Jogos educacionais: uma contribuição para o ensino de teste de software

Pedro Henrique Dias Valle

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Pedro Henrique Dias Valle

Jogos educacionais: uma contribuição para o ensino de teste de software

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. José Carlos Maldonado

USP – São Carlos
Fevereiro de 2017

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

V181j Valle, Pedro Henrique Dias
 Jogos educacionais: uma contribuição para
o ensino de teste de software / Pedro Henrique
Dias Valle; orientador José Carlos Maldonado. -
São Carlos - SP, 2017.
 148u p.

 Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática Computacional)
- Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2017.

 1. Jogos Educacionais. 2. Ensino de Teste de
Software. 3. Técnicas de Teste de Software. I.
Maldonado, José Carlos, orient. II. Título.

Pedro Henrique Dias Valle

Educational games: a contribution to the software testing education

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. José Carlos Maldonado

USP – São Carlos
February 2017

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida e pela sua misericórdia infinita, por sempre me ajudar nos momentos mais difíceis e por ter me dado a oportunidade de cursar o mestrado.

À minha família, em especial minha mãe Luzmaia, meu pai Randolpho e minha avó Aurora, pelo amor, incentivo e carinho. Obrigado por sempre acreditarem em mim. Sem vocês esse sonho não se realizaria.

Ao meu orientador José Carlos Maldonado pela orientação, apoio e diversas reflexões e a todos os professores que contribuíram para minha formação.

Aos meus amigos Renato, Raphael, Danillo, Renan, Luana e Pedro Alexandre que sempre me ouviram, orientaram e me apoiaram nos momentos mais difíceis.

Aos integrantes do Laboratório de Engenharia de Software (Labes), em especial meus amigos: Nilton, Iohan, Ricardo, Joice, Draylson, Silvana, Jacson, Cristiane, Armando, Diogenes, Carlos, Alinne, Lina, Lilian, Brauner, Eliane, Jorge e Delcio pelos momentos de descontração e companhia durante essa jornada.

Ao grupo de jovens Huiós pelas orações.

A todos os funcionários do ICMC, pelos serviços prestados.

Aos membros da banca pela disponibilidade e por aceitarem participar da minha banca de defesa.

Ao Instituto de Ciências Matemáticas e de Computação e a Universidade de São Paulo.

Por fim, à CAPES pelo apoio financeiro.

A todos, muito obrigado.

*“Dance como se ninguém estivesse vendo,
ame como se você jamais fosse se ferir,
cante como se ninguém estivesse ouvindo.
Viva como se a terra fosse o céu.”*

(Willian W. Purkey)

RESUMO

VALLE, P. H. D. **Jogos educacionais: uma contribuição para o ensino de teste de software.** 2017. 166 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

O teste de software é considerado uma importante atividade na garantia da qualidade de produtos de software. No entanto, há uma carência de profissionais qualificados nessa área. Isso pode ser ocasionado pela dificuldade de ensinar teste de software por meio de abordagens que utilizem apenas aulas teóricas e ferramentas de teste tradicionais. Além disso, há uma desmotivação decorrente do ambiente de trabalho e das estratégias de alocação e responsabilidade desses profissionais nas equipes de desenvolvimento e teste. Para amenizar esses problemas, têm sido utilizadas outras abordagens de apoio ao ensino de teste de software, tais como: jogos educacionais, ensino de teste com programação, módulos educacionais, entre outras. O objetivo deste projeto de mestrado foi desenvolver um jogo educacional, denominado Testing Game, para auxiliar o ensino de teste de software, especificamente: teste funcional, teste estrutural e teste de mutação. Para auxiliar o desenvolvimento do Testing Game, foi realizado um mapeamento sistemático para selecionar um motor de jogos. Na primeira versão do jogo foi utilizado o motor de jogos Cocos2D e na segunda versão foi utilizado o Construct 2. Para avaliar a eficiência do Testing Game, realizou-se um estudo de viabilidade com o intuito de avaliar a qualidade com relação à motivação, experiência do usuário e aprendizagem sob o ponto de vista dos estudantes. Além disso, avaliou-se a usabilidade do Testing Game. Aproximadamente 85,64% das pessoas que participaram do estudo avaliaram a qualidade do jogo de forma positiva com relação à motivação, experiência do usuário e aprendizagem sob o ponto de vista dos estudantes. Quanto à usabilidade do jogo, foram identificados poucos problemas, o que possibilita a liberação do jogo. Por meio deste trabalho, percebeu-se que o jogo Testing Game poderia ser utilizado como um recurso complementar de apoio ao ensino de teste de software, e sua efetividade ser avaliada.

Palavras-chave: Jogos Educacionais, Ensino de Teste de Software, Técnicas de Teste de Software.

ABSTRACT

VALLE, P. H. D. **Educational games: a contribution to the software testing education.** 2017. 166 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Software testing is a relevant activity to provide evidences of quality of software products. However, there is a lack of qualified professionals in this area. This can be caused due to difficulty in teaching software testing through approaches that use only theoretical classes traditional tools. In addition, there is a lack of motivation due to the work environment and the strategies of allocation and responsibility of these professionals in development and testing teams. To mitigate these problems, approaches have been used to support software testing education, such as: educational games, integrated teaching of software testing with programming, educational modules, among others. The objective of this master's thesis was to develop an educational game named Testing Game, addressing the following topics: functional testing, structural testing and mutation testing. To support the development of the Testing Game, we performed a systematic mapping aiming at selecting a game engine. In the first game version, we used Cocos2D and in the second one we used Construct 2. To evaluate the efficiency of the game, we conducted a feasibility study to evaluate the quality regarding motivation, user experience and learning from the point of view of the students. Moreover, we also evaluate the usability of the Testing Game. Approximately 85.64% of people who participated in the study assessed the quality of the game in a positive perspective regarding motivation, user experience and learning from the point of view of the students. Regarding the usability of the game, students identified minor problems were identified, which allows the release of the game. Through this work, we realize that the game Testing Game can be used as a complementary resource to support software testing education, and its effectiveness be evaluated.

