UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DO CURSO DE GRADUAÇÃO EM TECNOLOGIA EM SISTEMAS PARA INTERNET

ROGERIO CRISTIANO MIDDING

AVALIAÇÃO DA QUALIDADE INTERNA DE SOFTWARES DESENVOLVIDOS COM A TÉCNICA TDD

TRABALHO DE CONCLUSÃO DE CURSO

TOLEDO

2016

ROGERIO CRISTIANO MIDDING

AVALIAÇÃO DA QUALIDADE INTERNA DE SOFTWARES DESENVOLVIDOS COM A TÉCNICA TDD

Trabalho de conclusão de curso de graduação, apresentada ao Coordenação do Curso de Graduação em Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Wesley Klewerton Guez Assunção

TOLEDO

2016

RESUMO

MIDDING, Rogerio. AVALIAÇÃO DA QUALIDADE INTERNA DE SOFTWARES DESENVOLVIDOS COM A TÉCNICA TDD. 35 f. Trabalho de conclusão de curso – Coordenação do Curso de Graduação em Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Toledo, 2016.

O desenvolvimento dirigido a testes (*Test Driven Development - TDD*) é uma técnica de desenvolvimento de softwares que propõe a implementação do teste antes da codificação da funcionalidade em si. Na literatura observa-se que o TDD proporciona um aumento da qualidade externa, contudo existe uma carência de evidências sobre o impacto desta técnica na qualidade interna do software. Baseado nisso, o objetivo deste trabalho é avaliar a influência da técnica TDD em relação a qualidade interna do software. Para esta avaliação é proposto o uso de métricas de projeto de software que verificam atributos internos do produto, tais como coesão, acoplamento e complexidade. Essas métricas são utilizadas para medir produtos de software desenvolvidos com técnica TDD e produtos de softwares desenvolvidos da forma tradicional. Com os resultados das medições foi realizada uma avaliação empírica dos valores obtidos para atributos de qualidade interna em diferentes estudo de caso. Os resultados mostram que, de forma geral, a técnica TDD influencia positivamente na qualidade interna do software, entretanto houve casos em que os resultados obtidos dos softwares desenvolvido da forma tradicional foram melhores que os dos softwares desenvolvidos com TDD.

Palavras-chave: Desenvolvimento Dirigido a Testes, Medição de Software, Métricas de Projeto de Software, Qualidade Interna de Software

ABSTRACT

MIDDING, Rogerio. INTERNAL QUALITY ASSESSMENT IN SOFTWARES DEVELOPED BY USING TDD TECHNIQUE. 35 f. Trabalho de conclusão de curso – Coordenação do Curso de Graduação em Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Toledo, 2016.

Test Driven Development (TDD) is a software development technique that propose the implementation of the testing before functionality codification. In the literature we can observe that TDD leads to an improvement on product external quality, however, there is a lack of findings regarding its impact on product internal quality. Based on this, the goal of this study is to evaluate the influence of TDD on product internal quality. For this evaluation we propose the use of design metrics that verify product internal aspects, such as cohesion, coupling, and complexity. These metrics are used to measure and analyze products developed by using TDD and software products developed in a traditional way. With the results of the measurements we performed an empirical evaluation for the quality attributes values in different case studies. The results show that, in general, the TDD influences positively in the internal quality of softwares, however there were cases that the obtained results of softwares developed with in a traditional way were better than the ones developed with TDD.

Keywords: Test Driven Development, Software Measurement, Software Design Metrics, Internal Software Quality

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A baixa qualidade de produtos de software é um problema recorrente em empresas que trabalham com desenvolvimento de software. Este problema é responsável por elevar o custo necessário para manutenção e pela insatisfação dos usuários. Muitos erros são descobertos apenas em ambiente de produção e os defeitos vêm à tona, o que gera grande volume de manutenções no software. Essas manutenções podem causar outros erros que resultam em clientes insatisfeitos e custo de manutenção elevado (BECK, 2002).

O TDD (Desenvolvimento Dirigido a Testes, do inglês *Test Driven Development*), é uma técnica de desenvolvimento de software que coloca o teste como etapa inicial do processo de implementação. Os testes são desenvolvidos antes do código-fonte da funcionalidade, em um ciclo de etapas em que o resultado de cada etapa incrementa o produto final. Essa metodologia tem mostrado resultados promissores em relação a qualidade do produto, principalmente qualidade externa, que se refere às funcionalidades (MUNIR et al., 2014). Em relação a qualidade interna, observa-se principalmente uma alta cobertura do código-fonte pelos casos de teste (BISSI et al., 2016). Esta técnica está amplamente difundida e a sua utilização cresce devido aos resultados positivos obtidos em relação a qualidade do produto final. O TDD contribui para melhorar a qualidade nos processos e reduzir as manutenções necessárias.

Apesar dos benefícios apresentados pela técnica TDD, um estudo recente expõe a necessidade de analisar o impacto desta técnica em relação a outros aspectos da qualidade interna do software (BISSI et al., 2016). Estes mesmos autores mencionam a possibilidade da utilização de métricas que avaliam acoplamento, coesão e complexidade ciclomática e apontam que na literatura não encontram-se estudos que apresentam uma análise profunda em relação a técnica TDD e a qualidade interna do software.

O objetivo desse trabalho é realizar uma análise da qualidade interna de softwares produzidos com a técnica TDD utilizando-se métricas de projeto comumente aplicadas na Engenharia de Software, tais como, complexidade do código-fonte, acoplamento e coesão. Para obter conclusões a respeito do impacto da técnica TDD na qualidade interna, é proposta uma

avaliação empírica da qualidade interna de softwares desenvolvidos com a técnica TDD comparando com resultados das mesmas métricas aplicadas em softwares desenvolvidos sem utilização da técnica TDD. Para poder mensurar a qualidade interna, serão calculados atributos de qualidade internos (características) do softwares, tais como, reusabilidade, flexibilidade e efetividade utilizando os valores extraídos do código-fonte por meio das métricas internas de software.

As contribuições deste trabalho são:

- Seleção e utilização de um conjunto de métricas de código-fonte para avaliar características de qualidade internas (atributos de qualidade) de softwares desenvolvidos com TDD;
- Comparação de valores de atributos de qualidade de software desenvolvidos com TDD e softwares desenvolvidos de forma tradicional;

Este trabalho está organizado como segue. O Capítulo 2 apresenta conceitos de qualidade software indispensáveis para o desenvolvimento do trabalho. A técnica de desenvolvimento dirigido a testes é apresentada no capítulo 3. O desenvolvimento da avaliação empírica é descrita no Capítulo 4. No capítulo 5 estão os resultados e as análises feitas. O Capitulo 6 encerra o trabalho com as conclusões.

2 QUALIDADE DE SOFTWARE

Uma ideia comumente ligada à qualidade de software é que esta está relacionada apenas ao quesito de satisfação dos requisitos levantados, todavia existem outros fatores igualmente importantes. Para Sommervile (2011), a qualidade de software é um conceito subjetivo, que não implica apenas o quesito de correta implementação das funcionalidades. A qualidade de software pode ser medida por meio de um grupo de características específicas do produto, conhecidas como atributos de software (MALDONADO et al., 2001). Segundo O'Regan (2012), qualidade de software está relacionada ao nível de aptidão para o uso, contudo podemos definir qualidade de software mais profundamente.

Neste capítulo são apresentadas algumas definições de qualidade de software e conceitos de atributos de qualidade e métricas de software que são essenciais. Além disso, discutiremos o modelo de qualidade adotado no decorrer do trabalho.

2.1 ATRIBUTOS DE QUALIDADE DE SOFTWARE

Atributos de qualidade são características utilizadas para definir qualidade de um software. Segundo Chandrasekar et al. (2014), um atributo de qualidade é uma propriedade específica do software. Os atributos podem ser classificados como atributos de qualidade internos (ponto de vista dos desenvolvedores) e externos (ponto de vista dos usuários). É impossível um software conseguir adequar-se de forma a obter bons resultados em todos os atributos, pois ao otimizar um atributo outro será prejudicado. Por este motivo, cada software, conforme a sua aplicabilidade, deve priorizar métricas essenciais para seu uso (SOMMERVILE, 2011).

