UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

BRUNO HENRIQUE PACHULSKI CAMARA

UMA INVESTIGAÇÃO SOBRE O USO DE CRITÉRIOS DE TESTE NO DESENVOLVIMENTO BASEADO EM TESTES PARA O ENSINO DE PROGRAMAÇÃO

DISSERTAÇÃO

CORNÉLIO PROCÓPIO
2016

BRUNO HENRIQUE PACHULSKI CAMARA

UMA INVESTIGAÇÃO SOBRE O USO DE CRITÉRIOS DE TESTE NO DESENVOLVIMENTO BASEADO EM TESTES PARA O ENSINO DE PROGRAMAÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Dr. Marco Aurélio Graciotto Silva

CORNÉLIO PROCÓPIO
2016

Dados Internacionais de Catalogação na Publicação

P116 Camara, Bruno Henrique Pachulski
Uma investigação sobre o uso de critérios de teste no desenvolvimento baseado em testes para o ensino de programação / Bruno Henrique Pachulski Camara. Cornélio Procópio. UTFPR, 2016.
130. f. : il. ; 30 cm

Orientador: Dr. Marco Aurélio Graciotto Silva.
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Informática. Cornélio Procópio, 2016.
Bibliografia: f. 121 - 130.

1. Software - Testes. 2. Engenharia de software. 3. Programação (Computadores) - Ensino. 4. Informática - Dissertações. I. Silva, Marco Aurélio Graciotto, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD (22. ed.) 004

Biblioteca da UTFPR de Cornélio Procópio

Dedico este trabalho ao meu filho Samuel e minha esposa Jéssica.

Agradecimentos

Inicialmente agradeço a Deus por me permitir viver essa experiência magnífica de grande aprendizado. Sem a força vinda dEle não conseguiria desenvolver essa pesquisa.

Agradeço a minha esposa Jéssica, pela compreensão e apoio durante os últimos dois anos e em alguns momentos aturar meus falatórios sobre engenharia de software. Ao meu filho Samuel que mesmo sem compreender, com seu sorriso me dava forças para continuar e em muitas ocasiões um abraço para me acalmar. Não poderia deixar de citar minha família. Pai, mãe, Joice, Moreira, Gustavo, sempre solícitos em ajudar.

Meu orientado Prof. Marco, meu agradecimento eterno pela paciência, respeito, dedicação, companheirismo e ética.

A todos os colegas de classe, sem vocês teria sido muito mais difícil. Obrigado Anderson Fernandes e Douglas Nassif pelo companheirismo, amizade e apoio.

Obrigado a minha coordenadora do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, por acreditar em mim. Gostaria de agradecer também ao diretor Pedro Baer pelo apoio irrestrito da Faculdade Integrado, sem dúvidas foi essencial.

A todos, sou grato a Deus pela vida de vocês, espero um dia poder retribuir cada gesto de amizade.

Resumo

Camara, Bruno Henrique Pachulski. Uma investigação sobre o uso de critérios de teste no desenvolvimento baseado em testes para o ensino de programação . 2016. 130. f. Dissertação (Programa de Pós-Graduação em Informática), Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

Contexto: Estudantes de cursos da área de Ciência da Computação que estão iniciando em programação normalmente têm dificuldade de assimilar os conteúdos básicos. Alguns pesquisadores propõem a utilização do desenvolvimento baseado em testes (TDD) a fim de melhorar o ensino, mas sem o foco na qualidade do conjunto de casos de teste.
Objetivo: Este trabalho tem o objetivo de propor uma estratégia baseada no desenvolvimento baseado por testes em conjunto com técnicas de teste de software, guiando o aluno a desempenhar suas atividades de programação com mais qualidade e, desta forma, obter melhores resultados acadêmicos.
Método: Baseado em um mapeamento sistemático da literatura e na revisão de estudos que aplicam desenvolvimento baseado em testes (TDD) em âmbito educacional, foi definida uma estratégia para integrar critérios de teste ao TDD. Para avaliar os resultados, foram definidas medidas e instrumentos para a coleta de resultados. Foi executado um experimento científico com o público alvo para a obtenção das informações com a finalidade de avaliar a hipótese definida.
Resultados: Foi definida uma estratégia de utilização de técnicas de teste de software em conjunto com o TDD para alunos que estão iniciando em programação em cursos de Computação. A estratégia consiste na melhoria do conjunto de casos de teste durante a refatoração no TDD, considerando critérios estruturais baseado em fluxo de controle. Além disso, foram elaboradas ferramentas para a coleta de dados durante o desenvolvimento de programas e para análise dos dados. A estratégia permitiu a melhora do programa e dos casos de teste e foi avaliada pelos estudantes como um guia no desenvolvimento dos casos de teste e como um atalho para descoberta de defeitos.
Conclusões: Com a utilização da estratégia proposta, foi possível observar melhorias no conjunto de casos de teste e no software final. A estratégia definida pode ser replicada em

outros contextos e com o emprego de outras técnicas e critérios de teste de software para a avaliação da hipótese definida e generalização dos resultados.

Palavras-chaves: desenvolvimento baseado em testes; teste de software; critério de teste; ensino; introdução a programação.

Abstract

Camara, Bruno Henrique Pachulski. An investigation on using testing criteria in test-driven development for programming education. 2016. 130. f. Master's Thesis (Graduation Program in Informatics), Federal University of Technology – Paraná. Cornélio Procópio, PR, Brazil, 2016.

Background: Students of Computer Science courses usually have difficulty assimilating the basic contents of introductory programming disciplines. Some researchers propose the use of Test-Driven Development (TDD) to improve teaching, but with no focus in the test set quality.
Objective: The goal of this study is to define a strategy for test-driven development using software testing criteria, guiding students to improve the quality of their programming activities, and, consequently, academic results.
Method: Initially, we performed a systematic mapping study on TDD in the Computing education context, and a literature review about the usage of test criteria with TDD. Considering the evidences collected, we defined a strategy to integrate the usage of test criteria into TDD. In order to evaluate this strategy, we defined measurements and instruments for data collection. An experimental study was conducted with students to evaluate the strategy and the objective satisfaction.
Results: We defined a strategy to use software testing criteria with TDD for students that are learning how to program in Computing courses. The strategy consists into improving the test set while refactoring the code at TDD, considering control-flow-based structural testing criteria. Morever, we developed a set of tools to gather and analyze data produced by the students while using the strategy in the experimental study. The strategy lead to higher quality programs and test sets, and it was considered useful by the students, providing a guide for the creation of test cases, and helping them to discover errors.
Conclusions: It was possible to observe improvements into the quality of programs and of test sets. The strategy can be replicated, considering different contexts and evaluting more testing criteria, fostering further evaluation of the objective and generalization of the results.

Keywords: test-driven development; software testing; test criteria; education; introduction to programming

Lista de figuras

Lista de tabelas

Lista de acrônimos

- **ACM**: *Association for Computing Machinery*
- **AVA**: *Ambiente Virtual de Aprendizagem* 99
- **BDBComp**: Biblioteca Digital Brasileira de Computação
- **CBO**: *Coupling between object classes* 74
- **CFG**: *Control Flow Graph* 37
- **CONCCEPAR**: Congresso Científico da Região Centro-Ocidental do Paraná 119
- **FIE**: *Frontiers in Education Conference*
- **GFC**: *Grafo de Fluxo de Controle* 105
- **GQM**: *Goal Question Metric* 29
- **ICALT**: *International Conference on Advanced Learning Technologies*
- **ICSE**: *International Conference on Software Engineering*
- **IDE**: *Integrated development environment* 59
- **IEEE**: *Institute of Electrical and Electronics Engineers*
- **ITiCSE**: *Annual Conference on Integrating Technology into Computer Science Education*

- **LCOM**: *Lack of cohesion in methods* 74
- **PPGI**: *Programa de Pós-Graduação em Informática* 91
- **REST**: *Representational State Transfer* 82
- **SBIE**: Simpósio Brasileiro de Informática na Educação 47
- **SIGCSE**: *Technical Symposium on Computer Science Education*
- **TDD**: *Test-Driven Development* 27
- **TDDHQ**: *Test-Driven Development High Quality* 63
- **TDL**: *Test-Driven Learning* 56
- **URL**: *Uniform Resource Locator* 84
- **UTFPR**: Universidade Tecnológica Federal do Paraná 91
- **WMC**: *Weighted methods per class* 74

Lista de fragmentos de código

Sumário

Capítulo 1

Introdução

Em âmbito nacional são propostos diversos tipos de artefatos com intuito de colaborar com o ensino de novatos em programação, tais como: ferramentas, linguagens de programação, metodologias e técnicas de avaliação (AURELIANO; TEDESCO, 2012). Baseado nesta revisão sistemática da literatura desenvolvida por Aureliano e Tedesco (2012) é possível observar que pesquisadores nacionais têm dado foco em ferramentas de software, pois 19 estudos encontrados têm este tema como tópico de pesquisa.

Assim, é possível observar que, em geral, o ensino de programação é baseado em ferramentas ou na sintaxe de alguma linguagem de programação e não em conceitos abstratos da disciplina. Tal situação leva o acadêmico a direcionar sua atenção a questões técnicas (BARBOSA *et al.*, 2008). Desta forma, em diversos casos, mesmo sem assimilar completa e corretamente os conceitos da disciplina, os estudantes são capazes de resolver um algoritmo, entretanto sem a completa compreensão de qual é a lógica empregada (EDWARDS, 2003b). Mediante este fato, educadores da área de Ciência da Computação afirmam que as práticas atuais de ensino de programação não são eficazes: o acadêmico não pensa na solução, mas age na tentativa e erro (EDWARDS, 2003b, 2004a; JANZEN; SAIEDIAN, 2006b; SPACCO *et al.*, 2006).

Para mitigar este problema é proposta a *Reflection in action* (reflexão na ação), que promove a aprendizagem com a compreensão adequada dos problemas (EDWARDS, 2004a), evitando a prática da tentativa e erro (EDWARDS, 2004a; JANZEN; SAIEDIAN, 2006b). Em vez disso, as dúvidas encontradas em atividades de aprendizagem são tratadas com micro-experimentos, cujas hipóteses são cuidadosamente estudadas antes de tentar resolvê-los. Em disciplinas de introdução à programação, técnicas com base em testes de software ou *Test-Driven Development* (TDD) colaboram com a reflexão na ação, propicia

ainda o desenvolvimento de software incremental e colabora com a detecção de defeitos precocemente (EDWARDS, 2004a; JANZEN; SAIEDIAN, 2008; SOUZA *et al.*, 2015).

Por um lado, vários estudos relatam que a exigência de técnicas de teste de software em cursos de programação permitem que os alunos produzam software melhor, evidenciado pela maior cobertura do conjunto de casos de teste e pela detecção de defeitos (SPACCO *et al.*, 2006; SOUZA *et al.*, 2011b; BUFFARDI; EDWARDS, 2013).

O teste de software é o processo de execução de um programa com a intenção de evidenciar defeitos. Desta forma, é um processo que impacta diretamente na confiança de que o software funciona corretamente (MATHUR, 2008, p. 36). A atividade de teste de software, tal como a programação, não é considerada uma atividade trivial (LAHTINEN *et al.*, 2005), entretanto, pode ser utilizado como um guia para o desenvolvimento de software, provendo ao acadêmico uma forma de propor soluções (EDWARDS, 2004a). A utilização do teste de software como ferramenta de ensino é citada como uma forma de incentivar o raciocínio antes de programar (SHAW, 2000), propor uma situação e verificar sua validade (EDWARDS, 2004a). Possibilita ainda a capacidade de promover o cruzamento de casos de teste entre acadêmicos, a fim de verificar a qualidade de programas desenvolvidos ante os casos de teste de outro programador, professores e assistentes (SHAW, 2000).

Por outro lado, existem muitas abordagens utilizando TDD no contexto da educação: desenvolvimento de software para diferentes domínios (e mais motivadoras), tais como Web (SCHAUB, 2009), jogos (ISOMöTTöNEN; LAPPALAINEN, 2012); sistema inteligente de tutoria (HILTON; JANZEN, 2012), mecanismos de avaliação automática (EDWARDS, 2003b, 2004a; THORNTON *et al.*, 2008; BUFFARDI; EDWARDS, 2012), e outros. Entretanto, poucos estudos consideram a integração de ambos. Teste de software muitas vezes é utilizado para avaliação, considerando a cobertura do conjunto de casos de teste (SOUZA *et al.*, 2015; EDWARDS; SHAMS, 2014). No entanto, o uso explícito das técnicas de teste de software não é considerado ao projetar casos de teste, mesmo quando se usa TDD.

Recentemente, alguns estudos têm se concentrado na integração de TDD e testes de software, avaliando as melhorias na qualidade conjunto de teste (cobertura), o tempo necessário para o desenvolvimento e a qualidade do código fonte (CAUSEVIC *et al.*, 2012a, 2013c). Contudo, isso não tem sido investigado no contexto educacional, propondo assim uma estratégia que poderia ser utilizado com sucesso por estudantes.

Considerando este cenário, esta dissertação apresenta uma estratégia para integrar o TDD com técnicas de teste de software com o intuito de melhorar a reflexão na ação utilizando critérios de teste de software em conjunto com o TDD. Desta forma, foi proposta a alteração do ciclo tradicional de desenvolvimento do TDD, adicionando uma atividade com a finalidade de criar casos de teste criteriosamente. Esta atividade é complementar aos casos de teste

criados no início do TDD, evitando grandes mudanças no ciclo tradicional da técnica.

Este trabalho buscou utilizar dos benefícios comprovados cientificamente do TDD e propor uma abordagem com o objetivo de melhorar a qualidade dos casos de testes desenvolvidos com a prática da técnica, impactando na qualidade do software. Ao guiar o acadêmico em como os casos de teste devem ser implementados, é esperado que o mesmo encare os critérios como metas a serem atingidas. Entretanto, espera-se que, com o tempo, tais metas se tornem parte do procedimento comum de desenvolvimento de software, o que reforçará a *reflection in action* colaborando assim com a formação profissional do aluno. Com melhores casos de teste desenvolvidos, o código é testado com maior rigor e desta forma é esperado que o software resultante seja melhor.

O impacto desta mudança e da adoção de critérios estruturais básicos são intencionais, reduzindo a necessidade da introdução de conceitos que poderiam ser um obstáculo para alunos iniciantes. Ao mesmo tempo, espera-se que a abordagem proposta oriente os alunos na criação de melhores conjuntos de caso de teste, evitando abordagens *ad hoc* e melhorando a reflexão na ação.

Para atingir tal objetivo foi necessário atingir as seguintes metas:

1. Identificar as melhores práticas da aplicação de TDD no ensino;
2. Definir estratégias da aplicação de TDD no contexto no ensino;

Com o intuito de atingir as metas proposta, foram conduzidas as seguintes atividades:

1. Desenvolvimento de um mapeamento sistemático com o objetivo de identificar estudos com foco na aplicação de TDD no ensino de programação;
2. Execução e avaliação de experimentos preliminares para a avaliação da estratégia final;
3. Definição das medidas de avaliação dos resultados, da TDD Traking para captura das informações e ainda a modelagem de um projeto experimental a fim de avaliar as hipóteses levantadas;
4. Execução e avaliação da estratégia final.

Para cada uma das atividades supracitadas, obteve-se resultados específicos que corroboram com o objetivo principal desta dissertação. A partir do mapeamento sistemático da literatura, foi observado os artefatos de software e técnicas que são utilizados por pesquisadores no ensino de programação em conjunto com o TDD. Os estudos experimentais preliminares foram utilizados como base para a definição de um *Goal Question Metric* (GQM) e uma ferramenta para captura das informações necessárias para a avaliação dos resultados capturados na execução do projeto experimental, que foi utilizado para a avaliação da estratégia proposta. Os resultados da execução do projeto experimental final foram analisados com base nas medidas definidas e as questões de pesquisa foram respondidas.

O restante da presente dissertação está organizado da seguinte maneira. No Capítulo 2 são apresentados conceitos básicos sobre teste de software e desenvolvimento baseado em testes, bem como é apresentado um mapeamento sistemático da literatura sobre a utilização de TDD em ensino de programação. A definição da estratégia geral, o estudo experimental, o método de avaliação dos resultados e a ferramenta para capturar as informações necessárias para a avaliação são descritos na Capítulo 3. Os resultados dos estudos experimentais executados são descritos na Capítulo 5. A dissertação é encerrada no Capítulo 6 com a sumarização dos resultados e análise dos objetivos traçados.

Capítulo 2

Teste de software e desenvolvimento baseado em testes (TDD)

A relevância do teste de software no desenvolvimento de software é acentuada por ser tipicamente a última etapa com objetivo específico de encontrar defeitos no software. Esta atividade consiste em uma análise dinâmica que promove experimentos controlados em artefatos que compõem o produto com a finalidade de evidenciar defeitos de software (DELAMARO *et al.*, 2007, p. 2).

Já o TDD é uma técnica que orienta o desenvolvimento de software, guiando o praticante a escrever programas seguindo um ciclo de três etapas (BECK, 2002, p. 9). Primeiro, é desenvolvido um caso de teste para a próxima funcionalidade que se deseja desenvolver. Esta etapa é conhecida por *red*, pois estão definidos apenas os casos de teste que falham. Implementar o código para que os testes passem é o objetivo da segunda etapa, constituindo então a etapa *green*. O objetivo da última etapa é propor melhorias no código recém desenvolvido (refatoração), ato que não altera a funcionalidade. O ciclo se inicia novamente por uma nova funcionalidade e assim sucessivamente (BECK, 2002, p. 9).

Uma das motivações deste trabalho é promover a prática do TDD em conjunto com técnicas de teste de software como ferramenta para auxiliar no ensino de programação para iniciantes no contexto de cursos na área de Ciência da Computação. Neste contexto, o TDD corrobora com o desenvolvimento de código testável, visto que, em sua primeira etapa, é necessário que seja proposto um caso de teste que exercite a funcionalidade recém implementada. Todavia, é importante salientar que a técnica em si não garante a testabilidade, mas parte da premissa que, para que o código efetivo seja desenvolvido, é necessário que exista um caso de teste. A testabilidade é o objetivo e o TDD é uma ferramenta que pode

colaborar para este fim. Neste contexto, partindo do pressuposto que o TDD é baseado na construção de testes de unidade, pretende-se adicionar a prática de técnicas de teste em seu ciclo a fim de agregar conceitos de teste de software, elementos de testabilidade do código e fornecendo ferramentas para a localização de defeitos de software.

Desta forma, o intuito deste capítulo é promover a interseção entre as duas técnicas, visto que o TDD não é considerado uma técnica de teste de software e sim uma técnica de *design* de código (BECK, 2002; ANICHE; GEROSA, 2012). Assim, neste capítulo são abordados conceitos básicos de teste de software (Seção 2.1) e desenvolvimento baseado em testes (Seção 2.2), colaborando com a interseção proposta neste trabalho visando a utilização de técnicas de teste de software em conjunto com o TDD. Na Seção 2.3 são discutidas as estratégias que comumente são utilizadas em conjunto com o TDD para o ensino e na Seção 2.4 são apresentadas estratégias que utilizam técnicas de teste de software em conjunto com o TDD.

2.1. Teste de software

O teste de software tem por objetivo principal revelar a presença de defeitos (MYERS, 1979, 2004; MYERS *et al.*, 2013). Mediante tal definição é possível vislumbrar a possibilidade de que um software contenha defeitos que não foram evidenciados pelos testes efetuados. Parafraseando Armour (2000) e a terceira das "cinco ordens da ignorância": "Eu não sei que não sei". Técnicas de teste de software são utilizadas para revelar defeitos que não são conhecidos.

Engano, defeito, falha e erro são conceitos de suma importância e que se fazem necessários para compreender a atividade de teste (DELAMARO *et al.*, 2007, p. 2). Engano (*mistake*) é uma ação humana que introduz um defeito no software. Defeito (*fault*) é a introdução incorreta de instruções, o que pode causar um erro. Erro (*error*) é a diferença entre o valor esperado e o valor resultante. Falha (*failure*) é uma saída incorreta comparada a especificação (ISO, 2010, p. 220, p. 96). Conforme apresentado na Figura 2.1, por inúmeros motivos, o programador, durante a implementação, pode cometer um engano e introduzir um defeito no software, o qual poderá acarretar um erro que consiste na diferença do valor obtido e o valor esperado. Este por sua vez em um dado momento poderá ser exposto ao usuário, configurando uma falha.

Para evidenciar as falhas e erros, utilizam-se casos de teste. Um caso de teste é um conjunto de dados de entrada (valor do domínio de entrada) e de condições de execução desenvolvidos a fim de propor assertivas com o intuito de verificar se o resultado obtido é igual ao esperado (valor do domínio de saída) (ISO, 2010; MYERS, 2004, p. 368).

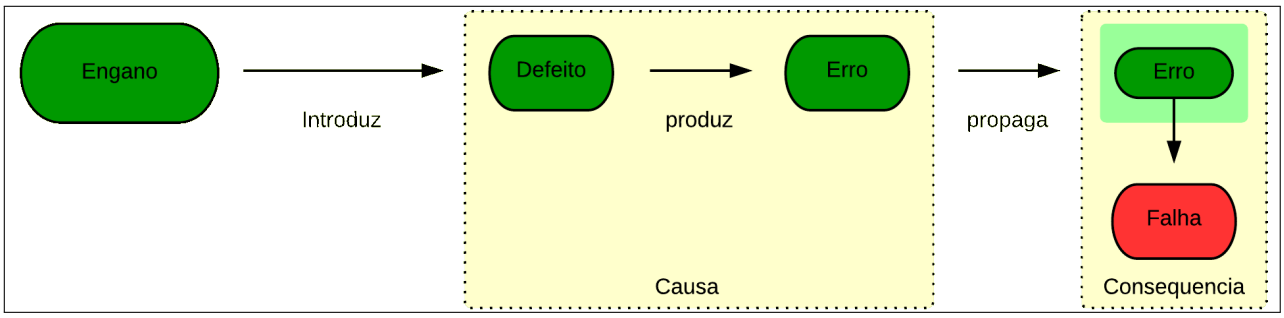

Figura 2.1. Taxonomia de defeito, adaptado de (VINCENZI *et al.*, 2007).

Desta forma, a assertiva proposta constitui um oráculo, o qual aponta o sucesso ou falha do caso de teste (BARESI; YOUNG, 2001), sendo assim possível evidenciar a descoberta de um defeito (MYERS, 2004). Deve-se salientar que, se o caso de teste falhar, possivelmente um defeito foi evidenciado e, desta forma, é necessário corrigi-lo para que o teste seja satisfeito.

2.1.1. Fases de teste de software

A fim de que o teste de software esteja presente em todas as etapas do processo de desenvolvimento do produto de software, esta atividade é tradicionalmente dividida em quatro momentos. São eles: unidade, integração, sistema e regressão.

A fase de desenvolvimento de testes de unidade é baseada na menor unidade do software, tal como é demonstrado na Figura 2.2. Neste momento, o foco principal é identificar defeitos lógicos e falhas (MALDONADO *et al.*, 1998; DELAMARO *et al.*, 2007; ANICHE, 2013; HARROLD; ROTHERMEL, 1994).

A norma ISO/IEC/IEEE 24765:2010 define que a unidade é um elemento testável que possa ser separado logicamente e que não possa ser subdividido em outros componentes (ISO, 2010, p. 148). Desta forma, para o paradigma orientado a objetos, alguns pesquisadores definem o método como a menor unidade entretanto outros pesquisadores, adotam a classe como menor unidade. No contexto da programação procedimental, são considerados comumente como unidade os procedimentos, funções e sub-rotinas.

Casos de teste que referenciem dois ou mais métodos de uma classe ou na programação procedimental dois ou mais procedimentos podem caracterizar teste de integração (VINCENZI, 2004; HARROLD; ROTHERMEL, 1994), tal como exemplificado na Figura 2.2. Neste contexto, a utilização de mais de uma unidade com o intuito de verificar a compatibilidade entre unidades configura a fase de teste de integração. Ela tem o objetivo de identificar defeitos associados à troca de informações entre as unidades. Por exemplo, quando em um caso de teste um objeto necessita de um retorno de um método de outra classe, é exercitado um conjunto de requisitos de teste que integra funcionalidade entre classes (LEUNG; WHITE,

1990).

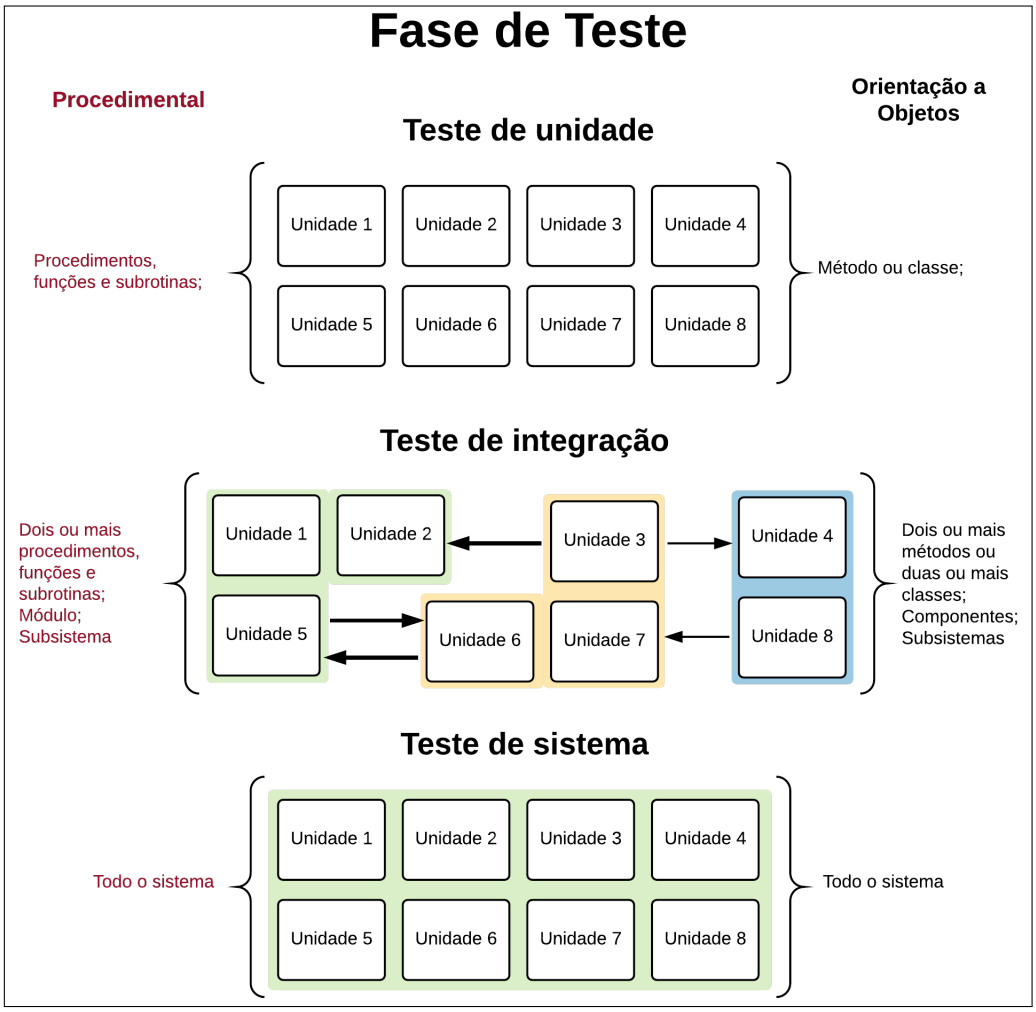

Figura 2.2. Ilustração das fases de unidade, integração e sistema, adaptado de (BINDER, 1999).

A fase de teste de sistema é realizada após a integração dos módulos e tem como objetivo comparar características especificadas no projeto com as encontradas no produto. Tal etapa de testes também tem por responsabilidade a verificação de requisitos funcionais e também os não funcionais (MALDONADO *et al.*, 1998; DELAMARO *et al.*, 2007). Nesta etapa destaca-se o teste de aceitação, que tem por objetivo verificar se o software está de fato concluído, suas funcionalidades e usabilidade. Normalmente é uma atividade onde são envolvidos usuários finais (DAVIS; VENKATESH, 2004).

Após a promoção de alterações ou adição de novas funcionalidades inicia-se a fase de regressão, ou seja, quando é necessário verificar se as alterações promovidas funcionam tal como especificado e se não foram introduzidos defeitos pelas modificações realizadas (MATHUR, 2008, p. 336).

Embora exista a ênfase em execução de testes a cada fase apenas, na prática, é

possível executar casos de teste das diferentes fases a qualquer momento do desenvolvimento. Desta forma, por exemplo, o teste de regressão pode ser visto como ortogonal às demais fases, permitindo a execução de conjuntos de casos de teste, independente da fase originária, a qualquer instante do desenvolvimento (MATHUR, 2008, p. 336).

2.1.2. Técnicas e critérios de teste de software

O desenvolvimento de conjuntos de casos de teste tem como objetivo buscar a evidência de defeitos que não se tem o conhecimento que existam (DELAMARO *et al.*, 2007, p. 5). Desta forma, são definidas técnicas para conduzir o testador na atividade de desenvolvimento de casos de teste, fornecendo uma abordagem padronizada e fundamentada teoricamente (DELAMARO *et al.*, 2007, p. 6).

O domínio de entrada e saída são todas as possibilidades de dados que podem ser computadas por um determinado programa (MYERS, 2004, p. 43). Myers (2004, p. 43) salienta que a melhor estratégia é desenvolver conjuntos de casos de testes o mais completo possível. Os critérios de teste de software têm por principal objetivo a diminuição do domínio de entrada a fim de tornar a atividade de teste viável, aumentando a possibilidade de encontrar defeitos (AMMANN; OFFUTT, 2008, p. 5).

Tal como apresentado na Figura 2.3, as técnicas de teste de software são definidas a partir da fonte de informação que será utilizada. Para cada técnica de teste são definidos critérios de teste, os quais utilizam a fonte de informação para derivar os requisitos de teste (DURELLI *et al.*, 2013; MALDONADO *et al.*, 1998; DELAMARO *et al.*, 2007). Dados os requisitos de teste, os casos de teste devem ser propostos ou gerados a fim de que os requisitos sejam satisfeitos. Mediante este fato, uma informação importante é qual a porcentagem de requisitos de teste cobertos pelos casos de teste. A partir da relação entre o proposto e o cumprido, é possível chegar a uma porcentagem de cobertura, medida essa denominada cobertura de teste (AMMANN; OFFUTT, 2008, p. 18).

As principais técnicas de teste de software são: funcional, estrutural e baseada em defeitos (DURELLI *et al.*, 2013; MALDONADO *et al.*, 1998; DELAMARO *et al.*, 2007). Para cada uma das técnicas são definidos critérios de teste (LI *et al.*, 2009), os quais podem ser aplicados em diversos artefatos de software (fonte de informação), incluindo especificações de requisitos, notações de modelagem e código fonte (MALDONADO *et al.*, 1998; DELAMARO *et al.*, 2007), e destes, tal como supracitado, derivados os requisitos de teste.

A técnica funcional é baseada na especificação de requisitos de software e sendo assim chamada de técnica de caixa preta, pois não se baseia no código para a definição dos requisitos de teste (MALDONADO *et al.*, 1998; DELAMARO *et al.*, 2007). Esta técnica é eficaz para

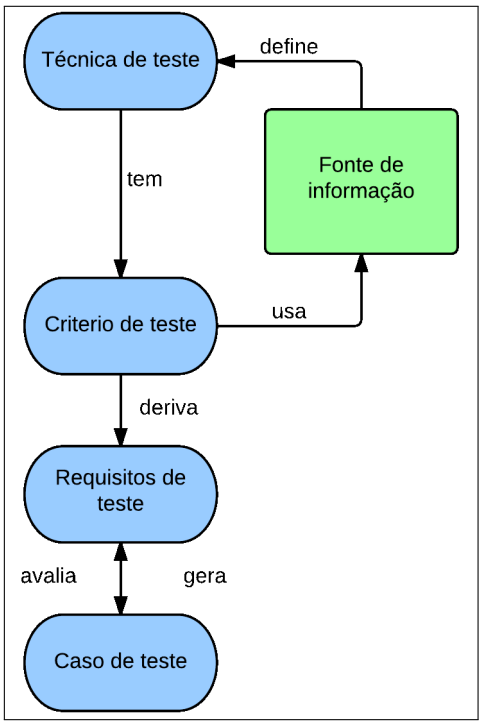

Figura 2.3. Relação entre as principais técnicas de teste de software, adaptado de (MALDONADO, 1991).

determinar se os requisitos de projeto foram implementados tal como especificado (MYERS, 2004). Algumas das técnicas funcionais mais utilizadas são discutidas a seguir.

O critério com base no particionamento em classes de equivalência tem por objetivo primário representar o domínio de entrada em pequenos grupos. Desta forma, divide-se determinado domínio em classes, as quais tem por objetivo revelar defeitos para dados de entrada equivalentes. Tais classes são ainda divididas em entradas e saídas tanto válidas quanto inválidas. Essa divisão visa confrontar a especificação com base nestes conjuntos. Para o desenvolvimento dos casos de teste deve ser considerado um ou mais valores das classes de equivalência, sendo que para as entradas válidas de cada classe deve ser considerada ao menos um caso de teste que satisfaça todos os requisitos das classes válidas. Para as entradas inválidas, devem ser desenvolvidos casos de testes que cubram uma entrada inválida por vez. Vale salientar que a divisão dos conjuntos é uma atividade do testador, o que torna a técnica dependente da habilidade e experiência deste profissional (MYERS, 2004, p. 52).

Outro critério da técnica funcional é Análise de Valor Limite. Ele utiliza como base o critério de particionamento em classes de equivalência e determina que, ao identificar as classes de equivalência, o testador deve considerar os valores limítrofes de cada classe e os valores próximos ao limite. Ou seja, são considerados valores máximos, mínimos, logo abaixo do máximo e logo acima do mínimo (MYERS, 2004, p. 59). Segundo Myers (2004), os valores limítrofes ou ainda os valores próximos são mais efetivos em revelar defeitos de software se

comparados aos outros valores da classe de equivalência.

Na técnica estrutural, os critérios e requisitos são derivados essencialmente a partir da estrutura do código fonte ou modelos do software. A partir do código fonte do programa é possível elaborar representações, como, grafo de fluxo de controle ou grafo de programa. A partir dos grafos são estabelecidos diversos critérios, os quais guiam o testador na derivação dos requisitos de teste e posteriormente no desenvolvimento dos conjuntos de testes. Por exemplo, com as representações é possível verificar os caminhos pelos quais os dados transitam, são comparados ou atribuídos. O fato dos requisitos de teste estarem ligados diretamente a estrutura de código fonte elimina questões de interpretação ou subjetividade, tal como na técnica funcional (DELAMARO *et al.*, 2007; MYERS, 2004).

Bem como apresentado para os critérios funcionais, existem critérios estruturais e dentre esses é importante destacar o critério Todos os Caminhos básicos e completos. No todos os caminhos básicos são considerados todos os nós, do inicial até o final, exceto os laços de repetição, este laço é definido pela complexidade ciclomática (critério de McCabe). O critério "Todos os caminhos completos" considera todos os caminhos de um *Control Flow Graph* (CFG). Neste contexto, no caso de um laço de repetição, o caminho que executa apenas uma repetição é diferente do caminho que executa duas vezes. Desta forma, este critério é o mais forte dentre os critérios estruturais, pois explora todos os caminhos possíveis de um programa (MYERS, 2004). Entretanto, mesmo para exemplos triviais é extremamente custoso, tanto para a definição dos casos de teste quanto para a execução. Além deste problema, programas podem conter linhas de código inacessíveis, o que impossibilita a execução dos critérios estruturais (MYERS, 2004).