Keywords: Educational Games, Software Testing Education, Software Testing Techniques.

LISTA DE ILUSTRAÇÕES

Figura 1 – Cenário típico da atividade de teste. Adaptado de (DELAMARO; MALDONADO; JINO, 2016)	29
Figura 2 – Funcionalmente do Teste Funcional	31
Figura 3 – Código do Programa BubbleSort em Java	33
Figura 4 – Grafo de Fluxo de Controle do programa Bubble Sort	34
Figura 5 – Grafo de Def-Uso do programa Bubble Sort	35
Figura 6 – Abordagens para Auxiliar o Ensino de Teste de Software (VALLE; BARBOSA; MALDONADO, 2015b)	39
Figura 7 – Elementos Essenciais à Qualidade de Jogos Educacionais. Adaptado de (ANNETTA, 2010).	46
Figura 8 – Visão Geral da Metodologia de Desenvolvimento de Recursos Educacionais (ROCHA <i>et al.</i> , 2016)	49
Figura 9 – Visão Geral das Atividades da Metodologia de Desenvolvimento de Recursos Educacionais (ROCHA <i>et al.</i> , 2016)	50
Figura 10 – <i>Workflow</i> do subprocesso de planejamento inicial (ROCHA <i>et al.</i> , 2016)	51
Figura 11 – <i>Workflow</i> do subprocesso de projeto iterativo (ROCHA <i>et al.</i> , 2016)	52
Figura 12 – Linha do Tempo com os Jogos Educacionais no Domínio de Teste de Software	54
Figura 13 – Tela inicial do jogo U-TEST (SILVA; THIRY, 2010)	55
Figura 14 – Tela inicial do Jogo das 7 Falhas (DINIZ; DAZZI, 2011b)	55
Figura 15 – Jogo JETS (Jogo da Equipe de Teste de Software) (SILVA; MULLER, 2012)	56
Figura 16 – Primeira fase do jogo <i>iTest Learning</i> (FARIAS <i>et al.</i> , 2012)	57
Figura 17 – Fase de Separar Conceitos do Jogo <i>iLearnTest</i> (RIBEIRO; PAIVA, 2015)	57
Figura 18 – Tabuleiro do Jogo <i>JoVeTest</i> (BARBOSA; NEVES; NETO, 2016)	58
Figura 19 – Metodologia utilizada para o desenvolvimento do jogo <i>Testing Game</i>	62
Figura 20 – Fase do <i>Testing Game</i> - Versão Inicial - sobre vetores com o tamanho válido	66
Figura 21 – Fase do <i>Testing Game</i> - Versão Inicial - sobre o critério Todos-Nós	66
Figura 22 – Visão Geral da Metodologia de Desenvolvimento de Recursos Educacionais (ROCHA <i>et al.</i> , 2016)	68
Figura 23 – Níveis do Jogo <i>Testing Game</i>	71
Figura 24 – SubFases do Teste Funcional	72
Figura 25 – Menu Superior do Jogo <i>Testing Game</i>	72
Figura 26 – Exemplos de imagens utilizadas no jogo <i>Testing Game</i>	73
Figura 27 – Interface do Ambiente de Desenvolvido do Construct 2	76

Figura 28 – Primeira subfase do Teste Funcional no Testing Game	78
Figura 29 – Segunda subfase do Teste Funcional no Testing Game	78
Figura 30 – Quinta subfase do Teste Funcional no Testing Game	79
Figura 31 – Primeira subfase do Teste Estrutural no Testing Game	81
Figura 32 – Quinta subfase do Teste Estrutural no Testing Game	82
Figura 33 – Sétima subfase do Teste Estrutural no Testing Game	83
Figura 34 – Oitava subfase do Teste Estrutural no Testing Game	83
Figura 35 – Primeira subfase do Teste de Mutação no Testing Game	85
Figura 36 – Quarta subfase do Teste de Mutação no Testing Game	86
Figura 37 – Desafios da Subfase 2 do Teste Estrutural no Testing Game	89
Figura 38 – Caso de teste válido na Subfase 2 do Teste Estrutural no Testing Game	89
Figura 39 – Moeda que representa a pontuação no Testing Game	90
Figura 40 – Segundo caso de teste válido na Subfase 2 do Teste Estrutural no Testing Game	90
Figura 41 – Caso de teste inválido na Subfase 2 do Teste Estrutural no Testing Game . . .	91
Figura 42 – Pontuação subtraída na Subfase do Teste Estrutural no Testing Game	91
Figura 43 – Terceiro caso de teste válido na Subfase 2 do Teste Estrutural no Testing Game	91
Figura 44 – Quarta subfase do Teste de Mutação no Testing Game	92
Figura 45 – Quarta subfase do Teste de Mutação no Testing Game	92
Figura 46 – Avaliação do Jogo com Relação ao Subcomponente Motivação	103
Figura 47 – Avaliação do Jogo com Relação ao Subcomponente Experiência do Usuário	104
Figura 48 – Avaliação do Jogo com Relação ao Subcomponente Aprendizagem	105
Figura 49 – Frequência de problemas encontrados por heurística no jogo Testing Game .	106
Figura 50 – Modelo de Avaliação de Jogos Educacionais. (Adaptado de Savi, Wange- nheim e Borgatto (2011))	144