Como a qualidade de software é um conceito subjetivo, para definição do mesmo faz-se necessário eleger alguns atributos de qualidade do software para serem avaliados. Para este trabalho, os atributos de qualidade utilizados para avaliar a qualidade são atributos internos, uma vez que o objetivo é evidenciar os efeitos do uso da técnica TDD na qualidade interna dos softwares. A Tabela 1 apresenta os quinze atributos que definem a qualidade de software, segundo Sommervile (2011):

Tabela 1: Atributos de Qualidade

Segurança	Compreensibilidade	Portabilidade
Proteção	Testabilidade	Usabilidade
Confiabilidade	Adaptabilidade	Reusabilidade
Resiliência	Modularidade	Eficiência
Robustez	Complexidade	Capacidade de aprendizado

Fonte: (SOMMERVILE, 2011 apud BOEHM, 1978)

A ISO 9126, padrão de avaliação de qualidade para tecnologia de informação, apresenta uma forma de medir qualidade de software. Essa avaliação leva em consideração as seguintes características (atributos de qualidade) do produto de software (O'REGAN, 2012):

- *Funcionalidade*: indica quão disponível é cada uma das funções requeridas;
- *Confiabilidade*: indica o quão confiável é o software;
- *Usabilidade*: indica o quão fácil o software é julgado a ser utilizado;
- *Eficiência*: indica a eficiência do software;
- *Facilidade de manutenção*: indica o nível de facilidade de se modificar e manter o produto;
- *Portabilidade*: indica a facilidade de transferir e utilizar o software em ambiente diferente.

Bansiya e Davis (2002) introduziram um modelo de qualidade para softwares desenvolvidos no paradigma orientado a objeto (QMOOD). Este modelo baseia-se em métricas de código-fonte para obter uma visão da qualidade interna do produto de software, portanto é uma abordagem que utiliza métricas internas. O modelo QMOOD será apresentado em detalhes na Seção 2.3. Esse viés na qualidade interna é justificada pelo fato de que um controle dos atributos de qualidade internos resultará em melhores resultados nos atributos de qualidade externos (KITCHENHAM; PFLEEGER, 1996).

2.2 MÉTRICAS DE SOFTWARE

Métricas, de forma geral, são maneiras de medir a características de um produto. Métricas de software são padrões quantitativos de medidas de vários aspectos do projeto de software (VICENTE, 2010). A utilização dessas métricas em um projeto de software pode apoiar estimativas, permite avaliar a qualidade do produto, a produtividade da equipe e o controle do projeto.

O software é um produto complexo e amplo, por isso é necessário utilizar diferentes métricas para medir um software e mostrar se o mesmo tem qualidade ou não. Inúmeras métricas foram propostas para medir diferentes características de um software. Uma métrica pode avaliar uma determinada característica, todavia, seu resultado não tem autonomia para definir a qualidade do projeto como um todo, pois o resultado obtido por essa métrica não irá determinar os resultados que outras métricas obterão, já que uma métrica não abrange todos os âmbitos a serem avaliados. Em muitos casos, o bom resultado de uma métrica terá efeito contrário em outro atributo de qualidade.

Segundo Morais et al. (2009), assim como os atributos, as métricas de software podem ser classificadas como internas e externas. As métricas externas são medidas através de experiências feitas com o software em execução para extrair dados, que podem variar de acordo com o contexto, e são utilizadas para medir atributos de qualidade referente às funcionalidades do software, como eficiência e robustez. As métricas internas são utilizadas para medir qualidade interna de um software através de dados estatísticos geralmente provenientes de análises do código-fonte e o contexto não influenciará nos resultados. Por exemplo, alguns atributos de qualidade internos que podem ser medidos são coesão, acoplamento e complexidade do código-fonte.

O mesmo autor (MORAIS et al., 2009) também faz uma subclassificação das métricas internas em:

- *Métricas de tamanho e complexidade*: são métricas referente a quantidade de linhas de código-fonte e referentes a complexidade, que é medida através da quantidade de interações realizadas. As métricas de código são muito utilizadas para estimativa de custo e as métricas de complexidade estão diretamente relacionadas a manutenabilidade;
- *Métricas de orientação a objetos*: são métricas voltadas para softwares produzidos no paradigma orientado a objetos, em que são usadas entidades e não algoritmos. Desta forma, é possível medir a qualidade interna relacionada a aplicação de conceitos de orientação a objetos como herança, polimorfismo (acoplamento) e coesão entre classes.

2.3 MODELO DE QUALIDADE UTILIZADO

Os atributos de qualidade interna serão avaliados no trabalho pois o objetivo é analisar qualidade interna do software. Este tipo de atributo está relacionado a requisitos que determinam a qualidade do processo de desenvolvimento, característica importante do desenvolvimento

de software (CHANDRASEKAR et al., 2014). Por isso foi adotado o modelo QMOOD proposto por Bansiya e Davis (2002), devido ao fato de se adequar mais ao propósito do trabalho, assim como foi realizado no trabalho relacionado Jetter (2007).

O QMOOD propõe quatro níveis com três relacionamentos entre eles, conforme a Figura 1 ilustra. No primeiro nível estão os atributos de qualidade utilizados, ou seja, as características relevantes para definir qualidade do software. O segundo nível representa as propriedades de design, os valores das propriedades são utilizados para calcular os atributos de qualidade (relação R12). O terceiro nível são as métricas de qualidade selecionados, as mesmas são descritas a seguir. A relação R23 é o uso das métricas para cálculo das propriedades de design. Finalmente, o quarto nível são os componentes de design, variam de acordo com a tecnologia utilizada, no caso da linguagem Java (tecnologia utilizada nos estudos de caso selecionados) são elementos como as classes, objetos e pacotes. A relação R34 simboliza que as métricas extraem os dados dos componentes de design orientados a objetos, tais como classes, objetos e métodos.

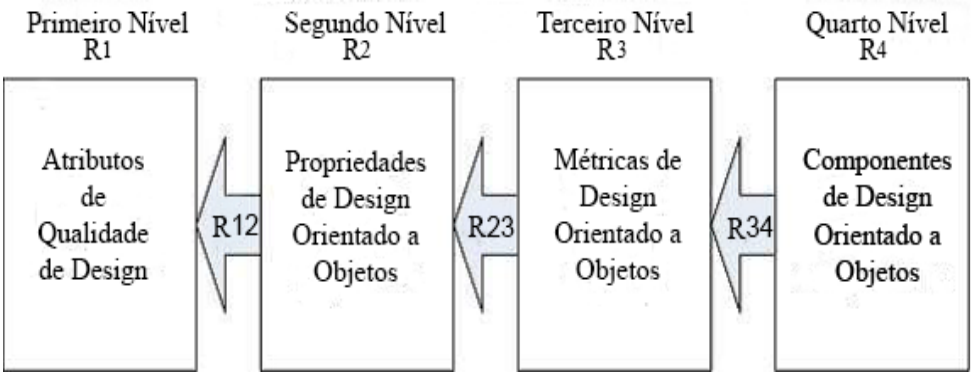

Figura 1: A figura ilustra os níveis do modelo QMOOD e as relações entre os mesmos
Fonte: (BANSIYA; DAVIS, 2002)

Os atributos de qualidade interna do primeiro nível são apresentados na Tabela 2, juntamente com uma descrição. Esses atributos foram propostos no modelo original por Bansiya e Davis (2002).