Desta forma, mediante o fato da execução de Todos os Caminhos não ser uma meta acessível, estabelecem-se diversos critérios mais fracos dentre os quais pode-se citar: Todos Nós e Todas Arestas. Para estes, considera-se o grafo de fluxo de controle, formado por comandos e desvios. O agrupamento de vários comandos em sequência forma um nó, ou seja, o nó caracteriza-se por uma sequência de comandos sem desvios. Os desvios caracterizam-se por comandos condicionais e laços e repetição, pelos quais o fluxo do programa pode sofrer variações mediante os valores de comparação (MALDONADO *et al.*, 1998).

O critério Todos Nós que tem o objetivo derivar casos de teste que exercitem todos os nós do CFG. Em outras palavras, se um programa é composto por 15 nós, é necessário que sejam criados casos de teste até que todos os nós sejam executados ao menos uma vez. É possível que apenas um caso de teste seja suficiente para cobrir todos os nós do grafo (MYERS, 2004).

Como o critério Todos Nós, o Todas Arestas tem por objetivo que todos os nós sejam alcançados ao menos uma vez e, em complemento, todas as arestas do grafo também devem

ser executadas ao menos uma vez (MYERS, 2004).

A fim de demonstrar que existe uma hierarquia de inclusão de critérios estruturais, Rapps e Weyuker (1985, 1982) classificaram os critérios (Figura 2.4). A relação de inclusão se dá mediante o conjunto de casos de testes adequados a dois critérios para um programa. Desta forma, se o critério b engloba todos os casos de teste do critério a, logo o critério b inclui o critério a (VINCENZI, 2004).

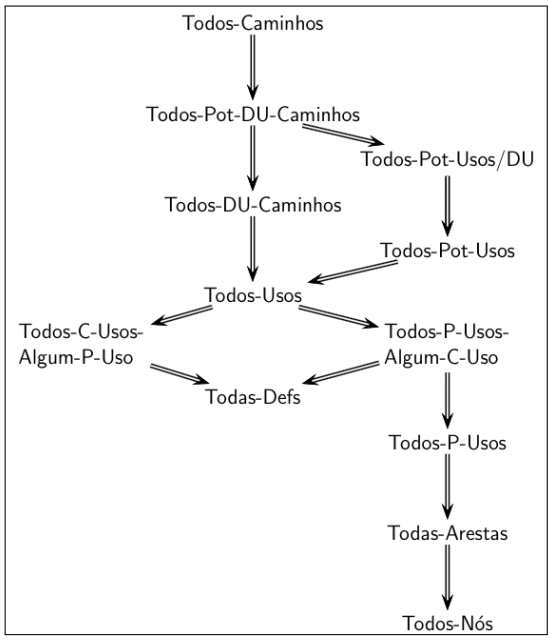

Figura 2.4. Relação de inclusão de critérios estruturais (MALDONADO, 1991).

Na técnica Baseado em Defeitos, os critérios e requisitos de teste são oriundos do conhecimento sobre defeitos típicos cometidos no processo de desenvolvimento de software. Esta teoria foi definida como hipótese do programador competente, na qual é definido que programadores experientes escrevem soluções corretas ou aproximadas do correto (DEMILLO *et al.*, 1978; BUDD, 1980). Desta forma, acredita-se que defeitos são inseridos nos programas por pequenos equívocos (DEMILLO *et al.*, 1978; BUDD, 1980; VINCENZI, 2004). DeMillo *et al.* (1978) também propuseram que defeitos complexos são relacionados a defeitos simples, caracterizando o efeito de acoplamento (*coupling effect*). Neste contexto, estudos comprovam que conjuntos de casos de teste que revelam defeitos simples são também capazes de revelar defeitos complexos (ACREE *et al.*, 1979; BUDD, 1980; VINCENZI, 2004).

Com base na teoria proposta por DeMillo *et al.* (1978), defeitos são inseridos nas aplicações mediante a pequenos desvios sintáticos, os quais alteram a semântica do programa, o que pode o levar a um comportamento incorreto (VINCENZI, 2004). Neste contexto é proposta Análise de mutantes, que é baseada na alteração do programa original a fim de introduzir pequenas alterações com o objetivo de alterar a semântica do programa (MALDONADO

et al., 1998; SHELTON *et al.*, 2012). A introdução destas falhas é ditada pelos operadores de mutação, que são "tipos" de defeitos que serão introduzidos, os quais são a nível de unidade (SHELTON *et al.*, 2012).

Os mutantes são os programas alterados, como se os defeitos estivessem sendo inseridos no programa original. Desta forma, o testador deve escolher casos de teste que demonstrem a diferença entre o programa original e o mutante (DELAMARO *et al.*, 2007). Existem também os mutantes equivalentes, que são os que não podem ser mortos, pois se comportam tal como o programa original. Uma métrica do teste de mutação é o escore de mutação (*Mutation Score*), que é a relação entre a quantidade de mutantes gerados e a quantidade de mutantes mortos menos a quantidade de mutantes equivalentes (DELAMARO *et al.*, 2007; MATHUR, 2008).

A análise de mutantes também é utilizada por diversos artigos científicos para medir a qualidade dos casos de teste de um projeto. Quando um defeito é inserido neste cenário, é esperado que os casos de teste o evidenciem. Se o mutante não for "morto", os casos de teste não detectaram o defeito inserido, ou seja não foram capazes de demonstrar a falha (DELAMARO *et al.*, 2007; INOZEMTSEVA; HOLMES, 2014).

2.1.3. Técnicas de desenvolvimento de casos de teste

O desenvolvimento de casos de teste pode acontecer a qualquer momento durante o desenvolvimento do projeto. Entretanto dois momentos são amplamente discutidos: o *test-first* e *test-last* (MYERS, 2004). No *test-first* o caso de teste é escrito (proposto) antes mesmo do código de implementação e no *test-last* o desenvolvimento dos casos de teste é praticado após a solução devidamente implementada.

Desta forma, no *test-first* é possível que o programador receba os casos de teste prontos e, a partir deles, implementar a solução. Desta forma, o conjunto de casos de teste é utilizado como um arcabouço para a implementação. Neste cenário, o código de teste vai influenciar o desenho da aplicação. Esta prática será discutida na próxima seção.

2.2. Desenvolvimento baseado em testes (TDD)

A prática do TDD (Figura 2.5) dá-se pela execução de um ciclo em que o programador escreve um pequeno teste, o qual em imediata execução deve falhar. Em seguida, efetua-se a implementação da solução para que o teste anteriormente escrito passe. O último passo é a refatoração, que consiste em simplificar ou propor melhorias para o código recém escrito. Tal processo é a execução típica do TDD, conhecido também Vermelho/Verde/Refatorar (BECK, 2002; MUNIR *et al.*, 2014; SHELTON *et al.*, 2012). Este ciclo pode ser observado na Figura 2.6,

onde são apresentados o fluxo do TDD em comparação ao *test-last*. No *test-last*, diferente do TDD, os casos de teste são escritos após o código.

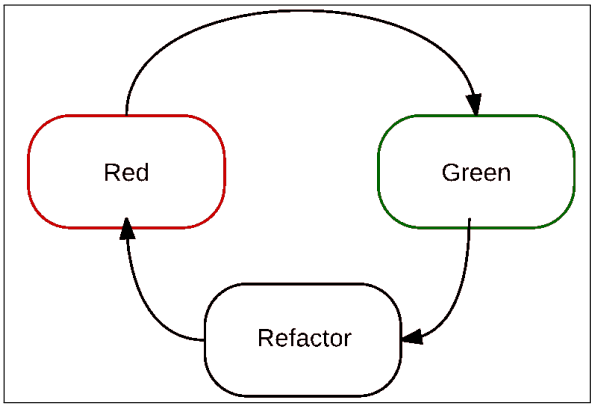

Figura 2.5. Fluxo do TDD tradicional.

O TDD estabelece que o código de teste é escrito antes da solução efetiva, o que o classifica também como uma técnica de *test-first*. Cabe destacar que *test-first* não é sinônimo de TDD, pois casos de teste podem ser propostos por uma equipe distinta e encaminhado para a equipe de desenvolvimento, fato que caracteriza que os casos de teste são escritos antes mesmo da implementação, mas não guiam (não interferem diretamente) no *design* das classes e métodos (GASPAR; LANGEVIN, 2007; HEINONEN *et al.*, 2013). Desta forma, é importante salientar que o TDD não é visto como uma técnica de teste de software, mas como técnica de *design* de código, o qual corrobora com código mais coeso e menos acoplado, visto que o TDD tem uma etapa para melhorar o código recém desenvolvido (Figura 2.5) (ANICHE; GEROSA, 2012; BECK, 2002; EDWARDS, 2003a).

A primeira etapa do TDD propicia o desenvolvimento de casos de teste e tem como objetivo garantir que atenda às especificações funcionais para qual a unidade foi escrita. Não é comum considerar a aplicação de critérios de teste de software em conjunto com TDD. Entretanto, estudos com este viés têm sido desenvolvidos (SHELTON *et al.*, 2012; CAUSEVIC *et al.*, 2013a).

Na prática do TDD, a segunda etapa propicia o desenvolvimento efetivo de código da aplicação, ou seja, o código que foi esboçado na primeira etapa é implementado. Na última etapa (terceira) é aplicada a refatoração do código, em que são propostas melhorias no código recém desenvolvido (BECK, 2002; ANICHE, 2013). Nestas etapas também não se observa a utilização de critérios de teste.

Dessa forma, observa-se que, no TDD, os casos de teste geralmente não têm foco na utilização de técnicas de teste de software tal como cobertura de critérios de teste (BECK, 2002; ANICHE; GEROSA, 2012; EDWARDS, 2003a). Na prática, observa-se que o aumento

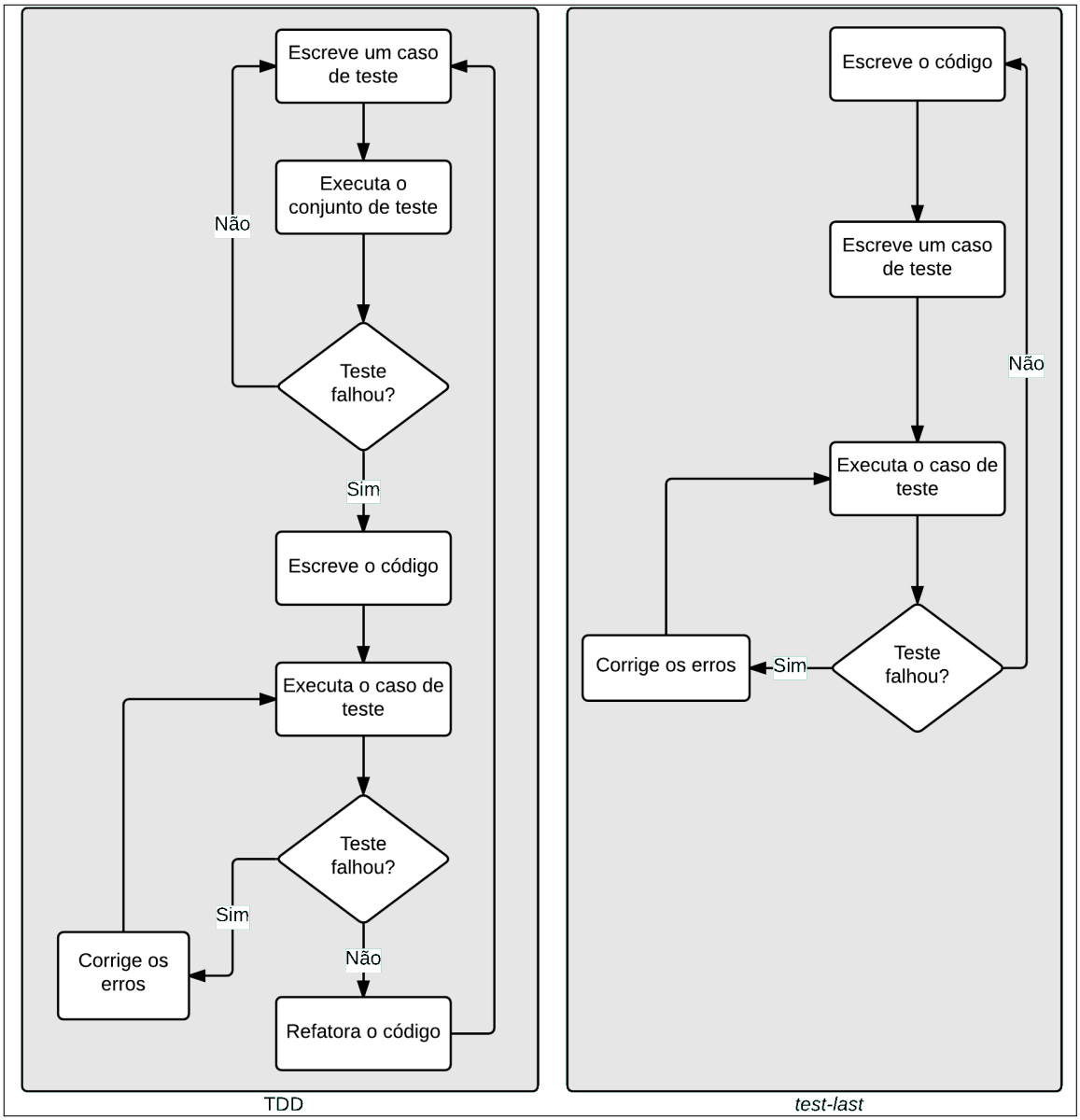

Figura 2.6. Comparação do TDD com *test-last*, adaptado de (MUNIR *et al.*, 2014).

da quantidade de casos de teste ocasiona o aumento da cobertura de forma acidental ao invés de intencional.

Ao considerar critérios durante a prática do TDD, o aumento da cobertura pode impactar na qualidade do conjunto de casos de teste e potencializar outras etapas de teste de software. Cabe observar que quantidade de casos de teste não é o foco, mas sim que os códigos sejam desenvolvidos conscientemente e que a qualidade seja impactada. Deste modo, torna-se viável a utilização de ferramentas que analisam cobertura de critérios simples, tais como: todos nós e todas arestas (SHELTON *et al.*, 2012).

Além das questões básicas sobre TDD, é necessário caracterizar outros aspectos e impactos da técnica. Diversos estudos são desenvolvidos para comparar TDD com *test-*

last. Neste contexto, Munir *et al.* (2014) desenvolveram uma revisão sistemática, a qual analisou 41 estudos publicados em reconhecidos eventos e periódicos entre os anos 2000 e 2011, dos quais, 61% são experimentos, 32% são estudos de caso e 7% questionários (*survey*). Uma das contribuições da pesquisa foi identificar quais variáveis são utilizadas em estudos experimentais que visam comparar resultados entre TDD e *Test-last Development*. Tais variáveis são indicadores para comparar as duas práticas. Desta forma, foram definidas 8 categorias, as quais são apresentadas na Figura 2.7: produtividade, qualidade externa, opinião do programador, qualidade interna do código, tempo de desenvolvimento, conformidade, opinião do desenvolvedor, tamanho e robustez.

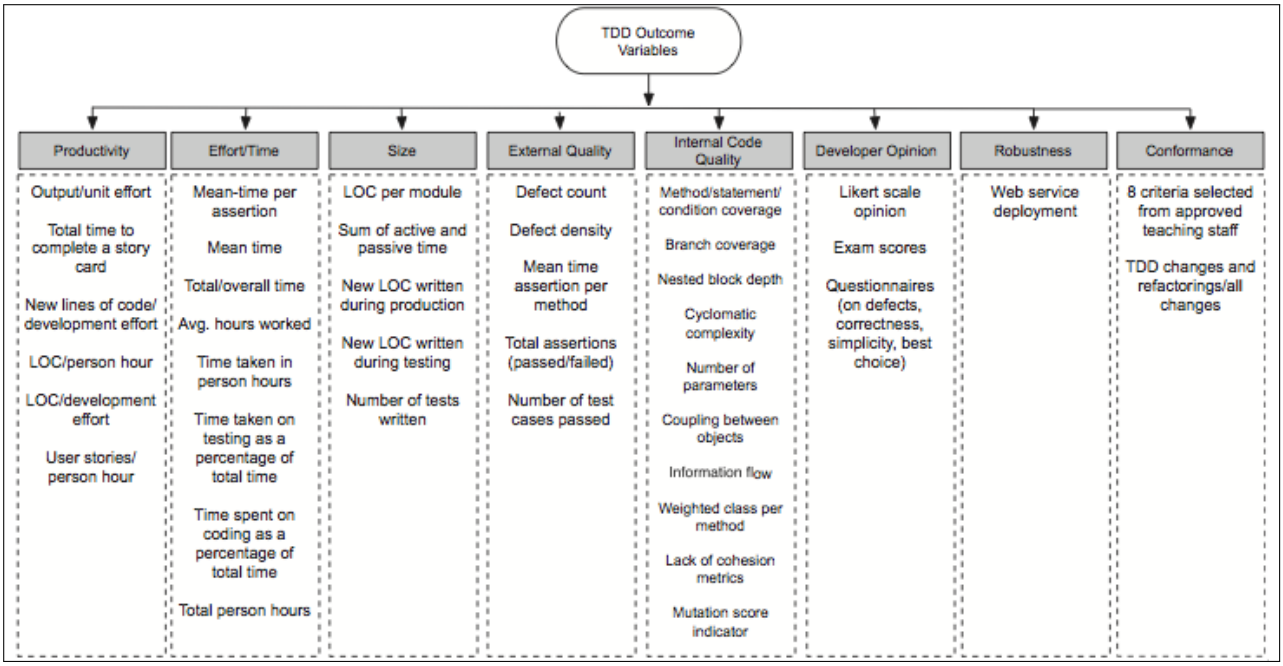

Figura 2.7. Categorias, variáveis e métricas utilizados para avaliar o TDD (MUNIR *et al.*, 2014).

Em geral, a produtividade é medida por uma relação baseada em atividades definidas (especificações, *user stories*), a partir das quais esperam-se artefatos de software resultantes. Em linhas gerais estes artefatos podem ser mensurados baseado em suas características. Para código por exemplo: quantidade de linhas de código por mês, número de casos de teste, medida de esforço de desenvolvimento, etc (MUNIR *et al.*, 2014). Esforço e tamanho são definidas por alguns pesquisadores como subcategorias de produtividades (MUNIR *et al.*, 2014).

Qualidade externa faz referência à qualidade de entrega do produto, o que tem por indicador principalmente defeitos e quantidade de casos de teste desenvolvidos para tal artefato. A qualidade externa tem por objetivo avaliar a estrutura do programa, código e cobertura de teste, níveis de acoplamento e de coesão, os quais afetam diretamente a produtividade e

manutenibilidade do código (MUNIR *et al.*, 2014).

A avaliação do TDD é geralmente alicerçada sobre atributos internos de qualidade do produto, observando-se métricas tais como: complexidade ciclomática do código (*Cyclomatic Complexity*), cobertura do teste, acoplamento entre os objetos, baixa coesão, etc (MUNIR *et al.*, 2014). O TDD geralmente é associado a corroborar a com diminuição do acoplamento e aumentar a coesão, atuando como ferramenta para o desenvolvimento do código orientado a objetos (ANICHE; GEROSA, 2012; BECK, 2002; EDWARDS, 2003a; ASTELS, 2003).

Para a categoria de robustez, foi investigado um estudo que abordou o uso do TDD a fim de colaborar com a resolução de problemas tal como entrada de dados inválidos (MUNIR *et al.*, 2014). Foi definida apenas uma métrica (*Web service deployment*), a qual faz referência especificamente ao objeto de estudo da pesquisa abordada (MUNIR *et al.*, 2014).

Na categoria conformidade, observa-se a correlação de prática do TDD com base no seguinte fluxo de desenvolvimento: 1) na etapa *red* só é permitido alterar métodos que são referenciados por casos de teste que falharam; 2) novos métodos só podem ser incluídos se forem referenciados por casos de teste; 3) a refatoração só pode alterar a estrutura do código e não seu comportamento (MUNIR *et al.*, 2014).

A categoria de opinião do desenvolvedor faz referência ao sentimento do participante quanto a um dado específico, referente a pesquisa em questão. Nesta categoria, pode ser observada a opinião do desenvolvedor quanto a uma prática proposta ou quanto a seu conhecimento sobre determinado tópico (MUNIR *et al.*, 2014).

Em linhas gerais, a pesquisa desenvolvida por Munir *et al.* (2014) demonstra as variáveis analisadas em estudos com o intuito de investigar a prática do TDD. A partir das variáveis propostas, é possível propor projetos experimentais baseados neste conjunto, de forma a padronizar os dados relevantes em um projeto em que o TDD é colocado a prova.

Tal como pode ser observado no estudo Munir *et al.* (2014) o TDD é utilizado em conjunto com outras técnicas e ferramentas. Desta forma, foi desenvolvida uma revisão sistemática da literatura com o objetivo de verificar quais técnicas e ferramentas são utilizadas em conjunto com o TDD no contexto do ensino de programação em cursos de computação.

Kollanus (2010) desenvolveu uma revisão sistemática da literatura com a finalidade de verificar se existem evidências empíricas quanto aos benefícios atribuídos ao TDD. Desta forma, foram selecionados 40 artigos para caracterizar a questão de pesquisa deste estudo, a qual foi dividida em três categorias de evidências para a discussão: qualidade externa, qualidade interna e produtividade. Dos estudos selecionados, 22 avaliam a qualidade externa, neste quesito 16 estudos relataram que o TDD corroborou com o aumento da qualidade, 5 estudos afirmaram que não foi verificado diferença significativa e apenas um estudo afirmou que o TDD impactou negativamente na qualidade externa. Em relação a qualidade interna, a

quantidade de estudos que relatam impacto positivo foi menor, 6 estudos relataram aumento de qualidade interna, 9 estudos afirmam que não ouve impacto significativo e apenas um estudo relata que a técnica proposta (TDD) teve impacto negativo. No referente a produtividade do TDD, 5 estudos apontaram que foram evidenciados resultados positivos, 7 estudos não observaram diferença significativa e 11 estudos afirmaram que o TDD não teve um bom desempenho se comparado com outra técnica.

Com base nos resultados os pesquisadores definiram seu parecer quanto as questões definidas. Referente a qualidade externa observou-se uma fraca evidência de melhora, quanto a qualidade interna, foi observado que existe uma pequena evidência de melhora e com relação a produtividade foi identificado a evidência moderada que o TDD impacta negativamente (KOLLANUS, 2010).

2.3. TDD em ensino de programação

A prática de TDD no ensino visa colaborar com a evolução do acadêmico em disciplinas de introdução a programação. O TDD é aplicável a pequenos projetos com indivíduos que estão iniciando na programação e ainda colabora para que o estudante sempre tenha uma versão executável do código (EDWARDS, 2003a). Este mapeamento sistemático foi realizado com a finalidade de coletar características técnicas e pedagógicas de estudos que exploram a aplicação do desenvolvimento guiado por teste, permitindo a seleção de métodos para uso TDD em ensino e a identificação de possíveis dificuldades para a implantação da técnica com critérios de teste de software.

Este mapeamento será utilizado como meio de identificar, avaliar e interpretar pesquisas disponíveis e relevantes relacionadas as questões de pesquisa que serão descritas. A partir das questões serão coletadas as evidências existentes a fim de identificar eventuais lacunas em pesquisa ou ainda fornecer estrutura para apoiar novas investigações (KITCHENHAM; CHARTERS, 2007), proporcionando benefícios tais como a redução de possíveis tendências na seleção de pesquisas e replicação da pesquisa (KITCHENHAM *et al.*, 2004; KITCHENHAM; CHARTERS, 2007; BIOLCHINI *et al.*, 2005; PETERSEN *et al.*, 2008).

2.3.1. Questão de pesquisa

Um mapeamento sistemático é baseado em um protocolo previamente estabelecido (BIOLCHINI *et al.*, 2005; KITCHENHAM *et al.*, 2004), que é definido inicialmente pelas questões de pesquisa e pelos métodos que serão empregados para conduzir o mapeamento (KITCHENHAM *et al.*, 2004).

Desta forma foram definidos dois objetivos da pesquisa: 1) identificação de métodos de aplicação e avaliação da prática do TDD no ambiente acadêmico; 2) identificação de dificuldades e benefícios encontradas na implantação da prática do TDD em ambiente acadêmico. Mediante tais objetivos foi definida a questão primária:

- Questão Primária: Quais são as técnicas utilizadas em conjunto com a prática do desenvolvimento baseado em testes para alunos que estão iniciando no desenvolvimento de software?

Além da intenção de identificar, de maneira geral, quais técnicas são empregadas em conjunto com a prática do TDD, esta revisão visou também identificar estudos que utilizam técnicas de teste de software em conjunto com a aplicação do desenvolvimento guiado por testes no ensino. Tais temas são tratados pelas questões secundárias:

- Questão Secundária 1: Quais técnicas de teste de software são utilizadas em conjunto com a prática do desenvolvimento baseado em testes para alunos que estão iniciando em programação em cursos de ciência da computação?
- Questão Secundária 2: Quais as dificuldades mencionadas na aplicação do desenvolvimento baseado em testes para alunos que estão iniciando no desenvolvimento de software?
- Questão Secundária 3: Quais os benefícios promovidos pela aplicação do desenvolvimento baseado em testes para alunos que estão iniciando no desenvolvimento de software?

Considerando a questão primária apresentada, foi caracterizada a população e a intervenção dos estudos a serem avaliados, conforme recomendado por Kitchenham e Charters (2007). Com o foco na população definida, foram observadas as técnicas aplicadas pelos estudos em conjunto com o TDD. A partir destes estudos foram observados seus resultados e extraídos os dados. Neste contexto foi definida como população disciplinas introdutórias de programação que apliquem o *Test-Driven Development* como ferramenta de ensino (intervenção).

Com o foco nas disciplinas introdutórias de programação, essencialmente em virtude da necessidade de observar como acadêmicos que estão iniciando no desenvolvimento de software, buscou-se identificar características de como o TDD tem sido utilizado como ferramenta pedagógica de ensino. Desta forma, o foco foi na identificação das técnicas (tais como: *pair-programming*, técnicas de teste de software, ferramentas de avaliação automática, entre outros) e particularidades de aplicação e avaliação dos resultados do desenvolvimento baseado em testes em âmbito acadêmico.

2.3.2. Seleção de fontes

Na Tabela 2.1 estão definidas as conferências e revistas que foram alvo para a localização dos estudos com foco em informática e educação, observando-se o Qualis na área de Ciência da Computação. Estas atividades foram desenvolvidas sob a orientação de um especialista.

Mediante a análise das conferências e revistas, foram identificados os mecanismos que as indexam. Desta forma, ACM Digital Library, IEEE Explore e ScienceDirect foram selecionados inicialmente. Além desses, o mecanismo de indexação Scopus foi incluído pois engloba estudos da ACM e IEEE. Com o intuito de localizar estudos publicados no SBIE, foi efetuada uma busca no Google Scholar, com uma restrição quanto ao endereço Web do BDBComp (http://www.lbd.dcc.ufmg.br/bdbcomp/) para que fossem localizados apenas estudos desta conferência.

Para a busca nas bases digitais definidas, foram selecionadas palavras-chave, as quais englobam as questões de pesquisa definidas. Inicialmente foi definida a palavra-chave *Test driven development* e suas possíveis variações: *test-driven development*, TDD e *test-first*. Apesar de *test-first* não representar exatamente a prática do TDD, alguns estudos utilizam esta palavra-chave. Para representar a população da intervenção, definiu-se *introduction to programming* (introdução a programação) e as siglas: CS0 (*Introduction to Computer Science*); CS1 (*Introduction to programming and computation*) e CS2 (*Introduction to data structures*), que são nomenclaturas geralmente utilizadas em países de língua inglesa.

Definidos os termos, foi elaborada uma expressão de busca genérica e a partir desta foram derivadas as demais variações que foram utilizadas para a pesquisa nas bases selecionadas. A expressão genérica é apresentada no Código-fonte 2.1.

Tabela 2.1. Conferências e revistas com foco em Computação

Sigla	**Nome**	**Qualis-CC**	**Base**
ICSE	*International Conference on Software Engineering*	A1	ACM, IEEE
SIGCSE	*Technical Symposium on Computer Science Education*	A2	ACM
FIE	*Frontiers in Education Conference*	B1	IEEE
ITiCSE	*Annual Conference on Integrating Technology into Computer Science Education*	B1	ACM
ICALT	*International Conference on Advanced Learning Technologies*	B1	IEEE
CSEE&T	Conference on Software Engineering Education and Training	B2	IEEE
SBIE	Simpósio Brasileiro de Informática na Educação	B2	BDBComp
-	IEEE Transactions on Software Engineering	A1	IEEE
-	IEEE Transactions on Education	B1	IEEE
-	Computing and Education	A1	ScienceDirect

```
("test driven development" OR  "test-driven development" OR TDD OR "test-first")
AND
(CS0 OR CS1 OR CS2 OR "Introduction to programming")
```

Código-fonte 2.1. Expressão de busca genérica.

Para a execução da pesquisa na ACM Digital Library, IEEE Explore, Scopus e ScienceDirect, a busca foi restrita a título, resumo e palavras-chaves.

Para alguns mecanismos de busca, devido a limitações de quantidade de caracteres e combinações de palavras-chave (termos), foi necessário que a expressão fosse seccionada, de forma a ser executada em sua totalidade. Para a completa execução da pesquisa, optou-se por dividir a população sendo elaboradas quatro derivações da consulta completa, tal como exemplificado no Código-fonte 2.2, a qual considera CS0 como população.

```
( Title:"test driven development" OR Abstract:"test driven development" OR
    Keywords:"test driven development" OR Title:"test-driven development" OR
    Abstract:"test-driven development" OR Keywords:"test-driven development" OR
    Title:"TDD" OR Abstract:"TDD" OR Keywords:"TDD" OR
    Title:"test-first" OR Abstract:"test-first" OR Keywords:"test-first" OR
    Title:"test driven design" OR Abstract:"test driven design" OR
    Keywords:"test driven design" )
AND (Title:CS0 OR Abstract:CS0 OR Keywords:CS0)
```

Código-fonte 2.2. Expressão de busca seccionada utilizada na ACM.

Por final, com o intuito de localizar estudos publicados no Simpósio Brasileiro de Informática na Educação (SBIE), foi executada a busca genérica tal como definida, traduzindo os termos para o Português. Entretanto, para esta a não foram localizados estudos usando a expressão definida inicialmente. Desta forma, a expressão foi reestruturada. Todas as palavras-chave foram mantidas, contudo, retirada a obrigatoriedade de que o estudo fizesse referência TDD e educação em Ciência da Computação, desta forma, ignorando se o estudo fazia referência explícita aos dois termos no mesmo estudo, atentando-se a isso apenas ao aplicar os critérios de seleção na próxima etapa do mapeamento sistemático. Foi então elaborada a expressão disposta no Código-fonte 2.3.

```
(    "test driven development" OR "test-driven development" OR TDD
    OR "test-first" OR "desenvolvimento guiado por testes" OR CS0
    OR CS1 OR CS2 OR "Introduction to programming" OR "algoritmo"
    OR "introdução a programação" OR "Introdução a ciência da computação"
    OR "avaliação automática" OR algoritmos OR "avaliador automático"
    OR "programação")
```

Código-fonte 2.3. Expressão de busca utilizada no Google Scholar para estudos do SBIE.

Mediante as execuções nas bases supracitadas, foram identificados 50 estudos que foram analisados nesta revisão, tal como disposto na próxima seção.

2.3.3. Seleção de estudos

Para a avaliação da relevância dos estudos foram definidos critérios de inclusão e exclusão, denominados critérios de seleção. Os critérios de inclusão são definidos como segue:

- I1 = Estudo apresenta a experiência da aplicação do desenvolvimento guiado por testes em ambiente acadêmico.
- I2 = Estudo apresenta descrições técnicas e pedagógicas e ainda critérios de avaliação para a prática do TDD em disciplinas iniciais de programação.

De forma semelhante, foram definidos critérios para a exclusão de estudos que não colaboram para com as questões de pesquisa definidas.

- E1 = Estudo não é relatado em artigo completo.
- E2 = Estudo não está disponível em Português ou Inglês.
- E3 = Estudo não aborda a prática do TDD no âmbito acadêmico.
- E4 = Estudo não apresenta como a prática do TDD foi conduzida.
- E5 = Estudo não deve ser um livro.

Com foco na qualidade dos estudos selecionados, foram também definidos critérios de qualidade para a avaliação dos estudos selecionados, os quais foram utilizados durante a leitura completa dos estudos com a finalidade de verificar se o estudo continha as informações necessárias para a classificação.

- Q1 = O estudo demonstra como a prática do desenvolvimento baseado em testes empregada foi avaliada, bem como o tipo de projeto experimental?

O processo de seleção dos estudos foi iniciado pela análise de título, resumo e palavras-chave. A partir desta análise, foram selecionados artigos para a leitura completa. Os artigos não selecionados nesta etapa foram excluídos. Em caso de dúvida sobre a relevância, o artigo foi incluso como pré-selecionado, a fim de não cometer falhas de exclusão de artigos que poderiam agregar valor à pesquisa. A partir da pré-seleção, os artigos selecionados foram lidos por completo e classificados conforme os critérios de inclusão e exclusão. Em caso de dúvida os artigos foram discutidos em reunião com um especialista, para inclusão ou possível exclusão.

Considerando as palavras chaves acima definidas, a pesquisa foi executada em três bases de pesquisa. Desta forma foram localizados 51 artigos, sendo um duplicado, totalizando 50 estudos, conforme apresentado na Tabela 2.2.

Tabela 2.2. Estudos localizados e incluídos por base.

Base	Localizado	Estudo Relevante
ACM Digital Library	28	17
Scopus	20	13
IEEE Explore	02	00
ScienceDirect	00	00
SBIE	17	00
Repetidos	17	12
Total	50	18

Considerando os 50 estudos recuperados, foram pré-selecionados 20 estudos para a leitura na íntegra, dos quais 2 foram excluídos por não se enquadrarem nos critérios de seleção. Desta forma, 18 estudos foram considerados relevantes para a presente pesquisa. Os 18 estudos selecionados foram publicados ao longo de 10 anos, com maior número de publicação nos anos de 2007, 2009 e 2012, como apresenta a Figura 2.8.

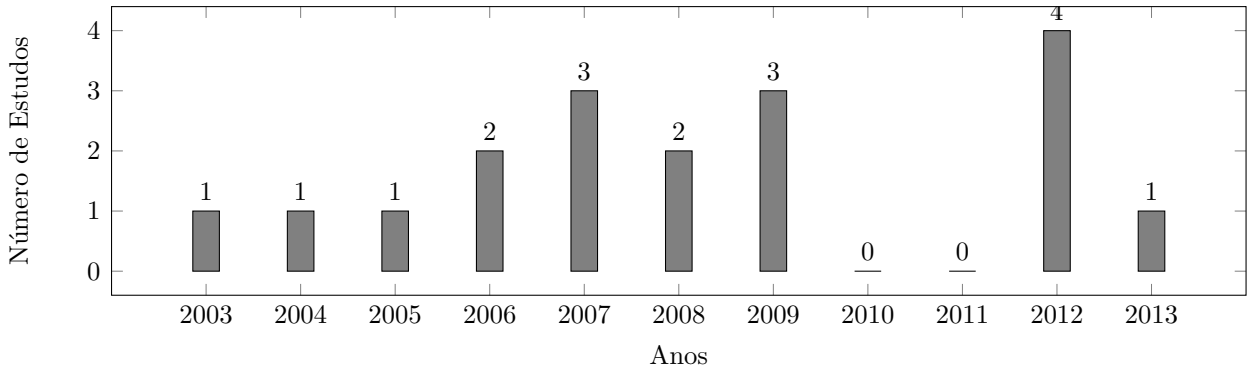

Figura 2.8. Visão temporal das publicações.

Em suma, os estudos selecionados estão dispostos na Tabela 2.3. Em relação a esses, é importante ressaltar que os estudos 1, 11, 12, 14, 16 apresentam estratégias para utilização do TDD, mas não aplicam estudo experimental para a validação de resultados.