LISTA DE TABELAS

Tabela 1 – Universidades brasileiras que tiveram seus currículos analisados (VALLE; BARBOSA; MALDONADO, 2015a)	41
Tabela 2 – Universidades do exterior que tiveram seus currículos analisados (VALLE; BARBOSA; MALDONADO, 2015a)	43
Tabela 3 – Características dos Jogos Educacionais no Domínio de Teste de Software . .	59
Tabela 4 – Conteúdos abordados no Jogo Testing Game	70
Tabela 5 – Motores de Jogos Identificados no Mapeamento Sistemático	75
Tabela 6 – Comparação do jogo Testing Game com jogos no domínio de teste de software	93
Tabela 7 – Perfil dos sujeitos que participaram da primeira etapa do estudo de viabilidade	100
Tabela 8 – Perfil dos sujeitos que participaram da segunda etapa do estudo de viabilidade	102
Tabela 9 – Lista de problemas encontradas na avaliação heurística do Jogo Testing Game	106
Tabela 10 – Relatório de Defeitos	142
Tabela 11 – Severidades para os problemas de usabilidade identificados na avaliação heurística	146
Tabela 12 – Conjunto de Heurísticas Proposto por Nielsen (1994) Para Avaliação da Usabilidade	146

SUMÁRIO

1	Introdução	21
1.1	Contextualização e Motivação	21
1.2	Objetivos	24
1.3	Metodologia	24
1.4	Organização	25
2	Fundamentação Teórica	27
2.1	Considerações Iniciais	27
2.2	Teste de Software	27
2.2.1	<i>Conceitos Básicos</i>	<i>28</i>
2.2.2	<i>Técnicas e Critérios de Teste</i>	<i>30</i>
2.2.2.1	<i>Técnica Funcional</i>	<i>30</i>
2.2.2.2	<i>Teste Estrutural</i>	<i>32</i>
2.2.2.3	<i>Teste Baseado em Defeitos</i>	<i>35</i>
2.3	Ensino de Teste de Software	36
2.3.1	<i>Mapeamento Sistemático sobre Abordagens para Auxiliar o Ensino de Teste de Software</i>	<i>38</i>
2.3.2	<i>Currículos das Principais Universidades do Brasil e do Exterior</i>	<i>40</i>
2.4	Jogos Educacionais	43
2.4.1	<i>Elementos Essenciais a Qualidade de Jogos Educacionais</i>	<i>45</i>
2.4.2	<i>Processos de Desenvolvimento de Jogos Educacionais</i>	<i>47</i>
2.4.3	<i>Classificação dos Jogos Educacionais</i>	<i>52</i>
2.4.4	<i>Jogos Educacionais no Domínio de Teste de Software</i>	<i>53</i>
2.5	Considerações Finais	59
3	Um Jogo Educacional Para o Ensino de Teste de Software	61
3.1	Considerações Iniciais	61
3.2	Metodologia	62
3.3	Jogo Testing Game	65
3.3.1	<i>Versão Inicial - Cocos2D</i>	<i>65</i>
3.3.2	<i>Segunda Versão - Construct 2</i>	<i>67</i>
3.4	Descrição do Jogo Testing Game	67
3.4.1	<i>Planejamento Inicial</i>	<i>69</i>
3.4.1.1	<i>Contexto e Problema</i>	<i>69</i>
3.4.1.2	<i>Público-Alvo</i>	<i>69</i>

3.4.1.3	<i>Conteúdo</i>	70
3.4.1.4	<i>Avaliação</i>	70
3.4.1.5	<i>Objetivos de Design</i>	71
3.4.1.6	<i>Objetivos de Arte</i>	72
3.4.1.7	<i>Objetivos Técnicos</i>	73
3.4.1.7.1	<i>Game Engine</i>	73
3.4.1.7.2	<i>Construct 2</i>	76
3.4.2	<i>Produção</i>	77
3.4.2.1	<i>Iteração 1 - Teste Funcional</i>	77
3.4.2.1.1	<i>Análise e Planejamento - Teste Funcional</i>	77
3.4.2.1.2	<i>Projeto Iterativo - Teste Funcional</i>	77
3.4.2.1.3	<i>Implementação Incremental - Teste Funcional</i>	79
3.4.2.1.4	<i>Integração, Testes e Revisão - Teste Funcional</i>	80
3.4.2.2	<i>Iteração 2 - Teste Estrutural</i>	80
3.4.2.2.1	<i>Análise e Planejamento da Iteração - Teste Estrutural</i>	80
3.4.2.2.2	<i>Projeto Iterativo - Teste Estrutural</i>	81
3.4.2.2.3	<i>Implementação Incremental - Teste Estrutural</i>	84
3.4.2.2.4	<i>Integração, Testes e Revisão - Teste Estrutural</i>	84
3.4.2.3	<i>Iteração 3 - Teste de Mutação</i>	84
3.4.2.3.1	<i>Análise e Planejamento - Teste de Mutação</i>	84
3.4.2.3.2	<i>Projeto Iterativo - Teste de Mutação</i>	85
3.4.2.3.3	<i>Implementação Incremental - Teste de Mutação</i>	86
3.4.2.3.4	<i>Integração, Testes e Revisão da Iteração - Teste de Mutação</i>	87
3.4.3	<i>Pós-Produção</i>	87
3.4.3.1	<i>Execução</i>	87
3.4.3.2	<i>Ambiente</i>	87
3.4.3.3	<i>Avaliação da Aprendizagem</i>	87
3.4.4	<i>Avaliação, Verificação e Validação</i>	88
3.4.5	<i>Gestão de Projetos</i>	88
3.4.5.1	<i>Licenciamento</i>	88
3.4.5.2	<i>Publicação</i>	88
3.5	<i>Exemplo de Aplicação do Jogo Testing Game</i>	89
3.6	<i>Comparação dos Jogos Educacionais no Domínio de Teste de Software</i>	92
3.7	<i>Considerações Finais</i>	93
4	<i>Avaliação da Qualidade e Usabilidade do Jogo Testing Game</i>	95
4.1	<i>Considerações Iniciais</i>	95
4.2	<i>Planejamento do Estudo de Viabilidade</i>	96
4.2.1	<i>Questões de Pesquisa</i>	96

