A NATUREZA EVOLUTIVA DOS *BAD SMELLS* EM SOFTWARE ORIENTADO POR OBJETOS - UM ESTUDO EMPÍRICO APLICANDO VALORES DE REFERÊNCIA PARA MÉTRICAS DE SOFTWARE

CHARLES HENRIQUE ALVARENGA

A NATUREZA EVOLUTIVA DOS *BAD SMELLS* EM SOFTWARE ORIENTADO POR OBJETOS - UM ESTUDO EMPÍRICO APLICANDO VALORES DE REFERÊNCIA PARA MÉTRICAS DE SOFTWARE

Proposta de 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: Mariza Andrade da Silva Bigonha
Coorientador: Kecia Aline Marques Ferreira

Belo Horizonte, MG

Outubro de 2015

Sumário

Capítulo 1

Introdução

Sistemas de software podem ter um tempo de vida longo, como por exemplo, sistemas de controle de tráfego aéreo que podem ter uma vida útil de 30 anos ou mais [Sommerville, 2010]. Para que um software seja útil durante toda sua vida é necessário que aconteçam evoluções nele. Essas ações ocorrem, por exemplo, devido às mudanças no negócio e mudanças nas expectativas dos usuários do sistema, gerando novas necessidades para o software existente. Essas evoluções e mudanças também podem ser desencadeadas por relatos de defeito no software ou por mudanças em outros sistemas [Sommerville, 2010].

Em algum estágio no ciclo de vida de um software, ele atinge um ponto de transição onde a realização de mudanças significativas e a implementação de novas exigências se tornam cada vez mais complexas e é necessário atenção e definição de ações para a sua melhoria da qualidade e, até mesmo, para retardar o seu envelhecimento [Parnas, 1994]. Sabe-se também que a estrutura e a qualidade de um software tende a degradar ao longo do tempo quando novas funcionalidades lhes são adicionadas, pois elas podem corromper a estrutura original do software, além de tornar mais difícil e caro realizar atividades de modificação no código [Sommerville, 2010].

Para modificar o código é necessário identificar os pontos susceptíveis a modificações, e essa atividade pode ser muito complexa ou mesmo uma tarefa inviável, principalmente se for considerado softwares de grande porte [Ferreira, 2011]. A fim de facilitar a identificação de áreas no código com possíveis problemas estruturais, Fowler [1999] introduziu o conceito de *bad smells* como um indicador de um possível problema estrutural no código que pode ser melhorado via refatoração no software. Refatoração diz respeito a mudanças realizadas na estrutura interna de um sistema para torná-lo mais fácil de entender e mais barato para modificar, mantendo seu comportamento observável [Sommerville, 2010]. Fowler [1999] apresenta em seu trabalho uma lista de

bad smells descrevendo como reconhecer tais anomalias no código, com o objetivo de facilitar a identificação de pontos para a realização de refatoração.

Vários estudos foram desenvolvidos a partir dos conceitos de Fowler [1999] para a identificação dos *bad smells*. Por exemplo, Marinescu [2005], definiu estratégias de detecção para o reconhecimento de *bad smells* no código utilizando métricas de software. Marinescu [2005] define estratégias de detecção como sendo uma condição lógica baseada em parâmetros por meio da qual são detectados fragmentos de código com propriedades específicas definidas nos *bad smells*. Lanza et al. [2007] desenvolveram um trabalho onde definiram novos *bad smells* e estratégias de detecção para a identificação de *bad smells* no código.

Segundo a definição de Marinescu [2005] e afirmada por Munro [2005] e Fontana et al. [2011], as estratégias de detecção de *bad smells* necessitam de métricas e estas precisam de valores de referência para identificar os desvios nos projetos. Neste campo de pesquisa, Ferreira [2011] desenvolveu um trabalho onde identifica valores de referência para 6 métricas de software orientado por objetos e Filó [2014] apresenta um catálogo de valores de referência para 18 métricas que podem ser utilizadas nas estratégias de detecção de *bad smells*.

Para Kruchten et al. [2012], os *bad smells* também são considerados indicadores do grau de dificuldade técnica e capacidade de manutenção de um software. Existem na literatura vários estudos que investigam a relevância dos *bad smells* para avaliação da qualidade de código, principalmente do ponto de vista do desenvolvedor [Arcoverde et al., 2011; Yamashita & Moonen, 2013; Palomba et al., 2014]. Muito embora um *bad smell* seja considerado um indicador de um possível problema e esteja diretamente ligado com a qualidade do software, ainda não há um consenso entre os pesquisadores da área em relação ao momento e por que eles são inseridos no código.