Tabela 2: Atributos de Qualidade Utilizados

Atributo de Qualidade	Definição
Reusabilidade	Representa as características do design orientado a objetos que permite ser reaplicado para outro fim sem esforço significativo.
Flexibilidade	Características que permite realização de mudanças no design. Capacidade de adaptar o design para possibilitar adição de recursos funcionais relacionados.
Compreensividade	Propriedade do design que o possibilita a ser facilmente aprendido e compreendido. Atributo bastante relacionado com a complexidade da estrutura do design.
Funcionalidade	Responsabilidades atribuídas pelas classes do design que ficam disponíveis por meio de interfaces públicas.
Extensibilidade	Refere-se a presença e uso de propriedades do design que permitem a inclusão de novos requisitos.
Efetividade	Refere-se a capacidade do design de alcançar a funcionalidade e o comportamento desejados por meio do uso dos conceitos de design orientado a objetos.

Fonte: (BANSIYA; DAVIS, 2002)

No segundo nível, as propriedades de design orientado a objetos estão na Tabela 3, a mesma especifica cada um deles e estabelece a relação com as métricas utilizadas neste trabalho para medi-las (relação R23).

Tabela 3: Propriedades de design e métricas utilizadas

Propriedade de Design	Definição	Medido com
Tamanho do design	Número de classes utilizadas no design.	Número de classes
Hierarquias	Usado para representar diferentes conceitos de generalização/especialização no design. É a quantidade de classes não herdadas que possuem subclasses no design.	Profundidade da árvore de herança
Abstração	Medida do aspecto generalização/especialização no design, classes que possuem um ou mais descendentes representam esta propriedade.	Abstração
Encapsulamento	Mede o controle de acesso aos dados e o comportamento com apenas um construtor. Em design orientado a objetos é basicamente definir os atributos como privados para evitar acesso direto de outros objetos.	1
Acoplamento	Define a interdependência entre objetos no design. Mede a quantidade de objetos que teriam que ser acessados por um objeto para que o mesmo realize a funcionalidade corretamente.	Instabilidade
Coesão	Avalia a relação de métodos e atributos em uma classe. Sobreposição forte dos parâmetros de métodos com os tipos de atributos são uma indicação de forte coesão.	1 / Falta de coesão dos métodos
Composição	Mede as relações "parte de", "possui", "consiste em" e "parte-todo" que são relações de agregação em design orientado a objetos.	Número de atributos
Herança	Mede o relacionamento "é um" entre as classes. Está relacionada com o nível de aninhamento entre as classes em uma hierarquia de herança.	1 – (Número de métodos sobrescritos / número de métodos)
Polimorfismo	Habilidade de substituir um objeto por outro de mesma interface em tempo de execução. É uma medida de serviços dinamicamente determinados em tempo de execução.	Número de métodos sobrescritos
Comunicação	Quantidade de métodos públicos disponíveis como serviços para outras classes. Mede os serviços que uma classe disponibiliza.	Número de métodos
Complexidade	Mede o grau de dificuldade em entender e compreender a estrutura interna e externa das classes e seus relacionamentos.	Peso dos métodos por classe

Fonte: (BANSIYA; DAVIS, 2002)

Apesar de existir vários tipos de métricas, apenas serão utilizadas métricas internas, dentre elas estão algumas métricas de tamanho e complexidade, mas principalmente as de orientação a objetos são utilizadas para os cálculos dos atributos de qualidade. Na sequência são apresentadas as métricas utilizadas neste trabalho, juntamente com uma descrição. O trabalho relacionado de Jetter (2007) adaptou o modelo proposto por Bansiya e Davis (2002), dessa forma foram selecionadas as seguintes métricas:

- *Número de classes*: representa a quantidade de classes presentes no projeto. A métrica proposta por Bansiya e Davis (2002) em seu modelo foi a mesma, todavia era chamada de "tamanho do design em classes". Torna-se parâmetro de escolha de projetos para realizar comparações de resultados obtidos pelos atributos de qualidade.

- *Profundidade da árvore de herança*: é a média de todas as classes e significa a distância média da classe Object na hierarquia de herança. A métrica proposta por Bansiya e Davis (2002) para medir hierarquia é "número de hierarquias", em seu modelo, assim como profundidade da árvore de herança, afeta positivamente o atributo funcionalidade.

- *Abstração*: a métrica de abstração representa o grau de abstração presente no sistema, é a relação entre o número de classes abstratas e o número de classes. A métrica que calcula abstração no modelo original é "número médio de ancestrais", da mesma forma que a métrica abstração influencia positivamente na extensibilidade e efetividade, todavia diminui a compreensividade do projeto.

- *Encapsulamento*: Não foi possível obter os valores para o encapsulamento, para não influenciar os valores foi fixado o valor neutro de 1 deste modelo para todos os estudos de caso. A métrica utilizada por Bansiya e Davis (2002) em seu modelo é "métrica de acesso aos dados". Essa propriedade afeta os atributos de flexibilidade, compreensividade e efetividade.

- *Instabilidade*: mede o acoplamento entre os objetos, mostra a interdependência das classes. Afeta quase todos os atributos de qualidade, mas um alto acoplamento compromete principalmente a reusabilidade, flexibilidade, compreensividade e extensibilidade. A métrica utilizada por Bansiya e Davis (2002) originalmente é "acoplamento direto das classes", ambas métricas são equivalentes.

- *Falta de coesão dos métodos (LCOM)*: o fator mais importante da coesão é a relação dos métodos e atributos de uma classe, a métrica LCOM considera exatamente essa questão. A métrica proposta por Bansiya e Davis (2002) foi "coesão entre os métodos", a diferença entre as métricas é que a LCOM aponta o valor negativo enquanto a coesão entre

os métodos resulta no valor positivo dessa propriedade. Por este motivo, a propriedade coesão é calculada como (1 / LCOM).

- *Número de atributos*: Bansiya e Davis (2002) usaram a métrica "medida de agregação" para medir a agregação de um projeto. Essa métrica é a quantidade de dados declarados com os tipos definidos pelo usuário. Em projetos orientados a objetos, essa é a definição da quantidade de atributos. Um alto valor de agregação tem um impacto positivo na flexibilidade e efetividade.

- *1 - (número de métodos sobrescritos / número de métodos)*: Para medir a herança, foi utilizada esse cálculo. No modelo original, a métrica utilizada é "medida da abstração funcional", que é definida como a proporção da quantidade de métodos sobrescritos pela quantidade total de métodos acessíveis. O valor da propriedade herança reflete a relação "é um" entre classes, o cálculo utilizado para representar essa propriedade também reflete esse conceito.

- *Número de métodos sobrescritos*: o polimorfismo deve ser expressado como medida do comportamento dinâmico. Bansiya e Davis (2002) utilizaram a métrica "número de métodos polimórficos" que era mais voltado para a superclasse. Todavia, a quantidade de métodos sobrescritos também podem representar polimorfismo pois sobrescrita de métodos permite comportamento dinâmico em tempo de execução. O polimorfismo afeta a flexibilidade, funcionalidade, extensibilidade e efetividade positivamente além de prejudicar a compreensividade.

- *Número de métodos*: para medir a comunicação entre as classes foi utilizada a métrica "tamanho da classe de interface" no modelo original. A métrica utilizada para adaptar tem o mesmo conceito com a diferença de considerar os métodos privados. Alto grau de comunicação produz impacto positivo na reusabilidade e funcionalidade.

- *Peso dos métodos por classe*: é uma métrica que permite apontar a complexidade com mais precisão, devido ao fato de levar-se em consideração *loops*, pontos de decisão no código e operações lógicas para o cálculo dessa métrica. No modelo original é proposto o uso da métrica "número de métodos" para medir a complexidade. Essa propriedade influencia negativamente a compreensibilidade do projeto.

O relacionamento R12 consiste no uso das propriedades de design para o cálculo dos atributos de qualidade, essa relação é apresentada na Tabela 4, que mostra detalhadamente como cada propriedade afeta em cada atributo de qualidade.