4.2.2	<i>Objetivo</i>	96
4.2.3	<i>Seleção dos Sujeitos</i>	96
4.2.4	<i>Instrumentação</i>	97
4.2.5	<i>Ameaças à Validade</i>	98
4.3	Execução do Estudo de Viabilidade	99
4.3.1	<i>Execução da Primeira Etapa - Avaliação da Qualidade do Jogo Testing Game</i>	100
4.3.2	<i>Execução da Segunda Etapa - Avaliação Heurística do Jogo Testing Game</i>	101
4.4	Análise dos Resultados	102
4.4.1	<i>Resultado da Avaliação da Qualidade do Jogo Testing Game</i>	103
4.4.2	<i>Resultado da Avaliação Heurística do Jogo Testing Game</i>	105
4.4.3	<i>Discussão dos Resultados</i>	107
4.5	Considerações Finais	108
5	Considerações Finais e Trabalhos Futuros	111
5.1	Visão Geral da Pesquisa	111
5.2	Contribuições	111
5.3	Trabalhos Futuros	112
5.4	Produção Científica	113
5.4.1	<i>Publicações Elaboradas</i>	113
5.4.2	<i>Publicações em Elaboração</i>	113
	Referências	115
	APÊNDICE A Pacote do Experimento de Avaliação da Usabilidade e Qualidade do Jogo Testing Game	125
A.1	Considerações Iniciais	125
A.2	Materiais utilizados nas duas etapas	125
A.2.1	<i>Termo de Consentimento</i>	125
A.2.1.0.1	Confidencialidade	126
A.2.1.0.2	Benefícios e Liberdade de Desistência	126
A.2.2	<i>Formulário de Feedback</i>	126
A.3	Materiais utilizados na primeira etapa do estudo de viabilidade (Avaliação da Qualidade)	128
A.3.1	<i>Formulário de Caracterização de Perfil</i>	128
A.3.2	<i>Modelo de Avaliação da Qualidade de Jogos Educacionais</i>	130
A.4	Materiais utilizados na segunda etapa do estudo de viabilidade (Avaliação da Usabilidade)	136
A.4.1	<i>Formulário de Caracterização de Perfil</i>	136
A.4.2	<i>Avaliação Heurística</i>	138

A.4.3	<i>Formulário de Defeitos:</i>	142
APÊNDICE B	Modelo de Avaliação da Qualidade de Jogos Educacionais e	
	Avaliação Heurística	143
B.1	Considerações Iniciais	143
B.2	Modelo de Avaliação de Jogos Educacionais na Engenharia de Software	143
B.3	Avaliação Heurística	145
B.4	Considerações Finais	147
ANEXO A	Artigos publicados	149

INTRODUÇÃO

1.1 Contextualização e Motivação

Nos últimos anos, os desenvolvedores de sistemas e produtos de software defrontam-se com a demanda de alta qualidade dos produtos e sistemas desenvolvidos e, conseqüentemente, as atividades de VV& T - Verificação, Validação e Teste têm sido aprimoradas, em particular as atividades de testes, com a utilização de técnicas, critérios e ferramentas para a execução de testes (DELAMARO; MALDONADO; JINO, 2016; SOMMERVILLE, 2011). O teste de software tem por objetivo executar programas ou modelos com entradas específicas, analisando se eles comportam de acordo com o esperado, realizando uma análise detalhada com o intuito de levá-los a falhar e, posteriormente, contribuir, com a depuração, para eliminar os defeitos que originaram as falhas (DELAMARO; MALDONADO; JINO, 2016). O teste envolve basicamente quatro fases: o planejamento de teste, o projeto de caso de teste, a execução de testes e a avaliação dos resultados dos testes. Essas etapas devem ser planejadas e desenvolvidas ao longo do processo de desenvolvimento de software (PRESSMAN, 2011; DELAMARO; MALDONADO; JINO, 2016), constituindo estratégias de teste.

A execução de todos os casos de teste, que em geral é impossível, pode demorar muito tempo para ser realizada. Assim, é necessário utilizar apenas subconjuntos de casos de teste para a execução de testes. Uma estratégia seria caracterizar subdomínios do domínio de entrada, com base em critérios de teste bem estabelecido, e a partir desses subdomínios escolher os casos de teste. De acordo com os critérios utilizados são obtidos subdomínios diferentes e, eventualmente, conjuntos de teste diferentes. Os critérios de testes, em nível de programa, podem ser classificados em três categorias: Funcional, Estrutural e Baseado em Defeitos (DELAMARO; MALDONADO; JINO, 2016).

Similarmente ao processo de desenvolvimento de software, estabeleceu-se um modelo de maturidade específico para essa atividade, o modelo TMMi (*Testing Maturity Model*). O

principal objetivo do TMMi é fornecer uma estrutura para avaliar a maturidade dos processos de teste de software. Esse modelo foi validado e está fundamentado em normas estabelecidas, padrões e outros *frameworks* (como, IEEE 829, ISTQB, TPI, TMap, CMMi) (VEENENDAAL; CANNEGIETER, 2012).

Apesar do teste ser reconhecido como uma atividade importante no desenvolvimento de software (DELAMARO; MALDONADO; JINO, 2016; SILVA; GOMES; MATOS, 2012), há uma grande carência por profissionais especializados nessa área (SOUZA D. M.; MALDONADO; BARBOSA, 2012), pois esses apresentam dificuldades na utilização de técnicas, critérios e ferramentas propostas para a execução dos testes em função da deficiência na formação ou por desmotivação decorrente do ambiente de trabalho e das estratégias de alocação e responsabilização desses profissionais nas equipes de desenvolvimento e teste (VALLE; BARBOSA; MALDONADO, 2015a).