Considera-se até então que as atividades de manutenção e evolução do software são responsáveis pela frequente causa da inclusão de *bad smells* nos sistemas [Tufano et al., 2015; Lehman & Ramil, 2001]. Porém, estudos recentes contrariam tal afirmação, apresentando resultados que demonstram que a maioria dos *bad smells* são introduzidos no código quando o software é criado [Chatzigeorgiou & Manakos, 2010; Tufano et al., 2015].

Nesse contexto, o objetivo principal deste trabalho de pesquisa é realizar um estudo empírico aplicando valores de referência para métricas de software a fim de analisar a natureza evolutiva dos *bad smells* em softwares orientados por objetos, investigando quando *bad smells* são introduzidos nos softwares.

1.1 Objetivos Específicos

Para atender o objetivo principal desta proposta de dissertação, especificado na Seção 1, os seguintes objetivos específicos são definidos para este trabalho:

1. fazer uma revisão bibliográfica acerca de métricas, valores referências, *bad smells* e evolução de software;

2. identificar os *bad smells*, métricas, valores de referência para as métricas e as estratégias de detecção dos *bad smells*;

3. descrever os padrões evolutivos do software do ponto de vista de *bad smells* a partir dos resultados observados no estudo empírico realizado.

1.2 Motivação

Nos últimos anos, houve um crescimento no interesse de tentar compreender melhor os padrões de evolução de um software. Este fenômeno pode ser observado no trabalho realizado por Syeed et al. [2013] que mostra que o número de artigos publicados sobre o tema cresceu 80% em 2012 se comparado com o número de artigos publicados sobre o tema em 2000. Vários fatores estão relacionados diretamente com a qualidade do software, dentre eles destacam se os *bad smells*. Porém, como apresentado na seção 1, ainda existem poucos estudos que analisam a evolução dessas anomalias de código [Tufano et al., 2015; Chatzigeorgiou & Manakos, 2010; Olbrich et al., 2009], inclusive com discordância entre eles sobre o momento em que *bad smells* são inseridos no código.

Diante desses fatos, a motivação deste trabalhoé o fato de que é importante que as equipes de manutenção e evolução de software tenham subsídios e informação sobre o surgimento e evolução dos *bad smells*, para que ações e atividades de refatoração possam ser planejadas e priorizadas corretamente. Esse trabalho visa contribuir com a consolidação de conhecimento nesta área.

1.3 Contribuição Pretendida

Como resultado deste trabalho de dissertação espera-se alcançar as seguintes contribuições principais:

1. identificação de quando os *bad smells* são introduzidos nos projetos de software;

2. discussão sobre a evolução dos *bad smells* na estrutura interna do software orientado por objetos;

3. orientação, para que as equipes de desenvolvimento de software direcionem seus esforços para as atividades de refatoração e ações de melhoria da qualidade do software a partir do entendimento da evolução dos *bad smells*.

1.4 Organização da proposta

O restante desta proposta de dissertação está organizada como mostrado a seguir. Capítulo 2 inicialmente define alguns conceitos importantes para o entendimento dos temas abordados nesse projeto de pesquisa, como: métricas, *bad smells* e valores referência; apresenta uma revisão bibliográfica preliminar mostrando os trabalhos relacionados com o tema proposto para essa dissertação, enfatizando aqueles mais importantes a fim de fornecer subsídio para que seja possível avançar no estado da arte.

Capítulo 3 descreve a solução proposta para o problema enunciado no fim do Capítulo 1; apresenta a metodologia que será usada no desenvolvimento desse trabalho de pesquisa, o cronograma das atividades; e termina com uma proposta de sumário para esta dissertação.

Capítulo 2

Revisão da Literatura

Neste capítulo são introduzidos alguns conceitos fundamentais para o entendimento do trabalho que se propõe a desenvolver nesta dissertação. É apresentada uma revisão sucinta da literatura e uma revisão dos trabalhos com foco nas soluções que envolvem os objetivos deste trabalho.