2.3.4. Análise e interpretação dos dados

No contexto pedagógico foram observados qual o momento de aplicação da técnica proposta no estudo e também a estratégia e suas características. Desta forma, observou-se que a maior parcela de aplicação de TDD dá-se nas disciplinas CS1 e CS2, tal como apresentado na Figura 2.9.

Para a coleta dos dados técnicos, foi utilizada como base as categorias e variáveis definidas por Munir *et al.* (2014), apresentada na Figura 2.7. Esta classificação configura-se relevante pois apresenta categorias, variáveis e métricas que são utilizados por pesquisadores

Tabela 2.3. Estudos selecionados.

Cód.	Referência	Título
01	Adams (2009)	Test-driven Data Structures: Revitalizing CS2
02	Edwards (2003b)	Rethinking Computer Science Education from a Test-first Perspective
03	Janzen e Saiedian (2006b)	Test-driven Learning: Intrinsic Integration of Testing into the CS/SE Curriculum
04	McKinney e Denton (2006)	Developing Collaborative Skills Early in the CS Curriculum in a Laboratory Environment
05	Shams e Edwards (2013)	Toward Practical Mutation Analysis for Evaluating the Quality of Student-written Software Tests
06	Allevato e Edwards (2012)	RoboLIFT: Engaging CS2 Students with Testable, Automatically Evaluated Android Applications
07	Briggs e Girard (2007)	Tools and Techniques for Test-driven Learning in CS1
08	Buffardi e Edwards (2012)	Exploring Influences on Student Adherence to Test-driven Development
09	Desai *et al.* (2009)	Implications of Integrating Test-driven Development into CS1/CS2 Curricula
10	Edwards (2004b)	Using Software Testing to Move Students from Trial-and-error to Reflection-in-action
11	Edwards e Pérez-Quiñones (2007)	Experiences Using Test-driven Development with an Automated Grader
12	Hilton e Janzen (2012)	On Teaching Arrays with Test-driven Learning in WebIDE
13	Isomöttönen e Lappalainen (2012)	CSI with Games and an Emphasis on TDD and Unit Testing: Piling a Trend Upon a Trend
14	Janzen e Saiedian (2008)	Test-driven Learning in Early Programming Courses
15	Marrero e Settle (2005)	Testing First: Emphasizing Testing in Early Programming Courses
16	Schaub (2009)	Teaching CS1 with Web Applications and Test-driven Development
17	Thornton *et al.* (2007)	Helping students test programs that have graphical user interfaces
18	Thornton *et al.* (2008)	Supporting Student-written Tests of GUI Programs

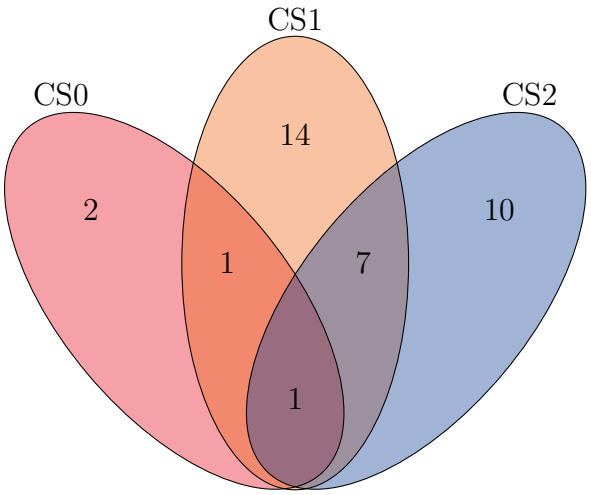

Figura 2.9. Distribuição dos estudos por disciplina.

em projetos experimentais para avaliar as vantagens do TDD sobre o *test-last*.

A Tabela 2.4 apresenta os dados extraídos dos estudos analisados mediante a classificação definida por Munir *et al.* (2014). Para facilitar a visualização, apenas os estudos que apresentam estudos experimentais a fim de validar a proposta e que utilizaram ao menos uma das variáveis definidas para avaliação de seus resultados foram dispostos na tabela. A revisão sistemática proposta por Munir *et al.* (2014) apresenta grande quantidade de categorias e variáveis utilizadas na avaliação do TDD. Todavia, no contexto definido neste mapeamento, as categorias definidas na (Tabela 2.4) sumarizam como a técnica proposta na intervenção é observada no âmbito acadêmico.

Com base nas variáveis utilizadas na avaliação das estratégias propostas, é possível

Tabela 2.4. Variáveis coletadas na revisão sistemática segundo a classificação de (MUNIR *et al.*, 2014).

Categoria	Variável	2	3	4	5	6	7	8	9	10	13	15	17	18
Produtividade	Novas linhas de código					x								
Esforço/Tempo	Tempo de desenvolvimento			x	x	x				x	x			
Tamanho	Linhas de Código								x					
	Número de casos de teste					x				x	x			
Qualidade Externa	Quantidade de defeitos			x	x	x								
	Total de asserções			x			x		x	x				
	C.T. que passaram					x	x		x	x	x			
Qualidade Interna	Análise de Mutação						x							
	Cobertura de Nós							x		x				
	Cobertura de Arestas							x		x				
	Cobertura de Métodos							x		x				
Opinião	Escala de Likert	x	x					x	x					
	Notas dos exames	x			x	x	x		x	x		x	x	x
	Questionários	x						x			x	x	x	

observar que as categorias mais utilizadas são esforço, qualidade interna e opinião do desenvolvedor. Esta constatação está ligada ao objetivo de aplicação das estratégias no contexto do ensino, visto que o aumento da qualidade interna e externa pode impactar de modo positivo ou negativo na opinião do desenvolvedor. A variável com destaque neste contexto é a "Nota dos exames", foi utilizada em 09 estudos, e pode ser chave quanto à aceitação do TDD. Neste contexto, a melhora do desempenho acadêmico do indivíduo pode ser um motivador para a adesão ao TDD.

Mediante análise dos dados e das características das estratégias de aplicação do TDD, os estudos foram classificados em 5 categorias, os quais são discutidas a seguir.

2.3.4.1. Avaliação automática

Edwards (2003b) tem uma extensa pesquisa sobre a utilização de TDD e avaliação automática de trabalhos de alunos em disciplinas iniciais de programação em computação (EDWARDS, 2003b, 2004b; THORNTON *et al.*, 2007; EDWARDS; PéREZ-QUIñONES, 2007; THORNTON *et al.*, 2008; BUFFARDI; EDWARDS, 2012). A proposta deste pesquisador é utilizar a abordagem do TDD em aulas práticas, a partir da primeira disciplina com foco em programação (CS1). Desde suas primeiras linhas de código, o acadêmico deve ser capaz de demonstrar a regularidade de seu código através do desenvolvimento de casos de teste que verificam a qualidade do código de produção (EDWARDS, 2003b).

Desta forma, é necessário que o aluno receba um retorno rápido se seus programas estão atingindo a qualidade mínima requerida pela atividade. Mediante este fato, é proposta uma ferramenta que tem por objetivo receber o código do aluno e o avaliar quanto a validade

e integridade do conjunto de casos de teste do aluno, dando retorno a respeito de quais casos de teste estão corretos ou incorretos e dicas de como melhorar o conjunto. Quanto a estilo de codificação e qualidade fornecer retorno quanto a possíveis melhorias.

Mediante os objetivos propostos, foi desenvolvida a ferramenta Web-CAT, que é uma aplicação Web que recebe o código de implementação bem como o conjunto de casos de teste. Após o envido do trabalho, este é compilado e avaliado em quatro dimensões: exatidão, integridade dos casos de teste, validade do teste e de qualidade de código (EDWARDS, 2003b).

A fim de verificar os efeitos da ferramenta, os autores propuseram um experimento no ano de 2003, em que estudantes enviaram todas as suas atividades para a Web-CAT. Com o intuito de comparar os valores, foram utilizadas as resoluções propostas pelos alunos de 2001, visto que são os mesmos exercícios. Entretanto, é importante salientar que em 2001 os acadêmicos não escreveram casos de teste. Desta forma, foram produzidos casos de testes para possibilitar a análise destes valores antigos pela Web-CAT (EDWARDS, 2004b). A partir dos dados apresentados, a turma de 2001 teve uma quantidade considerável de falhas (11,8%) a mais que a de 2003. Além disso, a média de avaliação do professor da turma de 2003 foi melhor se comparado à nota de 2001. Entretanto, ficou claro que o acadêmico de 2003 precisou investir mais tempo de desenvolvimento nas atividades (EDWARDS, 2004b).

Em outro estudo, Edwards e Pérez-Quiñones (2007) afirmam que, após anos de estudos foi observado que, apesar da ferramenta possibilitar a análise do programa mediante conjunto de testes dos professores, os acadêmicos devem ver valor em seus próprios casos de teste. Se uma ferramenta automatizada é utilizada, o aluno deve entender que o sistema os ajuda a aperfeiçoar o seu código e também como um sistema que o avalia. Entretanto, é necessário definir número máximo de submissões a fim de evitar o excesso de envios por parte do aluno, com o intuito de prevenir que o aluno programe na tentativa e erro (EDWARDS; PéREZ-QUIñONES, 2007). Destacam ainda que a prática do TDD influencia os acadêmicos a produzirem melhor qualidade de código, mas que convencer o aluno a praticar TDD é uma tarefa que precisa ser estudada com mais afinco, visto que os métodos de aplicação do TDD não têm sido foco de estudos (BUFFARDI; EDWARDS, 2012).

Com o intuito de avaliar a qualidade dos casos de teste desenvolvidos durante os exercícios enviados a Web-CAT, Shams e Edwards (2013) propõem a utilização da análise de mutantes na análise da qualidade do conjunto de casos de teste. Todavia, o uso de critério de teste levanta alguns desafios que dificultam a sua aplicação neste cenário, os quais são avaliados e propostas possíveis soluções, que foram avaliadas em comparações de envios de acadêmicos para a Web-CAT. Dados os resultados os pesquisadores levantam questionamentos quanto à proposta, dos quais é importante salientar o uso da geração de mutantes. Desta forma, os autores propuseram que os mutantes fossem gerados a partir da solução de referência,

que é proposta pelo professor. No entanto, é possível que a eficácia da técnica seja afetada, visto que os mutantes não são gerados a partir do programa do aluno (SHAMS; EDWARDS, 2013).

2.3.4.2. Programação com elementos de motivação

A proposta da inclusão de elementos de motivação em disciplinas iniciais de programação vem da necessidade de tornar as atividades nestas disciplinas atuais e relevantes para o restante do curso. Todavia este esforço não tem objetivo de adicionar conteúdo extra a essas disciplinas (ALLEVATO; EDWARDS, 2012). Neste contexto, foram encontradas estratégias que utilizam o desenvolvimento de interfaces gráficas, programação de jogos e programação web.

Esta revisão encontrou três estudos com a proposta da utilização de interfaces gráficas, as quais propõe o desenvolvimento de aplicações que sejam executadas em ambientes *desktop* (THORNTON *et al.*, 2007, 2008) e também para a plataforma Android (ALLEVATO; EDWARDS, 2012).

Em ambas as situações os pesquisadores utilizaram bibliotecas com o objetivo de facilitar o desenvolvimento das interfaces gráficas, visto que iniciantes necessitam ter complexidade reduzida em termos de conceitos e programação e de configuração (THORNTON *et al.*, 2007). Cabe salientar que o foco das duas abordagens não é o ensino do desenvolvimento de interfaces gráficas e sim dos conceitos básicos definidos para as disciplinas (ALLEVATO; EDWARDS, 2012; THORNTON *et al.*, 2008, 2007).

Na abordagem que propõe o desenvolvimento de interfaces desktop foi utilizado uma biblioteca chamada ObjectDraw em conjunto com uma biblioteca de testes automatizados para interface gráfica chamada Abbot GUI (baseada no JUnit). Cabe salientar que adaptações foram propostas a fim de facilitar a utilização destas ferramentas por parte do acadêmico (THORNTON *et al.*, 2007). O ambiente de desenvolvimento utilizado pelos acadêmicos nesta proposta é o BlueJ, por se tratar de uma IDE que tem suporte a testes unitários e ao ObjectDraw. O ambiente de avaliação automática (Web-CAT) foi modificado a fim de comportar os testes de unidade escritos para avaliar a interface gráfica (THORNTON *et al.*, 2007).

A proposta de avaliação desta abordagem se deu na disciplina CS1. A avaliação da foi feita com base na comparação dos resultados acadêmicos de três turmas, sendo que a primeira não utilizou programação gráfica, a segunda utilizou o ObjectDraw sem a necessidade do desenvolvimento de casos de testes para a interface gráfica e a terceira utilizou o ObjectDraw e ainda desenvolveu casos de teste (THORNTON *et al.*, 2008).

Com base nos dados das três turmas, foi possível observar que a terceira turma desenvolveu mais casos de teste do que as outras, além de ter iniciado as submissões antes enviando a versão final mais cedo se comparado aos outros anos. Os acadêmicos envolvidos, quando questionados, disseram que acharam a abordagem divertida ou interessante (THORNTON *et al.*, 2008).

A abordagem que propõe o desenvolvimento de aplicações para a plataforma Android é um estudo que foi desenvolvido recentemente e tem pesquisadores em comum. A motivação da proposta é baseada na ascensão das plataformas móveis. A plataforma Google Android foi selecionada pois é baseada na plataforma Java, a mesma linguagem de programação já utilizada nas disciplinas que os pesquisadores aplicaram experimentos (ALLEVATO; EDWARDS, 2012).

A fim de propiciar o desenvolvimento de casos de teste para interfaces do Android, os autores desenvolveram uma ferramenta chamada RoboLIFT. Esta ferramenta é uma extensão da ferramenta LIFT desenvolvida pelos mesmos pesquisadores para a utilização de acadêmicos iniciantes e é compatível com JUnit. Quando um caso de testes é desenvolvido utilizando a RoboLIFT, é possível visualizar no emulador sua execução, fato este que pode colaborar com a diminuição da rejeição da técnica pelo acadêmico (ALLEVATO; EDWARDS, 2012).

Para cada tarefa disponibilizada no curso, são disponibilizados projetos a fim de serem utilizados como esqueletos, com o intuito de diminuir a complexidade da criação do projeto. Tal como a abordagem de implementação de interface gráfica para desktop, os acadêmicos submetem seus exercícios e, para a execução dos casos de teste que envolvem interface gráfica, foi necessária a implementação de um módulo que fosse capaz de executar este conjunto (ALLEVATO; EDWARDS, 2012).

Com o intuito de comparar os resultados, foram utilizados dados de exercícios aplicados para turma de CS2, desenvolvidos com a abordagem de programação de interfaces gráficas de 2010 a 2011, totalizando três turmas e dados coletados mediante a aplicação da abordagem do desenvolvimento de aplicações com Android (ALLEVATO; EDWARDS, 2012).

Para avaliar os resultados obtidos nesta abordagem, foi utilizada a cobertura de teste, neste contexto, o grupo de controle teve menor cobertura que a intervenção, a diferença foi considerada significativa estatisticamente. Outra métrica utilizada é quanto ao percentual de correção do programa, que é medida pela porcentagem de testes de referência que o programa do aluno consegue passar. Para tal, os valores do grupo de intervenção também foram melhores, entretanto sem diferença significativa (ALLEVATO; EDWARDS, 2012).

A adoção do ensino de programação com foco no desenvolvimento de jogos em disciplinas iniciais foi o foco do estudo proposto por Isomöttönen e Lappalainen (2012). Os autores salientam que desenvolver jogos em disciplinas como CS1 pode ter um viés positivo quanto à motivação do acadêmico com impacto direto na retenção acadêmica, sem propiciar

a diminuição de conteúdo programático previsto (ISOMöTTöNEN; LAPPALAINEN, 2012).

Esta abordagem é proposta em uma turma de CS1, com foco no ensino de programação procedural, utilizando como base a linguagem de programação C#. Durante as aulas, os professores utilizam o TDD e os alunos são incentivados a utilizar a abordagem, entretanto sem sofrer penalidades em avaliações por não utilizar. A ferramenta utilizada para a execução dos testes automatizados é a COMTEST (LAPPALAINEN *et al.*, 2010), a qual permite a escrita de testes de maneira simplificada (ISOMöTTöNEN; LAPPALAINEN, 2012).

A análise dos resultados se deu a partir de um questionário que os alunos responderam na disciplina em que a abordagem foi aplicada. Especificamente quanto ao foco proposto, a avaliação do TDD demonstrou que 68% dos acadêmicos desenvolveram o seu jogo utilizando TDD. A partir de análises dos comentários dos alunos foi possível identificar que os mesmos acreditam que o TDD colaborou para a detecção de defeitos e para com a compreensão do código que estavam desenvolvendo (ISOMöTTöNEN; LAPPALAINEN, 2012).

De forma geral, os autores salientam que os alunos que praticaram TDD têm dificuldades no início, entretanto com o passar do tempo a abordagem é satisfatória. Alguns alunos preferem a depuração a escrever testes por acreditarem que o desenvolvimento de casos de teste seja irrelevante. Foi citado também que receber casos de teste prontos (*test-first*) é um modelo a ser considerado. Os autores ainda identificaram que problemas técnicos com IDE e bibliotecas de teste podem afastar os acadêmicos do *Test-Driven Development* (ISOMöTTöNEN; LAPPALAINEN, 2012).

A proposta da utilização de ferramentas com foco na Web proposta por Schaub (2009) tem o objetivo centrado em retenção. O proposto é entusiasmar o acadêmico com a introdução do desenvolvimento Web na disciplina CS1. Sua motivação vem da experiência que o acadêmico já possui de utilizar este meio (SCHAUB, 2009).

Os alunos utilizam entradas por meio de "console" e geram saídas em HTML. Cabe salientar que os acadêmicos não são instruídos sobre HTML, apenas recebendo exemplos de como os arquivos de saída podem ser e, a partir destes, utilizam a criatividade para produzir suas próprias saídas (SCHAUB, 2009).

Com o intuito de reduzir o esforço necessário para que os alunos adotem o TDD, optou-se por não utilizar um *framework* de testes. Para tal, foi utilizado um comando específico que permite adotar o TDD sem a utilização de ferramentas adicionais. Esta disciplina não focou em questões específicas de arquitetura de sistemas Web (SCHAUB, 2009).

O pesquisador salienta que, antes da aplicação desta proposta, a turma de CS1 tinham de 80 a 85% de aprovação. Após a implantação a diferença não foi significativa. Entretanto, os acadêmicos participantes demonstraram mais interesse e normalmente seus esforços os levavam a completar os exercícios e ainda ir além da simples conclusão. É importante salientar

que este estudo não apresentou um projeto experimental com o intuito de provar a abordagem explanada (SCHAUB, 2009).

2.3.4.3. *Test-Driven Learning*

A proposta desta abordagem baseia-se em três princípios bem definidos de como o professor deve se portar durante o ensino de programação. Os princípios são: ensine baseado em exemplos; apresente exemplos com testes automatizados; inicie sempre pelos testes. A proposta dos pesquisadores é de apresentar uma metodologia que tenha aderência desde a iniciantes até para programadores profissionais com foco em: teste de software, utilizar simples *frameworks* de teste; encorajar a prática do TDD; melhorar a qualidade do código do aluno. Cabe salientar que a proposta tem por princípio acadêmicos que programem utilizando o paradigma de orientação a objetos (JANZEN; SAIEDIAN, 2006b).

No primeiro estudo apresentado (JANZEN; SAIEDIAN, 2006b), foi proposto um pequeno experimento em uma turma de CS1 a fim de comparar alunos que utilizaram da prática proposta com alunos que não utilizaram. Em uma enquete, os acadêmicos que utilizaram o *Test-Driven Learning* (TDL) tiveram 10% de vantagem em comparação ao grupo de controle. Os pesquisadores aplicaram outro questionário a fim de coletar informações quanto outros indivíduos (CS2, SE e Indústria), o qual demonstrou que programadores mais experientes são mais reservados a adoção do TDL (JANZEN; SAIEDIAN, 2006b).

Em um estudo subsequente à proposta, foi executado um experimento com acadêmicos participantes das disciplinas CS1 e CS2. Os participantes iniciaram o desenvolvimento do primeiro projeto do experimento divididos pelo professor em dois grupos: um utilizando TDD e outro *test-last*. No segundo projeto os grupos foram invertidos (JANZEN; SAIEDIAN, 2008).

Os resultados dos alunos de CS1 demonstraram que, no primeiro projeto, os acadêmicos que utilizaram o TDD desenvolveram mais casos de teste que o grupo de controle, e quando ouve a troca, o grupo passou a praticar *test-last*. Ao final do experimento este grupo produziu uma quantidade expressiva a mais de casos de teste se comparado ao primeiro grupo que praticou *test-last*. Segundo o autor, estes dados podem evidenciar um viés positivo que a prática do TDD pode proporcionar (JANZEN; SAIEDIAN, 2006b). Na execução com alunos de CS2, onde foi possível escolher a abordagem a ser utilizada, apenas 6 estudantes escolheram participar do grupo de TDD e os outros optaram pelo *test-last*. Nesta execução, os acadêmicos do grupo de TDD utilizaram 12% a menos de tempo que o grupo de *test-last* e ainda escreveram mais linhas de código (JANZEN; SAIEDIAN, 2008). Os resultados dos dois estudos não demonstraram diferenças estatísticas a fim de comprovar que a abordagem proposta é eficiente. Entretanto, efeitos positivos foram demonstrados nos resultados.

2.3.4.4. Sistema inteligente de tutoria

Com o intuito de facilitar o aprendizado de acadêmicos iniciais em desenvolvimento de software, Hilton e Janzen (2012) propuseram a utilização de um IDE web que contém um sistema de tutoria inteligente com o propósito de ensinar programação utilizando o TDD. A abordagem proposta é baseada da abordagem TDL proposta pelo segundo autor deste estudo e definida na Seção 2.3.4.3. A WebIDE atua também como um avaliador automático, tal como proposto na Seção 2.3.4.1.

Para verificar se a abordagem proposta atinge o objetivo proposto, foi desenvolvida na WebIDE uma lição sobre vetores, a qual aborda as características e conceitos básicos. O experimento contou com acadêmicos da disciplina CS0, os quais foram separados em dois grupos (controle e intervenção). Os grupos foram treinados sobre vetores: o grupo de controle segundo os meios tradicionais e o outro utilizando a WebIDE (HILTON; JANZEN, 2012).

Com a execução do questionário após o treinamento, o grupo de intervenção demonstrou que em questionários mais complexos teve um aproveitamento até 19% acima do grupo de controle (HILTON; JANZEN, 2012). Desta forma, a intervenção demonstrou que a abordagem proposta pode colaborar para com o aprendizado do acadêmico.

2.3.4.5. Aprendizagem colaborativa

Segundo McKinney e Denton (2006), a aprendizagem colaborativa precoce leva a um maior interesse da turma. Mediante este fato, os pesquisadores sugerem que a utilização precoce de aprendizagem colaborativa em disciplinas iniciais de desenvolvimento de software pode aumentar o interesse e aproveitamento do acadêmico de forma a colaborar com a diminuição da taxa de evasão, tal como já abordado nesta proposta como um dos desafios enfrentados na área (LETHBRIDGE *et al.*, 2007; SHAW, 2000). Mediante este fato, o proposto tem como foco propiciar a aprendizagem colaborativa em disciplinas iniciais de programação utilizando práticas ágeis em atividades semestrais em laboratório.

A proposta foi aplicada em uma turma de CS1, as aulas foram dedicadas para desenvolver um único projeto ao longo do semestre, que se constitui no desenvolvimento de um programa para gerenciar um sistema para reciclagem. Durante as aulas os acadêmicos trabalharam em duplas, praticando programação em pares e TDD (MCKINNEY; DENTON, 2006). Durante a aplicação dos laboratórios, os instrutores orientam os acadêmicos em suas deficiências, de maneira a corrigir possíveis problemas a tempo de evitar desistências ou fracassos das equipes (MCKINNEY; DENTON, 2006).

A avaliação dos resultados se deu por dados coletados em dois momentos: no meio e final do semestre. Em cada um desses momentos os acadêmicos avaliaram suas duplas

quantitativamente e seus grupos qualitativamente a partir da listagem de pontos fortes e fracos. Os resultados quantitativos demonstraram que as habilidades das equipes evoluíram durante o semestre e os qualitativos demonstraram que os pontos fracos diminuíram com reflexo no aumento das qualidades (MCKINNEY; DENTON, 2006).

2.3.4.6. Considerações do mapeamento sistemático

A questão primária deste estudo buscava evidenciar quais as técnicas utilizadas em conjunto com o TDD no contexto de alunos que estão iniciando no desenvolvimento de software, desta forma, foram identificadas as cinco categorias na Seção 2.3.4, as quais são: avaliação automática, programação com elementos de motivação, *Test-Driven Learning*, sistema inteligente de tutoria, aprendizagem colaborativa. As categorias, suas características, benefícios e dificuldades discutidas na Seção 2.3.4 apresentam as técnicas aplicadas em conjunto com o TDD no ambiente educacional (Seção 2.3.1).

Na categoria de avaliação automática, os trabalhos objetivam que o aluno receba retorno rápido sobre as atividades que estão desenvolvendo e ainda quanto à correção de suas atividades durante o desenvolvimento. Esta estratégia está se desenvolvendo no sentido da avaliação da qualidade do conjunto de casos de teste dos alunos com a finalidade de propor melhorias no conjunto de casos de teste. Os acadêmicos que as utilizam normalmente desenvolvem mais casos de teste e com menor quantidade de falhas.

No contexto da utilização de elementos de motivação, foram localizadas estratégias que utilizam o desenvolvimento de aplicações *desktop*, móveis, *web* e jogos. Estratégias como essas são utilizadas com intuito de engajar o acadêmico e impactar diretamente na retenção acadêmica, envolvendo os alunos no desenvolvimento de aplicações interessantes. Essas abordagens utilizam bibliotecas simplificadas, que tornam acessíveis aos alunos iniciantes. Em conjunto com essas propostas também foram propostas a utilização de IDE com foco em alunos iniciantes e ferramentas de avaliação automática (THORNTON *et al.*, 2008).

A proposta da utilização do TDL é baseada em três princípios: ensine baseando em exemplos, apresente exemplos com testes automatizados e inicie sempre pelos testes. Ela tem como objetivo central melhorar a qualidade do código do aluno. Os resultados observados nesta revisão demonstram que os praticantes demoraram menos tempo no desenvolvimento e escreveram mais código, evidenciando melhora na produtividade. Com base no TDL, foi proposto um sistema inteligente de tutoria o qual propôs a utilização de uma IDE *web* a qual tinha características de um avaliador automático. Foi demonstrado que a estratégia obteve resultados expressivos quanto em comparação ao grupo de controle.

A utilização de métodos ágeis de desenvolvimento de software foi proposta pela categoria de aprendizagem colaborativa. O intuito da proposta foi envolver os acadêmicos

em equipes com a finalidade de desenvolver um único projeto durante o semestre simulando assim um projeto de software real. O professor atuava como o gestor do projeto, guiando as atividades e resolvendo problemas com o objetivo que todos os grupos entreguem os projetos.

A primeira questão secundária foi definida objetivando encontrar quais técnicas de teste de software são utilizadas em conjunto com a prática do TDD com alunos que estão iniciando em programação. Todavia, não foram localizados estudos que utilizassem a aplicação de técnicas de teste de software em conjunto com o TDD como intervenção. Desta forma, esta linha de pesquisa é incipiente no âmbito educacional. É importante ressaltar que foi localizado um estudo que utilizava critérios de teste em trabalhos submetidos a uma ferramenta de avaliação automática com o objetivo de avaliar a qualidade do conjunto de casos de teste (SHAMS; EDWARDS, 2013).

A segunda questão secundária buscou evidências de dificuldades mencionadas na aplicação de estratégias que utilizassem o TDD como técnica de desenvolvimento. Neste sentido, foi apontado que a inexperiência pode causar a dificuldade na adaptação com um *Integrated development environment* (IDE) complexo e também dificuldades na utilização de bibliotecas para criação do conjunto de casos de teste (ALLEVATO; EDWARDS, 2012; THORNTON *et al.*, 2008, 2007; SCHAUB, 2009). Neste contexto, foi proposto o uso de ferramentas mais simples e bibliotecas com foco em alunos iniciantes (THORNTON *et al.*, 2007; SCHAUB, 2009). Janzen e Saiedian (2006b) citaram que programadores mais experientes têm mais dificuldades de se adaptar com a abordagem proposta pelo TDD e desta forma propuseram que o ensino de TDD deve iniciar em disciplinas iniciais. Isomöttönen e Lappalainen (2012) salientam que os alunos que praticaram TDD têm dificuldades no início, entretanto com o passar do tempo a abordagem é satisfatória (ISOMöTTöNEN; LAPPALAINEN, 2012).

A terceira questão secundária tinha por objetivo buscar os benefícios promovidos pelo TDD para alunos que estão iniciando no desenvolvimento de software. Foi relatado que geralmente no início os acadêmicos demonstraram dificuldade, mas, com o passar do tempo, a abordagem demonstra vantagens. Recorrentemente observou-se melhoria substancial das habilidades de programação e de aproveitamento e as propostas foram consideradas interessantes e divertidas (ISOMöTTöNEN; LAPPALAINEN, 2012; THORNTON *et al.*, 2008; ISOMöTTöNEN; LAPPALAINEN, 2012; SCHAUB, 2009; MCKINNEY; DENTON, 2006). Edwards (2004b) observou que a utilização do TDD impõe mais dedicação: os acadêmicos iniciaram o desenvolvimento das atividades antes e levaram mais tempo para a conclusão. Foi citado pelos próprios participantes que o TDD forneceu ferramentas para detecção de defeitos, bem como colaborou com a maior compreensão do software que estava sendo desenvolvido (ISOMöTTöNEN; LAPPALAINEN, 2012).

2.4. Desenvolvimento baseado em casos de testes desenvolvidos criteriosamente

Assim como já discutido, o TDD tem seu foco no projeto de classes e nas boas práticas de orientação a objetos (ANICHE; GEROSA, 2012; BECK, 2002; EDWARDS, 2003a; ASTELS, 2003). Entretanto, na presente proposta o TDD é tratado juntamente com conceitos e técnicas de teste de software com o intuito de verificar a relevância da utilização de técnicas de teste de software em conjunto com a prática do TDD em ambiente de ensino, especificamente em disciplina iniciais de programação, a fim de que o acadêmico tenha foco na qualidade do código desde o princípio de sua formação. Mediante este fato, os seguintes estudos têm propostas semelhantes em contextos diferentes, tal como a indústria.

2.4.1. TDD com critérios de mutação

Shelton *et al.* (2012) propuseram um estudo observacional, com base em uma alteração no ciclo do TDD tradicional, sendo adicionada uma etapa de desenvolvimento de casos de testes a partir do resultado da análise de mutantes. Desta forma, o objetivo é verificar se a etapa adicional colabora com a qualidade do software e não atrapalha o fluxo do TDD. A Figura 2.10 apresenta o fluxo do TDD com a adição da etapa de desenvolvimento de casos de teste com base na análise de mutantes. Neste estudo, é proposto que após a entrega da solução, o pesquisador utilizará a ferramenta MuJava para efetuar a geração mutantes. (SHELTON *et al.*, 2012).

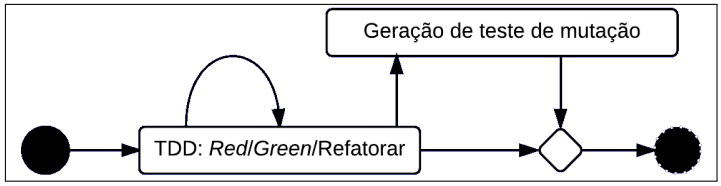

Figura 2.10. Fluxo de TDD proposto (SHELTON *et al.*, 2012).

No processo tradicional do TDD o programador escreve casos de teste baseado em especificações funcionais. Normalmente os programadores são pressionados a entregar o código o quanto antes, algo que não ajuda no desenvolvimento de código robusto e com qualidade. O desenvolvimento baseado em testes levanta alguns questionamentos, tais como: qual a cobertura, se maior quantidade de testes de unidade relataria erros adicionais e se a maior quantidade de códigos de unidade atrapalharia o ritmo de desenvolvimento. O TDD é baseado em ciclos frequentes de Vermelho/Verde/Refatorar e acrescentar a análise de mutantes a tal ciclo pode onerar o desenvolvedor (SHELTON *et al.*, 2012).

Neste contexto, foi proposto um estudo experimental para verificar a validade da técnica proposta. O estudo contou com a participação de três programadores com mais de 10 anos de experiência e com proficiência na prática do TDD.

Todos os programadores escreveram mais de 860 linhas de código para solucionar o problema proposto, os quais codificaram 72 testes unitários, sendo em média de 24 testes para cada programador. A média de cobertura dos testes foi de 81%, a cobertura máxima foi 100%. Para os três programadores foram gerados 644 mutantes, obtendo a pontuação de mutação de 40,5% em média. A partir dos defeitos identificados foram desenvolvidos casos de testes para matar os mutantes vivos (SHELTON *et al.*, 2012).

Foram gerados em média 30 casos de teste e 91 no total, elevando a cobertura de código (*source coverage*) média de 91% para 100% e a pontuação de mutação foi elevada de 40% para 96,9%. Todavia, o resultado mais expressivo é que foram encontrados 20 defeitos após a etapa de análise de mutantes, sendo em média 7 defeitos por programador (SHELTON *et al.*, 2012).

Além do estudo observacional desenvolvido, foram aplicadas perguntas aos programadores, as quais eram possíveis responder entre sim e não (SHELTON *et al.*, 2012). São elas:

1. O processo proposto atrapalhou o ritmo de desenvolvimento?
2. Os novos testes ajudaram na refatoração do código?
3. Os novos testes ajudaram na detecção de defeitos?
4. O relatório foi útil?
5. Você usa ferramentas de cobertura?

Os resultados demonstraram melhoria na qualidade do conjunto de casos de teste e que os programadores receberam bem a alteração no fluxo tradicional do TDD, o qual aumenta o esforço empregado, entretanto melhora a qualidade final do software. Desta forma, os programadores avaliaram a técnica como favorável, visto que todos assinalaram "sim" para as questões 1, 2, 3 e 4.

2.4.2. *Test-Driven Development High Quality*

Também no contexto da utilização de técnicas de teste de software em conjunto com o TDD, Causevic *et al.* (2013a) propõem uma série de estudos. Estes fazem referência a utilização de TDD no contexto da indústria com o foco na utilização de técnicas de teste de software (CAUSEVIC *et al.*, 2012a, 2012b, 2012, 2013b).

Inicialmente estes pesquisadores propuseram um estudo a fim de verificar o impacto que o conhecimento de técnicas de teste de software pode ter mediante a prática do

TDD (CAUSEVIC *et al.*, 2012a). Desta forma, foi proposto um experimento, no qual participantes desenvolveram um determinado software e responderam um questionário e posteriormente foram treinados em técnicas de teste de software. Após o treinamento, foi desenvolvido outro software. Mediante esta dinâmica, os pesquisadores propuseram mensurar os impactos proporcionados pelo conhecimento adquirido em conceitos de teste de software.

Observou-se que o tempo de desenvolvimento não foi afetado, enquanto na cobertura de instruções observou-se melhora substancial, todavia a hipótese nula não pode ser refutada (CAUSEVIC *et al.*, 2012a).

Foi proposto também um estudo com o objetivo de comparar a qualidade dos artefatos produzidos com TDD e com *test-last* (CAUSEVIC *et al.*, 2012, 2012b). Desta maneira foi conduzido um experimento científico, no qual os participantes desenvolveram um software (*Bowling Game*) utilizando as técnicas propostas. Para avaliar os artefatos resultantes foram utilizados: cobertura de instruções, quantidade de defeitos, pontuação de mutação e quantidade de defeitos detectados (CAUSEVIC *et al.*, 2012, 2012b). Todavia, os pesquisadores verificaram que casos de teste que visaram executar funcionalidade de forma excepcional (*negative path*) encontravam mais defeitos. Os casos de teste referentes ao *negative path* representam 29% do total. Entretanto, a quantidade de defeitos por eles evidenciados representa 65%, ou seja, foram mais eficazes. Foi identificado também que os praticantes de TDD escrevem mais casos de teste que exercitam as funcionalidades tal como especificadas (*happy path*). Cabe enfatizar que os grupos que praticaram TDD foram mais rápidos em comparação aos grupos de *test-last* (CAUSEVIC *et al.*, 2012).