Para verificar como os profissionais de teste de software são capacitados Valle, Barbosa e Maldonado (2015a) realizam uma pesquisa nos currículos de referência propostos pela Sociedade Brasileira de Computação (SBC) e pela *Association for Computing Machinery* (ACM) para cursos de graduação da área de Computação. O currículo de referência tem por objetivo fornecer apoio e diretrizes para a definição, implantação e avaliação de cursos de graduação na área de Computação. Os currículos de referência propostos pela SBC indicam que os conteúdos de teste de software sejam abordados em um dos tópicos da disciplina de Engenharia de Software, denominado Verificação, Validação e Teste de Software (SBC, 2005).

Do mesmo modo, os currículos de referência propostos pela ACM (ACM; IEEE, 2013; LEBLANC *et al.*, 2006; MAO, 2008; TOPI *et al.*, 2010) sugerem que os conteúdos de teste de software sejam abordados em tópicos da disciplina de Engenharia de Software e de Fundamentos de Programação, sendo atribuída poucas horas aula para o ensino desse conteúdo. Logo, observa-se que os cursos de graduação da área de computação, em geral, não proporcionam que os estudantes obtenham uma visão integrada dos conteúdos de teste de software com outras disciplinas dos curso de graduação, tais como: Programação Orientada a Objetos, Estrutura de Dados, Reúso, entre outras (VALLE; BARBOSA; MALDONADO, 2015a).

Para caracterizar essa situação Valle, Barbosa e Maldonado (2015a) analisaram os currículos das melhores universidades do Brasil e exterior. Nessa análise, os autores observaram que, em geral, o teste de software é abordado na disciplina de Engenharia de Software dos cursos de graduação em Computação e que algumas universidades brasileiras ofertam uma disciplina específica para o ensino de teste de software, porém os conteúdos de teste são abordados de forma isolada das demais disciplinas dos cursos. Semelhantemente ao cenário nacional, as universidades do exterior, em geral, abordam o teste de software nas disciplinas de Engenharia de Software e Fundamentos de Programação, conforme sugerido pelos currículos de referência propostos pela SBC e ACM (VALLE; BARBOSA; MALDONADO, 2015a).

Além disso, percebe-se que há uma escassez de profissionais qualificados em teste de

software. Isso pode estar relacionada com a desvalorização desses profissionais. Geralmente, seus salários são menores se comparados com os demais cargos de uma empresa de desenvolvimento de software, por exemplo, desenvolvedor. Outro fator que pode influenciar na escassez de profissionais qualificados é a falta de reconhecimento do trabalho de um profissional de teste de software. Além disso, há uma carência de ambientes que motivem os estudantes a realizarem atividades relacionadas com o teste de software, uma vez que há, reconhecidamente, uma dificuldade e ineficiência em ensinar esse conteúdo por meio de abordagens tradicionais que utilizam apenas aulas teóricas (SMITH *et al.*, 2012; SOUZA *et al.*, 2013).

Para amenizar os problemas citados anteriormente, algumas iniciativas têm sido propostas para capacitar e motivar os profissionais de teste de software, tais como: módulos educacionais, teste por pares, desenvolvimento dirigido por testes, ensino de teste com programação, jogos educacionais, entre outras (VALLE; BARBOSA; MALDONADO, 2015b). Dentre essas iniciativas, encontram-se os jogos educacionais no domínio de teste de software. Alguns jogos educacionais nesse domínio identificados por Valle, Barbosa e Maldonado (2015b) são: *iTest Learning* (BEZERRA *et al.*, 2014), Jogo das 7 Falhas (DINIZ; DAZZI, 2011b), TestEG (OLIVEIRA, 2013), JETS (SILVA; MULLER, 2012), U-TEST (THIRY; ZOUCAS; SILVA, 2011). Nos últimos anos, os jogos educacionais têm despertado grande interesse por parte dos pesquisadores de tecnologia educacional devido ao crescimento da indústria de jogos eletrônicos. Além disso, esses jogos são considerados importantes ferramentas que facilitam a aprendizagem de conceitos e ideias sobre o conteúdos educacionais (FREITAS; MAHARG, 2011). Os jogos educacionais motivam e proporcionam prazeres aos jogadores enquanto eles jogam, permitindo que os estudantes adquiram conhecimentos educacionais combinados com a diversão (NETO; FONSECA, 2013).

Nessa perspectiva, o ensino conjunto de teste de software com programação também tem sido utilizado para capacitar e motivar os estudantes (COLLOFELLO; VEHATHIRI, 2005; EDWARDS, 2003; BARBOSA *et al.*, 2008; CLARKE *et al.*, 2010; RAMAKRISHNAN, 1999; SOUZA D. M.; MALDONADO; BARBOSA, 2012). O ensino conjunto desses conteúdos pode ajudar no desenvolvimento das habilidades de análise e compreensão dos estudantes, pois para testar é necessário que os estudantes conheçam o comportamento de seus programas (EDWARDS, 2004; SOUZA D. M.; MALDONADO; BARBOSA, 2012). Para isso, é necessário que o teste de software seja ensinado nos primeiros anos dos cursos de graduação em Computação. Segundo Corte e Maldonado (2006) e Dvornik *et al.* (2011) o teste de software deve ser ensinado o mais cedo possível durante os cursos de graduação em Computação (BARBOSA *et al.*, 2008; PATTERSON; KÖLLING; ROSENBERG, 2003; CORTE; MALDONADO, 2006).