2.1 Métricas

Em qualquer processo que tenha como meta atingir a qualidade de um produto final, é imprescindível que haja medição. Sommerville [2010] define medição como um valor numérico relacionado a algum atributo de software, permitindo a comparação entre atributos da mesma natureza. Sommerville ainda argumenta que por meio das métricas é possível controlar um software, melhorá-lo, planejá-lo e identificar componentes com anomalias. [Filó, 2014] define que as medidas são utilizadas para permitir que modelos criados tenham sua qualidade avaliada, bem como permitir que se possa verificar se o padrão de qualidade estabelecido está sendo atingido no fim do processo.

No contexto de Engenharia de Software, as métricas possuem um papel fundamental na qualidade, pois fornecem aos engenheiros de software dados que lhes permitem avaliar os processos, projetos e produtos [Ferreira, 2011].

Existem 2 conjuntos de métricas para orientação por objetos disponível na literatura que são popularmente conhecidos como Métricas CK [Chidamber & Kemerer, 1994] e Métricas MOOD [Brito e Abreu & Carapuca, 1994] e se destacam pela grande relevância em outras pesquisas. O conjunto de métricas CK é composto por 6 métricas de software orientado por objetos, enquanto no conjunto de métricas MOOD são definidas 8 métricas de software orientado por objetos, que avaliam aspectos de herança, coesão, encapsulamento, acoplamento, polimorfismo e reúso de software.

2.2 *Bad Smells*

Fowler [1999] define o conceito de *bad smells* como um indicador de um possível problema estrutural no código que pode ser melhorado via refatoração no software. Ele apresenta em seu trabalho uma lista de 22 *bad smells* e mostra como identificar essas anomalias no código. A seguir são apresentadas as definições dos *bad smells* relacionados com o trabalho de pesquisa proposto.

- *Duplicated Code* significa que uma mesma estrutura de código aparece em mais de um local no software [Fowler, 1999].
- *Feature Envy* diz que um método possui maior relação com outra classe do que com a classe na qual está localizado. Portanto, esse fato aumenta o grau de acoplamento do método com outras classes e diminui o grau de coesão em relação aos demais métodos de sua classe de origem [Fowler, 1999].
- *God Class* refere-se a classes que tende a centralizar a inteligência e a responsabilidade no sistema [Lanza et al., 2007].
- *Shotgun Surgery* refere-se a classes que ao serem alteradas causam impacto em outras classes [Lanza et al., 2007].
- *Long Parameter List* ocorre em método com uma longa lista de parâmetros, o que pode estar associado à dificuldade de compreensão do método [Fowler, 1999].
- *State Checking* se manifesta como instruções condicionais que selecionam um caminho de execução com base no estado de um objeto [Chatzigeorgiou & Manakos, 2010].
- *Long Method* são métodos com um código grande, alta complexidade e baixa coesão, demandando mais tempo e um esforço maior para seu entendimento e manutenção [Fowler, 1999].
- *Large Class* refere-se a uma classe que está tentando fazer demais. Classes com essa característica possuem muitas variáveis de instância ou métodos [Fowler, 1999].
- *Class Data Should be Private* é uma classe expondo seus atributos, violando o princípio da ocultação de informações [Tufano et al., 2015].
- *Complex Class* é uma classe que tem uma alta complexidade ciclomática [Brown et al., 1998].

- *Functional Decomposition* refere-se a uma classe onde a herança e polimorfismo são mal utilizados, declarando muitos campos privados e implementado poucos métodos [Brown et al., 1998].

- *Spaghetti Code* é uma classe sem estrutura que declara longos métodos sem parâmetros [Brown et al., 1998].

2.3 Valores Referência

Para um efetivo uso das métricas na Engenharia de Software é necessário que se tenha definido valores referência [Ferreira, 2011]. A definição desses valores permite identificar pontos de referência em uma métrica para que ela seja interpretada corretamente [Marinescu, 2005].

Marinescu [2005] definiu valores referência para métricas no paradigma orientado por objetos para a identificação de *bad smells*. Nesse campo de pesquisa, Ferreira [2011] identifica valores referência para 6 métricas de software orientado por objetos: *Lack of Cohesion in Methods (LCOM), Depth of Inheritance Tree (DIT), Coupling Factor (COF)*, Número de Conexões Aferentes, Número de Métodos Públicos e Número de Atributos Públicos. Filó [2014] apresenta um catálogo de valores de referência para 18 métricas que podem ser utilizadas nas estratégias de detecção de *bad smells*, dentre elas: *Number of Children (NOC) e Cyclomatic complexity (VG)*. Oliveira et al. [2014] desenvolveu um trabalho que descreve um método para derivação de valores referência relativos para 7 métricas de classe, dentre elas: *Number of methods (NOM), Number of Lines of Code (LOC), Fan-out, Response For a Class (RFC)* e *Weighted Method Count (WMC)*.