Seguindo a linha deste estudo, foi proposto um terceiro, desta vez no contexto industrial, o qual tem por objetivo comprar os resultados de TDD, *test-last* e TDD+. Esta variação do TDD, denominada TDD+, introduziu a utilização do desenvolvimento de testes seguindo o conceito do *negative path* junto a prática do TDD (CAUSEVIC *et al.*, 2013b). Entretanto, não foi definido um fluxo a ser seguido no TDD+, apenas encorajando o uso do *negative path* ao criar casos de teste. O projeto experimental conduzido neste estudo é uma variação do estudo acima relatado (CAUSEVIC *et al.*, 2012). Neste experimento participaram mais de 100 indivíduos de uma empresa da Índia (CAUSEVIC *et al.*, 2013b). Os resultados obtidos foram semelhantes ao do estudo preliminar, em que 71% dos casos de testes eram seguindo as funcionalidades especificadas (*happy path*) e 29% pelo *negative path*. O grupo que utilizou TDD+ criou mais casos de teste utilizando o *negative path*, entretanto o grupo de *test-last* foi o que mais encontrou defeitos com esses casos de teste. Em sumo, durante o experimento, 75% dos defeitos foram evidenciados pelo *negative path*. Estes resultados evidenciaram a importância que casos de teste negativos tem como em um conjunto de testes (CAUSEVIC *et al.*, 2013b). Causevic *et al.* (2013b) afirmam que neste

estudo experimental não houve diferença estatística significativa em qualidade do conjunto de casos de teste entre os grupos TDD, TDD+ e *test-last*.

Com base nos resultados citados, foi proposto o método *Test-Driven Development High Quality* (TDDHQ) (CAUSEVIC *et al.*, 2013a), que investiga como os desenvolvedores podem se beneficiar de conhecimentos adicionais sobre técnicas de projeto de teste de software. Com o objetivo de aumentar a cobertura de defeitos e a qualidade do conjunto de casos de teste, foi proposta uma modificação no fluxo tradicional do TDD, tal como apresentado na Figura 2.11. Antes do início da implementação é necessário que seja elicitado um Aspecto de Melhoria (*Quality Improvement Aspect*), o qual faz referência a aspectos de qualidade que podem ser reforçados, tais como: funcionalidade, desempenho, segurança, usabilidade, robustez, etc. Baseado no aspecto selecionado, o programador seleciona a técnica de teste de software que será empregada a fim de satisfazer as características desse aspecto. O fluxo de execução é seguido até que não existam mais casos de teste derivados da técnica de teste selecionada. Então o programador pode selecionar outro aspecto de melhoria ou então optar por implementar novas funcionalidades (CAUSEVIC *et al.*, 2013a).

Os autores propuseram ainda um pequeno experimento controlado a fim de verificar a eficácia de detecção de defeitos da técnica proposta. A execução se deu durante uma disciplina de teste de software, em que um grupo de alunos utilizou TDDHQ e o outro grupo utilizou o ciclo de desenvolvimento tradicional. Baseado nos resultados obtidos durante a execução não foi possível refutar a hipótese de que a prática do TDDHQ tem melhores resultados se comparado ao ciclo tradicional do TDD. Entretanto, o grupo que utilizou a técnica proposta teve aparentemente melhores resultados (CAUSEVIC *et al.*, 2013a).

Um dado que deve ser observado quanto à técnica funcional é que o estudo utiliza a técnica de caixa branca em conjunto da prática do TDD. Este demonstrou que as entradas inválidas foram mais efetivas na detecção de defeitos que os casos de teste que exploravam entradas válidas (CAUSEVIC *et al.*, 2013a).

2.4.3. TDD com foco em qualidade de teste

Inicialmente, este estudo (FUCCI; TURHAN, 2013) buscou identificar se os praticantes de TDD escrevem mais casos de teste, produzem artefatos com maior qualidade e se são mais produtivos. Fucci e Turhan (2013) fazem um estudo que replica um experimento científico (ERDOGMUS *et al.*, 2005) com o objetivo de comparar *test-first* e *test-last*. Inicialmente os pesquisadores utilizaram um formulário com o objetivo de avaliar conhecimentos prévio.

Os alunos foram separados em grupos de *test-first* e *test-last*, alguns indivíduos executaram o experimento individualmente mas houveram casos de execução em dupla. O software utilizado como atividade a ser desenvolvida foi o *Bowling Game*. Para a avaliação dos

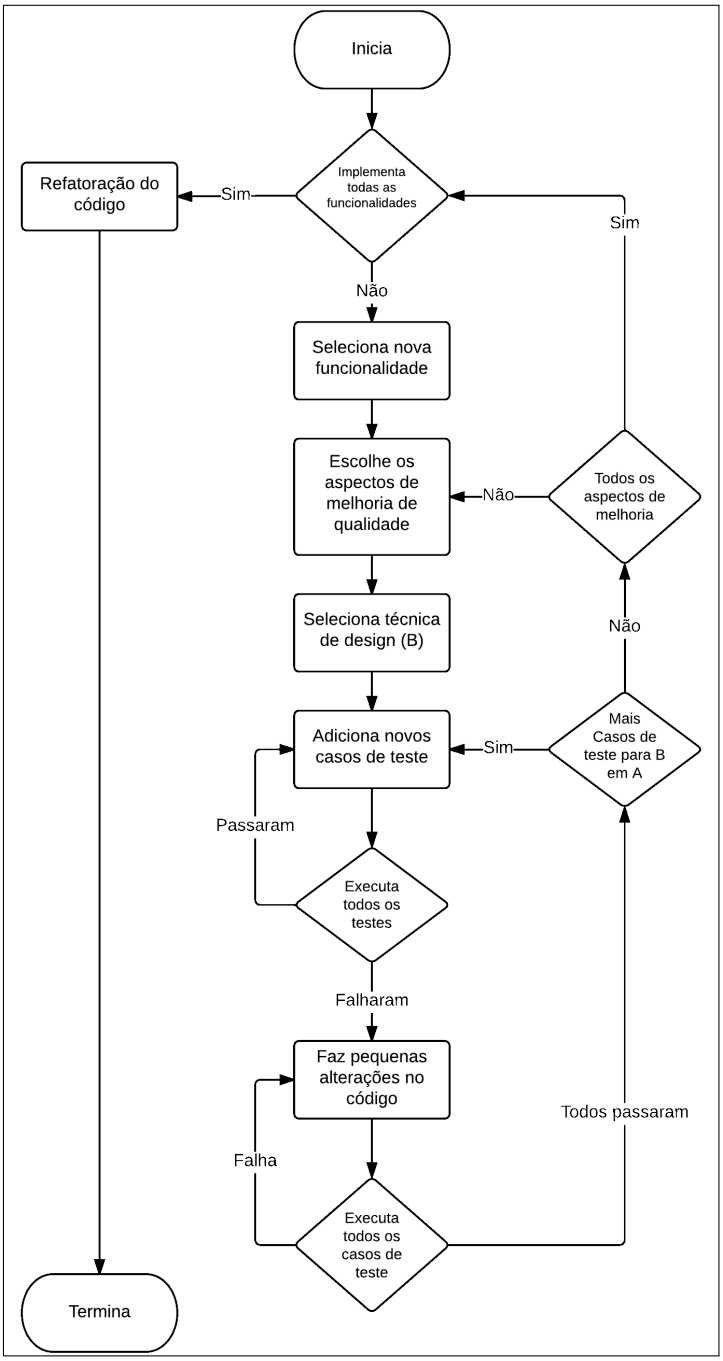

Figura 2.11. Fluxo de execução do TDD proposto pelo TDDHQ (CAUSEVIC *et al.*, 2013a).

resultados foram capturados quantidade de casos de teste, assertivas por caso de teste e que defeitos são assertivas que falharam, tempo de desenvolvimento e ainda quantidade de *user stories* concluídas. Este estudo experimental não identificou diferença significativa em relação a qualidade interna e externa entre os artefatos produzidos pelos dois grupos. Além disso foi afirmado que a programação por pares não influenciou nos resultados obtidos (FUCCI; TURHAN, 2013).

Fucci e Turhan (2014) conduziram outro estudo com base no experimento definido por Erdogmus *et al.* (2005). Este segue a mesma linha do estudo experimental mencionado, todavia com algumas alterações. Neste estudo foi praticado apenas TDD e a programação em pares foi encorajada. Como resultado, foi observado que os resultados obtidos na replicação seguiram consistentes com o estudo original. Não foi obtida correlação entre testes e qualidade, conforme os resultados do estudo original. Todavia, foi possível observar que a maior quantidade de casos de teste corrobora com a produtividade e qualidade. Desta forma, os pesquisadores propõem a criação de maiores conjuntos de testes (FUCCI; TURHAN, 2014).

2.5. Considerações finais

Os conceitos básicos sobre teste de software explorados neste capítulo são apresentados na Figura 2.12. Este mapa conceitual propicia uma visão geral sobre os tópicos de teste de software mencionados. No mapa é possível observar o TDD definido como um processo da prática de teste de software que pode utilizar técnicas de teste de software e normalmente utiliza a fase de unidade para derivar o conjunto de casos de teste. No contexto dessa dissertação o TDD é utilizado tal como a técnica *test-first* como um meio de aplicação de técnicas de teste de software. Durante a execução do TDD na fase de unidade.

Tal como destacado neste capítulo, alguns autores propõe a utilização de critérios de teste de software em conjunto com o TDD com o objetivo de potencializar a técnica (SHELTON *et al.*, 2012; CAUSEVIC *et al.*, 2013a). Desta forma, foram propostas alterações no ciclo

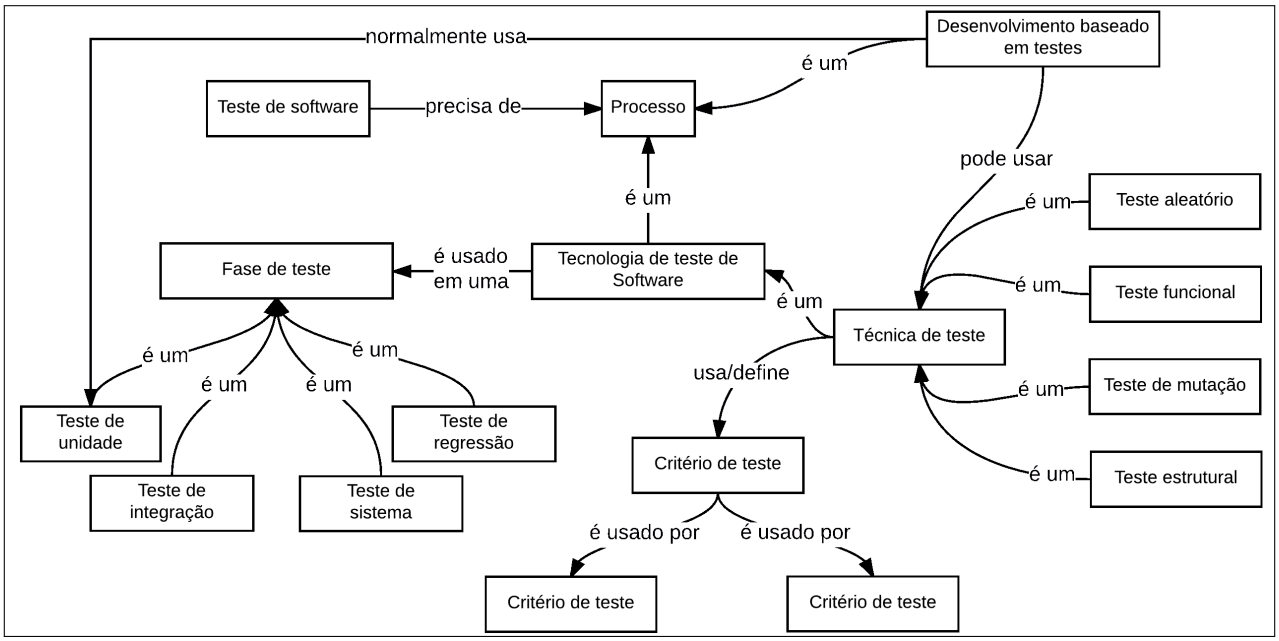

Figura 2.12. Mapa conceitual sobre teste de software, baseado em Durelli *et al.* (2013).

tradicional do TDD e adicionadas etapas para a utilização de critérios de teste funcional e de mutação. Os duas propostas citadas foram aplicadas na indústria, em contexto diferente da presente dissertação.

Baseado nos dados obtidos no mapeamento da literatura discutida neste capítulo e nos demais trabalhos citados que propõem a utilização do TDD, foi possível observar que a aplicação de técnicas de teste de software em conjunto com o TDD é incipiente no domínio educacional. Não foram localizadas estratégias propostas para a utilização no ensino de programação. Com base na forma de análise e ainda nos relatos dos pesquisadores quanto a experiência foi definida a estratégia apresentada no próximo capítulo.

Capítulo 3

Método

Com base no levantamento do estado da arte do TDD no contexto do aprendizado e também em relação a utilização em conjunto com técnicas de teste de software, foram observados benefícios e possíveis problemas. Neste contexto, foi definida uma estratégia inicial, a qual foi testada em experimentos científicos e, a partir dos resultados preliminares, foi definida a estratégia descrita neste capítulo.

Uma importante regra do TDD é "Se você não consegue escrever um teste para o que está codificando, então não deveria estar pensando em codificar" (GEORGE; WILLIAMS, 2004). Na verdade, a testabilidade muitas vezes é considerada um guia para a qualidade. No TDD, o atributo qualidade é imposto pelo uso extensivo de testes de unidade, guiando a criação e modificação de código. No entanto, apesar da importância dos casos de teste, eles não são criados com base em um critério de teste.

Naturalmente o TDD propõe que os requisitos funcionais do software sejam a entrada para o desenvolvimento dos casos de teste. Neste contexto, o programador utiliza as funcionalidades especificadas para derivar seu conjunto de casos de teste da maneira que lhe for satisfatória (*ad hoc*). Desta forma, o programador utiliza a mesma fonte de informação para derivar seus casos de teste, todavia sem a utilização de critérios funcionais e também, sem observar possíveis exceções a funcionalidade.

Estudos mostram que grande parte dos defeitos são evidenciados por casos de teste que se propõe à exercitar as funcionalidades implementadas seguindo o *negative path* (CAUSEVIC *et al.*, 2012b), ou seja, o simples incentivo a desenvolver casos de teste com o objetivo de exercitar exceções foi suficiente para evidenciar mais defeitos (CAUSEVIC *et al.*, 2012).

No TDD a etapa de refatoração caracteriza a melhoria do código já desenvolvido. Em seu fluxo tradicional, com base em uma análise do que já foi desenvolvido, o programador decide

se o código deve ser modificado a fim de melhorar a solução, diminuir acoplamento, aumentar a coesão, dentre outros fatores. Entretanto, nenhuma regra é definida especificamente. Logo, é possível que à refatoração não atinja o objetivo de melhoria. Tal fato pode ser atribuído a inexperiência ou até à falta de interesse de propor uma solução melhor em termos de *design*. Desta forma, considerou-se a alteração da atividade de refatoração adotando a utilização de técnicas de teste de software para derivar casos de teste com o objetivo inicial de guiar os estudantes a complementar o conjunto de casos de teste previamente proposto tornando-o mais robusto. Todavia, é necessário que seja definida uma estratégia coerente quanto a qual critério será utilizado e qual a cobertura mínima deve ser atingida considerando o contexto de aplicação.

Observa-se que a etapa que contempla tradicionalmente a criação de casos de teste no TDD é a *red*. Ela não foi considerada para a alteração em virtude de viés criativo que ela propõe. Nesta etapa o programador cria a estrutura de classes e métodos que será utilizada na construção da solução a ser implementada. Existe a possibilidade que a técnica proposta insira um viés no *design* da aplicação, visto que o praticante utilizará critérios de testes na refatoração. Tal como observado em outros estudos, o simples conhecimento de técnicas de teste de software impacta na forma com que o programador propõe seus casos de teste (CAUSEVIC *et al.*, 2012a). Neste contexto, é possível que os critérios utilizados durante o desenvolvimento, utilizando o ciclo proposto impactem também no primeiro estágio.

Desta forma, com base no ciclo tradicional do TDD, apresentado na Figura 3.1, propõe-se a adição de uma etapa de desenvolvimento de casos de teste em concorrência com a etapa de refatoração.

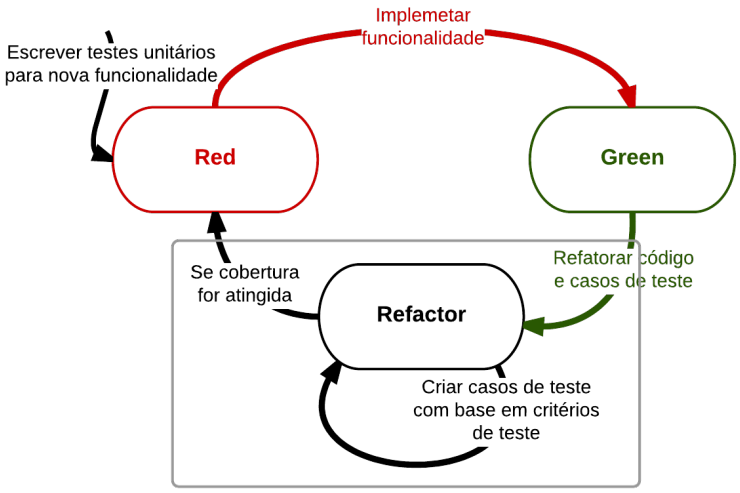

Figura 3.1. Fluxo proposto para o TDD pautado em critérios de teste.

A adoção de critérios de teste não proporciona apenas o aumento da qualidade, mas também dará suporte para os estudantes aprender em simples técnicas de teste e a melhorar o

design de seu código. Além disso, após satisfazer os critérios mais simples, é possível utilizar critérios mais fortes. Assim, esta estratégia colabora com a melhoria incremental da qualidade.

3.1. Estratégia geral

A proposta da utilização de critérios de teste de software descrita nesta dissertação altera o ciclo tradicional de execução do TDD, estendendo a etapa de *refactor* com a finalidade de que o conjunto de casos de teste também seja melhorado. Inicialmente é necessário que seja definida qual a estratégia será utilizada. Para tal foram definidas três perguntas que auxiliarão nesta tarefa.

1. Qual critério de teste de software será utilizado?
2. Qual métrica deve ser atingida para caracterizar o final da iteração?
3. Como o praticante avaliará a cobertura durante a prática do TDD?

Em primeiro lugar, é necessário que seja selecionado os critérios que serão utilizados, observando o contexto de utilização da estratégia. No caso da utilização de mais de um critério, é necessário que seja definido se existe uma ordem de cobertura ou regra durante a execução. Inicialmente, para que seja possível utilizar o arcabouço proposto, é necessário que as duas questões subsequentes sejam contempladas a fim de fornecer subsídios para que o praticante obtenha êxito. O objetivo da proposta é melhorar a qualidade do código desenvolvido, bem como o código de produção, entretanto, sem prejudicar a prática do TDD. A seleção de um critério incompatível com o público de aplicação pode inviabilizar a prática.

Após a definição do critério a ser utilizado, é necessário que sejam definidas quais medidas devem ser observadas durante a execução que caracterizará o final do ciclo de refatoração e desta forma o reinício do ciclo do TDD. A proposta deste estudo é a utilização de critérios de teste de software. Desta forma, é necessário que seja fornecida uma forma de avaliação de cobertura do conjunto de casos de teste. Visto que a técnica é baseada na cobertura, é necessário que o praticante consiga mensurar, a cada execução do conjunto de casos de teste, a sua cobertura e, com base nisso, seguir a estratégia proposta.

A partir da definição da estratégia, tal como apresentado na Figura 3.1, a execução do TDD acontecerá como o definido tradicionalmente até que a etapa de refatoração seja iniciada. Durante esta etapa o praticante poderá escolher se inicialmente será feita a refatoração de código ou o conjunto de casos de teste será melhorado. É esperado que a refatoração poderá impactar na cobertura e "refatoração" dos casos de teste pode evidenciar defeitos. Deste modo, espera-se potencializar a etapa de "melhoria" naturalmente proposta, transformando-a em um novo ciclo interno de validação funcional e melhoria de forma geral do software. Este ciclo deve se estender até que o critério estabelecido como saída deste ciclo seja atingido: no

contexto deste trabalho, até que a cobertura mínima seja atingida.

3.2. Estratégia com critérios estruturais

No contexto da presente dissertação em que se objetiva propor uma estratégia para alunos que estão iniciando no desenvolvimento de aplicações, é necessário que sejam utilizados conceitos simples de teste de software. Todavia, apesar da utilização de critérios simples, é importante ressaltar que estudos demonstram que conjuntos de casos de teste simples têm capacidade de revelar defeitos complexos (ACREE *et al.*, 1979; BUDD, 1980). Além disso, é necessário preservar o fluxo tradicional do TDD.

Pautado no arcabouço proposto, foi definida uma estratégia baseada em critérios de teste de software estrutural que utilizam GFC para derivar os requisitos de teste. Seguindo o modelo previamente descrito, a estratégia se constitui tal como definido a seguir.

1. Qual critério de teste de software será utilizado?
    - Serão utilizados os critérios de teste de software baseado em CFG: Todos-Nós e Todas-Arestas.
2. Qual métrica deve ser atingida para caracterizar o final da iteração?
    - Serão observadas as coberturas nos dois critérios. Todavia o acadêmico deve seguir uma ordem de cobertura. É necessário atingir 100% do critério Todos-Nós para então implementar casos de teste com o objetivo de atingir 100% de cobertura do critério Todas-Arestas.
3. Como o praticante avaliará a cobertura durante a prática do TDD?
    - Durante o desenvolvimento da aplicação proposta o indivíduo deverá utilizar o IDE Eclipse em conjunto com os plugins JUnit e Eclemma, que fornecerão as informações necessárias sobre cobertura dos critérios definidos para o desenvolvimento utilizando o arcabouço proposto.

Com base nessa definição, foi desenvolvido um diagrama que demonstra como será o fluxo de desenvolvimento proposto (Figura 3.2). Neste modelo, observa-se que durante a etapa de refatoração o aluno deverá trabalhar em paralelo com a atividade de refatoração de código e melhoria do conjunto de casos de teste derivados a partir dos critérios Todos-Nós e Todas-Arestas. Desta forma, no contexto da melhoria do conjunto de casos de teste é necessário inicialmente que 100% do critério Todos-Nós seja atingido. Após isso inicia-se o desenvolvimento dos casos de teste com o intuito de atingir 100% do critério Todas-Arestas.

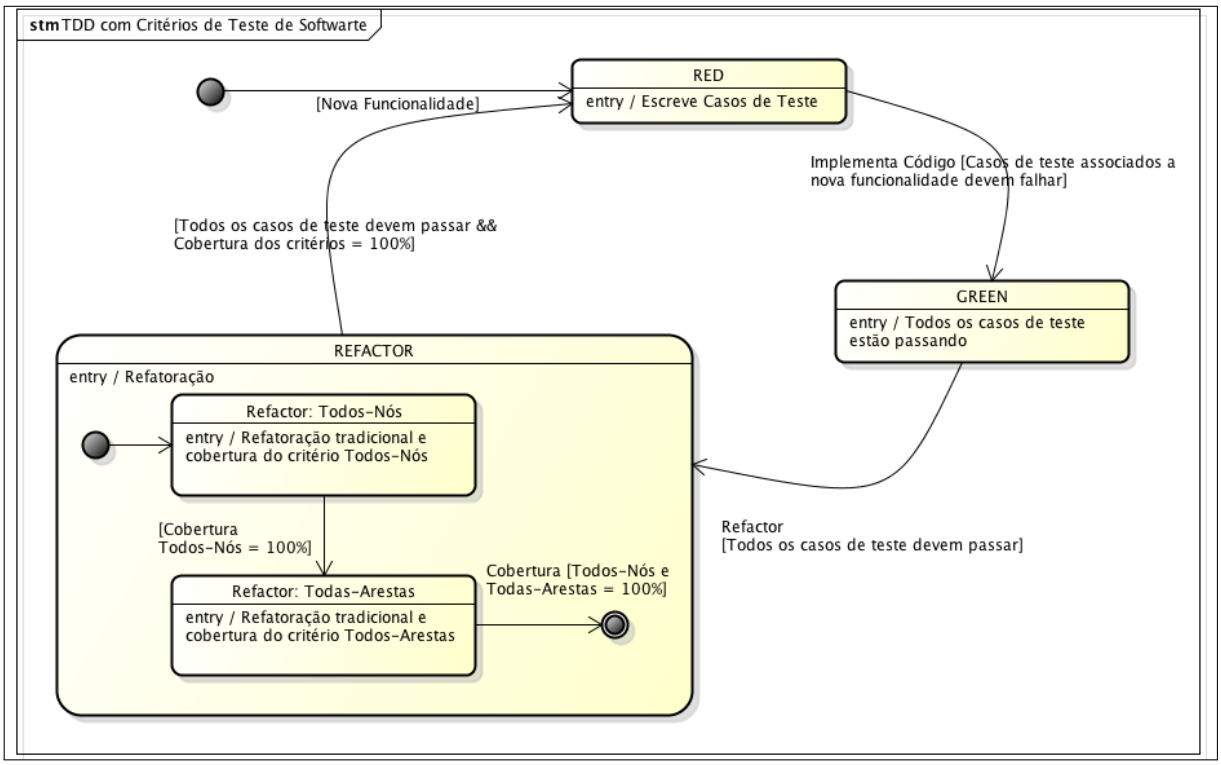

Figura 3.2. TDD com critérios estruturais.

3.3. Definição das medidas

Para verificar se o objetivo foi atingido, deve-se capturar evidências que permitam avaliar a satisfação do objetivo (BASILI, 1992). Deste modo, para avaliar se a técnica proposta atinge os objetivos esperados, foi necessário estabelecer a meta almejada. Para verificar se a meta foi atingida, é necessário definir questões que caracterizem o objetivo traçado a com base na perspectiva traçada. Por fim, deve-se definir as métricas que serão utilizadas para a avaliação das questões.

A partir de um delineamento de objetivo, questões e métricas, Basili, propôs o *Goal Question Metric* (BASILI, 1992). O pesquisador parte do pressuposto que para avaliar uma técnica ou processo, primeiramente é necessário que este seja definido e então metas devem ser traçadas e, com base nas metas definir questões que as avaliem. A partir das questões é necessário que sejam estabelecidas as métricas que serão utilizadas com o objetivo de responder as questões. Por final, é necessário que seja fornecido um arcabouço para a interpretação dos dados (BASILI, 1992).

O GQM é baseado em uma estrutura hierárquica em que inicialmente é definida uma meta (*Goal*), a qual pode ser relacionada a diversos modelos de qualidade ou baseado em pontos de vista em relação a ambientes. Com base na meta são definidas questões (*Questions*), as quais são utilizadas com a finalidade de caracterizar a forma que a meta será avaliada a

meta. As métricas (*Metric*) são associadas a uma ou mais perguntas e tem como propósito responder de forma quantitativa as questões. Uma métrica pode ser associada a mais de uma questão (BASILI, 1992).

- **Objeto de estudo:** O objeto de estudo definido para o presente trabalho é a verificar se a estratégia proposta corrobora com a qualidade do código de teste e também com o código de produção, assim beneficiando alunos.
- **Propósito** A técnica proposta tem por objetivo melhorar a qualidade do conjunto de casos de teste do acadêmico em disciplinas iniciais e consequentemente impactar em seus resultados acadêmicos. Desta forma, foi proposto que durante a execução tradicional do TDD sejam desenvolvidos casos de teste tal com o objetivo de evidenciar defeitos. Tal fato adiciona a utilização de técnicas de teste de software a prática que inicialmente tem o objetivo de promover a testabilidade do código.
- **Foco de qualidade** A abordagem proposta visa impactar na qualidade do conjunto de testes desenvolvido durante o exercício da técnica. Todavia, espera-se que a etapa de desenvolvimento de casos de teste não interrompa o ciclo tradicional do TDD, mantendo assim os benefícios que a técnica proporciona tanto em âmbito educacional, quanto na indústria.
- **Ponto de vista** Serão observados os pontos de vista do Aluno e do Professor, visto que serão coletadas informações da resolução dos problemas propostos aos alunos. Além disso, os alunos participantes responderão questionários em dois momentos durante a execução do estudo experimental: antes do treinamento e após a resolução dos exercícios.
- **Contexto** O foco da intervenção são alunos que estão cursando disciplinas introdutórias de programação em cursos com foco em Ciência da Computação. Desta forma, é proposta a utilização do TDD pautado em técnicas de teste de software como ferramenta de ensino. A intervenção proposta baseia-se na alteração do ciclo tradicional do TDD, com a finalidade de adicionar uma etapa de criação de casos de teste baseado em critérios de teste de software.

As perguntas visam caracterizar o objeto de medição (produto, processo, recursos) no que diz respeito a um problema de qualidade e para determinar a sua qualidade do ponto de vista selecionado (BASILI, 1992). Com base no objetivo de medição e no ponto de vista proposto, foram definidas as seguintes questões, as quais tem por finalidade verificar a melhoria do desempenho dos indivíduos da intervenção (BASILI; ROMBACH, 1988).

Q01. O código resultante é melhor?
Q02. O conjunto de testes é melhor?
Q03. Estão praticando TDD corretamente?

Q04. Desempenho acadêmico é melhor?

As métricas para avaliação das questões foram definidas com base na classificação de estudos que comparam *test-last* e TDD (MUNIR *et al.*, 2014; KOLLANUS, 2010). As métricas utilizadas nestes estudos foram selecionadas em virtude da relevância no contexto de aplicação e também pela viabilidade de obtenção das informações. Estas foram obtidas a partir de duas fontes: código ou questionário. Foram observados variáveis das categorias: qualidade interna, qualidade externa (ZHANG *et al.*, 2006; CRISPIN, 2006), produtividade (GEORGE; WILLIAMS, 2004; ZHANG *et al.*, 2006), tamanho (JANZEN; SAIEDIAN, 2006a) e opinião(CRISPIN, 2006), tal como definido por Munir *et al.* (2014). A relação entre métrica e questão são dispostas na Tabela 3.1. Esta tabela apresenta quais métricas serão utilizadas para responder as questões definidas, qual a fonte de obtenção de cada métrica e também qual sua classificação.

Tabela 3.1. Métricas utilizadas no GQM

Classificação	**Fonte**	**Métrica**	**Q01**	**Q02**	**Q03**	**Q04**
Qualidade interna	Código	Complexidade do código	X		X	
	Código	Acoplamento	X			
	Código	Coesão	X			
	Código	Cobertura em relação a critérios		X		
Qualidade externa	Código	Quantidade de defeitos	X	X		
	Código	Total de asserções		X	X	
Produtividade	Código	Tempo de desenvolvimento			X	
Tamanho	Código	Tamanho do conjunto de testes		X	X	
Opinião	Questionário	Opinião do aluno				X
	Questionário	Resultados acadêmicos				X

Com base no objetivo, questões e métricas definidas foi elaborado um diagrama (Figura 3.3) que demonstra graficamente com quais questões o objetivo será avaliado e ainda quais as métricas para responder cada questão. Nas próximas subseções, são explicadas as razões para as métricas escolhidas para cada questão.

3.3.1. O código resultante é melhor?

Com o objetivo a responder a Q01, devem ser observadas métricas de qualidade interna (complexidade do código, acoplamento e coesão) e externa (quantidade de defeitos e total de assertivas), produtividade (tempo de desenvolvimento) e tamanho (tamanho do conjunto de testes).

Qualidade interna observa variáveis relacionadas ao *design* de código e atributos relacionados a maneira com que o código foi construído. Desta forma, variáveis desta categoria

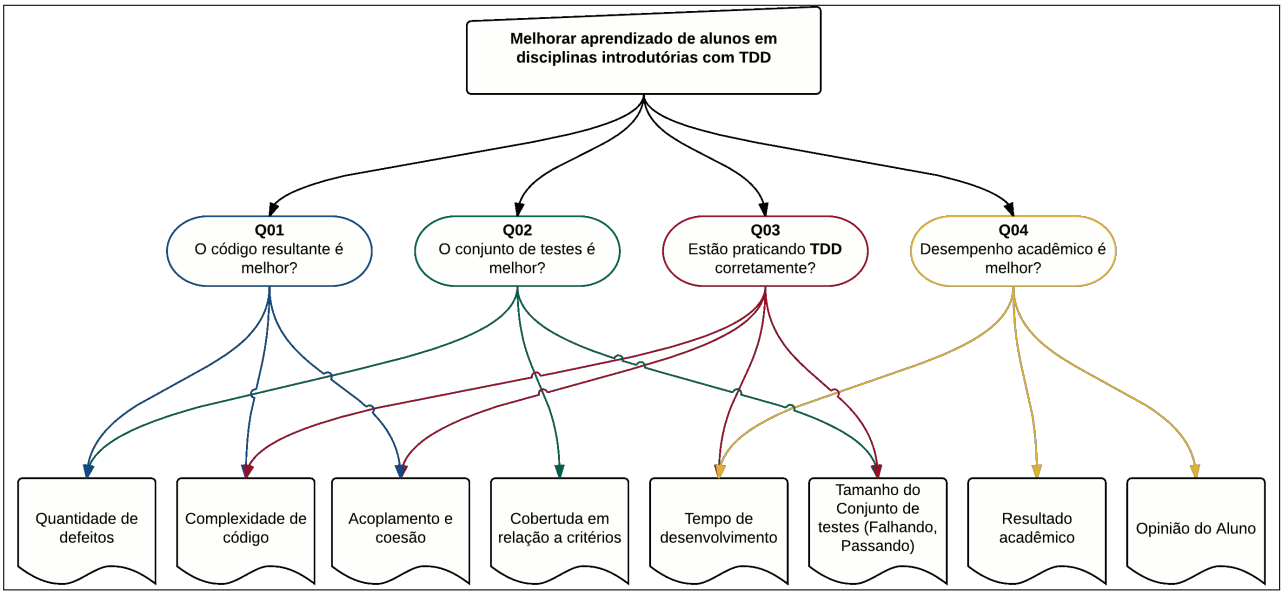

Figura 3.3. Diagrama *Goal Question Metric*.

são comumente utilizadas para avaliar estudos que avaliam a utilização de TDD (GASPAR; LANGEVIN, 2007; HEINONEN *et al.*, 2013; ANICHE; GEROSA, 2012; BECK, 2002; EDWARDS, 2003a).

No que tange as variáveis que avaliam informações quanto à qualidade interna do código (Q01), algumas questões são necessárias para responder esta pergunta.

- O código é menos complexo?
- O código é menos acoplado e mais coeso?
- A cobertura quanto aos critérios utilizados é maior?

Desta forma, no contexto da qualidade interna, o código será melhor quando atingir melhores índices em *Weighted methods per class* (WMC) que é referente a complexidade do código, *Coupling between object classes* (CBO) que mede o acoplamento e *Lack of cohesion in methods* (LCOM) que mensura a coesão (CHIDAMBER; KEMERER, 1994). Estas questões serão utilizadas para caracterizar se de algum modo a técnica proposta impactou negativamente ou potencializou os efeitos comumente atribuídos ao TDD (MUNIR *et al.*, 2014).

Qualidade externa faz referência às características de entrega do software, indicando primariamente informações quanto a defeitos e asserções. Nesta categoria, deverão ser observados: quantidade de defeitos e total de assertivas. Desta forma, as seguintes questões são necessárias para responder quanto à qualidade externa.

- A quantidade de defeitos detectado durante o desenvolvimento foi maior?