Tabela 4: Relação entre as métricas e as propriedades

Atributo de Qualidade	Cálculo Computacional
Reusabilidade	-0.25 * Acoplamento + 0.25 * Coesão + 0.5 * Comunicação + 0.5 * Tamanho do design
Flexibilidade	0.25 * Encapsulamento - 0.25 * Acoplamento + 0.5 * Composição + 0.5 * Polimorfismo
Compreensibilidade	-0.33 $*$ Abstração + 0.33 $*$ Encapsulamento - 0.33 $*$ Acoplamento + 0.33 $*$ Coesão - 0.33 $*$ Polimorfismo - 0.33 $*$ Complexidade - 0.33 $*$ Tamanho do design
Funcionalidade	0.12 $*$ Coesão + 0.22 $*$ Polimorfismo + 0.22 $*$ Comunicação + 0.22 $*$ Hierarquia
Extensibilidade	0.5 $*$ Abstração - 0.5 $*$ Acoplamento + 0.5 $*$ Herança + 0.5 $*$ Polimorfismo
Efetividade	0.2 $*$ Abstração + 0.2 $*$ Encapsulamento + 0.2 $*$ Composição + 0.2 $*$ Herança + 0.2 $*$ Polimorfismo

Fonte: (BANSIYA; DAVIS, 2002)

A Tabela 5 ilustra a correlação entre as propriedades de design e os atributos de qualidade, as setas para cima representam uma influência positiva da propriedade no atributo de qualidade (quanto mais alta melhor), em contrapartida, as setas para baixo representam uma influência negativa (os valores das propriedades devem ser baixos). Importante notar que apesar de uma propriedade influenciar positivamente em algum atributo, pode prejudicar o valor de outro, isso comprova a complexidade de um software e a inviabilidade do mesmo apresentar bons resultados em todos os atributos de qualidade.

Tabela 5: Relação entre as métricas e os atributos de qualidade

	REU	FLE	COM	FUN	EXT	EFE
Tamanho do design	↑		↓	↑		
Hierarquia				↑		
Abstração			↓		↑	↑
Encapsulamento		↑	↑			↑
Acoplamento	↓	↓	↓		↓	
Coesão	↑		↑	↑		
Composição		↑				↑
Herança					↑	↑
Polimorfismo		↑	↓	↑	↑	↑
Comunicação	↑			↑		
Complexidade			↓			

REU: Reusabilidade, FLE: Flexibilidade, COM: Compreensividade, FUN: Funcionalidade, EXT: Extensibilidade, EFE: Efetividade

Fonte: (JETTER, 2007)

3 DESENVOLVIMENTO DIRIGIDO A TESTES

Neste capítulo é apresentada a técnica de desenvolvimento dirigido a testes (TDD), com um breve histórico e definição, além dos benefícios que sua aplicação implicam.

3.1 HISTÓRICO E DEFINIÇÃO

O Desenvolvimento Dirigido a Testes (*Test Driven Development* - TDD) é uma metodologia de desenvolvimento de software originado do método *Extreme Programming*[1] do Manifesto Ágil[2], além de ser baseada em uma técnica de desenvolvimento antiga em que a codificação dos testes é feita antes da implementação (BORGES, 2006). Desenvolver testes antes das funcionalidades é a principal característica do TDD, que tem como objetivo garantir efetividade nos processos. Todavia, apesar de a ideia ser antiga, essa técnica como metodologia é recente, devido ao fato de ter sido disseminada, de forma a tornar-se uma técnica bem elaborada com a possibilidade de ser usada por massas e não apenas como aventuras testadas pelos pioneiros (KOSKELA, 2008).

O TDD consiste no desenvolvimento dos testes antes do software propriamente dito. Considera-se que será escrito teste para cada funcionalidade do produto de software em iterações pequenas e rápidas. Segundo Koskela (2008), a efetividade do processo pode ser resumida em: “apenas escreva código-fonte para fazer um teste passar”.

[1] http://www.extremeprogramming.org/
[2] http://www.manifestoagil.com.br/

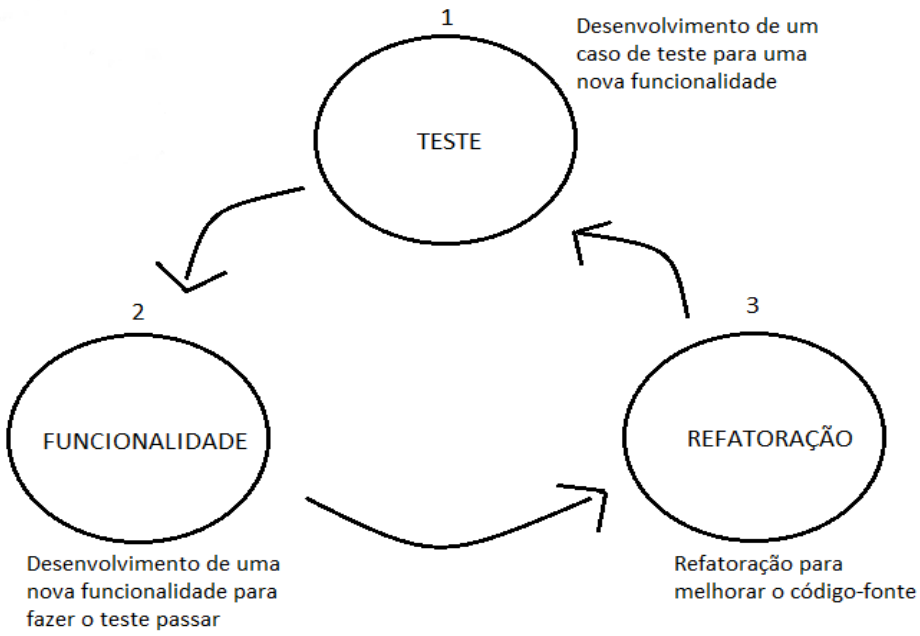

Figura 2: A figura mostra o ciclo de etapas de funcionamento do TDD

A Figura 2 ilustra o ciclo do TDD, que ocorre da seguinte maneira: 1) O desenvolvedor escolhe uma funcionalidade do projeto e desenvolve seu teste. Com o teste finalizado, inicialmente ele não passa, importante ressaltar que essa é a ideia pregada pelo TDD: obrigar o programador a pensar a respeito da funcionalidade e projeto já na construção do teste, mesmo antes da implementação. Desta forma, o desenvolvedor está apto para realizar o próximo passo. 2) Com o conhecimento das operações que a implementação precisará realizar detalhadamente, este passo consiste em construir a implementação do código-fonte da funcionalidade para que o teste passe. Então o teste poderá ser executado para verificar se a funcionalidade está de acordo com os requisitos levantados. 3) Assim que o teste passar, finalmente, deve-se refatorar o código-fonte de modo a realizar uma revisão para aprimorá-lo e melhorar o design (BORGES, 2006).

3.2 BENEFÍCIOS DO TDD

Além de ajudar na implementação do código-fonte, os testes gerados com o TDD podem ser re-executados a qualquer momento, o que ajuda a garantir que a implementação de uma determinada funcionalidade não passará a apresentar resultados inesperados após alguma refatoração (VICENTE, 2010). O TDD também traz benefícios em relação à organização, pois produz um código-fonte fácil de testar, além de evitar sobrescrita de código-fonte causado por suposições equivocadas. Estas vantagens são possíveis de se obter devido ao fato de o desen-

volvedor saber exatamente os passos que devem ser seguidos, por ter codificado os testes antes da implementação (KOSKELA, 2008).

