráculo	7	7	7	7
Falsos Positivos	2	2	3	0
Falsos Negativos	4	5	6	3
Cobertura	**43%**	**29%**	**14%**	**57%**
Precisão	**60%**	**50%**	**25%**	**100%**
Métricas Úteis	NCO CDLOC LOCC NCC	NCO CDLOC	LOCC NOCA NOCO NCC NCO	CDLOC LOCC
Métricas Não Úteis	NOCO NOCA	NOCA NOCO LOCC NCC	CDLOC	NOCA NOCO NCC NCO

Métrica	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse
Participante	**S26**	**S27**	**S28**	**S29**
Tempo (minutos)	4	5	10	12
Métodos Identificados	**MediaController.handleCommand (ok)** MediaController.incrementCountViews (F+) MediaController.playVideoMedia (F+) MediaListController.showMediaList (F+)	**AlbumController.handleCommand (ok)** **MediaController.handleCommand (ok)** **MediaListController.showMediaList (ok)**	**MediaController.handleCommand (ok)** MediaAccessor.loadAlbuns (F+) **AlbumController.handleCommand (ok)** MediaAccessor.addMediaData (F+) MediaController.incrementeCountViews (F+)	**Nenhuma**
Número de Acertos	1	3	2	0
Métodos do Oráculo	7	7	7	7
Falsos Positivos	3	0	3	0
Falsos Negativos	6	4	5	7
Cobertura	**14%**	**43%**	**29%**	**0%**
Precisão	**25%**	**100%**	**40%**	**0%**
Métricas Úteis	NCO	NCO	NOCO LOCC NCC	NCO
Métricas Não Úteis	NOCA NOCO LOCC CDLOC NCC	NOCA NOCO LOCC CDLOC NCC	NOCA CDLOC NCO	NOCA NOCO LOCC CDLOC NCC

Métrica	Métricas Híbridas	Métricas Híbridas	Métricas Híbridas	Métricas Híbridas
Participante	**S38**	**S39**	**S40**	**S41**
Tempo (minutos)	5	6	3	12
Métodos Identificados	**AlbumController.handleCommand (ok)** **MainUIMidlet.startApp (ok)** MediaAccessor.loadAlbuns (F+) MediaAccessor.updateMediaInfo (F+)	**MediaController.handleCommand (ok)** **MainUIMidlet.startAPP (ok)** **MusicPlayController.handleCommand (ok)**	MediaController.goToPreviousScreen (F+) MediaController.incrementContView (F+) **MediaController.showImage (ok)**	**MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **AlbumController.handleCommand (ok)**
Número de Acertos	2	3	1	4
Métodos do Oráculo	7	7	7	7
Falsos Positivos	2	0	2	0
Falsos Negativos	5	4	6	3
Cobertura	**29%**	**43%**	**14%**	**57%**
Precisão	**50%**	**100%**	**33%**	**100%**
Métricas Úteis	LOC CYCLO	LOC LCOM	WMC PAR	LOC
Métricas Não Úteis	LCOM NOA NOO PAR WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO	CBO NOA NOO PAR WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO	NOA NOO CBO LCOM PAR WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO	CBO LCOM NOA NOO PAR WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO

Métrica	Métricas Híbridas	Métricas Híbridas
Participante	**S42**	**S43**
Tempo (minutos)	14	7
Métodos Identificados	**MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **MainUIMidlet.startApp (ok)** **MediaListController.showMediaList (ok)**	**MediaController.handleCommand (ok)** **MediaListController.showMediaList (ok)** **MusicPlayController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)**
Número de Acertos	6	4
Métodos do Oráculo	7	7
Falsos Positivos	0	0
Falsos Negativos	1	3
Cobertura	**86%**	**57%**
Precisão	**100%**	**100%**
Métricas Úteis	LOC PAR NCC	LOCC LOC CYCLO
Métricas Não Úteis	CBO LCOM NOA NOO WMC CYCLO NOCA NOCO CDLOC NCO	NCC NOA CBO LCOM NOO PAR WMC NOCA NOCO CDLOC NCO

3a Replicação de Experimentp(UFBA)
(Experimento aplicado a 10 alunos de Pós-Graduação)

Métrica	**Métricas Tradicionais**	**Métricas Tradicionais**	Métricas Tradicionais
Participante	**S14**	**S15**	**S16**
Tempo (minutos)	8	12	7
Métodos Identificados	**MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MusicPlayController.handleCommad (ok)** **MainUiMidlet.startApp (ok)**	**MediaController.handleCommand (ok)** BaseController.baseToPreviousScreen (F+) MediaAccessor.updateMediaInfo(F+) MediaController.playMusicMedia (F+)	**MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **MediaListController.showMediaList (ok)** **AlbumController.handleCommand (ok)** AlbumController.resetMediaData (F+) MediaAccessor.loadalbums (F+)
Número de Acertos	5	4	5
Métodos do Oráculo	7	7	7
Falsos Positivos	0	3	2
Falsos Negativos	2	3	2
Cobertura	**71%**	**57%**	**71%**
Precisão	**100%**	**57%**	**71%**
Métricas Úteis	LOC	CBO LOC NOA	LOC CYCLO
Métricas Não Úteis	CBO LCOM NOA NOO PAR WMC CYCLO	LCOM NOO PAR WMC CYCLO	NOA CBO LCOM NOO PAR WMC