2.4 Trabalhos Relacionados

Nessa seção são descritos os trabalhos que inspiraram este projeto de pesquisa.

Stephane et al. [2009] realizaram uma análise exploratória do ciclo de vida do *bad smell God Class* com o objetivo de identificar como eles são introduzidos e removidos do código. Para isso, o estudo utiliza uma abordagem Bayesiana [Khomh et al., 2009] para identificar a presença dos *bad smells* nas versões dos dois softwares analisados: Xerces [2015] e Eclipse [2015]. Nos resultados apresentados foi possível identificar que o crescimento dos dois sistemas é relativamente linear e, ao comparar o crescimento do *bad smell* no software Eclipse, observou-se que o número de *God Class* não cresce

linearmente com o software e permanece estável. Para o software Xerces, há um significativo aumento da quantidade de *God Class* à medida que o software cresce. Os resultados também indicam que 19% dos *bad smells* identificados no Eclipse foram removidos durante a evolução do software, e 30% dos *bad smells* foram removidos do sistema Xerces.

Olbrich et al. [2009] realizaram um estudo de caso em dois sistemas Java orientado por objetos, Lucene [2015] e Xerces [2015]. Nesse estudo foram analisados a evolução e o impacto dos *bad smells God Class* e *Shotgun Surgery*. A análise foi feita com base em cada intervalo de 50 versões desses softwares e foram utilizadas as estratégias de detecção e valores referência definidos por Marinescu [2005] para a identificação dos *bad smells*. Os resultados do estudo indicaram que para o *bad smell God Class* existe uma grande correlação entre a taxa de crescimento do sistema e o aumento no número dos *bad smells*. Para o *bad smells Shotgun Surgery* o resultado foi diferente, a taxa de crescimento do *bad smell* foi maior do que a taxa de crescimento do software. Outro resultado interessante identificado no trabalho diz respeito àquelas classes que possuem o *bad smell God Class*. Essas classes são alteradas com maior frequência do que aquelas classes que não possuem esse *bad smell*. Isso acontece porque como estas classes são as mais importantes no sistema, naturalmente sofrem manutenções constantemente.

Uma das limitações do trabalho desenvolvido por Olbrich et al. [2009] é que no estudo foram considerados apenas dois softwares e dois *bad smells*. Observa-se, pelos resultados apresentados, que eles são diferentes para cada um dos dois softwares, o que impossibilita chegar a uma conclusão sobre os resultados obtidos.

Chatzigeorgiou & Manakos [2010] realizaram um trabalho para investigar a evolução dos *bad smells* em códigos orientado por objetos e analisaram as versões de 2 softwares para 4 tipos de *bad smells*: *Long Method, Feature Envy, State Checking* e *God Class*. O trabalho busca responder questões relacionadas com o aumento dos *bad smells* ao longo do tempo, a remoção deles, em que momento eles foram inseridos e quanto tempo eles sobrevivem no código. Para identificação dos *bad smells* foi utilizada a ferramenta de detecção automática JDeodorant [2015]. Os resultados do trabalho mostram que 55% dos *bad smells* no software JFreeChar foram introduzidos quando os arquivos foram criados, enquanto no software JFlex apenas 14,44% foram introduzidos no início. Os resultados também mostram que existem *bad smells* que permanecem no código até a versão final do software.

Um dos principais pontos de reflexão em relação ao trabalho desenvolvido por Chatzigeorgiou & Manakos [2010] está relacionado com a utilização da ferramenta JDeodorant [2015] para a identificação dos *bad smells*, pois essa ferramenta e a maioria das outras ferramentas discordam em seus resultados na identificação dos *bad smells*

[Fontana et al., 2011]. Outro fator que merece atenção está relacionado com o baixo número de softwares usado no estudo realizado, considerando principalmente que os resultados apresentados foram muito distantes um do outro.