Espera-se que, durante o desenvolvimento, o indivíduo detecte mais defeitos em virtude do exercício proposto durante a refatoração. Neste contexto, quanto mais defeitos

localizados durante o processo, melhor o código.

3.3.2. O conjunto de testes é melhor?

Para fornecer subsídios para responder a Q02, deverão ser utilizadas as métricas de qualidade interna (Cobertura em relação a critérios), qualidade externa (quantidade de defeitos e total de assertivas) e tamanho (Tamanho do conjunto de testes).

A primeira forma de análise proposta neste estudo é a observação da quantidade de asserções (qualidade externa), tanto na falha quanto no sucesso, visto que na primeira etapa do TDD deve ocorrer uma falha em virtude da característica da técnica. Após a falha, o conjunto deve ser executado com sucesso. Desta forma, caracteriza-se o fluxo de execução correto do TDD. Todavia, no contexto de avaliação da qualidade do conjunto de testes, é importante verificar o esforço mediante quantidade de execuções durante o processo de desenvolvimento e a detecção de defeitos.

- A quantidade de asserções definidas durante o desenvolvimento do praticante da intervenção é maior que o praticamente do TDD tradicional?

Produtividade em geral é definida como a relação entre saída e entrada (PETERSEN, 2011), apontando a capacidade de produção do indivíduo. Desta forma, uma das variáveis que pode ser utilizada é a quantidade de casos de teste desenvolvido durante o processo de desenvolvimento. Todavia, é importante salientar que o simples tamanho do conjunto não é significante para a qualidade do conjunto de casos de teste. Deste modo, faz-se necessário adicionar outros atributos com a finalidade de caracterizar um bom conjunto de casos de teste. A quantidade de casos de teste desenvolvido por um acadêmico durante o TDD pode ser observada de diversos panoramas. Todavia, na avaliação quanto a técnica proposta, espera-se que o conjunto de casos de teste desenvolvido pela intervenção proposta seja menor, entretanto com maior qualidade. Então, utilizando variável de outro contexto (qualidade interna), espera-se que seja obtida maior cobertura com menor quantidade de casos de teste.

- O tamanho do conjunto de testes é menor, entretanto, com a cobertura maior?

Para o presente estudo, a eficácia é um dos pontos com impacto significativo, assim, um ponto de atenção é quanto a quantidade de defeitos localizados durante o processo de desenvolvimento. Verificar se a quantidade de defeitos (qualidade externa) encontrados foi maior utilizando a intervenção evidencia a eficácia ante outras técnicas.

- A quantidade de defeitos encontrados é maior?

A união das três questões, as quais observam a eficácia e a eficiência, corrobora para a comparação entre os grupos de intervenção e controle. Espera-se que o indivíduo da

intervenção tenha maior quantidade de asserções em virtude do fluxo de refatoração proposto. Todavia, este fluxo colabora com um conjunto de casos de testes menor, entretanto com maior cobertura e que evidencie maior quantidade de defeitos.

3.3.3. Estão praticando TDD corretamente?

A intervenção proposta no presente trabalho tem como técnica base o TDD e por esse motivo é necessário que seja possível verificar qual foi o fluxo de desenvolvimento do aluno durante a execução experimental, verificando se o TDD foi executado corretamente. Desta forma, a cada iteração deve ser capturada a etapa atual do TDD e as demais informações da execução do conjunto de casos de teste. Deste modo, com a possibilidade de acompanhamento de qual o ciclo seguido pelo aluno é possível verificar se as etapas têm as características definidas pelo TDD.

- Com base na sucessão das etapas do TDD e as características do conjunto de casos de teste, o aluno praticou a técnica corretamente?

Logo, é necessário que seja analisado, a linha do tempo de execução das etapas do TDD que o participante executou durante o estudo experimental, bem como seja verificado se as características básicas de cada etapa são contempladas:

- *RED:* Deve ser introduzido ao menos um caso de teste, o qual deve falhar na execução;
- *GREEN:* Deve ser introduzido código para que todos os casos de teste sejam executados com sucesso;
- *Refactor:* A última execução da refatoração deve ter a complexidade melhor que a inicial;

Ao longo do desenvolvimento o conjunto de casos de teste deve crescer e a cada um dos ciclos deve ter tempo de permanência relevante. Com base no tempo que o aluno executa cada uma das etapas, é possível observar se está agindo na tentativa e erro, algo que se pretende mitigar.

3.3.4. Desempenho acadêmico é melhor?

Grande parte dos estudos que utilizam TDD como intervenção buscam avaliar a satisfação ou rejeição do aluno quanto a execução da técnica proposta. A avaliação do desempenho acadêmico (Q04) deve observar os resultados acadêmicos do aluno, bem como a avaliação que o professor faz da proposta. Estes dados serão fornecidos pelo professor da disciplina em que o projeto experimental for executado.

- Com base em sua experiência como professor da presente disciplina, a técnica proposta atingiu seu objetivo?
- Com base na opinião do aluno, o software produzido é melhor?

Com base nos dados qualitativos apurados a questão será caracterizada com a finalidade de verificar se o resultado acadêmico foi satisfatório. É possível que em dadas circunstâncias a técnica produza resultados positivos, mas não convença o professor responsável.

3.4. Protocolo dos estudos

De forma geral, este estudo deve ser aplicado a acadêmicos que estão iniciando no desenvolvimento de software em cursos de Ciência da Computação.

O método de investigação experimental escolhido para esta proposta foi o estudo controlado, modelado tal como proposto por Pfleeger (1994) (PFLEEGER, 1995a, 1995d, 1995b, 1995c). O estudo controlado foi escolhido por propiciar: controle na execução, controle na medição, controle da investigação, facilidade para repetição. Além disso ele é apropriado para confirmar teorias, tal como apresentada nesta dissertação. As seguintes hipóteses foram definidas para este estudo:

- Hipótese: A qualidade do código de teste, e consequentemente do código de produção, pode ser melhorada com a adoção de critérios de teste de software aliado a prática do TDD.
- Hipótese nula: A adoção de critérios de teste de software em conjunto com o TDD não traz melhorias relevantes ao código de produção.

Com a finalidade de avaliar as hipóteses definidas foi definido um estudo experimental que é composto pelas etapas: questionário de conhecimentos, divisão dos grupos, treinamento, desenvolvimento do software, questionário de avaliação e avaliação dos resultados. Tal como expostas em Figura 3.4, cada uma das etapas é executada sequencialmente.

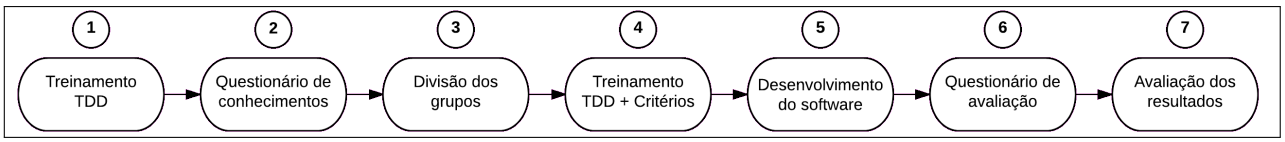

Figura 3.4. Fluxo de execução do estudo experimental.

Desta forma, inicialmente os acadêmicos devem ser treinados (Etapa 1) [1] no fluxo tradicional TDD e ainda devem conhecer o IDE de desenvolvimento (Etapa 5) e os *plugins* que

[1] Material disponível em: `http://www.magsilva.pro.br/projects/tdd/`.

serão utilizados. Neste contexto, é importante salientar que o ambiente de desenvolvimento utilizado em ambos os grupos será o mesmo.

Após o treinamento, os alunos devem responder um questionário (Etapa 2) que captará informações com o objetivo de traçar um perfil dos indivíduos submetidos ao estudo experimental, identificando experiências prévias em Computação, TDD e técnicas de teste de software. A partir do perfil, os acadêmicos devem ser separados aleatoriamente em dois grupos (Etapa 3): "Grupo C" e "Grupo I".

- "Grupo C" (controle): deve ser responsável por desenvolver um software utilizando o fluxo tradicional do TDD.
- "Grupo I" (intervenção): deve ser responsável por desenvolver um software utilizando o TDD pautado em critério de teste de software.

Após a divisão dos grupos, o grupo de intervenção (Grupo I) deve ser treinado (Etapa 4) para utilizar o fluxo de desenvolvimento proposto para o TDD pautado em critérios de teste de software (Seção 3.2).

O software a ser desenvolvido (Etapa 5) deve ser dividido em iterações, as quais estabelecem um conjunto de funcionalidades a serem implementadas, alteradas ou removidas do software durante a execução do estudo experimental. O processo inicia-se com o desenvolvimento da primeira iteração. As demais iterações devem ser reveladas a partir da conclusão da primeira e assim sucessivamente. A partir da segunda iteração podem ser propostas alterações em funcionalidades já especificadas.

Cabe destacar que os participantes não devem ter conhecimento de todas as iterações que serão propostas (alterações ou novas implementações) desde o início do experimento. Tal como proposto por Shelton *et al.* (2012), deve ser simulado um ambiente típico de desenvolvimento de software. As informações pertinentes às próximas iterações devem ser disponibilizadas apenas quando o indivíduo (ou grupo) terminar o ciclo de refatoração, para que assim o ciclo seja reiniciado.

Os dois grupos desenvolverão a mesma atividade, entretanto utilizando fluxos de desenvolvimento distintos (com ou sem critérios de teste). O "Grupo C" não deve saber como o "Grupo I" desenvolverá sua solução e vice-versa. Tal postura visa proteger o experimento de possível influência na motivação com que indivíduo efetuará as implementações propostas no experimento.

Após o desenvolvimento do software proposto, cada acadêmico deve responder um questionário (Etapa 6), permitindo a captura de informações quanto a sua opinião sobre a efetividade da técnica proposta e a experiência como: praticidade de aplicação da técnica e se o ciclo do TDD tradicional foi interrompido mediante a alteração proposta. Visto que tais

informações inicialmente não podem ser coletadas a partir do código ou de variáveis coletadas durante a aplicação do experimento. Além disso serão validadas informações do primeiro questionário.

A última etapa do estudo experimental é a avaliação dos resultados (Etapa 7). Esta etapa utilizará os dados que serão capturados nas etapas de questionário de conhecimento (Etapa 2), desenvolvimento do software (Etapa 5) e questionário de avaliação (Etapa 7). A hipótese definida neste estudo experimental será avaliada segundo as questões e métricas definidas na Seção 3.3.

Algumas práticas podem invalidar a execução do indivíduo para a avaliação dos resultados deste estudo. Neste contexto, os estudos que contiverem casos de teste falhando ou soluções que contenham erro, casos que impossibilitam a avaliação, devem ser desconsiderados. A combinação desses dados com demais informações do projeto, possivelmente capturados com auxílio de ferramentas, auxiliam o tratamento desta questão. Outro fator que será considerado durante a análise é a execução segundo a técnica proposta durante o estudo experimental. Se a prática destoar do proposto o estudo poderá ser desconsiderado.

3.5. Considerações finais

O presente capítulo definiu o modelo geral da estratégia que é baseado na intersecção entre TDD e técnicas de teste de software para aplicação no contexto acadêmico. Foi definida uma instância que utiliza critérios estruturais baseados em fluxo de controle para derivar o conjunto de casos de teste tal como definido no fluxo proposto. Desta forma, com base na estratégia e o objetivo desta dissertação foi definido um GQM para auxiliar na definição das questões a serem analisadas e as métricas utilizadas para responder as questões.

A partir da definição do GQM, foi desenvolvida a ferramenta TDD Tracking para a captura das medidas referente às métricas estabelecidas durante o desenvolvimento do software proposto na execução do estudo experimental. Para facilitar a análise também foi desenvolvido um módulo de análise que colabora com a organização e sumarização dos dados capturados.

Nos próximos capítulos serão apresentadas a ferramenta e as execuções experimentais que utilizam o projeto experimental definido neste capítulo.

Capítulo 4

Ferramenta TDD Tracking

O conjunto de ferramentas TDD Tracking é um conjunto de ferramentas que foi proposto com a finalidade de suprir a necessidade de uma ferramenta que obtenha as informações estabelecidas com o GQM (Seção 3.3) e as organize para que estes dados possam ser analisados posteriormente.

A motivação para o desenvolvimento da ferramenta em questão surgiu da proposta da utilização de técnicas de teste de software em conjunto com o TDD. A ferramenta foi proposta com a finalidade de monitorar o comportamento do indivíduo durante o desenvolvimento observando a evolução da solução proposta a cada execução do conjunto de casos de teste, obtendo informações detalhadas sobre o código. No contexto desta dissertação, a simples observação durante o processo de desenvolvimento não é uma opção plausível em virtude da escala que se pretende aplicar estudos experimentais.

A ferramenta é dividida em três partes: plugin, servidor e ferramenta de análise. Como pode ser observado em Figura 4.1, cada um dos módulos da ferramenta captura ou fornece informações para a atividade subsequente.

Os módulos da ferramenta são utilizados durante o desenvolvimento da solução e posteriormente na análise dos resultados. Desta forma, o *plugin* e o servidor são utilizados durante o desenvolvimento do software proposto no estudo experimental. Durante este período, a cada execução do conjunto de casos de teste, a ferramenta capta informações tais como: estágio atual do TDD (RED, GREEN ou *Refactor*), código-fonte atual, horário e ainda todas informações geradas por dois plugins disponíveis para Eclipse, JUnit e Eclemma. Todos os dados são armazenados em uma pasta oculta no projeto criado no eclipse. O *plugin* ainda gerencia o envio de todas as informações para o servidor.

O servidor atua como um repositório, que recebe todas as informações captadas em

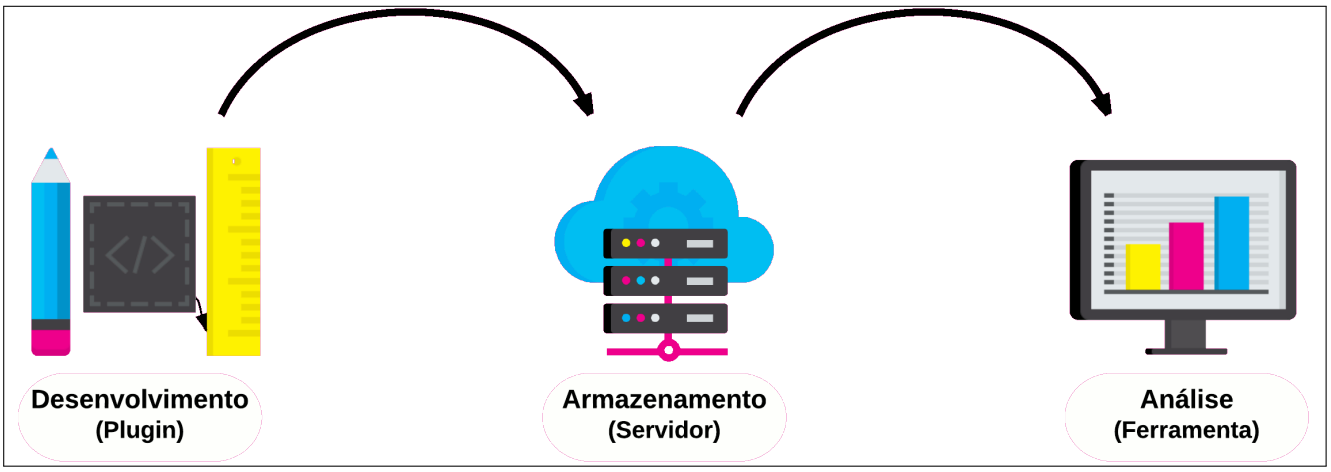

Figura 4.1. Ilustração da ferramenta de captura de informações.

todos os projetos que estão sendo desenvolvidos por meio de serviços *Representational State Transfer* (REST) e os armazena em uma base relacional (APACHE Derby). O servidor é um módulo opcional visto que o *plugin* armazena as informações captadas na mesma estrutura do projeto do Eclipse. Todavia, o intuito é de prover facilidade na obtenção dos arquivos desenvolvidos durante a execução do experimento.

A ferramenta de análise tem objetivo de sumarizar as informações de cada um dos projetos desenvolvidos pelos participantes durante o experimento. Esta ferramenta possibilita ainda adicionar informações que não são captadas durante o desenvolvimento, tais como; horário de início de cada iteração, grupo do participante (controle ou intervenção) ou ainda marcar o projeto como inválido (desta forma o descartando).

Com todos os dados definidos, esta demonstra graficamente a linha do tempo de desenvolvimento do aluno contendo informações do estágio do TDD, período de desenvolvimento (tempo), cobertura dos critérios de teste (qualidade interna), quantidade de casos de teste, asserções e horário de início de cada iteração.

4.1. *Plugin*

O software de captura das informações foi desenvolvido como um *plugin* [1] para o IDE Eclipse. Primeiramente, para iniciar a operação, é necessário que a captura seja "formalmente" iniciada. O desenvolvedor deve criar um projeto Java no Eclipse. Assim que o projeto estiver criado é necessário que o projeto seja selecionado e, utilizando o botão direito selecione a opção "TDD Criteria" e então "Inicializar o TDD Criteria". Se o servidor for utilizado no processo de desenvolvimento para armazenamento das informações captadas, é necessário que o aluno configure o ip do servidor utilizando o menu de contexto do projeto, selecionando a opção

[1] Disponível em: https://github.com/bhpachulski/TddCriteriaPlugin

"TDD Criteria" e então "Alterar IP do Servidor".

Durante o desenvolvimento do software, ao seguir o fluxo do TDD, o aluno executa os casos de teste utilizando os botões que são dispostos pelo *plugin* no IDE do Eclipse, tal como pode ser observado na Figura 4.2. Estes plugins são referentes a cada uma das etapas do TDD, sendo eles: *RED*, *GREEN* e *Refactor*.

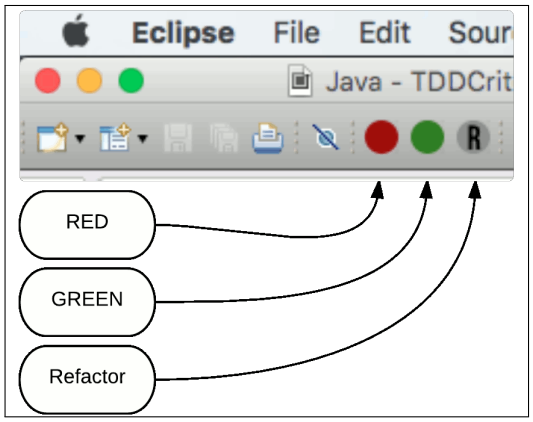

Figura 4.2. Ferramenta: botões de captura de estágio do TDD.

Ao acionar o botão, o *plugin* registra o estágio atual do TDD de acordo com o botão acionado, informações fornecidas pelo JUnit e Eclemma e ainda mostra para o indivíduo na IDE as ações padrão do Eclemma e JUnit. Neste mesmo momento são criados os arquivos de acompanhamento, os quais são:

- Arquivo de acompanhamento do TDD: arquivo onde são registrados o estágio do TDD que o usuário está executando e o horário.
- Relatório do JUnit: foi implementada uma estrutura do Eclipse que trabalha como um gatilho para o JUnit (*Extension Point*). Desta forma, ao final da execução dos casos de teste, o resultado é sumarizado em um objeto Java (TestSuiteSession) e este é transformado em XML.
- Relatório do Eclemma: foi necessário modificar o *plugin* Eclemma para que, ao final da execução, o relatório no formato XML fosse persistido. Este relatório é padrão do plugin, entretanto não oferece o recurso de extensão tal como o do JUnit.

Como exemplificado na Figura 4.3, após o aluno executar o conjunto de casos de teste utilizando os botões (Figura 4.2) destinados a execução do TDD, são exibidas as informações dos *plugins* JUnit e Eclemma. Além disso, o Eclemma demonstra no código a porção coberta e não coberta pelo conjunto de casos de teste.

Pelo fato do *plugin* armazenar as informações coletadas no projeto do Eclipse, é possível sua operação sem conexão com a Internet (*offline*) e, quando houver a necessidade de sincronização ou a disponibilidade do servidor na rede, todas as informações sejam

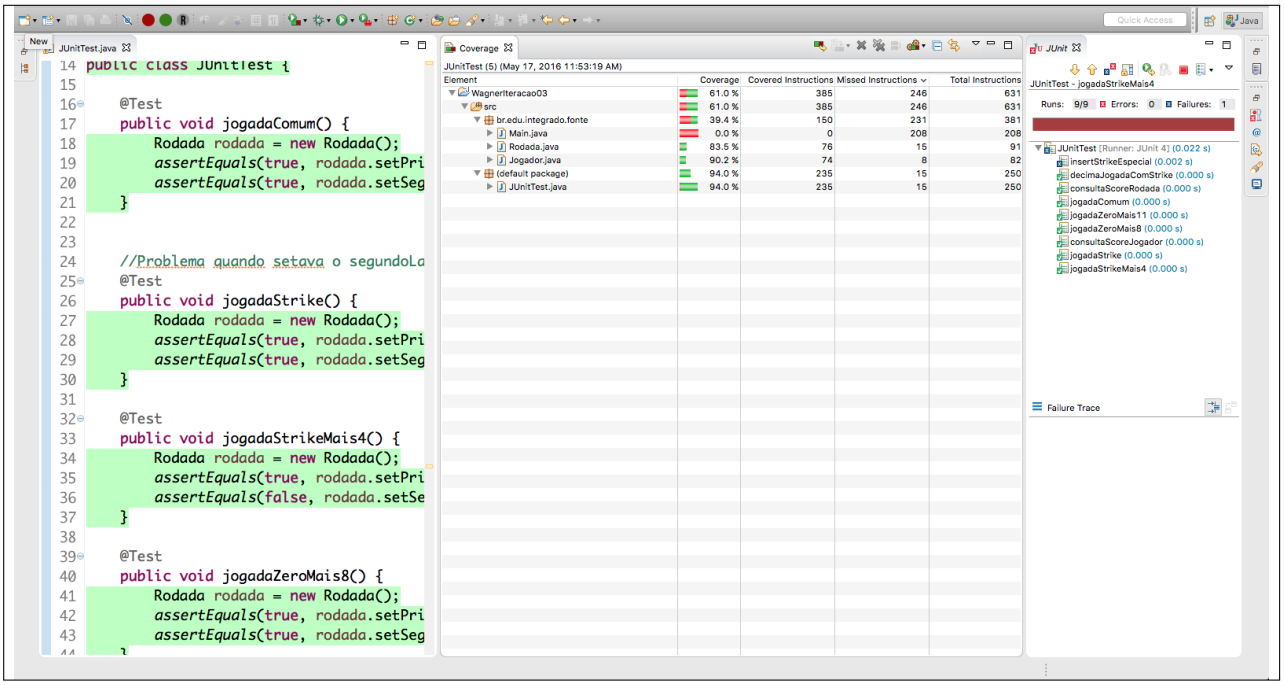

Figura 4.3. Execução do *plugin*.

sincronizadas. Caso não seja utilizado o servidor, é necessário que o projeto seja copiado para que o software de análise exiba seus relatórios.

A estrutura de armazenamento dos arquivos que o *plugin* utiliza está disposta na Figura 4.4. Nela é possível observar que, a partir do momento que o *plugin* é iniciado, a estrutura básica é criada, a qual contém arquivos de configuração, registro de eventos (log) e diretórios que abrigarão os arquivos de acompanhamento. A partir da criação, a cada execução do conjunto de casos de teste os arquivos de acompanhamento dos estágios do TDD, JUnit e Eclemma serão gerados e armazenados mesmo se utilizando os botões fornecidos pelo *plugin* (*RED*, *GREEN* e *Refactor*).

4.2. Servidor

O servidor opera como um facilitador para a organização dos arquivos de acompanhamento gerados pelo *plugin*. Tal como exemplificado na Figura 4.5 o servidor fornece um serviço REST que recebe todos os arquivos gerados no decorrer da utilização do *plugin*. Todos os arquivos recebidos são armazenados conforme a classificação indicada pelo *plugin*, as quais são referentes a: participante (identificador único), nome do projeto, estágio do TDD, tipo do arquivo (JUnit, Eclemma) e informações sobre eventos.

Ao final da execução experimental ou da prática de exercícios utilizando a ferramenta em conjunto com servidor, o servidor fornece links para download dos dados de cada um dos participantes por meio de um *Uniform Resource Locator* (URL).

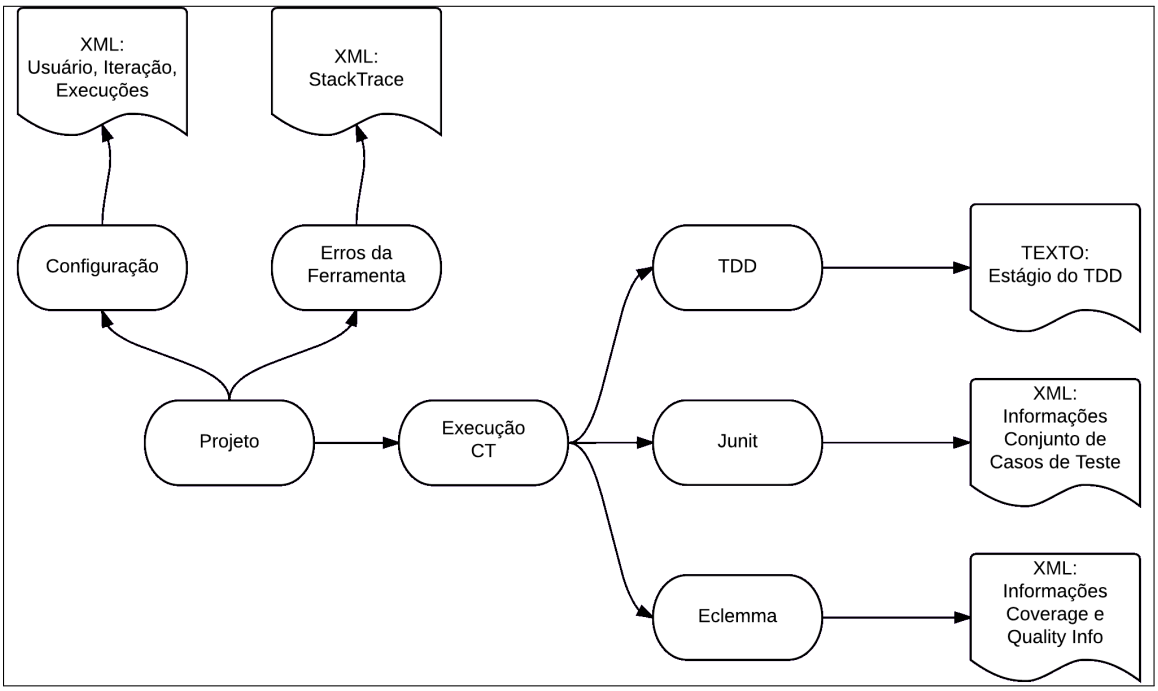

Figura 4.4. Ferramenta: estrutura de armazenamento.

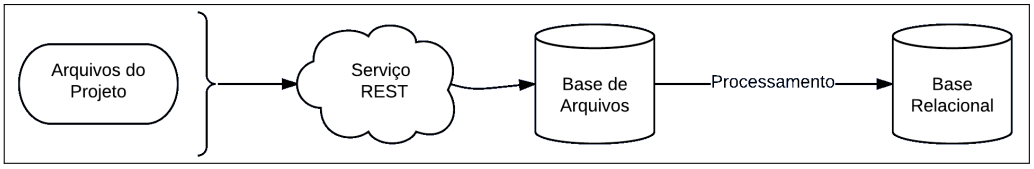

Figura 4.5. Ferramenta: serviço REST do servidor.

4.3. Ferramenta de Análise

Para utilizar a ferramenta de análise o pesquisador terá que selecionar o diretório base dos arquivos e executar a leitura dos arquivos. Após coletar os dados para análise, é necessário selecionar a pasta base do projeto do Eclipse para que os dados sejam exibidos, tal como exemplificado na Figura 4.6. A ferramenta de análise fornece, após a execução, dois gráficos que demonstram o ciclo de desenvolvimento do projeto com base em duas dimensões, casos de teste e cobertura. As duas dimensões inicialmente demonstram o fluxo de desenvolvimento do projeto, sendo organizado pelo horário de início com término na última execução do conjunto.

Na visão de "Casos de teste", como pode ser observado na Figura 4.7 o valor vertical é referente à quantidade de casos de testes e, ao utilizar o mouse para verificar cada uma das execuções, são informados dados da execução quanto a: estágio do TDD, data e hora da execução, cobertura de instruções e ramos, quantidade de casos de teste (CT), quantidade de casos de teste executados com falha (CT F) e com sucesso (CT S) e quantidade de linhas do software naquele estágio. Este gráfico visa demonstrar de forma simplificada a linha do tempo

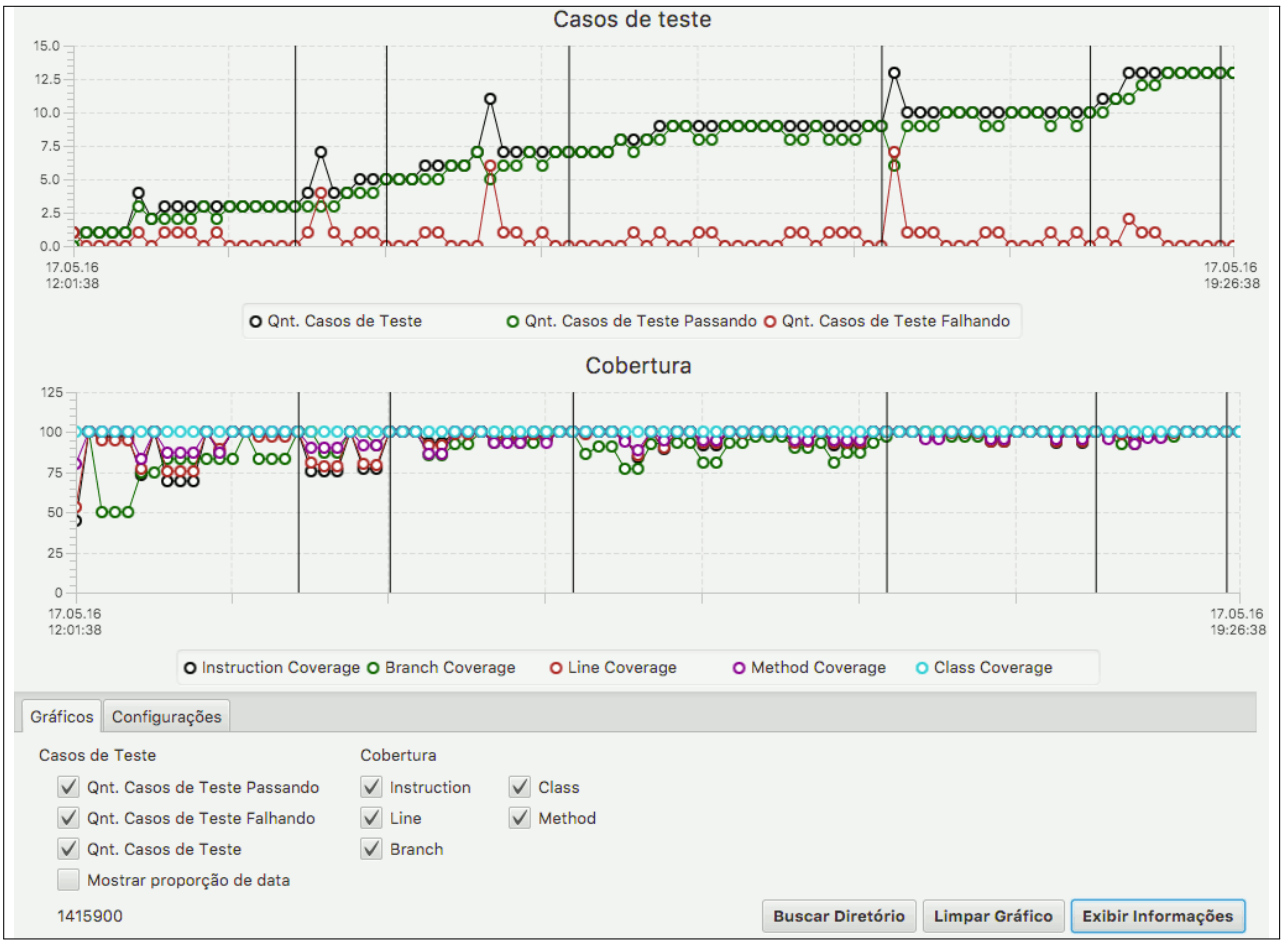

Figura 4.6. Ferramenta de análise das informações.

do desenvolvimento do projeto de um estudante, fornecendo subsídios para a análise do fluxo do TDD. Na Figura 4.7 observa-se o desenvolvimento de um software utilizando TDD em relação a quantidade de casos de teste. A linha verde é referente a quantidade de casos de teste executados com sucesso, a vermelha quanto a quantidade de casos de teste que falharam e a preta é referente a quantidade total de casos de teste. Neste exemplo, é possível observar que ao longo do desenvolvimento do software a quantidade de casos de teste aumentou. A execução em destaque demonstra que a etapa executada naquele momento foi a *RED*. Desta forma, neste momento o conjunto de casos de teste era composto por 5 casos de teste dos quais, 4 foram executados com sucesso e um falhou.

O gráfico de análise de cobertura (vide Figura 4.8) é suporte para o primeiro, fornecendo informações sobre cobertura de instruções, ramos, linha, métodos e classe. A união das duas visões da linha de desenvolvimento do aluno provê subsídios para a avaliação da pratica do TDD bem como da aplicação do TDD pautado em critérios de teste de software.

Como supracitado, é possível alterar algumas configurações no projeto, tais como observável na Figura 4.9. Deste modo, após a captura das informações é possível:

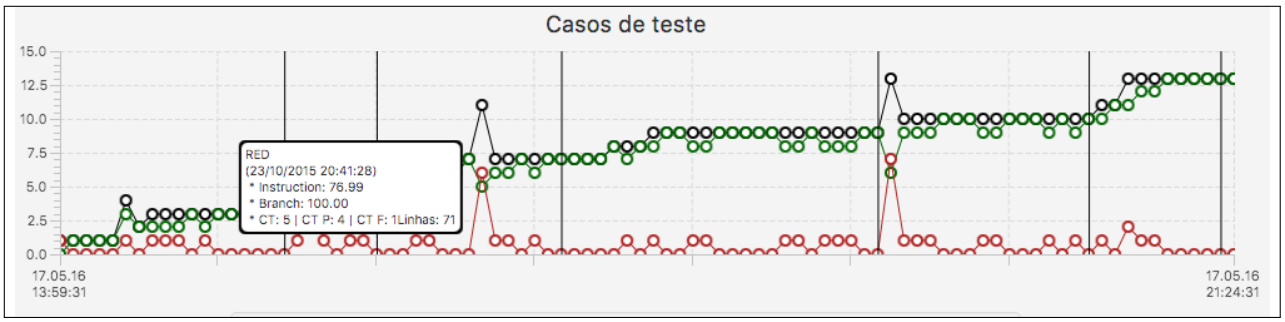

Figura 4.7. Gráfico de análise do fluxo do TDD.

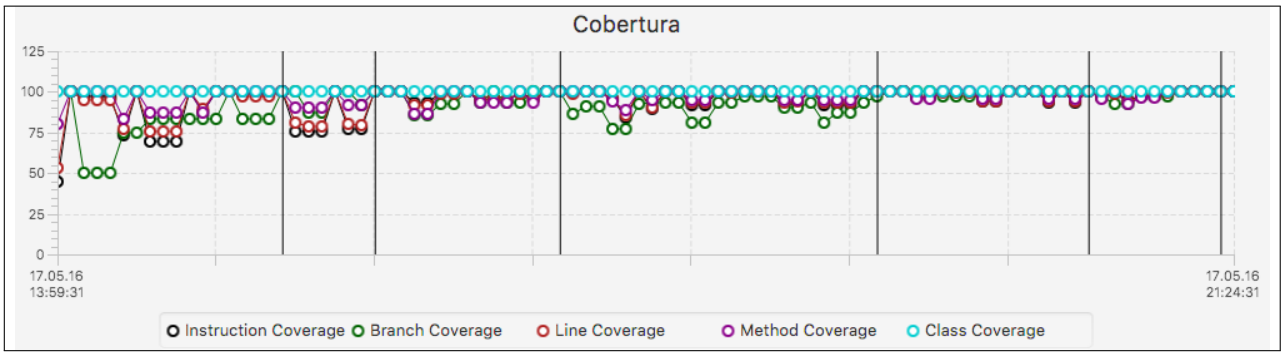

Figura 4.8. Gráfico de análise de cobertura.