Vicente (2010) destaca várias vantagens obtidas com o uso de TDD:

- *Confiança*: os testes validam cada funcionalidade e estão disponíveis para verificar a cada alteração realizada;
- *Flexibilidade*: facilita a manutenção, pois o desenvolvedor não precisa se preocupar em causar erros em outra parte do software;
- *Documentação*: se bem desenvolvidos, os testes podem documentar o software de forma eficiente, deve-se tomar o cuidado de mantê-los atualizados;
- *Facilidade de depuração*: se o TDD for feito de maneira correta, o software fica modularizado, o que facilita a tarefa de encontrar o local do erro, sem precisar executar no modo de depuração por muito tempo, pois o erro foi inserido recentemente;
- *Melhora do projeto*: códigos-fonte testáveis possuem classes e métodos com poucas dependências de outras classes. Código-fonte resultante é limpo e flexível, o que facilita a manutenção do mesmo;
- *Código-fonte que funciona*: o fato de o código-fonte passar nos testes construídos garantem a eficiência do software;
- *Código-fonte entregue em tempo*: facilita a tarefa de estimar o tempo necessário de produzir o software.

4 AVALIAÇÃO EMPÍRICA

Neste capítulo será descrita como foi realizada a avaliação empírica da qualidade interna de softwares produzidos com TDD. Para chegar aos resultados obtidos, foram analisados softwares produzidos com a técnica TDD e também softwares produzidos sem o uso da técnica em questão.

4.1 ESTUDOS DE CASO

Conforme discutido na literatura, estudos de caso desenvolvidos com a técnica TDD não estão explicitamente disponibilizados para estudos empíricos. Para tal, buscou-se indicações de softwares desenvolvidos com TDD em fóruns de discussões. Outro critério adotado para selecionar estudos de casos foi a disponibilização de tais softwares em uma plataforma de disponibilização de códigos-fonte, tal como o github[1]. Os estudos de caso selecionados desenvolvidos com a técnica TDD (Grupo 1) estão apresentados na Tabela 6.

Para conduzir a avaliação empírica, decidiu-se utilizar como comparação um conjunto de softwares que não apresentam evidências de serem desenvolvidos utilizando-se a técnica TDD. Estes estudos de caso não desenvolvidos com TDD serão utilizados como parâmetros para comparação da qualidade interna dos softwares desenvolvidos com TDD. A Tabela 7 apresenta o conjunto de estudos de caso utilizados como base de comparação (Grupo 2).

Ao importar os estudos de caso, notou-se que alguns deles são organizados em vários projetos independentes, nesses casos as métricas foram computadas separadamente e consequentemente os atributos de qualidade também foram calculados individualmente. Dessa forma, para realizar a análise empírica dos resultados obtidos foram considerados os valores dos atributos de qualidade separadamente.

[1] http://github.com – rede social em que os desenvolvedores podem armazenar seus projetos de forma gratuita para códigos-fonte abertos.

Tabela 6: Estudos de caso Grupo 1

Est. de caso	Projetos	Descrição	URL
Dindwarf	dimdwarf-aop dimdwarf-api dimdwarf-api-internal end-to-end-tests launcher test-utils	Servidor de aplicativos distribuídos para Java.	http://github.com/orfjackal/dimdwarf
Mockito	mockito	Mais popular framework para realizar mock de testes unitários desenvolvidos em java.	http://github.com/mockito/mockito
Fitnesse	fitnesse	Servidor web, wiki e ferramenta de teste automatizada totalmente integrada para software.	http://github.com/unclebob/fitnesse
Apache Commons Lang	commons-lang	Provedor de bibliotecas Java.	http://github.com/apache/commons-lang
JUnit	junit-jupiter-api junit-platform-commons junit-platform-console junit-platform-engine junit-platform-launcher junit-platform-runner	JUnit é um framework simples para escrever testes repetíveis.	https://github.com/junit-team/junit5

Fonte: Autoria própria

4.2 FERRAMENTAS

Como ambiente de desenvolvimento integrado (IDE) utilizou-se o Eclipse Kepler Service Release 2 e JDK versão 8. Nesta IDE foram importados os estudos de caso utilizando-se a ferramenta git[2], que permite copiar/clonar todos os artefatos de implementação de um software.

Para computar as métricas de software dos estudos de caso, utilizou-se o plugin da IDE Eclipse chamado Metrics[3] na versão 1.3.6. Este plugin implementa as métricas de software necessárias para calcular os atributos de qualidade apresentados no capítulo 2 e é o mesmo utilizado em no trabalho relacionado de Jetter (2007).

[2]http://git-scm.com - ferramenta utilizada para controle de versionamento.
[3]http://marketplace.eclipse.org/content/eclipse-metrics

Tabela 7: Estudos de caso Grupo 2

Est. de caso	Projetos	Descrição	URL
MyBatis3	chapter01 chapter02 chapter03 chapter04 chapter05	Projeto referente a um livro que ensina utilizar o MyBatis3 (framework de persistência de classes com suporte a customização SQL).	https://github.com/sivaprasadreddy/Java-Persistence-with-MyBatis3
dl4j-examples	nd4j-examples datavec-examples dl4j-cuda-specifc-examples dl4j-spark	Exemplos desenvolvidos com a biblioteca deeplearning4j (biblioteca de aprendizagem Java e Scala distribuída).	https://github.com/deeplearning4j/dl4j-examples
JMetal	jmetal	Framework de otimização multi-objetivo com metaheurísticas.	https://github.com/jMetal/jMetal
JBoss	jboss	Um sistema modular de carregamento de classes.	https://github.com/jboss-modules/jboss-modules
JHotDraw	jhotdraw-core jhotdraw-samples-mini jhotdraw-utils jhotdrae-xml nanoxml-patched	Biblioteca de desenho 2D	https://github.com/wumpz/jhotdraw

Fonte: Autoria própria

4.3 COLETA DE DADOS

Para realizar a coleta de dados, inicialmente foi necessário resolver vários problemas de compilação dos projetos, quando existiam. Estes problemas ocorreram principalmente devido a problemas de importação de dependências dos projetos e versão do JDK. Na sequência utilizou-se o plugin Metrics para computar os valores das métricas definidas. Os valores obtidos para cada métrica foram utilizados para popular uma planilha eletrônica para realizar o cálculo das propriedades de design e os atributos de qualidade.

Para computar os valores das métricas, é necessário importar os projetos na IDE eclipse e para cada estudo de caso deve-se habilitar a opção do Metrics e realizar um *build* nos projetos. Dessa forma as métricas serão calculadas para o determinado projeto e os valores são mostrados na visão do plugin chamada Metrics View.

O Metrics computa os valores das métricas de forma absoluta (para as métricas que é possível) e também disponibiliza os valores médios de cada métrica (a unidade depende da métrica em questão). Dentre as métricas utilizadas, as que possuem valores absolutos utilizados no cálculo dos atributos de qualidade são as seguintes: linhas de código, número de classes, número de atributos, número de métodos e número de métodos sobrescritos. As métricas em que o valor utilizado é uma média ou resultado de cálculos são: profundidade da árvore de herança, abstração, instabilidade, falta de coesão entre métodos e peso dos métodos por classe.

Somente com os resultados dos atributos de qualidade dos estudos de caso com TDD não é possível chegar a conclusões em relação a qualidade dos projetos, pois não existe uma classificação genérica de intervalos de valores considerados bons ou ruins para cada atributo. Por esse motivo é necessário considerar outro grupo de estudos de caso desenvolvidos da forma tradicional (testes depois da implementação) e extrair os dados das mesmas métricas para calcular os atributos de qualidade interna e, desta forma, realizar uma avaliação empírica para comparar os valores dos atributos de qualidade obtidos em softwares desenvolvidos com a técnica dirigida a testes com os resultados encontrados com softwares desenvolvidos da forma tradicional.

5 RESULTADOS E ANÁLISES

Neste capítulo são apresentados os dados computados das métricas dos estudos de caso, os valores das propriedades, além dos resultados de cada atributo de qualidade para cada projeto juntamente com as respectivas médias. Em seguida é realizada uma análise desses dados a respeito da qualidade interna dos estudos de caso utilizados.

5.1 RESULTADOS

Os dados coletados das métricas nos estudos de caso dos grupos 1 e 2 são apresentados nas tabelas 8 e 9 respectivamente. Os valores das propriedades de design calculados para cada estudo de caso são representados nas tabelas 10 e 11. Com os valores das propriedades, foram calculados os atributos de qualidade para cada estudo de caso e os resultados obtidos estão detalhados na tabela 12.