Ensinar conceitos de teste de software nos primeiros anos dos cursos de graduação em Computação pode melhorar o raciocínio dos estudantes com relação aos programas desenvolvidos e pode favorecer o uso das técnicas de teste no processo de desenvolvimento de software (SOUZA *et al.*, 2013). O trabalho proposto por Barriocanal *et al.* (2002) demonstra que ensinar teste de

software no início dos cursos de graduação em Computação pode melhorar a qualidade do código e facilitar o ensino de teste de software e de fundamentos de programação. Segundo Edwards e Shams (2014) e Souza, Isotani e Barbosa (2015) o ensino conjunto de teste de software com programação pode contribuir para o ensino de ambos conteúdos. Na programação, a atividade de teste pode aperfeiçoar a capacidade de compreensão e análise dos estudantes e no teste de software pode-se propor a criação de uma “cultura de teste de software” adequada entre os estudantes. Além disso, o ensino conjunto de teste de software com programação pode motivar uma reforma nos currículos da área de computação, com o intuito de introduzir aspectos de qualidade no processo de desenvolvimento de software antecipadamente.

1.2 Objetivos

O objetivo geral deste projeto de mestrado foi desenvolver um jogo educacional, denominado Testing Game, para auxiliar o ensino de teste de software com programação. O jogo Testing Game está disponível para sua utilização por meio da plataforma Web. Esse jogo aborda conteúdos de teste de software que contemplam as técnicas de teste funcional, estrutural e baseado em defeitos. A abordagem de ensino de teste com programação foi considerada neste trabalho. No jogo, há situações em que o jogador deve testar apenas programas com estrutura sequencial, em seguida programas com programas com estrutura condicional e por fim, programas com estrutura de repetição.

Desta forma, os jogadores podem observar que a atividade de teste de software pode ser considerada desde os seus primeiros programas. Além disso, estratégia de teste incremental foi considerada, na qual os critérios mais fracos são utilizados inicialmente e os critérios mais forte são utilizados posteriormente. Os resultados obtidos por meio de um estudo de viabilidade demonstraram que o jogo apresenta uma boa qualidade e usabilidade sob o ponto de vista dos estudantes. Desta forma, acredita-se que esse jogo educacional possa ser utilizado com um recurso complementar aos conteúdos de teste de software abordados em sala de aula ou ser inserido durante as aulas de teste de software com um instrumento facilitador da aprendizagem.

1.3 Metodologia

Para a condução deste projeto de mestrado foi realizada uma revisão da literatura para identificar os principais conceitos e pesquisas relacionados ao tema deste projeto de mestrado. Para isso, foram identificados os principais conceitos sobre teste de software, técnicas e critérios de teste. Além disso, foram identificados os principais conceitos, aplicações e classificação de jogos educacionais. Para caracterizar o estado da arte referente ao ensino de teste de software, realizou-se um mapeamento sistemático com objetivo de identificar as principais abordagens de apoio ao ensino de teste de software (VALLE; BARBOSA; MALDONADO, 2015b). Além disso, realizou-se uma análise nos currículos das melhores universidades do Brasil e do exterior

para conhecer como o teste de software é abordado nas universidades (VALLE; BARBOSA; MALDONADO, 2015a).

Após caracterizar o estado da arte referente ao ensino de teste de software, observou-se a necessidade de desenvolver novas tecnologias de apoio ao ensino de teste de software. Sendo assim, o objetivo deste projeto foi desenvolver um jogo educacional de apoio ao ensino de teste de software com programação. Para isso, realizou-se um mapeamento sistemático para identificar motores de jogos que pudessem auxiliar o desenvolvimento do jogo. A primeira versão do jogo utilizou o motor de jogos Cocos2D e a segunda versão do jogo utilizou o Construct 2. Para a descrição do Testing Game foi utilizada uma metodologia de desenvolvimento de recursos educacionais focada no desenvolvimento de jogos educacionais e artefatos gamificados (ROCHA *et al.*, 2016).

Para avaliar o jogo, foi utilizada uma metodologia experimental definida por Shull, Carver e Travassos (2001) que é composta por quatro etapas: i) estudo de viabilidade que verifica a viabilidade prática da tecnologia avaliada; ii) estudo de observação que identifica as principais dificuldades e limitações da tecnologia avaliada por meio de observações de pesquisadores; iii) estudo de caso (ciclo de vida) que caracterizara aplicação da tecnologia avaliada no contexto de ciclo de vida de forma isolada; e iv) estudo de caso (indústria) que analisa a aplicação da tecnologia avaliada na indústria. Neste projeto de mestrado foi utilizado apenas a primeira etapa, a saber: estudo de viabilidade.

O estudo de viabilidade foi dividido em duas etapas. A primeira etapa teve por objetivo avaliar a qualidade do jogo em relação à motivação, experiência do usuário e aprendizagem, sob o ponto de vista dos estudantes, para isso foi utilizado o modelo de avaliação de jogos educacionais na Engenharia de Software definido por Savi, Wangenheim e Borgatto (2011). A segunda etapa teve por objetivo avaliar a usabilidade do jogo por meio do conjunto de heurísticas proposto por Nielsen (1994). A metodologia utilizada para condução deste projeto de mestrado é melhor detalhada na Seção 3.2.