1. Selecionar com base no horário das execuções qual foi a execução que encerra a iteração;
2. Selecionar qual o grupo que o projeto faz parte: intervenção ou controle;
3. Marcar o projeto como excluído do projeto experimental, com a finalidade de que seja ignorado em pesquisas que observem diversos projetos de apenas uma vez.

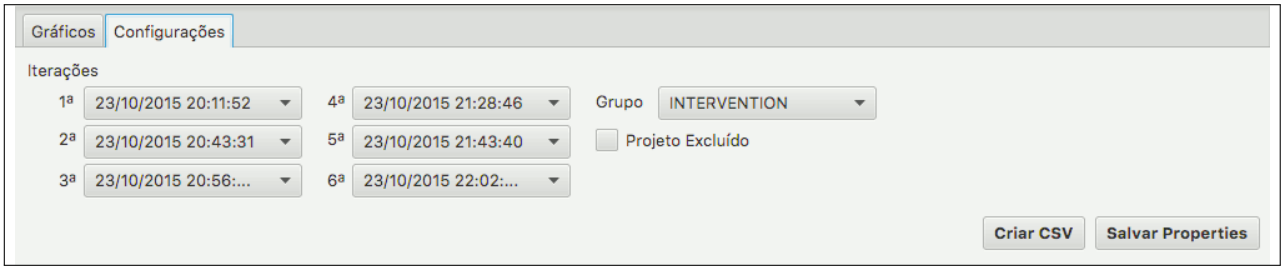

Figura 4.9. Segunda aba de configurações da ferramenta de análise.

Como apresentado, a ferramenta tem por objetivo dar suporte a execução de experimentos científicos que busquem avaliar o desenvolvimento com olhar no conjunto de casos de teste. O foco inicial previsto é o TDD, todavia, é possível utiliza-la também em outro contexto, visto que não é obrigatório o uso dos botões *RED*, *GREEN* e *Refactor*.

4.4. Ferramentas relacionadas

Existem algumas ferramentas com o objetivo de apoiar a prática do TDD ou de monitorar o desenvolvimento de aplicações utilizando teste de software. Todavia, não foi encontrada uma solução que satisfazia as necessidades especificadas na Seção 3.3.

No contexto do TDD foi encontrado um *plugin* para o Eclipse chamado Pulse que monitora a maneira que o desenvolvedor está praticando o TDD, demonstrando graficamente o ciclo do TDD e cada uma de suas etapas. Além disso é possível visualizar o tempo despendido em cada uma das etapas e ao final é possível gerar uma imagem com toda a sessão de prática. Neste *plugin* não é necessário que o praticante utilize um botão específico para inferir a etapa do TDD que está sendo praticado. Desta forma, com base no resultado dos casos de teste e no código é inferida a etapa que o programador está praticando.

Com o foco em teste de software, Beller *et al.* (2015b) propuseram a ferramenta WatchDog disponível para os IDE Eclipse e IntelliJ. Esta viabiliza medir o tempo que o desenvolvedor utiliza desenvolvendo código-fonte e código de teste, qual a periodicidade em que o desenvolvedor executa seu conjunto de casos de teste, quanto tempo ao todo o desenvolvedor gastou rodando o conjunto de casos de teste e ainda quais as reações do programador quando os casos de teste falham. Recentemente foi adicionada também a análise do ciclo do TDD (BELLER *et al.*, 2015a). Desta forma, observando-se o programador segue corretamente o fluxo proposto e desenvolve o código como é necessário. Entretanto toda informação capturada pelo WatchDog é enviada para um repositório da equipe de desenvolvimento, não sendo disponibilizado para cruzamento com outras informações capturadas por outras ferramentas.

Outra iniciativa de coleta de dados é o *BlueJ Blackbox Data Collection Project*, o qual tem por objetivo coletar dados de como a IDE BlueJ é utilizada, com a finalidade de entender como os estudantes aprendem a programar. Além do código fonte são enviadas informações quanto ao uso do módulo onde o código fonte é escrito, uso do depurador e das demais funcionalidades fornecidas pelo IDE. Estas informações são enviadas para os pesquisadores com a finalidade de melhoria e ampliação das funcionalidades da ferramenta e ainda da análise e desenvolvimento de pesquisa científica. Especificamente quanto a testes unitários, a ferramenta registra apenas dados em relação a edição do código e a quantidade de vezes que o conjunto de casos de teste é executado (BROWN *et al.*, 2014).

Assim como o BlueJ, com foco em disciplinas iniciais de programação e a utilização de técnicas de teste de software, destacam-se as ferramentas de avaliação automática. O objetivo é que o aluno submeta resoluções de exercícios e receba um retorno rápido se seus programas estão atingindo a qualidade mínima requerida pela atividade. Normalmente avalia-se quanto

à validade e integridade dos casos de teste, estilo de codificação e qualidade do código do aluno (EDWARDS, 2003b, 2003b, 2004b; THORNTON *et al.*, 2007; EDWARDS; PéREZ-QUIñONES, 2007; THORNTON *et al.*, 2008; BUFFARDI; EDWARDS, 2012; SOUZA *et al.*, 2011a). Entretanto as ferramentas não captam as informações com a finalidade de obter as métricas necessárias a fim de responder as questões definidas na Seção 3.3.

4.5. Limitações e futuras implementações

Durante a primeira utilização da ferramenta com alunos a ferramenta comportou-se tal como esperado. Todavia é necessário ressaltar alguns pontos observados durante a experiência.

Atualmente a ferramenta não obriga o aluno a seguir o fluxo correto do TDD. Durante o desenvolvimento é possível que o aluno siga o fluxo incorretamente. Em virtude do objetivo educacional, é necessário que a ferramenta ajude o acadêmico a seguir o ciclo corretamente (*RED -> GREEN -> Refactor*). Objetiva-se que tal funcionalidade possa ser habilitada e desabilitada, fornecendo desta forma duas formas de operação.

Pretende-se também implementar a validação dos estágios do TDD durante o desenvolvimento com o objetivo de guiar o acadêmico na correta execução do ciclo do TDD. A verificação do ciclo é feita por observação, desta forma, pretende-se automatizar esta tarefa. Por exemplo, para que a etapa *RED* seja válida, é necessário que exista ao menos um caso de teste falhando, bem como na etapa *GREEN* não existam casos de teste falhando. No *Refactor* tradicional não existem muitas obrigatoriedades. Entretanto, no caso do TDD com critérios, a cobertura de critérios estruturais pode ser utilizada como uma forma de validação da refatoração, aumentando as evidências para responder o questionamento sobre a correta execução do TDD.

A operação de análise necessita ser aperfeiçoada, a fim de que leia a partir dos registros no servidor os projetos desenvolvidos. Além disso, as iterações de início e término de cada iteração são informadas pelos acadêmicos durante o desenvolvimento da aplicação e posteriormente apontada utilizando a ferramenta. Objetiva-se que o início das iterações seja apontado com a utilização do *plugin* e ainda que a ferramenta de análise seja capaz de inferir o início e final das iterações a fim de comparação.

No cenário atual é necessário que seja adicionado ao Eclipse a última versão do JUnit e uma versão modificada do *plugin* Eclemma. Foi necessário implementar algumas modificações com a finalidade de gerar arquivo de acompanhamento do Eclemma. Desta forma, vemos a necessidade de que sejam implementados *Extension Points* no Eclemma que permitam a extensão de forma similar à possibilitada pelo *plugin* JUnit.

Capítulo 5

Resultados

Este capítulo apresenta os resultados obtidos nas execuções experimentais aplicadas com a finalidade de verificar a hipótese levantada na Seção 3.4. Foram executados três estudos preliminares objetivando o refinamento do estudo experimental e a definição da estratégia de TDD com critérios de teste de software, as quais são apresentadas na Seção 5.1. Os treinamentos, atividades, estratégias de teste empregadas e resultados também são discutidos. Após a apresentação dos estudos preliminares na Seção 5.1.5 são apresentados o que foi possível observar com as execuções e o que foi trabalhado no desenvolvimento do protocolo final de execução. O protocolo final, apresentado na Seção 3.4, foi executado e é apresentado na Seção 5.2, em que são discutidos o treinamento, a atividade desenvolvida, a estratégia utilizada e os resultados. Por final, são discutidas as ameaças à validade.

5.1. Avaliações preliminares

Com o intuito de avaliar e refinar o experimento proposto e a estratégia preliminar, foram aplicados três experimentos. A primeira execução foi aplicada em abril de 2014 para acadêmicos do quinto período do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, da Faculdade Integrado de Campo Mourão/Paraná. A segunda execução do experimento foi desenvolvida junto aos acadêmicos do *Programa de Pós-Graduação em Informática* (PPGI) da Universidade Tecnológica Federal do Paraná (UTFPR), Campus Cornélio Procópio, durante a disciplina de Tópicos em Engenharia de Software. A terceira execução foi executada em novembro de 2014 para acadêmicos do sexto período do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, da Faculdade Integrado de Campo Mourão/Paraná. Tais execuções são descritas e os pontos fortes e fracos serão discutidos.

5.1.1. Execução 01: alunos avançados da Faculdade Integrado

Neste estudo, foram selecionados acadêmicos da Faculdade Integrado de Campo Mourão, matriculados no quinto período do curso de Tecnologia em Análise e Desenvolvimento de Sistemas. Os alunos receberam um breve treinamento para desempenhar a técnica proposta.

Para a garantir o equilíbrio, os dois grupos possuíam o mesmo número de acadêmicos, selecionados aleatoriamente. Desta forma, os grupos configuraram-se tal como apresentado na Tabela 5.1. As médias de notas de disciplinas técnicas e exatas são aproximadas, o que demonstra o equilíbrio quanto ao conhecimento e aproveitamento acadêmico.

Tabela 5.1. Relação de participantes da primeira execução.

Aluno	**Grupo**	**Perfil**	**Média Técnica**	**Média Exatas**
Aluno 01	C	Aluno	8,8	8,5
Aluno 02	C	Aluno	8,0	8,1
Média	C	-	**8,4**	**8,3**
Aluno 09	I	Aluno	8,2	8,0
Aluno 10	I	Aluno	8,1	8,3
Média	I	-	**8,1**	**8,1**

*Os alunos do Grupo I (intervenção) iniciam no 09 em virtude da relação estabelecida para comparação na Tabela 5.6.

5.1.1.1. Treinamento

Os acadêmicos selecionados assistiram uma videoaula de TDD, que demonstrou o processo clássico de desenvolvimento de software e posteriormente demonstrou a aplicação da prática do TDD e seus benefícios. No decorrer do vídeo, o instrutor desenvolve um programa exemplo que também foi implementado pelos alunos. Posteriormente foi proposto outro exercício para os envolvidos que foi desenvolvido utilizando os conceitos recém adquiridos.

Os códigos de implementação e os casos de teste provenientes do desenvolvimento dos exercícios foram avaliados para verificar se a solução foi escrita corretamente. Foi possível observar que os quatro alunos que participaram da etapa de treinamento obtiveram resultados satisfatórios, ou seja, o conhecimento foi comprovado. Desta forma, foi possível validar que só participaram do experimento indivíduos que possuíam proficiência em TDD.

5.1.1.2. Atividade desenvolvida

O software a ser desenvolvido durante o experimento é baseado no proposto por Martin (2014) e é conhecido como *Bowling Game*, sendo que sua funcionalidade principal é calcular a pontuação (*score*) de um jogador em um jogo de boliche. Este software é um exemplo

utilizado amplamente disseminado na comunidade praticante, utilizado para se iniciar no desenvolvimento com TDD e também em outros projetos experimentais que avaliam a prática do TDD (CAUSEVIC *et al.*, 2013a; XU; RAJLICH, 2006). O software foi simplificado e dividido em três iterações, as quais abrangem os cenários abaixo:

- Iteração 01: O jogo consiste em 10 rodadas e, a cada rodada, o jogador tem direito a dois lances para atingir o a pontuação máxima (10). O *score* do jogo pode ser consultado a qualquer momento;
- Iteração 02: Se na primeira tentativa a pontuação máxima for atingida (*strike*), o jogador não terá direito ao segundo lance, entretanto a pontuação da próxima rodada será dobrada;
- Iteração 03: O jogador terá duas jogadas extra se fizer *strike* na décima rodada. Se a pontuação máxima for atingida nas duas tentativas (*spare*) da última rodada o jogador terá direito a uma jogada extra.

5.1.1.3. Técnica de teste de software empregada

O ciclo do TDD pautado em critérios de teste para desenvolver o software proposto é iniciado por uma nova funcionalidade e segue o ciclo bem como definido tradicionalmente. Quando a etapa de refatoração for cumprida, é proposto que sejam implementados casos de teste baseados no critério **Todos Nós** até que a cobertura seja 100% e, então, o desenvolvedor deverá implementar 100% da cobertura para o critério **Todas Arestas** e assim prosseguir para o desenvolvimento de novas funcionalidades, tal como pode ser observado na Figura 3.1, replicada na Figura 5.1 para facilitar a leitura.

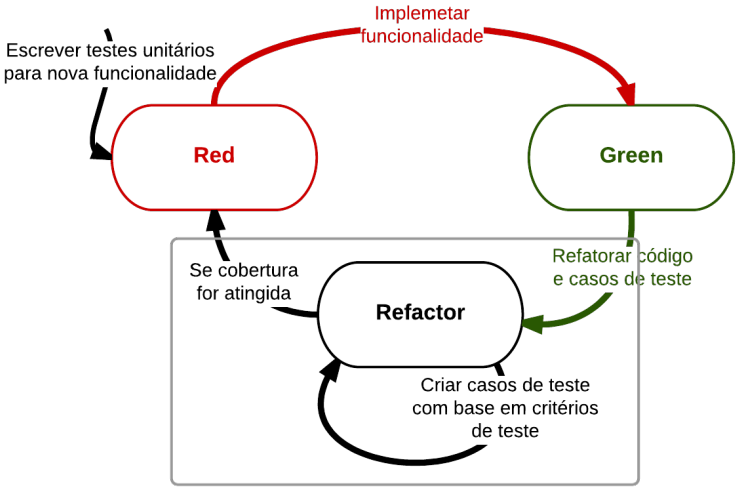

Figura 5.1. Fluxo proposto para o TDD pautado em critérios de teste.

5.1.1.4. Análise dos Resultados

A partir da aplicação do experimento, foram avaliadas variáveis como classificadas por Munir *et al.* (2014): Quantidade de casos de teste (Produtividade), Tempo para a conclusão do experimento (Tempo), Número de casos de teste escrito (Tamanho), Cobertura de testes (Qualidade Interna) e Quantidade de defeitos encontrados (Qualidade Externa).

O experimento foi cronometrado e, ao final de cada iteração, o tempo de cada grupo foi registrado, bem como a quantidade de defeitos encontrado. Tais informações estão dispostas na Tabela 5.2.

A primeira iteração foi das três a que consumiu mais tempo dos programadores durante todo o experimento. Nesta etapa se propõe a estrutura da qual todas as demais funcionalidades serão anexadas. Em ambos os grupos o tempo foi superior a duas horas, entretanto o "Grupo I" se destaca por consumir 25 minutos a mais que o "Grupo C".

Em ambos os grupos, foi satisfatória a quantidade de defeitos encontrados em virtude do desenvolvimento prévio dos casos de teste. Foram encontrados quinze defeitos pelo "Grupo C", enquanto o "Grupo I" encontrou vinte e dois defeitos, os quais podem ser atribuídos aos desenvolvimento dos casos de teste observando os critérios de teste propostos. O "Grupo I" encontrou sete erros e teve mais cobertura de testes a mais que o "Grupo C". Este resultado mostra indícios que os critérios colaboraram efetivamente para a qualidade dos casos de teste e, em consequência, com qualidade da implementação.

Tabela 5.2. Tempo de desenvolvimento e quantidade de defeitos da Execução 01.

Aluno	Grupo		Tempo	Quant. Def.
Aluno 1	**C**	Iter. 1	2h15m	5
		Iter. 2	0h25m	2
		Iter. 3	0h30m	1
Aluno 2	**C**	Iter. 1	1h55m	2
		Iter. 2	0h27m	3
		Iter. 3	0h22m	2
Aluno 3	**I**	Iter. 1	2h40m	7
		Iter. 2	1h00m	5
		Iter. 3	0h30m	3
Aluno 4	**I**	Iter. 1	2h30m	5
		Iter. 2	0h45m	4
		Iter. 3	0h40m	3

Quando questionado sobre a eficiência da utilização de critérios de teste proposta, o "Grupo I" demonstrou que a estratégia o ajudou a pensar melhor em como testar o código e a verificar quais as porções de código ainda necessitavam ser testadas. Segundo um dos acadêmicos do "Grupo I", normalmente ele testava apenas o que o TDD propunha e, com a

prática proposta, é estabelecido um roteiro de como escrever os casos de teste.

Baseado nos resultados obtidos nesta primeira execução, é possível observar que o TDD pautado em critérios de teste propõe outro desafio ao acadêmico, além do desenvolvimento do software proposto. Os critérios foram encarados como um novo objetivo, que é o de testar o software tal como proposto, assim como em desenvolvimento de software: enquanto todos os requisitos não forem contemplados o software não está pronto. Neste cenário, enquanto todos os requisitos de teste não forem atingidos, os casos de teste não estão prontos. Quanto ao resultado efetivo, os números demonstraram que a quantidade de defeitos encontrados e a cobertura do "Grupo I" foi superior.

5.1.2. Execução 02: Alunos avançados da UTFPR

Neste estudo participaram dez alunos do PPGI da UTFPR que estavam cursando a disciplina de Tópicos em Engenharia de Software do quarto trimestre do ano de 2014. A turma era composta por graduados em cursos com base de computação, entretanto com diferentes níveis de proficiência em programação.

A fim de viabilizar a participação de todos na execução, foi proposta a formação de duplas, caracterizando então a prática de programação em pares (*pair-programming*). As duplas foram formadas a partir de um levantamento que os alunos responderam no ingresso na disciplina, o qual questionava quanto ao nível de proficiência em programação e em qual linguagem. Desta forma, as duplas foram montadas unindo alunos com muito e com pouco conhecimento em programação formando cinco grupos que compõem o experimento, conforme exposto na Tabela 5.3. Os dados do levantamento também foram utilizados para a escolha da linguagem que seria utilizada na execução do experimento.

Tabela 5.3. Composição dos grupos da segunda execução.

Grupo	Membro Experiente	Membro
Grupo 01	P 01	P 02
Grupo 02	P 03	P 04
Grupo 03	P 05	P 06
Grupo 04	P 07	P 08
Grupo 05	P 09	P 10

A fim de proporcionar um ambiente aproximado da real aplicação do *pair-programming*, durante a modelagem desta aplicação foi mensurado o tempo de execução com base em outra aplicação dos mesmos exercícios (ANICHE; GEROSA, 2012). Desta forma, foi estipulado 45 minutos por iteração, com a inversão dos papéis aos 20 minutos. Entretanto, a execução foi interrompida aproximadamente aos 80 minutos, proporcionando 4 trocas de papéis.

5.1.2.1. Treinamento

Os participantes do experimento assistiram uma breve sessão de treinamento, em que foi demonstrado de maneira sucinta o fluxo do TDD e como proceder com a prática no IDE escolhido para a execução (Netbeans). Durante o treinamento foram abordados os dois critérios de teste de software utilizados (Todos Nós e Todas Arestas) durante a prática do TDD.

5.1.2.2. Atividade desenvolvida

Para esta execução foram propostos dois problemas a serem desenvolvidos, os quais já foram utilizados com o propósito de avaliar estudos com foco no TDD (ANICHE; GEROSA, 2012). Os exercícios propostos são situações típicas do mundo real. Cada dupla desenvolveu apenas um exercício.

- **Exercício 1 - Calculadora de Salário:** O participante deve implementar uma calculadora de salário de funcionários. Um funcionário contém nome, e-mail, salário-base e cargo. De acordo com seu cargo, a regra para cálculo do salário líquido é diferente.
- **Exercício 2 - Processador de Boletos** Nesse exercício, o participante deverá implementar um processador de boletos. O objetivo desse processador é verificar todos os boletos e, caso o valor da soma de todos os boletos seja maior que o valor da fatura, então essa fatura deverá ser considerada como paga.

5.1.2.3. Técnica de teste de software empregada

De semelhante modo a primeira execução do experimento proposto (Seção 5.1.1.3), o ciclo natural do TDD é alterado com o intuito de incluir o desenvolvimento de casos de teste derivados de critérios de teste de software. Para esta execução foi alterado apenas os critérios empregados, inicialmente o participante deve cobrir 100% **Todas Nós** e posteriormente 100% **Todos Arestas**, tal como apresentado na Figura 3.1.

5.1.2.4. Análise dos resultados

O procedimento de execução do experimento transcorreu sem muitas dúvidas. Entretanto dois grupos (01 e 02) não completaram o desenvolvimento das funcionalidades propostas. Em virtude disso não foram considerados os dados de cobertura deles. A média de cobertura de desvios foi elevada, visto que os dois problemas propostos não necessitam de muitas condicionais para a resolução. Entretanto, a cobertura de linhas não teve elevada cobertura

mesmo sendo um critério simples. Todavia os participantes não utilizaram ferramentas para verificar a taxa de cobertura, o que justifica inicialmente a baixa cobertura de linhas.

Durante a execução os acadêmicos alternaram na codificação por 4 vezes, ou seja, cada membro do grupo programou durante aproximadamente 40 minutos. O resultado de cobertura e tempo podem estão dispostos na Tabela 5.4.

Tabela 5.4. Composição e resultados dos grupos da segunda execução.

Grupo	**Exercício**	**Todas Linhas**	**Todos Desvios**	**Tempo**	**Completo**
Grupo 01	1	-	-	-	Não
Grupo 02	1	-	-	-	Não
Grupo 03	2	75,6%	92.9%	60m	Sim
Grupo 04	2	70,5%	100.0%	90m	Sim
Grupo 05	2	78,5%	100.0%	75m	Sim
Média	-	**74.8%**	**97.6%**	**75m**	-

Ao final da execução os participantes responderam um pequeno questionário composta pelas questões abaixo expostas. O resultado tabulado das questões está exposto na Tabela 5.5.

1. Nível de Programação (0-10)?
2. Conhecimento em Teste de Software (0-10)?
3. Nível em TDD (0-10)?
4. Ciclo do TDD foi interrompido?

Tabela 5.5. Resultados do questionário da segunda execução.

Questão	**P01**	**P02**	**P03**	**P04**	**P05**	**P06**	**P07**	**P08**	**P09**	**P10**
1	10	3	3	2	7	3	9	5	8	2
2	7	2	8	2	8	3	7	1	6	1
3	7	2	2	2	7	4	5	1	1	1
4	Não	Não	Não	Não	Não	Não	Não	Não	Não	Não

* As colunas “P01” (Participante 01) até “P10” (Participante 10) faz referência aos participantes do experimento.

Com base na Tabela 5.5 é possível observar que os indivíduos possuíam nível de programação, conhecimento em técnicas de teste de software e em TDD diversificado, com grande variação de conhecimento. Este grupo, tal como as outras execuções preliminares, não configura o público alvo a qual a técnica foi proposta. Todos os participantes responderam que o ciclo do TDD não foi interrompido pela adição de técnicas de teste de software na refatoração. Desta forma, é necessário verificar se este fato pode demonstrar que a intervenção não obstrui a técnica tradicional, mas a potencializa.

5.1.3. Execução 03: alunos concluintes da Faculdade Integrado

Para esta execução, foram selecionados acadêmicos concluintes do curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Faculdade Integrado de Campo Mourão/PR. Estes alunos cursavam a disciplina de Tópicos Especiais em Desenvolvimento de Sistemas que é ministrada no último semestre do curso. Os grupos que empregaram a técnica proposta e o grupo de controle foram separados aleatoriamente.

5.1.3.1. Treinamento

Os acadêmicos que cursavam esta disciplina receberam treinamento sobre a prática do TDD e desenvolveram seis exercícios utilizando a técnica, sendo três durante as aulas e outros três exercícios fora do ambiente da faculdade. Todos foram entregues como atividades da disciplina no Ambiente Virtual de Aprendizagem (AVA) da instituição, dos quais foram avaliados o código de implementação e casos de teste a fim de verificar se a prática do TDD estava sendo desenvolvida corretamente. Desta forma, foram excluídos quatro participantes da execução do experimento por não praticar o TDD corretamente.

5.1.3.2. Atividade desenvolvida

Assim como explanado na Seção 5.1.1, a atividade desenvolvida nesta execução foi o *Bowling Game*. Entretanto, como em diversos estudos, a entrega das iterações foi feita a partir do AVA da instituição (EDWARDS, 2004b; THORNTON *et al.*, 2008; SHAMS; EDWARDS, 2013). As iterações foram simuladas de forma que o acadêmico só tivesse acesso à especificação funcional da segunda iteração mediante a entrega da primeira e assim sucessivamente.

5.1.3.3. Técnica de teste de software empregada

Tal como as duas execuções supracitadas, a proposta foi adicionar o desenvolvimento de casos de teste pautados em critérios de teste de software. A técnica utilizada na terceira execução foi a mesma proposta na Seção 5.1.1.3. Durante a refatoração, o acadêmico deve atingir 100% da cobertura dos critérios Todos-Nós e Todas-Arestas em conjunto com a refatoração tradicional.

5.1.3.4. Análise dos resultados

Para esta execução do experimento, tal como a primeira (Seção 5.1.1), foram analisados: Quantidade de casos de teste, Tempo para a conclusão do experimento, número de casos de

teste, cobertura de teste. Em virtude da entrega ser mediante o envio para o *Ambiente Virtual de Aprendizagem* (AVA), a variável de quantidade de defeitos encontrados não foi coletada.

Analisando os valores de cobertura, apresentados na linha E3 da Tabela 5.6, tal como o primeiro experimento, o "Grupo I" (intervenção), teve valores mais elevados, principalmente nas duas primeiras iterações, em que três participantes, atingiram 100% no critério "Todas as arestas". Quando observados os valores de cobertura de todos os nós, o grupo de intervenção também tem melhores resultados, média de 85,3%, enquanto o grupo de controle tem 73,9%. Entretanto, mesmo o "Grupo I" tendo maiores coberturas para os critérios associados com a intervenção proposta, o "Grupo C" tem maior média de casos de teste desenvolvidos. Mediante este fato, pode-se supor que a adoção de critérios de teste de software colabora para a economia de tempo quando o acadêmico tem que propor os casos de teste.

A quantidade de casos de teste não é um indicador que garanta a qualidade do conjunto de casos de teste. Desta forma, mesmo a quantidade do "Grupo C" sendo maior, seus indicadores de cobertura foram menores. Mediante este fato, outro dado importante a ser indagado é qual a qualidade do conjunto de casos de teste dos grupos e quão relevante são os casos de teste desenvolvidos pelo grupo de controle.

5.1.4. Sumarização dos resultados preliminares

A primeira e terceira execuções contaram com características semelhantes em participantes, atividade e estratégia. Desta forma, seus resultados foram agrupados na Tabela 5.6. Nas duas execuções foram analisados atributos de qualidade interna: número de casos de teste e cobertura de critério de teste de software. Nestes estudos, foram considerados cobertura de instruções e ramos. Em geral, considerando os dois experimentos, o grupo de intervenção obteve maior cobertura nos dois critérios analisados com conjuntos de casos de testes menores em comparação ao grupo que praticou o TDD tradicional.

Na Tabela 5.6, considerando apenas a primeira execução, é possível verificar de fato que a eficácia do conjunto de casos de testes do grupo de intervenção obteve em comparação ao grupo tradicional. Tendo em vista que o Grupo C1 desenvolveu 3,5 casos de teste cobrindo 92,25% de instruções e 49,55% de ramos, enquanto o Grupo I1 desenvolveu em média dois casos de teste, cobrindo 97.05% de instruções e 89,30% dos ramos. Nas próximas iterações o Grupo C1 adicionou em média um caso de teste por iteração. Entretanto, o Grupo I1 dobrou o tamanho (embora ainda menor que o Grupo C1) e na terceira iteração, foi adicionado apenas um caso de teste. Considerando a cobertura dos requisitos de teste na segunda e terceira iteração, os grupos C1 e I1 mantiveram médias similares quanto a cobertura de instruções, entretanto o mesmo não ocorreu para a cobertura de ramos: o Grupo I1 sempre manteve a cobertura alta mais alta que o Grupo C1, com no mínimo 20% de diferença.

Tabela 5.6. Resultados das execuções 01 e 03.

			Iteração 1			**Iteração 2**			**Iteração 3**		
			Statements	Branches	CT	Statements	Branches	CT	Statements	Branches	CT
E01	Aluno 01	C1	93,3	54,8	5	96,7	93,3	6	95,4	71,4	7
	Aluno 02	C1	91,2	44,3	2	87,7	43,0	3	88,6	44,4	4
	Média	**C1**	**92.25**	**49.55**	**3.5**	**92.20**	**68.15**	**4.5**	**92.00**	**57.90**	**5.5**
	Aluno 09	I1	95,4	78,6	2	82,7	75,0	4	95,1	89,6	6
	Aluno 10	I1	98,7	100,0	2	98,9	100,0	4	98,7	100,0	4
	Média	**I1**	**97,05**	**89,30**	**2,0**	**90,80**	**87,50**	**4,0**	**96,90**	**94,80**	**5,0**
	I1 - C1		**4,85**	**39,75**	**-1,5**	**-1,40**	**19,35**	**-0,5**	**4,90**	**36,90**	**-0,5**
E03	Aluno 03	A2	77.8	33.3	1	85.2	66.7	2	82.2	68.8	3
	Aluno 04	C2	93,4	100,0	1	96,4	100,0	2	96,4	100,0	4
	Aluno 05	C2	89,4	62,5	1	79,2	77,8	1	89,0	61,8	1
	Aluno 06	C2	91,1	66,7	4	85,1	66,7	2	89,1	71,4	3
	Aluno 07	C2	80,0	50,0	2	94,9	83,3	7	95,5	83,3	10
	Aluno 08	C2	91,3	75,0	11	92,8	73,1	11	97,1	73,3	12
	Média	**C2**	**87.17**	**64.58**	**3.3**	**87.17**	**77.93**	**4.1**	**91.55**	**76.43**	**5.5**
	Aluno 11	B2	89,6	100	1	80,1	100	1	93,6	100	4
	Aluno 12	B2	88,2	100	2	92,9	100	5	87,1	80,6	9
	Aluno 13	B2	93,2	100	3	94,2	100	4	96,9	100	8
	Aluno 14	B2	98,2	90	3	99,4	81,2	3	99,0	88,7	3
	Aluno 15	B2	100	100	3	100	100	4	99,8	88,8	5
	Média	**C2**	**93,84**	**98,00**	**2,4**	**93,32**	**96,24**	**3,4**	**95,28**	**91,62**	**5,8**
	I2 - C2		**6,67**	**33,42**	**-0,9**	**6,15**	**18,31**	**-0,7**	**3,73**	**15,19**	**0,3**

* Grupo C TDD tradicional e Grupo I TDD+Critérios de Teste.

Na segunda execução, a diferença para a cobertura de instruções e ramos foi mais evidente. O Grupo C2 atingiu 64,58% de cobertura de ramos enquanto o Grupo I atingiu 98,00% utilizando menos casos de testes na primeira iteração. Para a cobertura de instruções a diferença foi menor, mas o Grupo I2 atingiu cobertura maior.

Com base na média dos grupos na primeira iteração, as quais podem ser observadas na Tabela 5.7, o Grupo C desenvolveu em média 3.38 casos de teste com 88.44% de cobertura de instruções e 60,83% de cobertura contra uma média de 2,29 casos de teste com 94,76% de cobertura de instruções e 95,51% de cobertura de ramos do grupo I. Na segunda e terceira iteração, a diferença é pequena, os dois grupo desenvolveram em média 5 casos de teste, entretanto com diferença significativa entre a cobertura dos requisitos de teste: 89,75% contra 92,60% quanto a cobertura de instruções e 75,49% contra 93,75% para cobertura de ramos. De fato, a diferença da cobertura de instruções entre os grupos C e I não é estatisticamente significante. Este resultado está de acordo com estudos que avaliam a cobertura para casos de teste escritos por estudantes utilizando TDD (EDWARDS; SHAMS, 2014). Contudo, para cobertura de ramos, a diferença é significante a um nível de 0,5 aplicando Mann-Whitney Test U (agrupando os dados dos dois estudos). Agrupando os dados de todas iterações, cobertura

de instruções também é significante.

Tabela 5.7. Resumo da execução 01 e 02.

	Todos-Nós				**Todas-Arestas**				**Casos de Teste**			
	TDD	CTDD	Dife.	p-value	TDD	CTDD	Dife.	p-value	TDD	CTDD	Dife.	p-value
Iter. 1	88,44	94,76	6,32	0,094	60,83	95,51	34,69	**0,006**	3,38	2,29	-1,09	1,000
Iter. 2	89,75	92,60	2,85	0,397	75,49	93,74	18,26	**0,031**	4,25	3,57	-0,68	0,860
Iter. 3	91,66	95,74	4,42	0,189	71,80	92,53	20,73	**0,014**	5,50	5,57	0,07	0,682
Iter. 1+2+3	89,95	94,37	3,85	**0,007**	69,37	93,93	24,56	**0,00009**	4,38	3,81	-0,57	0,818

* CTDD é referente ao TDD pautado em critérios de teste.

Um dos resultados esperados é a melhoria da qualidade do software em virtude da aplicação dos critérios de teste de software sem interromper o TDD, apesar do esforço adicional para criar melhores casos de testes e impactar na atividade de refatoração. Observou-se que o fluxo proposto atua como um guia de criação de casos de teste, possibilitando a definição de um conjunto de casos de testes mais consistente com respeito ao tamanho do conjunto e da qualidade (cobertura). A curto prazo, o esforço economizado na definição de menos casos de teste é um resultado interessante da técnica. A longo prazo, a maior qualidade do conjunto de casos de teste pode contribuir com a redução do esforço necessário em atividades específicas de teste de software ou possibilitar o uso de técnicas de software mais sofisticadas, considerando critérios mais fortes e teste de sistema.

Após o experimento, os estudantes foram indagados quanto ao uso do TDD. Em geral, foi observado que os acadêmicos consideraram os casos de teste como uma parte integrante do software, que eles não poderiam entregar o software sem implementar as funcionalidades e o todo o conjunto de casos de teste funcionando. Com respeito ao TDD com critérios, os dois grupos consideraram a técnica como satisfatória, satisfazendo os objetivos estabelecidos pelos requisitos definidos. Deste modo, enquanto os requisitos funcionais e de teste para iteração não foram cumpridos a contento do estabelecido pela técnica, o software não está pronto para ser entregue.

Um estudante, que já tinha experiência com o TDD, salientou que estava acostumado a utilizar técnica *ad hoc* para criar os casos de teste, todavia a intervenção proposta proveu melhor experiência, estabelecendo um guia para escrever os casos de teste. Além disso, quando questionados sobre a eficácia da prática de critérios de teste em conjunto com a refatoração no TDD, os alunos concordaram que a abordagem realmente colaborou com a finalidade de melhorar o conjunto de casos de teste no sentido de guiar qual a parte do software requer um esforço maior em melhorar o conjunto de casos de teste.

5.1.5. Lições aprendidas

Durante a execução dos estudos experimentais preliminares e na análise dos resultados, algumas falhas foram evidentes. Desta forma, estas foram alvo de esforços a fim que fossem mitigadas na definição do projeto experimental final (Seção 3.4).