Tabela 8: Dados extraídos dos estudos de caso do grupo 1

Projetos	LDC	NDC	PAH	Abstração	Instabilidade	LCOM	NDA	NDM	NDMS	PMC
fitnesse	40172	790	1,622	0,141	0,529	0,196	1363	4592	275	10,056
mockito	12221	305	1,482	0,258	0,409	0,110	302	1061	81	7,295
commons-io	9994	119	2,118	0,104	0,735	0,163	204	772	173	19,118
junit-platform-engine	1879	35	1,029	0,239	0,294	0,193	61	186	50	8,114
junit-jupiter-api	1478	7	2,714	0,868	0,246	0,119	4	14	2	42,857
junit-platform-commons	1152	15	1,533	0,375	0,097	0,100	9	45	3	13,600
junit-platform-console	1136	15	1,067	0,094	0,705	0,365	51	127	1	12,200
junit-platform-launcher	1109	18	1,000	0,215	0,398	0,324	50	113	5	8,611
junit-platform-runner	529	3	1,333	0,750	0,750	0,606	10	49	0	24,667
test-utils	205	6	1,667	0,000	0,286	0,000	2	14	0	6,000
dimdwarf-aop	164	6	1,667	0,222	0,643	0,218	10	19	1	5,667
dimdwarf-api-internal	59	2	1,000	0,667	0,057	0,000	1	7	3	4,000
launcher	32	1	1,000	0,000	1,000	0,000	0	0	0	5,000
end-to-end-tests	17	2	1,500	0,000	0,333	0,000	0	2	0	1,000
Média /Soma	70147	1324	1,481	0,281	0,463	0,171	1,561	7001	594	12,013

LDC: Linhas de Código, NDC: Número de Classes, PAH: Profundidade Árvore de Herança, LCOM: Falta de Coesão entre Métodos, NDA: Número de Atributos, NDM: Número de Métodos, NDMS: Número de Métodos Sobrescritos, PMC: Peso Métodos por Classe

Fonte: Autoria própria.

Tabela 9: Dados extraídos dos estudos de caso do grupo 2

Projetos	LDC	NDC	PAH	Abstração	Instabilidade	LCOM	NDA	NDM	NDMS	PMC
jhotdraw-core launcher	47892	450	2,756	0,169	0,353	0,242	1040	3952	588	19,196
Jmetal	35246	456	1,956	0,211	0,424	0,233	1343	2494	94	12,430
Jboss	14539	133	1,406	0,263	0,452	0,254	332	1023	175	24,436
jhotdraw-utils	6497	39	1,641	0,010	0,064	0,322	122	397	24	36,179
nanoxml-patched	3305	18	1,611	0,250	0,526	0,285	50	199	4	31,167
jhotdraw-samples-mini	2886	36	2,778	0,000	1,000	0,131	118	92	8	4,833
jhotdraw-xml	1433	10	1,100	0,182	0,435	0,387	28	124	5	34,700
chapter04	1030	13	1,154	0,143	0,621	0,292	27	112	4	10,615
chapter05	931	14	1,143	0,125	0,700	0,319	32	114	4	9,786
chapter03	831	14	1,143	0,167	0,629	0,271	28	91	4	8,214
dl4j-spark	464	4	1,000	0,000	0,833	0,000	8	5	0	12,500
chapter01	421	6	1,000	0,200	0,567	0,111	5	8	1	8,333
chapter02	415	8	1,250	0,167	0,583	0,249	9	37	2	6,875
datavec-examples	300	4	1,000	0,000	1,000	0,000	0	0	0	3,000
nd4j-examples	254	6	1,000	0,000	1,000	0,000	0	0	0	1,000
dl4j-cuda-specific-examples	114	1	1,000	0,000	1,000	0,000	0	0	0	3,000
Média /Soma	116558	1212	1,434	0,118	0,637	0,194	2,592	8648	913	14,142

LDC: Linhas de Código, NDC: Número de Classes, PAH: Profundidade Árvore de Herança, LCOM: Falta de Coesão entre Métodos, NDA: Número de Atributos, NDM: Número de Métodos, NDMS: Número de Métodos Sobrescritos, PMC: Peso Métodos por Classe

Fonte: Autoria própria.

Tabela 10: Propriedades de design dos estudos de caso do grupo 1

Projetos	ENC	TAM	HIER	ABST	ACOP	COE	COMP	HER	POL	COMUN
fitnesse	1	790	1,622	0,141	0,529	5,102	1363	0,940	275	4592
mockito	1	305	1,482	0,258	0,409	9,091	302	0,924	81	1061
commons-io	1	119	2,118	0,104	0,735	6,135	204	0,776	173	772
junit-platform-engine	1	35	1,029	0,239	0,294	5,181	61	0,731	50	186
junit-jupiter-api	1	7	2,714	0,868	0,246	8,403	4	0,857	2	14
junit-platform-commons	1	15	1,533	0,375	0,097	10,000	9	0,933	3	45
junit-platform-console	1	15	1,067	0,094	0,705	2,740	51	0,992	1	127
junit-platform-launcher	1	18	1,000	0,215	0,398	3,086	50	0,956	5	113
junit-platform-runner	1	3	1,333	0,750	0,750	1,650	10	1,000	0	49
test-utils	1	6	1,667	0,000	0,286	0,000	2	1,000	0	14
dimdwarf-aop	1	6	1,667	0,222	0,643	4,587	10	0,947	1	19
dimdwarf-api-internal	1	2	1,000	0,667	0,057	0,000	1	0,571	3	7
launcher	1	1	1,000	0,000	1,000	0,000	0	0,000	0	0
end-to-end-tests	1	2	1,500	0,000	0,333	0,000	0	1,000	0	2
Média /Soma	1	1324	1,481	0,281	0,463	5,848	1,561	0,915	594	5,288

ENC: Encapsulamento, TAM: Tamanho do Design, HIER: Hierarquia, ABST: Abstração, ACOP: Acoplamento, COE: Coesão, COMP: Composição, HER: Herança, POL: Polimorfismo, COMUN: Comunicação

Fonte: Autoria própria.

Tabela 11: Propriedades de design dos estudos de caso do grupo 2

Projetos	ENC	TAM	HIER	ABST	ACOP	COE	COMP	HER	POL	COMUN
jhotdraw-core	1	450	2,756	0,169	0,353	4,132	1040	0,851	588	3952
Jmetal	1	456	1,956	0,211	0,424	4,292	1343	0,962	94	2494
Jboss	1	133	1,406	0,263	0,452	3,937	332	0,829	175	1023
jhotdraw-utils	1	39	1,641	0,010	0,064	3,106	122	0,940	24	397
nanoxml-patched	1	18	1,611	0,250	0,526	3,509	50	0,980	4	199
jhotdraw-samples-mini	1	36	2,778	0,000	1,000	7,634	118	0,913	8	92
jhotdraw-xml	1	10	1,100	0,182	0,435	2,584	28	0,960	5	124
chapter04	1	13	1,154	0,143	0,621	3,425	27	0,964	4	112
chapter05	1	14	1,143	0,125	0,700	3,135	32	0,965	4	114
chapter03	1	14	1,143	0,167	0,629	3,690	28	0,956	4	91
dl4j-spark	1	4	1,000	0,000	0,833	0,000	8	1,000	0	5
chapter01	1	6	1,000	0,200	0,567	9,009	5	0,875	1	8
chapter02	1	8	1,250	0,167	0,583	4,016	9	0,946	2	37
datavec-examples	1	4	1,000	0,000	1,000	0,000	0	0,000	0	0
nd4j-examples	1	6	1,000	0,000	1,000	0,000	0	0,000	0	0
dl4j-cuda-specific-examples	1	1	1,000	0,000	1,000	0,000	0	0,000	0	0
Média /Soma	1	1212	1,434	0,118	0,637	3,279	2,592	0,759	913	7,135

ENC: Encapsulamento, TAM: Tamanho do Design, HIER: Hierarquia, ABST: Abstração, ACOP: Acoplamento, COE: Coesão, COMP: Composição, HER: Herança, POL: Polimorfismo, COMUN: Comunicação

Fonte: Autoria própria.