Arcoverde et al. [2011] realizaram uma pesquisa com 33 desenvolvedores de software com o objetivo de compreender a longevidade dos *bad smells* no código. O estudo foi baseado na hipótese de que os desenvolvedores não se preocupam com os *bad smells* no código e que muitos desses *bad smells* parecem ser ignorados pelos desenvolvedores em atividades de refatoração e não são removidos do código. Os resultados preliminares da pesquisa mostram que alguns desenvolvedores adiam refatorações no código com receio que elas possam parar o código do cliente, principalmente quando são realizadas mudanças de escopo de forma global no software. Outro ponto importante é que os desenvolvedores encontram dificuldades de visualizar as mudanças utilizando as ferramentas de refatoração disponíveis atualmente.

Embora Arcoverde et al. [2011] tenham buscado compreender a longevidade dos *bad smells*, seus os resultados focaram somente nas possíveis razões de permanência dos *bad smells* no código, e não identificaram quando os *bad smells* são inseridos no código.

O trabalho desenvolvido por Fontana et al. [2011] analisa o panorama atual das ferramentas para detecção automática de *bad smells* e visa expor as regiões do código mais afetadas pela degradação estrutural do software, bem como a relevância dos resultados dessas ferramentas para a evolução do software. Nesse processo, os autores utilizaram 4 ferramentas de detecção automática de *bad smells*: JDeodorant [2015], Infusion [2015], PMD [2015] e Checkstyle [2015] e 6 versões do software orientado por objetos GanttProject [2015] para analisar os 6 tipos de *bad smells: Duplicated Code, Feature Envy, God Class, Large Class, Long Parameter List e Long Method*. Os resultados mostram que a maioria dos *bad smells* foram removidos do código até a última versão do software analisado e que o restante permanece no código. Como parte dos resultados, os autores também analisaram se a presença dos *bad smells* estão relacionados com alguma característica do código que foi inserido ao longo da evolução do software ou até mesmo com o processo de desenvolvimento do software. Os resultados indicam que na maioria dos casos quando há atividades para a inclusão de novas funcionalidades, há também a inclusão de alguns *bad smells*.

Fontana et al. [2011] analisaram se características do código ou do processo de desenvolvimento de software estão relacionados com o surgimento dos *bad smells*. Entretanto, os resultados obtidos não foram suficientes para estabelecer a correlação entre esses fatores. Um dos fatores que pode ter contribuído para isso é que o estudo não analisou o software desde sua primeira versão de criação, tendo considerado somente

6 versões quando o software já estava consolidado, o que não lhes permitiu identificar se os *bad smells* haviam sido inseridos quando o arquivo foi criado. Outro ponto de atenção é que o trabalho foi realizado com base nos resultados das ferramentas de detecção automática de *bad smells*. Contudo essas ferramentas interpretam os *bad smells* de formas diferentes e, assim, podem gerar resultados diferentes.

Tufano et al. [2015] realizaram um trabalho onde foram analisados 200 projetos de código aberto com o objetivo de identificar quando e porque um código começa a ter *bad smells*. O estudo se concentrou em 5 *bad smells*: *Blob Class, Class Data Should be Private, Complex Class, Functional Decomposition* e *Spaghetti Code*. Para a identificação desses *bad smells*, foram utilizadas as estratégias da DECOR [Moha et al., 2010]. Em linhas gerais essa estratégia descreve um método com todos os passos necessários para a especificação e detecção de *bad smells*. Os resultados do estudo mostram que a maioria dos *bad smells* são introduzidos no código no momento em que os arquivos são criados, contrariando o senso comum que considera que os *bad smells* geralmente são inseridos durante as atividades de manutenção, evolução e próximos do *deadline* final de entrega do produto de software. Outro resultado que contraria o senso comum diz respeito a constatação de que a maioria dos *bad smells* são inseridos no código pelos próprios "donos"das classes ou aqueles que realizam um fluxo grande de trabalho nelas e não pelos desenvolvedores novatos.

O trabalho desenvolvido por Tufano et al. [2015], levanta uma controvérsia sobre o momento do aparecimento do *bad smell* no código em relação a outros trabalhos publicados, principalmente porque contraria as leis de Lehman [Lehman & Ramil, 2001], ao postular que a maioria dos *bad smells* são inseridos no código no momento em que os arquivos são criados e que a maioria deles são incluídos pelos profissionais mais experientes.