Neste contexto, foi identificado é que necessário que seja definido um pacote experimental para garantir que em todas as execuções seja possível obter um conjunto de dados homogêneo. As execuções não possuíam um conjunto de variáveis homogêneo, dificultando a comparação dos resultados dos três conjuntos de dados. Neste contexto, foi proposta a ferramenta descrita na **??**, a qual auxilia na captura das informações durante o processo de desenvolvimento do software. Além disso, foram propostos dois questionários que capturam informações a respeito da população antes e após o experimento.

Nas avaliações preliminares, os participantes das execuções supracitados não utilizaram uma ferramenta que os auxiliasse na verificação da cobertura durante o desenvolvimento, desta forma, dificultando a análise do acadêmico ao cobrir os critérios definidos. Possivelmente adicionando um problema a estratégia, ao tentar cobrir os critérios, o aluno pode ter desenvolvido casos de teste *ad hoc*. Mediante este fato, foi incluído a utilização do Eclemma, *plugin* que demonstra informações a respeito de cobertura de critérios de teste.

Durante a análise dos dados preliminares foi possível observar que alguns trechos de códigos gerados automaticamente pelo IDE de desenvolvimento (*getters* e *setters*, por exemplo) podem alterar os resultados reais do experimento, visto que, se tais métodos não forem utilizados nos casos de teste, a cobertura será menor.

Durante a avaliação dos resultados dos experimentos, é necessário avaliar o ciclo de desenvolvimento executado pelo acadêmico. Observar apenas o software resultante, com seu conjunto de casos de teste não garante que o aluno tenha utilizado o ciclo do TDD. Desta forma, a ferramenta proposta capta qual o ciclo de desenvolvimento que o acadêmico executou, proporcionando assim a possibilidade de posteriormente analisar o fluxo executado.

5.2. Avaliação da proposta

A partir dos estudos preliminares executados e das lições aprendidas nestes estudos, foi definido o estudo experimental final. Este estudo foi executado na Faculdade Integrado em Campo Mourão/Paraná, no mês de outubro, com 17 alunos do segundo período do curso de Tecnologia em Análise e desenvolvimento de Sistemas. O público foco da estratégia proposta na presente dissertação são alunos que estão iniciando no desenvolvimento de software. Desta forma, esta execução tem o objetivo de avaliar as hipóteses descritas na Seção 3.4.

5.2.1. Treinamento

A execução experimental ocorreu durante a disciplina de estrutura de dados. Desta forma, desde o início da disciplina o IDE Eclipse foi utilizado na resolução dos exercícios propostos. Ao final da disciplina, durante uma semana, os alunos foram instruídos sobre o TDD, bem como desenvolveram exercícios para exercitar suas habilidades com a técnica. Durante os exercícios, os acadêmicos tiveram a oportunidade de utilizar o mesmo ambiente de desenvolvimento que seria utilizado durante a execução experimental, com os *plugins* Eclemma e JUnit. Neste momento foram apresentados apenas os conceitos básicos sobre cobertura de critérios, para que fosse possível entender as informações apresentadas pelo Eclemma.

Após o treinamento inicial de TDD, os alunos responderam um questionário sobre seus conhecimentos de programação, teste de software e TDD, com as seguintes questões:

Q01 Como você avalia suas habilidades em programação?
Q02 Como você avalia suas habilidades no TDD?
Q03 Como você classifica seus conhecimentos sobre a técnica de teste de software funcional?
Q04 Como você classifica seus conhecimentos sobre a técnica de teste de software estrutural?
Q05 Acabei de receber um requisito novo no software que estou desenvolvendo. Já escrevi o teste e executei meus testes. O teste que eu escrevi falhou. O que devo fazer agora?

A Tabela 5.8 apresenta a população da quarta execução experimental e as respostas do questionário preliminar. Nesta tabela é possível observar que os alunos do grupo de controle consideraram suas habilidades em relação a programação variando entre 3 e 4, já o grupo de intervenção considerou entre 2 e 4. Em relação a os conhecimento a respeito do TDD, o Grupo C, variou entre 1 e 4 e o Grupo I, entre 0 e 4. Em relação as técnicas de teste de software, consideraram seu conhecimento os dois grupos obtiveram variação entre 0 e 3, configurando pouco conhecimento. A média acadêmica dos dois grupos é aproximada, sendo que o Grupo C tem 8,1 e o Grupo I 8,4.

As questões Q01 e Q02 são de autoavaliação e questionaram os acadêmicos a respeito de seu nível de programação. Na disciplina em que este estudo experimental foi aplicado, os alunos tem experiência de programação há apenas um semestre e, desta forma seu conhecimento é básico. Mesmo assim, alguns alunos atribuíram nota 4 em uma escala de 0 a 5. Quando questionados quanto ao TDD, na questão Q02, a resposta foi mais realista, sendo atribuídas notas mais baixas. As questões Q03 e Q04 também são de autoavaliação e, com base nas respostas, é possível observar que a grande parte dos alunos considera seu conhecimento quanto a teste de software incipiente.

Após responder os questionários, os grupos foram divididos aleatoriamente. O Grupo I foi treinado para a utilizar o TDD com critérios de teste de software. Os alunos foram

Tabela 5.8. População da execução 04.

População			Questões				
Indivíduo	**Média A.**	**Grupo**	**Q01**	**Q02**	**Q03**	**Q04**	**Q05**
Aluno 01	8,25	C	3	2	2	3	Correto
Aluno 02	9,58	C	2	2	0	0	Correto
Aluno 03	8,67	C	3	3	1	1	Correto
Aluno 04	7,2	C	3	2	2	3	Correto
Aluno 05	6,99	C	3	2	1	1	Correto
Aluno 06	9,58	C	4	1	1	1	Errado
Aluno 07	5,1	C	3	1	2	2	Errado
Aluno 08	8,28	C	3	1	2	2	Correto
Aluno 09	8,63	C	4	2	4	4	Correto
Aluno 10	9,86	C	4	4	2	3	Correto
Aluno 11	9,3	I	4	4	3	2	Correto
Aluno 12	9,58	I	3	4	0	0	Correto
Aluno 13	9,16	I	2	2	1	1	Correto
Aluno 14	9,79	I	4	3	1	3	Errado
Aluno 15	10,0	I	3	1	1	0	Correto
Aluno 16	10,0	I	3	1	0	0	Correto
Aluno 17	9,79	I	3	0	0	0	Correto

instruídos durante apenas uma aula sobre os critérios a serem empregados e sua junção com o TDD. Após esta aula, foi entregue aos alunos um exercício para ser desenvolvido utilizando a intervenção proposta.

5.2.2. Atividade desenvolvida

A atividade proposta neste execução foi a mesma utilizada nas execuções preliminares 01 e 02 e também em diversos outros estudos que avaliam experimentalmente o TDD (CAUSEVIC *et al.*, 2013a; XU; RAJLICH, 2006; CAUSEVIC *et al.*, 2012, 2012b; SHELTON *et al.*, 2012). Todavia, nesta execução o *Bowling Game* foi dividido em seis iterações, com a finalidade de aumentar a quantidade de "entregas" feitas pelo acadêmico durante a execução experimental. São elas:

- Iteração 01: O jogo consiste em 10 rodadas e, a cada rodada, o jogador tem direito a dois lances para atingir o a pontuação máxima (10). O *score* do jogo pode ser consultado a qualquer momento.
- Iteração 02: O ato de derrubar os 10 pinos em uma rodada é chamado *spare*, desta forma, o jogador tem como bônus da rodada anterior a pontuação obtida na próxima jogada.

- Iteração 03: Se na primeira tentativa a pontuação máxima for atingida (*strike*), o jogador não terá direito ao segundo lance.
- Iteração 04: O bônus pontuação do *strike* é o valor das duas próximas jogadas.
- Iteração 05: O jogador terá duas jogadas extra se fizer *strike* na décima rodada.
- Iteração 06: Se atingir o *spare* terá direito a uma jogada.

5.2.3. Técnica de teste de software empregada

Nesta execução experimental foi utilizada a estratégia descrita na Seção 3.2, a qual propõe a utilização do TDD em conjunto com os critérios estruturais baseados em *Grafo de Fluxo de Controle* (GFC) Todos-Nós e Todas-Arestas. A estratégia empregada está definida na Figura 3.2 e novamente apresentada na Figura 5.2.

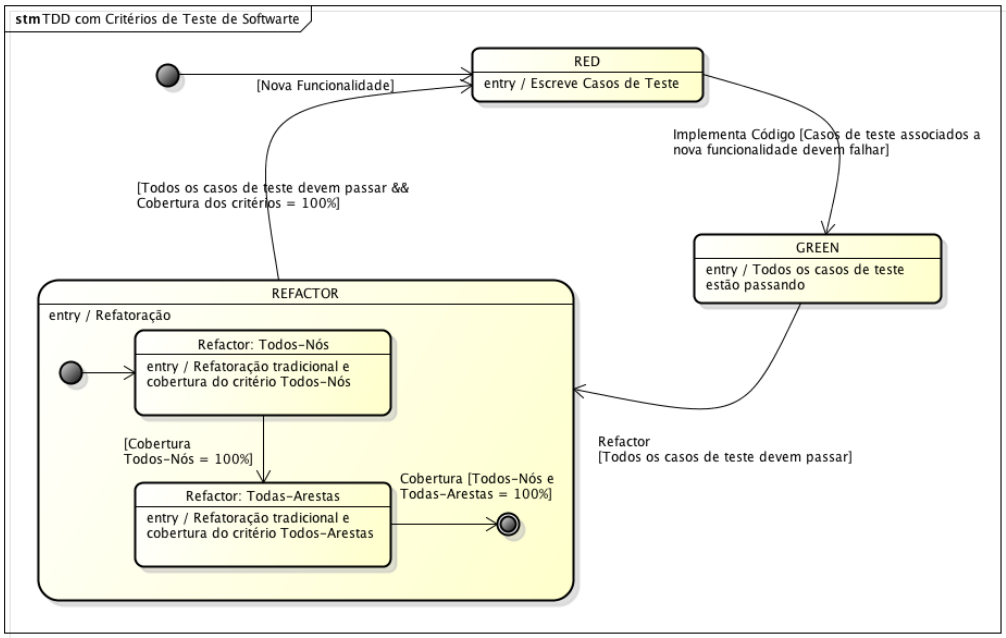

Figura 5.2. Fluxo proposto para o TDD pautado em critérios de teste.

5.2.4. Análise dos resultados

A análise dos resultados deste experimento foi feita com base no GQM definido na Seção 3.3. A análise iniciou-se pela conferência do ciclo de desenvolvimento executado pelo aluno, com a finalidade de verificar se o ciclo do TDD foi seguido corretamente. Em seguida, foi feita a análise do conjunto de casos de teste, do código resultante, da avaliação do desempenho acadêmico e da opinião do aluno quanto à participação no experimento e uso da estratégia.

5.2.4.1. Estão praticando TDD corretamente?

Neste contexto, foi observado que a maioria dos alunos dos grupos C e I ignoraram a fase de refatoração. Neste contexto podem ser atribuídas a falta de experiência para propor melhorias no código, pois o aluno está iniciando no desenvolvimento de software. Além disso, o problema proposto (*Bowling Game*) também pode colaborar com esta prática, fornecendo poucas possibilidades de melhoria. Todavia, é importante salientar que o aluno pode ter utilizado a ferramenta de captura de informações (*plugin*) incorretamente, informando a etapa incorreta durante o desenvolvimento.

Assumindo as possibilidades citadas, foram selecionadas as execuções que seguiram o fluxo mínimo proposto pelo TDD, propondo casos de teste na etapa *red* e efetuando a implementação na etapa *green*. Após isso, foram verificados também o tamanho do conjunto de casos de teste e as execuções do conjunto (com falhas e sucessos). Após esta análise, foram excluídas as execuções que não seguiram os requisitos estabelecidos. Desta forma, a Tabela 5.9 expõe as considerações quanto as execuções.

Tabela 5.9. Análise do ciclo do TDD de desenvolvimento dos participantes da execução 04.

Aluno	**Grupo**	**Excluído**	**Comentário**
Aluno 01	C	Não	
Aluno 02	C	Não	
Aluno 03	C	Sim	Execução incorreta
Aluno 04	C	Sim	Execução incorreta
Aluno 05	C	Não	
Aluno 06	C	Sim	Execução incorreta
Aluno 07	C	Não	
Aluno 08	C	Sim	Execução incorreta
Aluno 09	C	Sim	Execução incorreta
Aluno 10	C	Não	
Aluno 11	I	Não	
Aluno 12	I	Sim	Execução incorreta
Aluno 13	I	Sim	Execução incorreta
Aluno 14	I	Não	
Aluno 15	I	Não	
Aluno 16	I	Sim	Execução incorreta
Aluno 17	I	Não	

* Excluído: Alunos que foram excluídos pela execução incorreta do TDD.

5.2.4.2. O conjunto de testes é melhor?

Com base nos sete alunos que executaram o fluxo do TDD a contento, foram extraídos os dados das execuções, os quais estão dispostos na Tabela 5.10. Esta tabela apresenta as médias

das métricas estabelecidas para a avaliação dos grupos experimentais. Os dados dos grupos C e I são agrupados por iteração e para cada uma delas são apresentados dados referentes aos estágios do TDD. Além disso, a Tabela 5.11 resume todas as informações dos grupos por iteração e ainda demonstra a diferença das médias de todas as execuções dos dois grupos.

Tabela 5.10. Resultados da execução 04.

Grupo	Ite.	TDD	Tempo	CT	CT P	CT F	DEF	Todas-Arestas	Todos-Nós
C	1	Red	17,8	2,8	1,8	1,0	4,0	90,0	72,5
		Green	10,8	3,0	2,6	0,4	1,0	90,0	88,1
		Refactor	6,3	2,3	2,3	0,0	0,0	87,5	99,0
	2	Red	11,4	3,4	2,4	1,0	5,0	73,9	71,5
		Green	20,6	3,4	3,0	0,4	16,0	77,6	80,7
		Refactor	17,0	3,3	2,8	0,5	2,0	67,9	75,9
	3	Red	2,4	5,0	4,3	0,7	2,0	94,3	91,9
		Green	18,0	4,6	4,2	0,4	4,0	89,1	92,7
		Refactor	15,5	4,5	4,3	0,3	0,0	87,4	98,6
	4	Red	16,4	6,0	4,4	1,6	4,0	89,7	79,8
		Green	6,8	5,6	4,0	1,6	7,0	91,7	82,1
		Refactor	19,7	6,3	5,7	0,7	2,0	88,9	93,0
	5	Red	12,0	5,5	4,0	1,5	6,0	81,9	77,5
		Green	17,6	5,3	4,3	1,0	6,0	81,0	86,9
		Refactor	4,0	5,7	5,0	0,7	2,0	90,2	91,6
	6	Red	30,4	6,8	5,8	1,0	3,0	91,9	86,1
		Green	18,6	6,8	6,4	0,4	3,0	88,0	95,1
		Refactor	34,0	6,3	6,3	0,0	0,0	95,2	100,0
I	1	Red	16,8	2,3	1,3	1,0	2,0	83,3	74,5
		Green	18,3	2,0	2,0	0,0	0,0	83,3	97,6
		Refactor	28,5	2,5	2,0	0,5	0,0	93,8	81,5
	2	Red	4,5	4,3	3,8	0,5	2,0	90,0	85,8
		Green	6,3	8,0	7,3	0,8	3,0	100,0	100,0
		Refactor	0,0	3,7	3,7	0,0	0,0	97,2	99,8
	3	Red	14,8	5,5	4,3	1,3	3,0	96,4	78,0
		Green	15,0	5,5	5,5	0,0	0,0	98,2	99,9
		Refactor	2,0	7,0	7,0	0,0	1,0	100,0	100,0
	4	Red	11,3	8,0	7,0	1,0	1,0	96,4	85,8
		Green	15,3	7,0	7,0	0,0	0,0	98,3	100,0
		Refactor	13,5	7,0	7,0	0,0	0,0	98,4	100,0
	5	Red	15,0	8,5	8,0	0,5	3,0	99,3	96,5
		Green	13,8	8,3	8,3	0,0	0,0	100,0	100,0
		Refactor	5,8	8,3	8,3	0,0	3,0	100,0	100,0
	6	Red	14,5	12,5	11,5	1,0	1,0	98,7	90,5
		Green	9,5	9,8	9,5	0,3	0,0	99,3	94,3
		Refactor	9,5	12,5	12,5	0,0	0,0	100,0	100,0

*Ite. (Iteração), CT (Caso de teste), CT P (Caso de teste passando), CT F (Caso de teste falhando)

Durante a primeira iteração do desenvolvimento, o grupo de controle (C) desenvolveu

Tabela 5.11. Resumo dos resultados da execução 04.

Grupo	Ite.	Tempo	CT	CT P	CT F	DEF	Todos-Nós	Todas-Arestas
C	1	12,00	2,71	2,21	0,50	5,00	85,64	89,29
	2	16,29	3,36	2,71	0,64	23,00	76,05	73,51
	3	11,71	4,67	4,25	0,42	6,00	94,09	90,45
	4	13,46	5,92	4,54	1,38	13,00	83,75	90,28
	5	11,71	5,50	4,40	1,10	14,00	85,45	84,08
	6	26,69	6,67	6,17	0,50	6,00	92,74	91,15
	Média	15,31	4,80	4,05	0,76	11,17	86,29	86,46
I	1	19,70	2,20	1,70	0,50	2,00	85,11	85,42
	2	3,91	5,45	5,00	0,45	5,00	94,78	95,61
	3	12,30	5,67	5,11	0,56	4,00	91,16	97,86
	4	13,30	7,38	7,00	0,38	1,00	94,33	97,58
	5	11,50	8,33	8,17	0,17	6,00	98,84	99,75
	6	10,75	11,13	10,75	0,38	1,00	94,77	99,34
	Média	11,91	6,69	6,29	0,40	3,17	93,16	95,93
A - B		**-3.40**	**1.89**	**2.24**	**-0.35**	**-8.00**	**6.88**	**9.47**

mais casos de teste e obteve maior cobertura de ramos e menor cobertura de instruções. Todavia durante o restante do desenvolvimento, o grupo de intervenção (I) aumentou o tamanho do conjunto de testes a cada uma das iterações e também a cobertura de Todas-Arestas e Todos-Nós.

A cobertura de Todas-Arestas do Grupo C variou de 67,9% na refatoração da segunda iteração até 95,2% na refatoração da última iteração, enquanto a variação do Grupo I foi menor, de 93,8% na refatoração da primeira iteração para até 100% na refatoração da última iteração. Em relação à cobertura de instruções, o Grupo C obteve cobertura de 75,9% na refatoração da segunda iteração e aumentou para 100% na refatoração da última iteração enquanto o Grupo I obteve 81,5% de cobertura na refatoração da primeira iteração e na última iteração atingiu 100%. Com base na variação, foi possível observar que o grupo de intervenção manteve a cobertura alta, com no máximo 18.52% de variação durante todo o desenvolvimento do projeto enquanto o grupo de controle obteve variações de até 27,3%.

Em relação ao tempo investido na solução do problema proposto, inicialmente foi necessário que o grupo de intervenção investisse mais tempo no desenvolvimento, como pode ser observado na Figura 5.3. Nas primeiras duas o Grupo I investiu mais tempo no desenvolvimento. No decorrer do desenvolvimento, entre as iterações três e quatro o tempo investido foi semelhante. Todavia, nas duas últimas iterações o tempo investido pelo grupo de intervenção foi consideravelmente menor.

Durante o desenvolvimento, a cada iteração os alunos do Grupo C executaram em média 5,4 vezes o conjunto de casos de teste e os alunos do Grupo I executaram 4,13 vezes. A média de falhas de casos de teste por execução do Grupo C foi de 0,67 enquanto a do Grupo

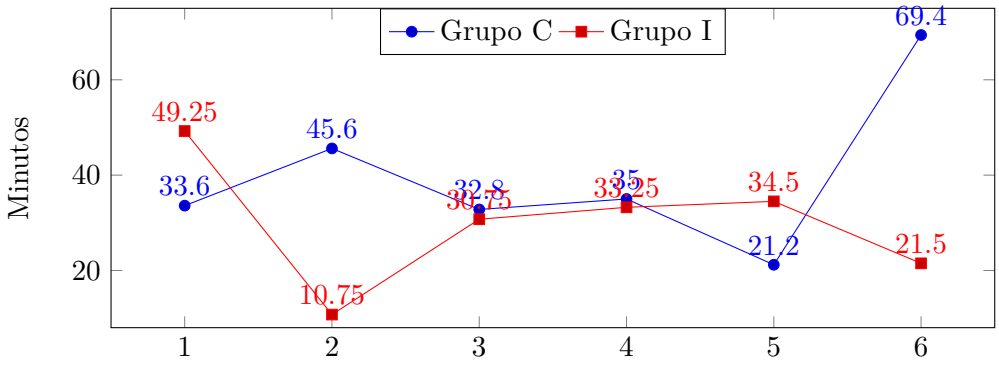

Figura 5.3. Visão do tempo de desenvolvimento na execução 04.

I foi de 0,26. Desta forma, os alunos do grupo de intervenção executaram o conjunto de casos de testes menos vezes e obtiveram menos falhas. Todavia, é necessário verificar se tal efeito pode ser atribuído a técnica proposta.

A Tabela 5.12 apresenta uma comparação estatística em relação a todas as iterações, estágios do TDD e métricas em relação a tempo, cobertura de testes e tamanho de conjunto de casos de teste que foram utilizadas no estudo. Esta tabela apresenta algumas diferenças entre o grupo de intervenção em relação ao grupo de controle. Cabe destacar que a amostra não é significativa, devido ao tamanho, mas esses resultados oferecem um indício quanto à qualidade do resultado obtido.

Em relação a cobertura, em diversos momentos o Grupo I (intervenção) obteve diferença significativa em relação ao grupo de controle (C). Neste contexto, durante a última iteração, as coberturas de instruções, ramos e complexidade apresentaram resultados significativos em relação ao grupo de controle.

De diferente modo se comparado com os estudos preliminares, o grupo de intervenção também desenvolveu mais casos de teste. Em alguns momentos durante o desenvolvimento do projeto, houve diferença significativa. Todavia, em relação ao tempo de desenvolvimento não houve diferença significativa. Desta forma, demonstrando que apesar do maior conjunto de casos de teste, o grupo de intervenção não foi prejudicado em relação a produtividade.

5.2.4.3. O código resultante é melhor?

Ao final das seis iterações, é necessário avaliar o software resultante com a finalidade de verificar se a intervenção promoveu também melhora no código de produção. Para a obtenção das métricas a partir do software final, foi utilizada a ferramenta JaBuTi (VINCENZI *et al.*, 2003), as quais estão dispostas na Tabela 5.13. Nesta tabela, são exibidas as métricas definidas por Chidamber e Kemerer (1994): acoplamento (CBO), complexidade (WMC) e coesão (LCOM).

Tabela 5.12. Resultados estatísticos da execução 04.

Ite.	Estágio	Tipo	P-Value	Sig.
1	Red	Instructions	0,462	Não
		Branches	0,083	Não
		C. de Teste	0,806	Não
		Tempo	0,157	Não
	Green	**Instructions**	**0,014**	**Sim**
		Branches	0,142	Não
		C. de Teste	0,624	Não
		Tempo	0,386	Não
	Refactor	Instructions	0,643	Não
		Branches	0,064	Não
		C. de Teste	0,165	Não
		Tempo	0,064	Não
2	Red	Instructions	0,624	Não
		Branches	0,462	Não
		C. de Teste	0,142	Não
		Tempo	0,513	Não
	Green	**Instructions**	**0,001**	**Sim**
		Branches	**0,007**	**Sim**
		C. de Teste	**0,027**	**Sim**
		Tempo	0,157	Não
	Refactor	Instructions	0,157	Não
		Branches	0,157	Não
		C. de Teste	0,289	Não
		Tempo	0,127	Não
3	Red	Instructions	0,327	Não
		Branches	0,086	Não
		C. de Teste	0,724	Não
		Tempo	0,083	Não
	Green	**Instructions**	**0,014**	**Sim**
		Branches	**0,014**	**Sim**
		C. de Teste	0,221	Não
		Tempo	0,355	Não
	Refactor	**Instructions**	**0,021**	**Sim**
		Branches	**0,021**	**Sim**
		C. de Teste	0,480	Não
		Tempo	0,480	Não

Ite.	Estágio	TIPO	P-Value	Sig.
4	Red	Instructions	0,462	Não
		Branches	0,086	Não
		C. de Teste	0,101	Não
		Tempo	0,655	Não
	Green	**Instructions**	**0,014**	**Sim**
		Branches	**0,014**	**Sim**
		C. de Teste	0,053	Não
		Tempo	0,480	Não
	Refactor	Instructions	0,083	Não
		Branches	0,083	Não
		C. de Teste	0,564	Não
		Tempo	0,655	Não
5	Red	Instructions	0,083	Não
		Branches	**0,043**	**Sim**
		C. de Teste	0,083	Não
		Tempo	0,221	Não
	Green	**Instructions**	**0,014**	**Sim**
		Branches	**0,014**	**Sim**
		C. de Teste	0,157	Não
		Tempo	0,289	Não
	Refactor	**Instructions**	**0,021**	**Sim**
		Branches	**0,021**	**Sim**
		C. de Teste	**0,013**	**Sim**
		Tempo	0,083	Não
6	Red	Instructions	0,699	Não
		Branches	0,053	Não
		C. de Teste	0,064	Não
		Tempo	0,165	Não
	Green	Instructions	0,327	Não
		Branches	**0,014**	**Sim**
		C. de Teste	0,086	Não
		Tempo	0,480	Não
	Refactor	**Instructions**	**0,001**	**Sim**
		Branches	0,083	Não
		C. de Teste	0,083	Não
		Tempo	0,121	Não

As métricas para análise de resultados do software final demonstraram que o grupo de intervenção obteve melhores resultados de acoplamento e uma pequena diferença em complexidade ciclomática. Todavia a coesão do grupo de controle foi menor que a do grupo de intervenção. Os resultados referentes a acoplamento e complexidade são equilibrados entre os grupos, entretanto em coesão alguns acadêmicos obtiveram valores com grande diferença.

Tabela 5.13. Resultados do software final.

Grupo	Aluno	Acoplamento	Coesão	Complexidade
C	Aluno 01	4.0	3.0	4.7
	Aluno 02	6,0	16,0	3,0
	Aluno 05	5,0	3,0	4,7
	Aluno 07	4,0	10,0	2,4
	Aluno 10	6,0	36,0	3,0
Média		**5,0**	**13,6**	**3,5**
I	Aluno 11	4,0	39,0	3,0
	Aluno 14	4,0	6,0	4,3
	Aluno 15	4,0	47,0	2,7
	Aluno 17	1,0	0,0	1,6
Média		**3,3**	**23,0**	**2,9**
p-value		**0,2168**	**0,2404**	**0,7122**

Tal fato pode ser observado no Aluno 10 do Grupo C e nos alunos 11 e 15 do Grupo I, os quais colaboraram para que a média dos grupos fossem elevadas.

Com base nos valores demonstrados na Tabela 5.13 é possível observar que em nenhuma das variáveis testadas com o Mann-Whitney Test U foi constatada diferença significativa. Entretanto, a intervenção desenvolveu uma solução menos acoplada com menor complexidade e, por outro lado, o grupo de controle propôs o software mais coeso. Neste contexto, desenvolver o código com menos acoplamento e mais coesão com menor complexidade é uma atividade complexa e são habilidades que são desenvolvidas ao longo da aprendizagem. Desta forma, na Seção 5.3 serão discutidas estratégias para obter melhores resultados para comparação de qualidade.

5.2.4.4. O desempenho acadêmico é melhor?

A opinião dos participantes do estudo experimental foi avaliada com um questionário respondido após a conclusão do software. O objetivo foi avaliar a satisfação do indivíduo quanto a experiência proposta. As questões Q01 e Q02 foram respondidas em uma escala de 0 a 5 e a Q09 foi disponibilizada como uma questão livre. As demais questões o aluno escolheu entre sim e não. As questões Q04, Q05, Q06, Q07 e Q08 foram destinada apenas ao grupo de intervenção. O questionário foi composto pelas seguintes questões:

Q01 O quanto a técnica proposta facilitou o desenvolvimento de casos de teste?
Q02 Você acha que sua produtividade foi prejudicada em virtude da técnica proposta?
Q03 Você utilizaria a técnica no desenvolvimento de futuros programas?
Q04 Após o período de adaptação da técnica, quando você precisa propor casos de teste em outras situações, você faz considerando os critérios de teste que aprendeu?

Q05 O desafio de atingir a cobertura mínima o motivou a melhorar seu conjunto de casos de teste?

Q06 Você acha que essa meta o estimulou a melhorar seu conjunto de casos de teste até que a meta fosse atingida?

Q07 A técnica me ajudou a localizar defeitos durante a refatoração?

Q08 Você se sentiu confortável ao praticar o TDD com critérios de teste de software?

Q09 Você tem algum comentário quanto a técnica proposta?

Na Tabela 5.14 estão dispostas as respostas dos participantes em relação ao questionário executado após o experimento. O grupo de controle definiu que o TDD facilitou consideravelmente o desenvolvimento do software proposto, com a variação entre 3 e 4. O grupo de intervenção definiu uma variação maior, entre 2 e 5. Em produtividade, o grupo de controle achou que o TDD não impactou no tempo e o grupo de intervenção respondeu que a técnica impactou em sua produtividade. Entretanto, com base na Figura 5.3 foi possível observar que o Grupo I executou a atividade em menos tempo.

Tabela 5.14. Resultados do questionário respondido após a execução 04.

Grupo	**Aluno**	**Q01**	**Q02**	**Q03**	**Q04**	**Q05**	**Q06**	**Q07**	**Q08**
C	Aluno 01	4,0	1,0	Sim	-	-	-	-	-
C	Aluno 02	4,0	0,0	Sim	-	-	-	-	-
C	Aluno 05	4,0	0,0	Sim	-	-	-	-	-
C	Aluno 07	3,0	3,0	Não	-	-	-	-	-
C	Aluno 10	3,0	1,0	Sim	-	-	-	-	-
I	Aluno 11	3,0	3,0	Sim	Sim	Sim	Sim	Sim	Sim
I	Aluno 14	5,0	2,0	Sim	Sim	Sim	Sim	Sim	Sim
I	Aluno 15	4,0	3,0	Sim	Sim	Sim	Sim	Sim	Sim
I	Aluno 17	2,0	3,0	Não	Não	Sim	Sim	Sim	Não

Todos os alunos do Grupo I consideraram as metas de cobertura como um desafio a ser atingido e este objetivo os ajudou a descobrir defeitos no código durante a refatoração. Além disso, assumiram que esta meta os motivou para melhorar o conjunto de casos de teste. O Aluno 17 foi o único do Grupo I que declarou que não utilizou os critérios de teste definido para derivar seu conjunto. Entretanto, considerou, assim como todos os outros, que os critérios foram como um desafio para que a meta fosse alcançada. O mesmo aluno declarou que não voltaria a utilizar a técnica proposta pois não se sentiu confortável.

Em relação a qualidade, os alunos do Grupo C ressaltaram que a técnica os ajudou a escrever código melhor durante o desenvolvimento. Foi salientado por um aluno do Grupo I que a estratégia o ajudou a entender o código que estava sendo desenvolvido e colaborou com a localização de defeitos.

Os resultados acadêmicos dos participantes do experimento após a execução do experimento mantiveram-se com pouca variação. Destacam-se neste ponto os alunos 05 (Grupo C) e 15 (Grupo I), que elevaram suas médias em média dois pontos. Todavia, não é possível afirmar que esta variação positiva é em virtude da prática do TDD ou ainda do TDD pautado em critérios de teste de software. Este ponto específico será discutido nas ameaças à validade.

5.3. Ameaças à validade

O estudo experimental da estratégia proposta nesta dissertação foi definido com base em estudos prévios que avaliam técnicas semelhantes. Todavia, é possível que existam variáveis relevantes que não foram consideradas para a avaliação dos resultados. Para reduzir este viés, foi proposto um GQM na Seção 3.3.

O TDD propõe a forma que o aluno desenvolve software. Desta forma, normalmente os alunos inicialmente têm problemas para aplicar o TDD. Mesmo que os alunos sejam treinados para a execução experimental, o TDD é uma técnica que necessita de prática para atingir a proficiência. Desta forma, um possível viés imputado pela falta de experiência com o TDD pode afetar a execução experimental, visto que a intervenção também tem como base a mesma técnica. Os alunos participantes da última execução receberam um treinamento durante maior período de tempo e foram executados vários exercícios em sala.

O desenvolvimento do software proposto é uma atividade que pode durar um longo tempo, o acadêmico pode ficar desgastado e desmotivado durante a atividade. Neste contexto, é possível que o resultado seja comprometido. A escolha de projetos simples, como o *Bowling Game*, amenizam este problema.

Outro fator que pode influenciar os resultados do presente trabalho é a utilização da ferramenta proposta para capturar os dados para avaliação. Se o aluno não utilizar a ferramenta como instruído pelo instrutor, os dados serão capturados incorretamente. Durante o último experimento executado (04), o acadêmico deveria indicar o estágio do TDD que estava sendo executado e a indicação incorreta desta informação pode alterar bruscamente os resultados. Com a finalidade de guiar o aluno a executar o TDD corretamente, é necessário que seja adicionado ao *plugin* uma validação de etapas, com o objetivo de demonstrar ao aluno, com base em seu histórico de execuções e nos dados capturados, qual etapa deve ser executada.

O *plugin* que captura as informações durante o estudo experimental necessita ser configurado corretamente e ainda o IDE deve conter as ferramentas necessárias para capturar informações esperadas. Além disso, durante o desenvolvimento o acadêmico necessita das

ferramentas descritas no protocolo do experimento para auxiliar no desenvolvimento. Com isso, espera-se evitar que o acadêmico da intervenção desenvolva casos de teste *ad hoc*. Para evitar esta ameaça, foi configurado um ambiente com todas as ferramentas necessárias para a execução do estudo experimental.

Como destacado na definição do protocolo experimental, o grupo de controle não deve ter o conhecimento da técnica aplicada pela intervenção. Todavia, durante o desenvolvimento do software é possível que os indivíduos troquem informações. Desta forma, o grupo de controle pode adicionar o viés esperado apenas para o grupo de intervenção. Tal fato pode ser agravado em virtude de ser aplicado durante uma disciplina, onde os alunos se conhecem. Esta ameaça pode ser mitigada em estudos futuros se o estudo experimental considerar apenas um grupo e em um primeiro momento os dois grupos apliquem apenas o TDD e posteriormente a estratégia proposta seja aplicada a todos os indivíduos envolvidos. Além disso, será possível estabelecer um comparativo sobre a qualidade do software final que o aluno desenvolve em ambos os experimentos.

Em virtude da execução experimental ser aplicada apenas durante um período em uma disciplina, não é possível afirmar que a prática da intervenção impactou no resultado acadêmico do indivíduo. Assim, para que tal comparação seja viável é necessário que seja aplicado em um período de tempo maior, tal como um bimestre, permitindo a comparação da prática do desenvolvimento comum com a estratégia estabelecida.

O grupo envolvido nos experimentos preliminares não representavam a população alvo da intervenção, mas a última execução considerou os acadêmicos com o perfil e ambiente alvo da intervenção. Entretanto, foram envolvidos poucos indivíduos. Para a generalização das hipóteses é necessário que sejam envolvidos mais acadêmicos, de outras instituições com currículos acadêmicos diferentes. Desta forma, os artefatos necessários para a execução do estudo experimental descrito na presente dissertação serão disponibilizados. Em estudos futuros, com base em um pacote experimental, espera-se que este estudo seja replicado por outros pesquisadores.

No contexto da execução dos estudos experimentais, é importante salientar que todas as execuções foram relatadas nesta dissertação foram aplicadas pela mesma pessoa que os definiu.