Tabela 12: Valores dos atributos de qualidade calculados para os grupos 1 e 2

Projetos	REU	FLE	COM	FUN	EXT	EFE
fitnesse	2692,143	819,118	-352,976	1245,509	137,776	328,016
mockito	685,170	191,648	-126,677	319,757	40,886	77,036
commons-io	446,850	188,566	-100,591	235,282	86,572	75,776
junit-platform-engine	111,722	55,677	-28,864	60,468	25,338	22,594
junit-jupiter-api	12,539	3,189	-14,377	6,665	1,740	1,745
junit-platform-commons	32,476	6,226	-6,954	15,397	2,106	2,862
junit-platform-console	71,509	26,074	-8,336	32,024	0,691	10,817
junit-platform-launcher	66,172	27,651	-9,285	30,510	2,886	11,434
junit-platform-runner	26,225	5,063	-8,750	11,931	0,500	2,550
test-utils	9,929	1,179	-3,724	4,767	0,357	0,800
dimdwarf-aop	13,486	5,589	-2,622	6,637	0,763	2,634
dimdwarf-api-internal	4,486	2,236	-2,879	2,860	2,091	1,248
launcher	0,250	0,000	-1,980	0,440	-0,500	0,200
end-to-end-tests	1,917	0,167	-0,770	1,210	0,334	0,400
Média	298,205	95,170	-47,770	140,961	21,539	38,437
jhotdraw-core	2201,945	814,162	-347,353	1098,902	294,334	326,004
Jmetal	1475,967	718,644	-184,065	670,625	47,375	287,835
Jboss	578,871	253,637	-108,311	293,602	87,820	101,818
jhotdraw-utils	218,760	73,234	-31,399	101,934	12,443	29,590
nanoxml-patched	109,246	27,119	-16,313	49,395	2,352	11,246
jhotdraw-samples-mini	65,658	63,000	-13,596	31,447	3,957	25,583
jhotdraw-xml	67,537	16,641	-15,422	31,132	2,853	7,028
chapter04	63,201	15,595	-7,905	29,045	2,243	6,621
chapter05	64,609	18,075	-8,077	29,668	2,195	7,618
chapter03	53,265	16,093	-7,366	24,674	2,247	6,825
dl4j-spark	4,292	4,042	-5,390	2,200	0,084	2,000
chapter01	9,111	3,108	-2,010	4,601	0,754	1,615
chapter02	23,358	5,604	-4,161	11,097	1,265	2,623
datavec-examples	1,750	0,000	-2,310	1,100	-0,500	0,200
nd4j-examples	2,750	0,000	-2,310	1,540	-0,500	0,200
dl4j-cuda-specific-examples	0,250	0,000	-1,320	0,440	-0,500	0,200
Média	308,786	126,810	-47,332	148,838	28,651	51,063

REU: Reusabilidade, FLE: Flexibilidade, COM: Compreensividade,

FUN: Funcionalidade, EXT: Extensibilidade, EFE: Efetividade

Fonte: Autoria própria.

5.2 ANÁLISE

Por meio dos valores das métricas de qualidade obtidos, é possível observar que o tamanho do projeto tem uma forte influência nos resultados, quanto maior o tamanho do projeto, a tendência é obter índices maiores nos atributos de qualidade. De forma geral, os estudos de caso utilizados no Grupo 1 (desenvolvidos com TDD) apresentam tamanhos menores em relação aos estudos de caso do Grupo 2 (desenvolvidos da forma tradicional).

Considerando apenas os dados extraídos das métricas, nota-se que os estudos de caso do Grupo 1 são mais modularizados que os estudos de caso do Grupo 2. Essa constatação é comprovada pelo fato de a soma das linhas de código de todos os estudos de caso do Grupo 1 (70147) ser menor que a soma dos casos de teste do Grupo 2 (116558) enquanto a quantidade de classes total do Grupo 1 (1324) é superior a quantidade de classes do Grupo 2 (1212). Essa conclusão fica muito mais evidente a partir da comparação das médias de linhas por classe dos estudos de caso, a média do Grupo 1 foi de aproximadamente 53 linhas por classe enquanto a essa média no Grupo 2 é superior a 96.

Uma informação importante, percebida por meio da observação dos valores computados pelas métricas, é que a quantidade média de atributos por classe do Grupo 1 é inferior ao Grupo 2 (1,56 atributos por classe contra 2,59). A quantidade de atributos indica o nível de composição, e quanto menor for o valor da média dessa propriedade, melhor é o impacto na efetividade e flexibilidade dos projetos. Outro ponto importante a destacar é o percentual de métodos sobrescritos, nesse quesito, os casos de estudo do Grupo 2 obtiveram resultados melhores aos do Grupo 1 (média de 11% do Grupo 2 e média de 8% do Grupo 1). A quantidade de métodos, métodos sobrescritos e a relação entre essas métricas indicam as propriedades de comunicação entre classes, polimorfismo e herança respectivamente.

Além disso, os graus de abstração e instabilidade apresentaram resultados consideravelmente diferentes de um grupo para o outro. Enquanto o Grupo 1 obteve média de 0,28 de abstração, o Grupo 2 apresentou média de 0,12 (diferença superior a 100%), dessa forma, devido à influência dessas propriedades, é possível afirmar que os softwares do Grupo 1 são mais extensíveis e eficientes, em contrapartida, o valor mais baixo dos estudos de caso do Grupo 2 indica maior compreensibilidade. Em relação ao grau de instabilidade, os resultados médios foram de 0,46 para o Grupo 1 e 0,64 para o Grupo 2, o que evidencia a maior qualidade interna do Grupo 1 nesse quesito, já que esta métrica indica o acoplamento do projeto.

Duas médias em que os dois grupos apresentaram médias parecidas foram nas métricas de falta de coesão entre os métodos (0,17 no Grupo 1 e 0,19 no Grupo 2) e peso dos métodos

por classe (12,01 no Grupo 1 e 14,14 no Grupo 2). No caso da métrica falta de coesão entre métodos, o valor deve ser o menor possível e será utilizado para calcular a coesão, assim como o valor do peso dos métodos por classe, que simboliza a complexidade do projeto. Em ambas as métricas a média apresentada pelo Grupo 1 é pouco melhor que a média do Grupo 2.

Por meio da observação dos resultados dos atributos de qualidade pode-se precipitadamente concluir que a qualidade interna dos estudos de caso do Grupo 2 é superior, principalmente devido às médias de cada atributo de qualidade do Grupo 2 serem superiores em relação ao Grupo 1. Todavia, é importante ressaltar o peso que o tamanho de cada projeto tem nos resultados dos cálculos dos atributos de qualidade, já que os valores são resultados de somas.

Devido ao impacto do tamanho do projeto nos valores dos atributos de qualidade é necessário ter cuidado ao realizar uma comparação entre os valores dos atributos de qualidade encontrados no Grupo 1 e no Grupo 2. Para que a comparação seja realista, deve-se comparar projetos com tamanhos parecidos, para isso deve-se observar a quantidade de linhas de código de cada projeto. Por exemplo, ao utilizar os maiores projetos de cada grupo (fitnesse do Grupo 1 com 40172 linhas e jhotdraw-core do Grupo 2 com 47892 linhas) para realizar uma comparação. Apesar de o tamanho do projeto do Grupo 2 seja consideravelmente maior, o mesmo obteve resultados superiores apenas em extensibilidade e compreensividade, enquanto o projeto do Grupo 1 apresenta valores melhores em efetividade, reusabilidade, funcionalidade e flexibilidade.