Lehman & Ramil [2001] define um conjunto de oito leis que "governam"a evolução de sistemas. A sétima lei diz que o declínio da qualidade do software ocorre ao longo do tempo com a inclusão de novas funcionalidades e principalmente com o aumento da complexidade do software. Um ponto de reflexão sobre o trabalho de Tufano et al. [2015] é que ele confia nos resultados da DECOR [Moha et al., 2010] para fazer a identificação dos *bad smells*, sem considerar que existem definidas várias estratégias de detecção de *bad smells* e elas produzem resultados diferentes [Fontana et al., 2011]. Além disso, o trabalho avalia uma grande quantidade de projetos sem levar em consideração o contexto de cada projeto, como por exemplo, o processo de desenvolvimento do software, o que também pode influenciar nos resultados finais.

2.5 Considerações

Há diferentes abordagens para estudar a evolução dos *bad smells* em sistemas de software. Alguns estudos utiliza ferramentas automatizadas para detecção dos *bad smells*, outros usam estratégias de detecção disponíveis na literatura, mas nenhum deles realiza um estudo empírico utilizando uma abordagem com foco na identificação de *bad smells* utilizando valores referência para métricas de software orientados por objetos como mecanismo de filtragem para a identificação.

O problema investigado neste trabalho de dissertação refere-se a identificar como *bad smells* evoluem em software orientado por objetos. Essa é uma questão em discussão na literatura, para a qual ainda não há uma resposta consensual que satisfaça a comunidade científica da área [Lehman & Ramil, 2001; Fontana et al., 2011; Tufano et al., 2015; Chatzigeorgiou & Manakos, 2010; Olbrich et al., 2009], principalmente porque os resultados identificados até o momento são conflitantes uns com os outros. Diante desses fatos, esse trabalho de pesquisa tem por objetivo contribuir com a solução para esse problema realizando um estudo empírico aplicando valores referência para métricas de software a fim de analisar a natureza evolutiva dos *bad smells* em softwares orientados por objetos, investigando quando estes *bad smells* são introduzidos pelos desenvolvedores.

Capítulo 3

Solução Proposta

A realização de mudanças significativas e a implementação de novos requisitos são ações necessárias para um software continuar útil. Realizar modificações em um software pode se tornar uma tarefa complexa ao longo do tempo. Portanto, é necessário compreender como um software evolui para a definição de ações para a melhoria da qualidade estrutural de software.

A solução proposta para este trabalho de dissertação é a realização de um estudo empírico com o objetivo de investigar a natureza evolutiva dos *bad smells* em softwares orientados por objetos utilizando valores referência para métricas para a identificação dos *bad smells* nas versões dos softwares analisados. A Seção 3.1 apresenta a metodologia que será empregada para a execução das atividades descritas na Seção 1.1, necessárias para o desenvolvimento da solução proposta.

Para cumprir esse objetivo, inicialmente serão escolhidos os softwares para serem analisados no estudo empírico. Como segunda etapa, serão selecionados os *bad smells*, métricas, valores referência para as métricas e as estratégias de detecção dos *bad smells*. Na terceira etapa será proposto um modelo e estratégia de realização do estudo empírico. Depois dessas etapas feitas, serão realizados o estudo empírico e a análise dos resultados.

3.1 Metodologia

1. A revisão bibliográfica será realizada com base em livros e artigos publicados em veículos reconhecidos. A revisão deverá se concentrar em trabalhos relacionados aos seguintes assuntos:

    a) evolução de software;

b) *bad smells* (conceitos, tipos, evolução);

c) estratégias de detecção de *bad smells*;

d) métricas de software;

e) valores de referência para métricas de software orientado por objetos.

2. Para a definição de quais softwares serão analisados, deverá ser realizada uma pesquisa dos repositórios de software disponíveis. Os softwares selecionados deverão obedecer no mínimo os critérios abaixo:

   a) ser desenvolvido seguindo os conceitos de orientação por objetos;

   b) ser desenvolvido em linguagem Java;

   c) possuir disponíveis para *download* as versões desenvolvidas do software ao longo do tempo de acordo com o tempo definido na metodologia de realização do estudo;

   d) possuir o registro do histórico da evolução do software e registros de mudanças.