5.4. Considerações finais

Os estudos experimentais relatados neste capítulo foram realizados com a finalidade de avaliar a estratégia proposta e refinar o protocolo experimental. Inicialmente foram executados três estudos preliminares com limitação quanto as variáveis capturadas, assim, limitando a análise.

A partir das execuções preliminares foi definido um projeto experimental final, com base no GQM, visando avaliar como o objetivo seria atingido.

Na última execução foram capturadas as variáveis analisadas nos estudos preliminares e ainda foram adicionadas informações quanto a execução do TDD. Desta forma, foi possível analisar os dados por iteração e estágio do TDD. Além disso, foram adicionados questionários para avaliar a experiência prévia dos alunos e também a opinião quanto a técnica empregada.

A quantidade de indivíduos que participaram do experimento não foi adequada para que a hipótese nula fosse refutada. Todavia, foi possível observar que o TDD pautado em critérios de teste de software adiciona qualidade no conjunto de casos de teste e guia o acadêmico no desenvolvimento do software proposto. A utilização de critérios de teste de software em conjunto foi possível sem obstruir a prática tradicional do TDD. O software final dos praticantes da intervenção foi menos acoplado e menos complexo, todavia o grupo de controle teve melhores resultados em relação a coesão. Em geral os acadêmicos demonstraram-se favoráveis à estratégia proposta.

Capítulo 6

Conclusões

A presente dissertação foi desenvolvida com intuito de propor uma estratégia com base no *Test-Driven Development* em conjunto com técnicas de teste de software no contexto do aprendizado de programação em cursos da área de ciência da computação.

Com o intuito de obter o estado da arte da utilização do TDD em ensino foi desenvolvido um mapeamento sistemático da literatura buscando evidenciar as estratégias utilizadas e as propostas de utilização de técnicas de teste de software em conjunto com o TDD. Com efeito foram definidas cinco categorias: Avaliação Automática, Programação com elementos de motivação, *Test-Driven Learning*, Sistema inteligente de tutoria e Aprendizagem colaborativa. Além disso, foram verificados os benefícios e críticas atribuídas as estratégias que utilizam o TDD como técnica de desenvolvimento. No cenário específico do ensino, não foram localizados estudos, todavia, foram identificadas iniciativas da intersecção entre TDD e teste de software em outros contextos. Quanto ao uso de técnicas de teste de software com TDD, foi realizado um levantamento bibliográfico, identificando o uso de técnicas baseadas em defeitos e estrutural.

A partir do estado da arte definido, foi proposta uma estratégia que une a prática do TDD com critérios de teste de software. Nesta estratégia o praticante utiliza o TDD tal como especificado até a refatoração. No estágio de refatoração, concomitantemente com a melhoria do software, o praticante aperfeiçoa também o conjunto de casos de teste com base em critérios de teste de software. Com base na estratégia geral definida, foi então proposta uma estratégia específica, a qual utiliza critérios estruturais baseados em grafo de fluxo de controle para incrementar o conjunto de casos de teste previamente definido.

A estratégia proposta teve por objetivo manter os benefícios amplamente atribuídos a prática do TDD e ainda adicionar a cultura da busca por defeitos inerente as técnicas de

teste de software. Todavia é necessário que o contexto seja considerado, colaborando com a qualidade do software desenvolvido e não dificultando o processo. O foco principal desta dissertação são alunos que estão iniciando no desenvolvimento de software. O arcabouço proposto pode ser aplicado em outros contextos e utilizadas outras técnicas de teste de software.

A partir da estratégia específica definida, foi definido um estudo experimental preliminar e a partir deste foram efetuadas três execuções. As execuções evidenciaram a necessidade da definição de um GQM para auxiliar na definição da estratégia de avaliação da proposta, o qual é demonstrado de forma simplificada na Figura 6.1.

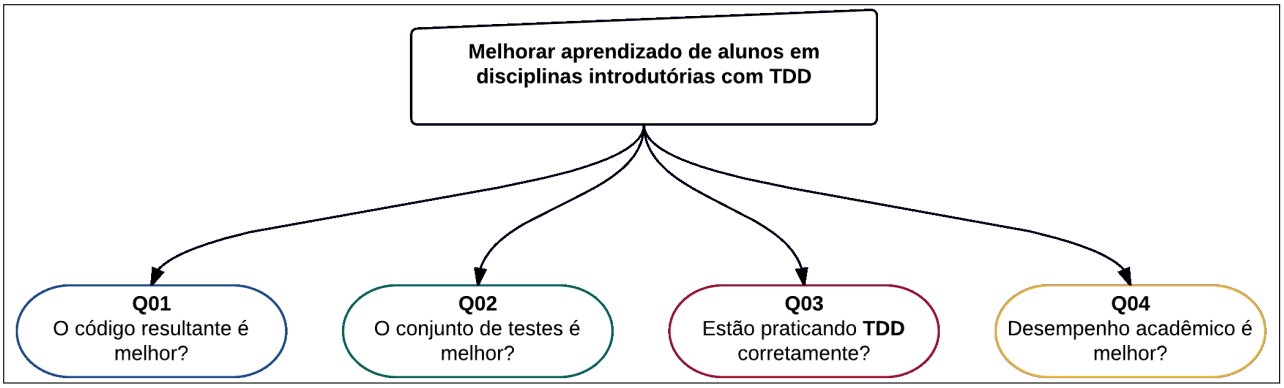

Figura 6.1. Diagrama *Goal Question Metric* resumido.

Durante a condução de experimentos científicos que avaliaram a execução do TDD, em comparação à prática tradicional de desenvolvimento de casos de teste (*test-last development*), a execução incorreta do TDD pode afetar o resultado do estudo. Este impacto pode ser acentuado se o experimento proposto for a comparação entre TDD e alguma variação da técnica, tal como os apresentados por CAUSEVIC *et al.*; SHELTON *et al.*; CAMARA; SILVA. Com o objetivo de mitigar este problema, foi proposta e construída uma ferramenta que captura as informações do desenvolvimento do software a cada execução do conjunto de casos de teste do acadêmico, obtendo algumas das informações necessárias para responder o conjunto de questões definidas no GQM.

A partir da coleta de dados, é necessário que sejam obtidos os dados capturados de modo simplificado, visto que a informação é captada e persistida em arquivos. Para isso, foi desenvolvida outra ferramenta que agrupa os dados e os apresenta com a finalidade de facilitar a análise das informações obtidas.

Com base nas informações captadas nas execuções experimentais, a hipótese proposta no protocolo do experimento foi avaliada. Os resultados obtidos serão discutidos a partir das questões definidas no GQM.

Q01 A comparação entre os softwares resultantes do experimento demonstrou que a

intervenção desenvolveu o software com menos acoplamento e com complexidade ciclomática menor. Entretanto, a coesão do grupo de controle foi melhor.

Q02 Os resultados demonstraram que a aplicação de critérios de teste de software pode melhorar a qualidade do software desenvolvido por estudantes e ainda apoiar a reflexão na ação. O TDD colabora com isso, mas nestes estudos observou-se que o uso de critérios de teste de software melhorou a qualidade do conjunto de casos de teste sem atrapalhar o fluxo tradicional do TDD.

Q03 Com base nas informações coletadas utilizando o *plugin* desenvolvido, foi possível analisar se o comportamento do acadêmico durante a execução experimental era compatível com a prática do TDD. Desta forma, os acadêmicos que não praticaram TDD como estabelecido pela técnica não foram considerados para a avaliação na execução 04, sendo que os dois grupos experimentais utilizaram TDD.

Q04 Os relatos quanto a utilização do TDD pautado em critérios de teste de software foram positivos. Todos os alunos da intervenção indicaram que os critérios foram um guia para melhorar o conjunto de casos de teste atuando como um motivador. Relacionaram ainda os critérios com a descoberta de defeitos durante o desenvolvimento.

Embora não seja possível refutar a hipótese nula com os resultados obtidos, é perceptível que a intervenção teve impacto positivo nos aspectos avaliados. Deste modo, é necessário que sejam definidas estratégias para mitigar as ameaças a validade identificadas e a partir disso disponibilizar um pacote experimental para que o estudo seja executado em maior escala. Além disso, é necessário que sejam definidas e avaliadas outras estratégias que combinem técnicas de teste. Utilizando critérios de teste que visam aumentar a força a cada estágio nas iterações e que sejam derivados de outras fontes de dados.

Os resultados obtidos durante o desenvolvimento da pesquisa relatada nesta dissertação foram publicados em eventos científicos:

- Com base nos resultados preliminares foi publicado no VI Congresso Científico da Região Centro-Ocidental do Paraná (CONCCEPAR) de 2015 um resumo com o título: Avaliação da eficácia da adoção de critérios de teste no contexto do aprendizado com Desenvolvimento baseado em Testes (CAMARA; SILVA, 2015).
- Com base nos resultados preliminares foi publicado um artigo científico no 47º ACM *Technical Symposium on Computing Science Education* no ano de 2016, com o título: *A Strategy to Combine Test-Driven Development and Test Criteria to Improve Learning of Programming Skills* (CAMARA; SILVA, 2016a).
- O relato da utilização da ferramenta desenvolvida foi publicado como resumo no VII CONCCEPAR de 2016 com o título: Uma ferramenta para apoiar o aprendizado de

programação com o desenvolvimento baseado em testes integrada com critérios de teste. Eleito o melhor dentre os resumos da área de computação (CAMARA; SILVA, 2016b).

O resultados das metas atingidas na presente dissertação são contribuições para a comunidade científica, são elas:

- O mapeamento sistemático buscou evidencias da utilização de técnicas de teste de software em conjunto com o TDD no contexto de ensino de programação. Não foram encontrados evidências da utilização de teste de software em conjunto com o TDD como estratégia de ensino de programação.
- Foram definidas medidas para avaliação de estudos experimentais que buscam avaliar a prática do TDD. Além disso, foi desenvolvida a TDD Traking uma ferramenta para capturar os dados necessários para a avaliação dos resultados.
- Foi definida uma estratégia geral e específica para a utilização de critérios de teste de software em conjunto com o TDD no contexto de ensino de programação.
- A estratégia foi executada por alunos em experimentos científicos e os resultados foram avaliados com base nas medidas definidas.

Algumas iniciativas propostas pela presente dissertação serão abordadas em trabalhos futuros. A ferramenta de captura de informações será otimizada para colaborar com o ensino do TDD sugestionando ao aluno o fluxo a ser seguido. Além disso, a integração entre o módulo de análise e o servidor será alterada para facilitar a coleta de informações e armazenamento de grandes volumes de dados. A ferramenta de análise será portada para ambiente Web a fim de facilitar a definição de um ambiente de replicação da estratégia e a obtenção das informações para análise.

No contexto da replicação do experimento, será estudada a utilização de outras técnicas de teste de software em conjunto com a estratégia definida. Na replicação do experimento, visa-se observar o efeito de saturação em relação a cobertura e assim utilizar outros critérios com mais força do que os de fluxo de controle. Neste sentido, seria possível avaliar o emprego de estratégias mais sofisticadas com os alunos, avaliar se o uso do TDD com critérios de teste pode contribuir com a melhoria do *design* do código em comparação ao TDD tradicional e se este esforço pode aliviar as atividades de teste de software em outros estágios de desenvolvimento de software.

Referências

ACREE, Allen Troy; BUDD, Timothy Alan; DEMILLO, Richard A.; LIPTON, Richard J.; SAYWARD, Frederick Gerald. *Mutation Analysis*. Atlanta, Georgia, EUA, set. 1979. 92 p.

ADAMS, Joel. Test-driven data structures: Revitalizing CS2. In: *40th ACM Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2009. p. 143–147. ISBN 978-1-60558-183-5.

ALLEVATO, Anthony; EDWARDS, Stephen H. Robolift: Engaging CS2 students with testable, automatically evaluated android applications. In: *43rd ACM Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2012. p. 547–552. ISBN 978-1-4503-1098-7.

AMMANN, Paul; OFFUTT, Jeff. *Introduction to software testing*. 1. ed. Cambridge, Reino Unido: Cambridge University, 2008. 344 p. ISBN 978-0521880381.

ANICHE, Mauricio. *Test-driven development: teste e design no mundo real*. 1. ed. São Paulo/SP: Casa do Código, 2013. 171 p. ISBN 978-85-66250-04-6.

ANICHE, Mauricio Finavaro; GEROSA, Marco Aurelio. How the Practice of TDD Influences Class Design in Object-Oriented Systems: Patterns of Unit Tests Feedback. In: *2012 26th Brazilian Symposium on Software Engineering*. Natal, Brasil: IEEE, 2012. p. 1–10. ISBN 978-0-7695-4868-5.

ARMOUR, Phillip G. The five orders of ignorance. *Communications of the ACM*, ACM, New York, NY, EUA, v. 43, n. 10, p. 17–20, out. 2000. ISSN 0001-0782.

ASTELS, David. *Test-Driven Development: A Practical Guide*. 1. ed. EUA: Prentice Hall PTR, 2003. 592 p. ISBN 978-0131016491.

AURELIANO, Viviane Cristina Oliveira; TEDESCO, Patrícia Cabral de Azevedo Restelli. Ensino-aprendizagem de programação para iniciantes: uma revisão sistemática da literatura focada no SBIE e WIE. In: *23º Simpósio Brasileiro de Informática na Educação*. Rio de Janeiro, Brasil: SBC, 2012. p. 1–10. ISSN 2316-6533.

BARBOSA, Ellen Francine; SILVA, Marco Aurélio Graciotto; CORTE, Camila Kozlowski Della; MALDONADO, José Carlos. Integrated teaching of programming foundations and software testing. In: *38th Annual Frontiers in Education Conference*. Saratoga Springs, NY, USA: IEEE, 2008. p. S1H–5 – S1H–10. ISBN 978-1-4244-1969-2. ISSN 0190-5848.

BARESI, Luciano; YOUNG, Michal. *Test Oracles*. Eugene, Oregon, EUA, ago. 2001. 55 p.

BASILI, V.R.; ROMBACH, H.D. The TAME project: towards improvement-oriented software environments. *Transactions on Software Engineering*, IEEE Computer Society, EUA, v. 14, n. 6, p. 758–773, jun. 1988. ISSN 0098-5589.

BASILI, Victor R. *Software Modeling and Measurement: The Goal/Question/Metric Paradigm*. College Park, MD, EUA, set. 1992. 24 p.

BECK, Kent. *Test-Driven Development: By Example*. 1. ed. EUA: Addison-Wesley Professional, 2002. 240 p. ISBN 978-0321146533.

BELLER, Moritz; GOUSIOS, Georgios; PANICHELLA, Annibale; ZAIDMAN, Andy. When, how, and why developers (do not) test in their IDEs. In: ACM. *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. EUA, 2015. p. 179–190.

BELLER, Moritz; GOUSIOS, Georgios; ZAIDMAN, Andy. How (much) do developers test? In: IEEE. *Proceedings of the 37th International Conference on Software Engineering (ICSE)*. IT, 2015. p. 559–562.

BINDER, R. V. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. [S.l.]: Addison Wesley Longman, Inc., 1999.

BIOLCHINI, Jorge; MIAN, Paula Gomes; NATALI, Ana Candida Cruz; TRAVASSOS, Guilherme Horta. *Systematic Review in Software Engineering*. Rio de Janeiro, RJ, Brasil, maio 2005.

BRIGGS, Tom; GIRARD, C. Dudley. Tools and techniques for test-driven learning in CS1. *J. Comput. Sci. Coll.*, Consortium for Computing Sciences in Colleges, EUA, v. 22, n. 3, p. 37–43, jan. 2007. ISSN 1937-4771.

BROWN, Neil Christopher Charles; KöLLING, Michael; MCCALL, Davin; UTTING, Ian. Blackbox: A large scale repository of novice programmers' activity. In: *45th ACM Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2014. (SIGCSE '14), p. 223–228. ISBN 978-1-4503-2605-6.

BUDD, T. A. *Mutation Analysis of Program Test Data*. Tese (Doutorado) — Yale University, New Haven, CT, maio 1980.

BUFFARDI, Kevin; EDWARDS, Stephen H. Exploring influences on student adherence to test-driven development. In: *17th ACM Annual Conference on Innovation and Technology in Computer Science Education*. New York, NY, EUA: ACM, 2012. p. 105–110. ISBN 978-1-4503-1246-2.

BUFFARDI, Kevin; EDWARDS, Stephen H. Effective and ineffective software testing behaviors by novice programmers. In: *9th Annual International ACM Conference on International Computing Education Research*. New York, NY, EUA: ACM, 2013. p. 83–90. ISBN 978-1-4503-2243-0.

CAMARA, Bruno Henrique Pachulski; SILVA, Marco Aurélio Graciotto. *Avaliação da eficácia da adoção de critérios de teste no contexto do aprendizado com Desenvolvimento baseado em Testes*. Maio 2015.

CAMARA, Bruno Henrique Pachulski; SILVA, Marco Aurélio Graciotto. A strategy to combine test-driven development and test criteria to improve learning of programming skills. In: *47th ACM Technical Symposium on Computing Science Education*. New York, NY, EUA: ACM, 2016. p. 443–448. ISBN 978-1-4503-3685-7.

CAMARA, Bruno Henrique Pachulski; SILVA, Marco Aurélio Graciotto. *Uma ferramenta para apoiar o aprendizado de programação com o desenvolvimento baseado em testes integrada com critérios de teste*. Maio 2016.

CAUSEVIC, A.; PUNNEKKAT, S.; SUNDMARK, D. Quality of testing in test driven development. In: *8th International Conference on the Quality of Information and Communications Technology*. PT: IEEE, 2012. p. 266–271. ISBN 9780769547770.

CAUSEVIC, A.; PUNNEKKAT, S.; SUNDMARK, D. TDD HQ: Achieving higher quality testing in test driven development. In: *39th Euromicro Conference Series on Software Engineering and Advanced Applications*. Santander, ES: IEEE, 2013. p. 33–36. ISBN 9780769550916.

CAUSEVIC, A.; SHUKLA, R.; PUNNEKKAT, S. Industrial study on test driven development: Challenges and experience. In: *1st International Workshop on Conducting Empirical Studies in Industry*. EUA: Springer, 2013. p. 15–20. ISBN 978-1-4673-6286-3.

CAUSEVIC, Adnan; SHUKLA, Rakesh; PUNNEKKAT, Sasikumar; SUNDMARK, Daniel. Effects of negative testing on TDD: An industrial experiment. In: BAUMEISTER, Hubert;

WEBER, Barbara (Ed.). *14th International Conference on Agile Software Development*. Vienna, IT: Springer Berlin Heidelberg, 2013. (Lecture Notes in Business Information Processing, v. 149), p. 91–105. ISBN 978-3-642-38313-7.

CAUSEVIC, A.; SUNDMARK, D.; PUNNEKKAT, S. Impact of test design technique knowledge on test driven development: A controlled experiment. In: *13th International Conference on Agile Software Development*. CH: Springer, 2012. (Lecture Notes in Business Information Processing, v. 111), p. 138–152. ISBN 9783642303494. ISSN 18651348.

CAUSEVIC, A.; SUNDMARK, D.; PUNNEKKAT, S. Test case quality in test driven development: A study design and a pilot experiment. In: *16th International Conference on Evaluation and Assessment in Software Engineering*. ES: IEEE, 2012. p. 223–227. ISBN 9781849195416.

CHIDAMBER, Shyam R.; KEMERER, Chris F. A metrics suite for object oriented design. *Transactions on Software Engineering*, IEEE Computer Society, EUA, v. 20, n. 6, p. 476–493, jun. 1994. ISSN 0098-5589.

CRISPIN, L. Driving software quality: How test-driven development impacts software quality. *IEEE Software*, v. 23, n. 6, p. 70–71, Nov 2006. ISSN 0740-7459.

DAVIS, F. D.; VENKATESH, V. Toward preprototype user acceptance testing of new information systems: implications for software project management. *IEEE Transactions on Engineering Management*, v. 51, n. 1, p. 31–46, Fev 2004. ISSN 0018-9391.

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario. *Introdução ao Teste de Software*. SP, Brasil: Elsevier, 2007. 394 p. (Campus).

DEMILLO, Richard A.; LIPTON, Richard J.; SAYWARD, Frederick G. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, v. 11, n. 4, p. 34–41, abr. 1978.

DESAI, Chetan; JANZEN, David S.; CLEMENTS, John. Implications of integrating test-driven development into CS1/CS2 curricula. In: *40th ACM Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2009. p. 148–152. ISBN 978-1-60558-183-5.

DURELLI, Vinicius Humberto Serapilha; ARAUJO, Rodrigo Fraxino; SILVA, Marco Aurelio Graciotto; OLIVEIRA, Rafael Alves Paes de; MALDONADO, Jose Carlos; DELAMARO, Marcio Eduardo. A scoping study on the 25 years of research into software testing in Brazil and an outlook on the future of the area. *Journal of Systems and Software*, Elsevier, Países Baixos, v. 86, n. 4, p. 934–950, abr. 2013. ISSN 0164-1212.

EDWARDS, Stephen H. Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing*, ACM, New York, NY, EUA, v. 3, n. 3, p. 1:1–1:24, set. 2003. ISSN 1531-4278.

EDWARDS, Stephen H. Rethinking computer science education from a test-first perspective. In: *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. New York, NY, EUA: ACM, 2003. p. 148–155. ISBN 1-58113-751-6.

EDWARDS, Stephen H. Using software testing to move students from trial-and-error to reflection-in-action. In: *35th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2004. p. 26–30. ISBN 1-58113-798-2.

EDWARDS, Stephen H. Using software testing to move students from trial-and-error to reflection-in-action. *SIGCSE Bull.*, ACM, New York, NY, EUA, v. 36, n. 1, p. 26–30, mar. 2004. ISSN 0097-8418.

EDWARDS, Stephen H.; PéREZ-QUIñONES, Manuel A. Experiences using test-driven development with an automated grader. *Journal of Computing Sciences in Colleges*, Consortium for Computing Sciences in Colleges, EUA, v. 22, n. 3, p. 44–50, jan. 2007. ISSN 1937-4771.

EDWARDS, Stephen H.; SHAMS, Zalia. Do student programmers all tend to write the same software tests? In: *19th Annual Conference on Innovation and Technology in Computer Science Education*. New York, NY, EUA: ACM, 2014. p. 171–176. ISBN 978-1-4503-2833-3.

ERDOGMUS, H.; MORISIO, Maurizio; TORCHIANO, Marco. On the effectiveness of the test-first approach to programming. *Transactions on Software Engineering*, IEEE Computer Society, EUA, v. 31, n. 3, p. 226–237, mar. 2005. ISSN 0098-5589.

FUCCI, D.; TURHAN, B. A replicated experiment on the effectiveness of test-first development. In: *7th International Symposium on Empirical Software Engineering and Measurement*. Baltimore, MD, EUA: [s.n.], 2013. p. 103–112. ISBN 978-0-7695-5056-5. ISSN 1938-6451.

FUCCI, Davide; TURHAN, Burak. On the role of tests in test-driven development: A differentiated and partial replication. *Empirical Software Engineering*, Springer, EUA, v. 19, n. 2, p. 277–302, abr. 2014. ISSN 1382-3256.

GASPAR, Alessio; LANGEVIN, Sarah. Restoring "coding with intention"in introductory programming courses. In: *ACM SIGITE Conference on Information Technology Education*. New York, NY, EUA: ACM, 2007. p. 91–98. ISBN 978-1-59593-920-3.

GEORGE, Boby; WILLIAMS, Laurie. A structured experiment of test-driven development. *Information and Software Technology*, v. 46, n. 5, p. 337 – 342, 2004. ISSN 0950-5849.

HARROLD, M. J.; ROTHERMEL, G. Performing data flow testing on classes. In: *ACM SIGSOFT Symposium on Foundations of Software Engineering*. New York: ACM Press, 1994. p. 154–163.

HEINONEN, Kenny; HIRVIKOSKI, Kasper; LUUKKAINEN, Matti; VIHAVAINEN, Arto. Learning agile software engineering practices using coding dojo. In: *14th Annual ACM SIGITE Conference on Information Technology Education*. New York, NY, EUA: ACM, 2013. p. 97–102. ISBN 978-1-4503-2239-3.

HILTON, Michael; JANZEN, David S. On teaching arrays with test-driven learning in webide. In: *17th ACM Annual Conference on Innovation and Technology in Computer Science Education*. New York, NY, EUA: ACM, 2012. p. 93–98. ISBN 978-1-4503-1246-2.

INOZEMTSEVA, Laura; HOLMES, Reid. Coverage is not strongly correlated with test suite effectiveness. In: *36th International Conference on Software Engineering*. New York, NY, EUA: ACM, 2014. p. 435–445. ISBN 978-1-4503-2756-5.

ISO. *ISO/IEC/IEEE 24765:2010 – Systems and software engineering – Vocabulary*. dez. 2010. 418 p.

ISOMöTTöNEN, Ville; LAPPALAINEN, Vesa. CSI with games and an emphasis on TDD and unit testing: Piling a trend upon a trend. *ACM Inroads*, ACM, New York, NY, EUA, v. 3, n. 3, p. 62–68, set. 2012. ISSN 2153-2184.

JANZEN, David; SAIEDIAN, Hossein. Test-driven learning in early programming courses. In: *39th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2008. p. 532–536. ISBN 978-1-59593-799-5.

JANZEN, D. S.; SAIEDIAN, H. On the influence of test-driven development on software design. In: *19th Conference on Software Engineering Education Training (CSEET'06)*. Oahu, Hawaii, EUA: IEEE, 2006. p. 141–148. ISSN 1093-0175.

JANZEN, David S.; SAIEDIAN, Hossein. Test-driven learning: Intrinsic integration of testing into the cs/se curriculum. In: *37th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2006. p. 254–258. ISBN 1-59593-259-3.

KITCHENHAM, Barbara; CHARTERS, Stuart. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. UK, 2007.

KITCHENHAM, Barbara A.; DYBA, Tore; JORGENSEN, Magne. Evidence-based software engineering. In: *26th Internacional Conference on Software Engineering*. GB: ACM, 2004. p. 273–281. ISBN 0-7695-2163-0. ISSN 0270-5257.

KOLLANUS, S. Test-driven development - still a promising approach? In: *7th International Conference on the Quality of Information and Communications Technology*. Porto: IEEE, 2010. p. 403–408. ISBN 9780769542416.

LAHTINEN, Essi.; ALA-MUTKA, Kirsti; JäRVINEN, Hannu-Matti. A study of the difficulties of novice programmers. In: *10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. New York, NY, EUA: ACM, 2005. p. 14–18. ISBN 1-59593-024-8.

LAPPALAINEN, Vesa; ITKONEN, Jonne; ISOMöTTöNEN, Ville; KOLLANUS, Sami. ComTest: a tool to impart TDD and unit testing to introductory level programming. In: *15th Annual Conference on Innovation and Technology in Computer Science Education*. USA: ACM, 2010. p. 63–67. ISBN 978-1-60558-820-9.

LETHBRIDGE, Timothy C.; DIAZ-HERRERA, Jorge; LEBLANC, Richard J. Jr.; THOMPSON, J. Barrie. Improving software practice through education: Challenges and future trends. In: *Future of Software Engineering*. EUA: IEEE Computer Society, 2007. p. 12–28. ISBN 0-7695-2829-5.

LEUNG, H. K. N.; WHITE, L. A study of integration testing and software regression at the integration level. In: *Conference on Software Maintenance-90*. San Diego, CA: IEEE, 1990. p. 290–301.

LI, Nan; PRAPHAMONTRIPONG, Upsorn; OFFUTT, Jeff. An Experimental Comparison of Four Unit Test Criteria: Mutation, Edge-Pair, All-Uses and Prime Path Coverage. In: *2009 International Conference on Software Testing, Verification, and Validation Workshops*. EUA: IEEE, 2009. p. 220–229. ISBN 978-0-7695-3671-2.

MALDONADO, J. C. *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. Tese (Doutorado) — DCA/FEEC/UNICAMP, Campinas, SP, Brazil, jul. 1991.

MALDONADO, J. C.; VINCENZI, A. M. R.; BARBOSA, E. F.; SOUZA, S. R. S.; DELAMARO, M. E. *Aspectos Teóricos e Empíricos de Teste de Cobertura de Software*. Campinas, SP, BR, jun. 1998.

MARRERO, Will; SETTLE, Amber. Testing first: Emphasizing testing in early programming courses. In: *10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. New York, NY, EUA: ACM, 2005. p. 4–8. ISBN 1-59593-024-8.

MARTIN, Robert C. *The Bowling Game Kata*. 2014. Disponível em: `http://butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata`.

MATHUR, Aditya P. *Foundations of Software Testing*. 1. ed. EUA: Pearson Education, 2008. 689 p. ISBN 978-8131716601.

MCKINNEY, Dawn; DENTON, Leo F. Developing collaborative skills early in the cs curriculum in a laboratory environment. In: *37th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2006. p. 138–142. ISBN 1-59593-259-3.

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

MYERS, Glenford. J. *The Art of Software Testing*. EUA: John Wiley & Sons, 1979. 177 p.

MYERS, Glenford J. *The Art of Software Testing*. 2. ed. EUA: John Wiley & Sons, 2004.

MYERS, Glenford J.; BADGETT, Tom; SANDLER, Corey. *The Art of Software Testing*. 3. ed. Hoboken, NJ, EUA: John Wiley & Sons, 2013. 240 p. ISBN 978-1-118-03196-4.

PETERSEN, Kai. Measuring and predicting software productivity: A systematic map and review. *Information and Software Technology*, v. 53, n. 4, p. 317 – 343, 2011. ISSN 0950-5849.

PETERSEN, Kai; FELDT, Robert; MUJTABA, Shahid; MATTSSON, Michael. Systematic mapping studies in software engineering. In: . IT: ACM, 2008. p. 68–77.

PFLEEGER, Shari Lawrence. Design and analysis in software engineering: the language of case studies and formal experiments. *SIGSOFT Software Engineering Notes*, ACM, New York, NY, EUA, v. 19, n. 4, p. 16–20, out. 1994. ISSN 0163-5948.

PFLEEGER, Shari Lawrence. Experimental design and analysis in software engineering: Part 2: How to set up and experiment. *SIGSOFT Software Engineering Notes*, ACM, New York, NY, EUA, v. 20, n. 1, p. 22–26, jan. 1995. ISSN 0163-5948.

PFLEEGER, Shari Lawrence. Experimental design and analysis in software engineering, part 4: choosing an experimental design. *SIGSOFT Software Engineering Notes*, ACM, New York, NY, EUA, v. 20, n. 3, p. 13–15, jul. 1995. ISSN 0163-5948.

PFLEEGER, Shari Lawrence. Experimental design and analysis in software engineering, part 5: analyzing the data. *SIGSOFT Software Engineering Notes*, ACM, New York, NY, EUA, v. 20, n. 5, p. 14–17, dez. 1995. ISSN 0163-5948.

PFLEEGER, Shari Lawrence. Experimental design and analysis in software engineering: Types of experimental design. *SIGSOFT Software Engineering Notes*, ACM, New York, NY, EUA, v. 20, n. 2, p. 14–16, abr. 1995. ISSN 0163-5948.

RAPPS, Sandra; WEYUKER, Elaine J. Data flow analysis techniques for program test data selection. In: *6th International Conference on Software Engineering*. Tokyo, Japan: IEEE Computer Society Press, 1982. p. 272–278.

RAPPS, Sandra; WEYUKER, Elaine J. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, v. 11, n. 4, p. 367–375, abr. 1985.

SCHAUB, Stephen. Teaching cs1 with web applications and test-driven development. *SIGCSE Bull.*, ACM, New York, NY, EUA, v. 41, n. 2, p. 113–117, jun. 2009. ISSN 0097-8418.

SHAMS, Zalia; EDWARDS, Stephen H. Toward practical mutation analysis for evaluating the quality of student-written software tests. In: *9th Annual International ACM Conference on International Computing Education Research*. New York, NY, EUA: ACM, 2013. p. 53–58. ISBN 978-1-4503-2243-0.

SHAW, Mary. Software engineering education: A roadmap. In: *22nd International Conference on Software Engineering*. New York, NY, EUA: ACM, 2000. p. 371–380. ISBN 1-58113-253-0.

SHELTON, W.; LI, Nan; AMMANN, P.; OFFUTT, J. Adding criteria-based tests to test driven development. In: *IEEE 5h International Conference on Software Testing, Verification and Validation*. Montreal, QC, Canada: IEEE, 2012. p. 878–886. ISBN 978-1-4577-1906-6.

SOUZA, D. M.; MALDONADO, J.C.; BARBOSA, E.F. Progtest: Apoio automatizado ao ensino integrado de programaçao e teste de software. In: *XXII Simposio Brasileiro de Informatica na Educacao - XVII Workshop de Informatica na Educacao, SBIE-WIE*. Aracaju, SE, Brazil: SBC, 2011. v. 11, p. 1893–1897. ISSN 2176-4301.

SOUZA, Draylson Micael de; MALDONADO, Jose Carlos; BARBOSA, Ellen Francine. ProgTest: An environment for the submission and evaluation of programming assignments based on testing activities. In: *24th Conference on Software Engineering Education and Training*. EUA: IEEE Computer Society, 2011. p. 1–10. ISBN 978-1-4577-0349-2. ISSN 1093-0175.

SOUZA, Draylson Micael de; ISOTANI, Seiji; BARBOSA, Ellen Francine. Teaching novice programmers using ProgTest. *International Journal of Knowledge and Learning*, InderScience, Reino Unido, v. 10, n. 1, p. 60–77, 2015. ISSN 1741-1009.

SPACCO, Jaime; HOVEMEYER, David; PUGH, William; EMAD, Fawzi; HOLLINGSWORTH, Jeffrey K.; PADUA-PEREZ, Nelson. Experiences with Marmoset: designing and using an advanced submission and testing system for programming courses. In: *11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. New York, NY, EUA: ACM, 2006. p. 13–17. ISBN 1-59593-055-8.

THORNTON, M.; EDWARDS, S. H.; TAN, R. P. Helping students test programs that have graphical user interfaces. In: SU, Hui Fang; TREMANTE, Andrés; WELSCH, Friedrich (Ed.). *5th International Conference on Education and Information Systems, Technologies and Applications 2007*. Orlando, FL, EUA: [s.n.], 2007. v. 2, p. 164–169. ISBN 9781934272251.

THORNTON, Matthew; EDWARDS, Stephen H.; TAN, Roy P.; PEREZ-QUINONES, Manuel A. Supporting student-written tests of GUI programs. In: *39th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, EUA: ACM, 2008. p. 537–541. ISBN 978-1-59593-799-5.

VINCENZI, Auri Marcelo Rizzo. *Object-oriented: definition, implementation and analysis of validation and testing resources*. São Carlos, SP, Brasil: Universidade de São Paulo, maio 2004.

VINCENZI, Auri Marcelo Rizzo; DELAMARO, Márcio Eduardo; HöHN, Erika Nina; MALDONADO, José Carlos. *Functional, Control and Data Flow, and Mutation Testing: Theory and Practice*. 2007. Summer course on testing.

VINCENZI, A. M. R.; WONG, W. E.; DELAMARO, M. E.; MALDONADO, J. C. JaBUTi: A coverage analysis tool for java programs. In: *Sessão de Ferramentas do Simpósio Brasileiro de Engenharia de Software (SBES)*. Manaus, AM, Brasil: SBC, 2003. p. 79–84.

XU, Shaochun; RAJLICH, V. Empirical validation of test-driven pair programming in game development. In: *5th IEEE/ACIS International Conference on Computer and Information Science, 2006 and 2006 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse.* Honolulu, HI, EUA: IEEE, 2006. p. 500–505.

ZHANG, Dongsong; ZHOU, Lina; BRIGGS, Robert O.; NUNAMAKER JR., Jay F. Instructional video in e-learning: Assessing the impact of interactive video on learning effectiveness. *Information and Management*, Elsevier Science, Amsterdam, Países Baixos, v. 43, p. 15–27, January 2006. ISSN 0378-7206.