Apesar de não analisar estilos de codificação, outros dois projetos compatíveis em tamanho para realizar comparação são o junit-platform-laucher (1109 linhas) do Grupo 1 e o chapter04 (1030 linhas) do Grupo 2. Neste caso o projeto do Grupo 1 apresenta resultados superiores em reusabilidade, flexibilidade, funcionalidade, extensibilidade e efetividade enquanto o projeto do Grupo 2 mostrou superioridade apenas no atributo de software compreensibilidade. Uma comparação em que o software do Grupo 2 apresenta melhores resultados é entre os estudos de caso mockito do Grupo 1 (com 12221 linhas) e jboss do Grupo 2 (com 14559 linhas), nesse caso nota-se que o projeto do Grupo 1 apresenta melhores resultados apenas em dois atributos de qualidade (reusabilidade e funcionalidade), já o projeto do Grupo 2 obteve resultados superiores em flexibilidade, compreensibilidade, extensibilidade e efetividade.

Por meio de uma comparação entre as médias obtidas por Grupo nas propriedades de software, nota-se a superioridade par parte do Grupo 1 em relação ao Grupo 2 na grande maioria. A média de complexidade do Grupo 1 foi 12,013 enquanto o índice médio dessa propriedade no Grupo 2 foi de 14,142 (diferença de 15%), a diferença no índice médio de acoplamento também é significante: 0,463 do Grupo 1 e 0,637 do Grupo 2 (aproximadamente

27%). Os índices médios de coesão e abstração apresentam ainda maior discrepância, a média de coesão do Grupo 1 é 44% maior que a do Grupo 2 (5,848 e 3,279), diferença considerável obtida mesmo com médias parecidas na métrica computada LCOM (0,17 do Grupo 1 e 0,19 do Grupo 2). A abstração, como citado anteriormente, com mais de 100% de diferença (0,281 do Grupo 1 e 0,118 do Grupo 2).

Os estudos de caso desenvolvidos com TDD também apresentam resultados superiores em relação a herança (média de 0,915 no Grupo 1 e 0,759 no Grupo 2), diferença de 17% nessa propriedade que é importante para a extensibilidade e efetividade. O Grupo 2 obteve média superior na quantidade de métodos públicos, fator importante para comunicação entre as classes (média de 7,135 métodos públicos por classe no Grupo 2 enquanto o Grupo obteve 5,288). Além disso, sobre o índice médio de polimorfismo, que é a relação entre métodos sobrescritos e métodos, o Grupo 2 obteve valores superiores ao do Grupo 1 (cerca de 11% nos estudos de caso do Grupo 2, enquanto esse índice alcançou apenas 8% no Grupo 1). Houve ainda, médias muito similares entre os grupos no quesito de hierarquia (1,481 do Grupo 1 e 1,434 do Grupo 2), essa propriedade é importante para o atributo de qualidade funcionalidade.

6 CONCLUSÃO

Como explicado no capítulo de análise, de forma geral, é possível afirmar que os softwares desenvolvidos com TDD (Grupo 1) apresentaram melhores resultados que os softwares desenvolvidos da forma convencional (Grupo 2) em relação a qualidade interna. Esta superioridade é evidente devido ao fato da maior modularização dos estudos de caso do Grupo 1, dessa maneira a complexidade e o acoplamento das classes são reduzidos, além disso, a coesão e abstração são elevados em relação aos estudos de caso do Grupo 2.

Os resultados são considerados satisfatórios e comprovam os benefícios que na teoria é esperado da aplicação da técnica de desenvolvimento dirigida a testes. Todavia, é importante ressaltar, como explicado no trabalho, ao otimizar alguns atributos de qualidade outros são prejudicados, por isso a superioridade dos resultados por parte dos estudos de caso desenvolvidos com TDD não é unânime nos atributos de qualidade. Além disso, essa superioridade não é uma regra, pois nota-se casos em que software desenvolvido da maneira tradicional obteve resultados superiores aos resultados de softwares desenvolvidos com TDD.

Visto que de forma geral o uso da técnica TDD traz consequências positivas tanto na qualidade interna e externa de softwares, o motivo da não utilização generalizada desta técnica é a cultura de desenvolvimento de software profundamente enraizada na abordagem *top-down*. As metodologias ágeis (incluindo a técnica TDD) surgiram como uma opção de desenvolvimento de software e têm-se mostrado uma alternativa a facilitar adaptações necessárias com a evolução da tecnologia. A tendência é o crescimento do uso das metodologias ágeis devido aos benefícios possibilitados pela sua aplicação.

6.1 TRABALHOS FUTUROS

Dentre as possibilidades de trabalhos futuros, destaca-se:

- *Teste de correlação*: realizar testes de correlação entre métricas de código-fonte de softwares desenvolvidos com e sem TDD, para avaliar a correlação estatística entre os dados.

- *Utilizar mais estudos de caso*: aplicar o modelo de qualidade em mais estudos de caso para ter uma base de comparação maior, possibilitando uma análise mais efetiva.
- *Análise mais aprofundadas*: Analisar de forma mais detalhada os resultados dos atributos de qualidade para verificar outros aspectos do software que influenciam na qualidade interna do projeto.

REFERÊNCIAS

BANSIYA, J.; DAVIS, C. G. A hierarchical model for object-oriented design quality assessment. **IEEE Transactions on software engineering**, IEEE, v. 28, n. 1, p. 4–17, 2002.

BECK, K. **Test Driven Development: By Example**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321146530.

BISSI, W.; NETO, A. G. S. S.; EMER, M. C. F. P. The effects of test driven development on internal quality, external quality and productivity: A systematic review. **Information and Software Technology**, v. 74, p. 45–54, 2016. ISSN 0950-5849.

BOEHM, B. W. **Characteristics of software quality**. [S.l.]: North-Holland, 1978.

BORGES, E. N. Conceitos e benefícios do test driven development. **Porto Alegre, Rio Grande do Sul, Brasil: Universidade Federal do Rio Grande do Sul. Acesso em**, v. 16, 2006.

CHANDRASEKAR, A.; SUDHARAJESH, M.; RAJESH, M. P. A research study on software quality attributes. **International Journal of Scientific and Research Publications**, Citeseer, v. 4, n. 1, p. 1–4, 2014.

JEFFRIES, R.; MELNIK, G. Guest editors' introduction: Tdd–the art of fearless programming. **Software, IEEE**, IEEE, v. 24, n. 3, p. 24–30, 2007.

JETTER, A. **Assessing software quality attributes**. Tese (Doutorado) — Citeseer, 2007.

KITCHENHAM, B.; PFLEEGER, S. L. Software quality: The elusive target. **IEEE software**, IEEE Computer Society, v. 13, n. 1, p. 12, 1996.

KOSKELA, L. **Test Driven: TDD and Acceptance TDD for Java Developers**. [S.l.]: Manning Publications Co, 2008.

MALDONADO, J. C.; ROCHA, A.; WEBER, K. Qualidade de software: teoria e prática. **São Paulo**, 2001.

MORAIS, C.; MEIRELLES, P.; KON, F. Kalibro: Uma ferramenta de configuraçao e interpretaçao de métricas de código-fonte. **Undergraduation course conclusion project, Universidade de Sao Paulo**, 2009.

MUNIR, H.; MOAYYED, M.; PETERSEN, K. Considering rigor and relevance when evaluating test driven development: A systematic review. **Information and Software Technology**, v. 56, n. 4, p. 375–394, 2014. ISSN 0950-5849.

O'REGAN, G. **A practical approach to software quality**. [S.l.]: Springer Science & Business Media, 2012.

SOMMERVILE, I. **Engenharia de Software**. [S.l.]: PEARSON EDUCATION, 2011.

VICENTE, A. A. **Definição e gerenciamento de métricas de teste no contexto de métodos ágeis**. Tese (Doutorado) — Universidade de São Paulo, 2010.