3. Para a definição de quais *bad smells* serão analisados durante sua evolução no software, será necessário escolhê-los em conjunto com as métricas, valores referências para as métricas e estratégias de detecção. A intenção é que sejam utilizados os *bad smells* mais citados e mais utilizados nos trabalhos disponíveis. Após esta definição deverão ser analisadas quais estratégicas de detecção deverão ser utilizadas para a identificação dos *bad smells* nos softwares analisados. As estratégias de detecção são compostas por métricas, e para a escolha e definição dessas métricas também deverá ser levada em consideração se as métricas possuem valores de referência definidos na literatura. As estratégias de detecção serão definidas com base nas estratégias disponíveis na literatura, e para os casos em que não existe a estratégia para o *bad smell*, ela será definida com base: (1) no conhecimento de orientação por objetos, (2) no objetivo da métrica e (3) do *bad smell*.

4. Ao definir o modelo e metodologia para a realização do estudo empírico, deverão ser levados em consideração os recursos, tempo e ferramentas disponíveis para a realização do estudo. Nessa etapa deverão ser definidos:

   a) a forma como os *bad smells* serão identificados no código;

   b) qual ferramenta será utilizada para a identificação das métricas com os valores de referência definidos em conjunto com a estratégia de detecção;

c) quais ferramentas serão utilizadas para importar e analisar as versões dos códigos dos softwares;

d) como será realizado a análise das versões dos softwares, se será via análise da documentação das alterações das versões ou por meio registros de *bugtrackers*;

e) após a identificação dos valores referência, identificar qual faixa desses valores deverá ser levada em consideração para a identificação do *bad smell* de acordo com a estratégia definida.

5. A execução do estudo empírico deverá seguir o modelo definido e planejado na etapa de elaboração da metodologia do estudo.

6. Para realizar a análise dos resultados, deverá ser definido qual será a forma de consolidação dos dados coletados sobre a evolução dos *bad smells* nas versões dos softwares analisados. Deverão ser definidos quais tipos de análise deverão ser realizadas individualmente e quais deverão ser analisadas de forma agrupada.

7. Após a análise dos dados, deverá ser elaborada uma discussão sobre os resultados obtidos a fim de produzir algumas diretrizes sobre a natureza evolutiva dos *bad smells* no código.

8. No fim do trabalho de dissertação deverão ser discutidos os principais resultados e contribuições alcançadas com o estudo e também listar os principais pontos que podem ser desenvolvidos futuramente para complementar a solução proposta neste trabalho.

3.2 Cronograma

Esta seção apresenta o cronograma para a execução da dissertação proposta nesse documento.

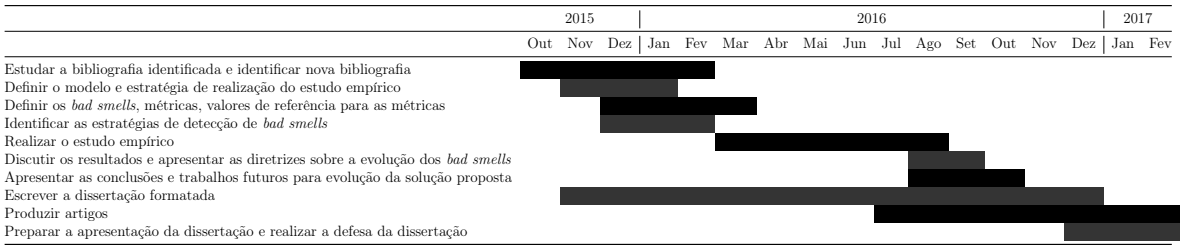

Tabela 3.1: Cronograma de atividades

3.3 Proposta de Sumário da Dissertação

A dissertação proposta será estruturada conforme o sumário a seguir:

Referências Bibliográficas

Arcoverde, R.; Garcia, A. & Figueiredo, E. (2011). Understanding the longevity of code smells: Preliminary results of an explanatory survey. Em *Proceedings of the 4th Workshop on Refactoring Tools*, WRT '11, pp. 33--36, New York, NY, USA. ACM.

Brito e Abreu, F. & Carapuca, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. Em *Proc. Int'l Conf. Software Quality (QSIC)*.

Brown, W. J.; Malveau, R. C.; W., H.; McCormick & Mowbray, T. J. (1998). *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons. ISBN 978-0471197133.

Chatzigeorgiou, A. & Manakos, A. (2010). Investigating the evolution of bad smells in object-oriented code. Em *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pp. 106–115.

Checkstyle (2015). Checkstyle. Disponível em: http://checkstyle.sourceforge.net/. Acesso em Agosto de 2015.

Chidamber, S. R. & Kemerer, C. F. (1994). A metrics suite for object oriented design. Em *IEEE Trans. Softw. Eng*, pp. 476--493. ISSN 0098-5589.