1.4 Organização

Além deste capítulo introdutório que contém a Contextualização, Motivação e Objetivos para este projeto de mestrado, o presente documento está organizado da seguinte forma: no Capítulo 2 são apresentados os principais conceitos relacionadas com o tema deste projeto de mestrado. Nesse capítulo são encontrados os conceitos básicos sobre ensino de teste de software e principais técnicas e critérios de teste que apoiam a realização da atividade de teste de software. Além disso, é dada uma visão geral sobre o ensino de teste de software, demonstrando as principais abordagens de apoio a esse conteúdo. Por fim, apresentam-se os conceitos sobre jogos educacionais e alguns jogos no domínio de teste de software. No Capítulo 3 é apresentada uma visão geral do jogo desenvolvido neste projeto de mestrado, denominado Testing Game,

apresentando suas principais funcionalidades e conteúdos de teste abordados. No Capítulo 4 é apresentado o estudo de viabilidade realizado para avaliar a qualidade do jogo, com relação à motivação, experiência do usuário e aprendizagem sob o ponto de vista dos estudantes e sua usabilidade. No Capítulo 5 é apresentada uma visão geral do projeto, contribuições, publicações e discutem-se possíveis trabalhos futuros. No Apêndice A é apresentado o pacote do experimento utilizado no estudo de viabilidade (Formulário de Consentimento, Formulário de Caracterização de Perfil, Modelo de Avaliação da Qualidade de Jogos Educacionais na Engenharia de Software, Conjunto de Heurísticas para Avaliar a Usabilidade e Questionário de *Feedback*). No Apêndice B é apresentada uma descrição sucinta do Modelo de Avaliação de Qualidade de Jogos Educacionais na Engenharia de Software e do Conjunto de Heurísticas para Avaliar a Usabilidade. Por fim, no Anexo A são apresentados os artigos publicados durante o mestrado.

FUNDAMENTAÇÃO TEÓRICA

2.1 Considerações Iniciais

Neste capítulo são apresentados os principais conceitos e pesquisas relacionados ao tema deste trabalho, identificados por meio de uma revisão da literatura.

Na Seção 2.2 são apresentados os principais conceitos e terminologias relacionados com o teste de software. Além disso, são discutidas as principais técnicas e critérios de teste de software. Na Seção 2.3 são apresentados os principais conceitos e abordagens de ensino de teste de software, e uma análise de currículos dos cursos de graduação em Ciência da Computação das universidades do Brasil e do exterior. Por fim, na Seção 2.4 são discutidos conceitos e aplicações sobre jogos educacionais, os principais elementos considerados essenciais à qualidade de jogos educacionais e alguns processos de desenvolvimento de jogos educacionais.

2.2 Teste de Software

Nos últimos anos, a indústria de software tem aprimorado o desenvolvimento de sistemas e produtos de software de alta qualidade. Para atender a essa exigência as atividades de VV& T - Verificação, Validação e Teste têm sido também aprimoradas, em especial as atividades de teste de software, com a utilização de técnicas, critérios e ferramentas para a execução de testes (SOMMERVILLE, 2011). O teste de software tem por objetivo executar programas ou modelos com entradas específicas, verificando se eles comportam-se de acordo com o esperado, com o intuito de levá-los a falhar e, posteriormente, contribuir por meio da depuração para eliminar os defeitos que originaram as falhas (DELAMARO; MALDONADO; JINO, 2016). O teste de software é amplamente reconhecido como uma atividade essencial em qualquer processo de desenvolvimento de software bem sucedido (FRASER; ARCURI, 2013).

2.2.1 Conceitos Básicos

Para garantir a qualidade de produtos de software algumas técnicas sistematizadas de gerenciamento da qualidade no desenvolvimento de software são adotadas (DELAMARO; MALDONADO; JINO, 2016; SOMMERVILLE, 2011). As atividades de verificação, validação e teste de software são importantes fatores que contribuem para a entrega de produtos de software de boa qualidade aos usuários finais (WAZLAWICK, 2013).

A atividade de **Validação** tem como principal objetivo assegurar que o software em desenvolvimento é o software correto de acordo com os requisitos dos usuários, em contrapartida à atividade de **Verificação** que visa avaliar se o produto está sendo desenvolvido corretamente (PRESSMAN, 2011). Sob outra perspectiva, o **Teste de Software** tem por objetivo executar programas ou modelos com entradas específicas avaliando se esses produtos se comportam conforme o esperado, realizando uma análise detalhada do software, com o intuito de identificar falhas durante suas análises e, posteriormente, por meio da depuração, eliminar os defeitos que as originam. O teste de software pode ser considerado uma atividade dinâmica, pois se baseia na execução de programas ou modelos (DELAMARO; MALDONADO; JINO, 2016).

O padrão IEEE (RADATZ; GERACI; KATKI, 1990) apresenta definições dos principais termos utilizados no contexto de teste de software, sendo eles:

- **Defeito** (*Fault*): É um passo, processo ou definição de dados incorretos, por exemplo, um comando incorreto;
- **Engano** (*Mistake*): É a ação humana que causa um resultado incorreto, por exemplo, uma decisão incorreta tomada pelo programador;
- **Erro** (*Error*): É o resultado incorreto na execução de um programa;
- **Falha** (*Failure*): É a produção de uma saída incorreta conforme a especificação.

Outros termos que são muito utilizados pelo engenheiros de software em ambiente de trabalho são (DELAMARO; MALDONADO; JINO, 2016):

- **Domínio de entrada:** É o conjunto de todos os possíveis valores que podem ser utilizados para executar um programa **P**, denotado por $D(P)$;
- **Dado de Teste:** É um elemento do domínio de entrada de um programa **P** para a execução de um determinado teste;
- **Caso de Teste:** É um par formado por uma entrada, um dado de teste, e por uma saída esperada para a execução de um determinado teste;
- **Conjunto de Teste:** É o conjunto de todos os casos de teste utilizados na execução da atividade de teste;

- **Oráculo:** É o instrumento que decide se a saída obtida de uma determinada execução de teste coincide com a saída esperada. Esse oráculo pode ser uma ferramenta automatizada ou o próprio testador.

Na Figura 1 é ilustrado um cenário típico da atividade de teste de software.