Métrica	Métricas de Interesse	Métricas de Interesse	Métricas de Interesse
Participante	**S30**	**S31**	**S32**
Tempo (minutos)	16	9	15
Métodos Identificados	**MediaController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **AlbumController.handleCommand (ok)** MediaController.playVideoMedia (F+)	**MediaController.handleCommand (ok)** MediaController.incrementCountViews (F+) MediaController.playVideoMedia (F+) **MediaListController.showMediaList (ok)**	**MediaController.handleCommand (ok)** MediaAccessor.loadAlbums (F+) MediaAccessor.loadMediaDataFromRMS (F+)
Número de Acertos	3	2	0
Métodos do Oráculo	7	7	7
Falsos Positivos	1	2	6
Falsos Negativos	4	5	7
Cobertura	**43%**	**29%**	**0%**
Precisão	**75%**	**0%**	**0%**
Métricas Úteis	NOCA NOCO LOCC CDLOC NCO	NCO LOCC	LOCC
Métricas Não Úteis	NCC	NOCA NOCO CDLOC NCO	NOCA NOCO CDLOC NCC NCO

Métrica	Métricas Híbridas	Métricas Híbridas	Métricas Híbridas
Participante	**S44**	**S45**	**S46**
Tempo (minutos)	8	9	11
Métodos Identificados	MediaAccessor.addMediaData (F+) MediaAccessor.createNewAlbum (F+) MediaAccessor.deleteAlbum (F+) MediaAccessor.deleteSingleMediaFromRMS (F+) MediaAccessor.loadMediaBytesFromRMS (F+) MediaAccessor. removeRecords (F+) MediaAccessor.mediaAccessor (F+) MediaAccessor.addMediaArrayOfBytes (F+) MediaAccessor.getAlbumNames (F+) MediaAccessor.ByteFromMediaInfo (F+) MediaAccessor.getMediaArrayOfBytes (F+) MediaAccessor.getMediaInfo (F+) MediaAccessor.getMediaInfoTable (F+) MediaAccessor.resetRecordStore (F+) MediaAccessor.setMediaInfo (F+)	**AlbumController.handleCommand (ok)** **MediaController.handleCommand (ok)** **MusicPlayController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **MediaListController.showMediaList (ok)**	**MediaController.handleCommand (ok)** **PhotoViewController.handleCommand (ok)** **AlbumController.handleCommand (ok)** **MainUiMidlet.startApp (ok)**
Número de Acertos	5	3	4
Métodos do Oráculo	7	7	7
Falsos Positivos	0	5	0
Falsos Negativos	2	4	3
Cobertura	**71%**	**43%**	**57%**
Precisão	**100%**	**38%**	**100%**
Métricas Úteis	NOCO NCO	LOC CYCLO NCO	LOC
Métricas Não Úteis	NOA NOO PAR WMC CYCLO NOCA LOCC CDLOC NCC	CBO LCOM NOA NOO PAR WMC NOCA NOCO LOCC CDLOC NCC	CBO LCOM NOA NOO PAR WMC CYCLO NOCA NOCO LOCC CDLOC NCC NCO

Anexo I

Passos para Instalação do ConcernMeBS

ConcernMeBs foi desenvolvido como um *plugin* do Eclipse 3.7.2 utilizando Java 6. O ConcernMeBs depende de três *plugins*:

1. Metrics Plugin 1.3.6;
2. ConcernMapper 2.0.3;
3. ConcernMorph 1.0.0.

Estes *plugins* devem ser instalados conjuntamente ao ConcernMeBs.

Para instalação do Metrics Plugin, execute o Eclipse. Vá ao menu Ajuda → Instalar novo software. Na caixa de diálogo aberta, adicione um novo site remoto com a URL (`http://metrics.sourceforge.net/update`), e siga as instruções para instalação.

Para a instalação de ConcernMapper, ConcernMopher e ConcermMeBs baixe os arquivos no *website* (`http://homepages.dcc.ufmg.br/~juliana.padilha/plugin/`):

- ConcernMapper.zip;
- ConcernMorph.zip;
- ConcermMeBs.zip.

Descompacte os arquivos e copie a pasta ConcernMapper, o arquivo JAR ConcernMorph e o arquivo JAR ConcernMeBs para dentro pasta "plugins" do Eclipse e reinicie o programa.