Eclipse (2015). Eclipse. Disponível em: http://eclipse.org. Acesso em Agosto de 2015.

Ferreira, K. A. M. (2011). *Um modelo de predição de amplitude da propagação de modificações contratuais em software orientado por objetos*. Tese de doutorado, DCC/UFMG.

Filó, T. G. S. (2014). Identificação de valores referência para métricas de softwares orientados por objetos. Dissertação de mestrado, DCC/UFMG.

Fontana, F. A.; Braione, P. & Zanoni, M. (2011). Automatic detection of bad smells in code: An experimental assessment. *Journal of Object Technology*.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA. ISBN 0-201-48567-2.

GanttProject (2015). Ganttproject. Disponível em:http://www.ganttproject.biz. Acesso em Agosto de 2015.

Infusion (2015). Infusion. Disponível em: http://www.ptidej.net/download. Acesso em Agosto de 2015.

JDeodorant (2015). Jdeodorant. Disponível em: http://www.jdeodorant.com/. Acesso em Agosto de 2015.

Khomh, F.; Vaucher, S.; Gueheneuc, Y.-G. & Sahraoui, H. (2009). A bayesian approach for the detection of code and design smells. Em *Quality Software, 2009. QSIC '09. 9th International Conference on*, pp. 305–314. ISSN 1550-6002.

Kruchten, P.; Nord, R. & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Software, IEEE*, 29(6):18–21. ISSN 0740-7459.

Lanza, M.; Ducasse, S. & Marinescu, R. (2007). *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer. ISBN 9783540395386.

Lehman, M. M. & Ramil, J. F. (2001). Rules and tools for software evolution planning and management. *Ann. Softw. Eng.*, 11(1):15--44. ISSN 1022-7091.

Lucene, A. (2015). Apache lucene. Disponível em: http://lucene.apache.org. Acesso em Agosto de 2015.

Marinescu, R. (2005). Measurement and quality in object-oriented design. Em *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pp. 701–704. ISSN 1063-6773.

Moha, N.; Gueheneuc, Y.; Duchien, L. & Meura (2010). Decor: A method for the specification and detection of code and design smells. *Software Engineering, IEEE Transactions on*, 36(1):20–36. ISSN 0098-5589.

Munro, M. (2005). Product metrics for automatic identification of "bad smell"design problems in java source-code. Em *Software Metrics, 2005. 11th IEEE International Symposium*, pp. 15–15. ISSN 1530-1435.

Olbrich, S.; Cruzes, D. S.; Basili, V. & Zazworka, N. (2009). The evolution and impact of code smells: A case study of two open source systems. Em *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, pp. 390--400, Washington, DC, USA. IEEE Computer Society.

Oliveira, P.; Valente, M. T. & Lima, F. P. (2014). Extracting relative thresholds for source code metrics. Em *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, pp. 254–263.

Palomba, F.; Bavota, G.; Di Penta, M.; Oliveto, R. & De Lucia, A. (2014). Do they really smell bad? a study on developers' perception of bad code smells. Em *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pp. 101–110. ISSN 1063-6773.

Parnas, D. L. (1994). Software aging. Em *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pp. 279--287, Los Alamitos, CA, USA. IEEE Computer Society Press.

PMD (2015). Pmd. Disponível em: http://pmd.sourceforge.net/. Acesso em Agosto de 2015.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley, Harlow, England, 9 edição. ISBN 978-0-13-703515-1.

Stephane, V. F. K.; Moha, N. & G, Y. (2009). Tracking design smells: Lessons from a study of god classes. *16th Working Conference on Reverse Engineering, IEEE Computer Society Press, Lille, France*.

Syeed, M.; Hammouda, I. & Syatä, T. (2013). Evolution of open source software projects: A systematic literature review. *Journal of Software*, 8(11).

Tufano, M.; Palomba, F.; Bavota, G.; Oliveto, R.; Penta, M. D.; Lucia, A. D. & Poshyvanyk, D. (2015). When and why your code starts to smell bad. *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015), Florence, Italy*.

Xerces, A. . J. (2015). Apache 2 j xerces. Disponível em: http://xerces.apache.org. Acesso em Agosto de 2015.

Yamashita, A. & Moonen, L. (2013). Do developers care about code smells? an exploratory survey. Em *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pp. 242–251.