Figura 1 – Cenário típico da atividade de teste. Adaptado de (DELAMARO; MALDONADO; JINO, 2016)

Para a execução da atividade de teste de software, primeiramente, é necessário definir um conjunto de casos de teste **T** e em seguida executar um programa em teste **P** com **T**. Posteriormente, verifica-se se algum erro foi revelado, se a saída obtida coincide com a saída esperada então nenhum erro foi identificado, caso contrário, algum erro foi revelado (DELAMARO; MALDONADO; JINO, 2016). Na Figura 1, o oráculo é o testador que baseado na especificação do programa (**S(P)**) decide se algum erro foi encontrado. No entanto, a atividade de teste é complexa e durante o processo de desenvolvimento de software, diferentes estratégias podem ser aplicadas, independentemente do processo de desenvolvimento de software utilizado. A seguir, apresenta-se uma breve descrição das principais estratégias de testes, a saber: teste de unidade, teste de integração, teste de sistema e teste de aceitação.

O **teste de unidade** tem por objetivo testar as subpartes de um sistema chamados de unidade, ou seja, as menores partes constituintes de um sistema (FEBBRARO *et al.*, 2013). Os testes de unidade testam as funcionalidade de forma independente e isolada. Isso facilita identificar qual unidade (por exemplo, métodos, funções e classes) não está funcionando corretamente. Os defeitos típicos encontrados nesse tipo de teste são os erros de lógica e de implementação. Por exemplo, algoritmos incorretos ou mal implementados e estruturas de dados incorretas que são identificadas em cada módulo do software (MESZAROS, 2007; SOUZA; FALBO; VIJAYKUMAR, 2015).

O **teste de integração** tem por objetivo testar as estruturas de comunicação entre as unidades que compõem o sistema, ou seja, identificar os possíveis problemas gerados quando as unidades, já testadas, são integradas. Para isso, é necessário elaborar um conjunto de teste de integração para exercitar as interações e as interfaces entre os módulos, ao contrário do teste de unidade que exercita as funcionalidades das unidades de software. Os testes de integração são também conhecidos como testes de componentes, pois tem por objetivo testar a interação entre

os componentes que juntos fornecem alguma funcionalidade (PRESSMAN, 2011; MESZAROS, 2007; SOUZA; FALBO; VIJAYKUMAR, 2015).

O **teste de sistema** tem por objetivo verificar se as funcionalidades foram implementadas corretamente. Para isso, deve-se avaliar os aspectos de correção, completude e coerência e os requisitos não funcionais (como segurança, performance e robustez) (DELAMARO; MALDONADO; JINO, 2016). O teste de sistema envolve a integração de componentes para a criação de uma versão do sistema. Esse tipo de teste verifica a compatibilidade dos componentes, analisando se eles interagem corretamente e transferem os dados corretamente por meio de suas interfaces (SOMMERVILLE, 2011; SOUZA; FALBO; VIJAYKUMAR, 2015).

O **teste de aceitação** tem por objetivo verificar o correto funcionamento do sistema em teste. Normalmente, costuma-se utilizar a interface final do sistema. Esse teste pode ser executado da mesma forma que o teste de sistema, porém a principal diferença é que o teste de aceitação é realizado pelo usuários final ou cliente e não pela a equipe desenvolvimento (WAZLAWICK, 2013; SOUZA; FALBO; VIJAYKUMAR, 2015).

2.2.2 Técnicas e Critérios de Teste

A atividade de teste de software possui diferentes critérios que definem quais são partes constituintes do produto que precisam ser testadas para eventualmente revelar a presença de defeitos no software. Cada critério divide o domínio de entrada em diferentes subdomínios e, conseqüentemente, diferentes, em geral infinitos, conjuntos de casos de teste podem ser derivados para satisfazer um determinado critério. Como, em geral, é impossível garantir a inexistência de defeitos no software, o teste é utilizado na prática para dar uma segurança da qualidade do produto desenvolvido (DELAMARO; MALDONADO; JINO, 2016; WAZLAWICK, 2013). Os critérios de teste de software, em nível de programa, podem ser classificados em três diferentes técnicas: **Funcional**, **Estrutural** e **Baseada em Defeitos**. O que distingue essas técnicas é a fonte utilizada para definir os requisitos de teste (DELAMARO; MALDONADO; JINO, 2016).

2.2.2.1 Técnica Funcional

O Teste Funcional, ou teste de caixa-preta, é baseado apenas na especificação do produto em teste. Os critérios desta técnica podem ser aplicados em qualquer fase do teste de software (unidade, integração, sistema e aceitação), independentemente do paradigma utilizado, pois não se analisam detalhes de implementação. Nessa técnica, o software é avaliado segundo o ponto de vista do usuário, uma vez que são fornecidas as entradas e posteriormente são avaliadas as saídas com o intuito de verificar se elas estão de acordo com os requisitos especificados (KHAN; SADIQ, 2011; DELAMARO; MALDONADO; JINO, 2016; JUNIOR *et al.*, 2012). Na Figura 2 é apresentado o funcionamento do Teste Funcional.

Para utilizar essa técnica, é necessário executar o programa em teste com os dados de

Figura 2 – Funcionalmente do Teste Funcional

teste selecionados do domínio de entrada. Nessa técnica, os detalhes de implementação não são considerados e consequentemente não se conhece a estrutura interna do programa em teste, logo essa técnica de teste de software se assemelha a uma caixa-preta. Por fim, é verificado se a saída obtida coincide com a saída esperada, se isso ocorrer, nenhum defeito foi identificado, caso contrário, algum defeito foi revelado (JUNIOR *et al.*, 2012).

O Teste Funcional identifica apenas erros relacionados com o mal funcionamento de software. Esses erros são revelados a partir das saídas dos produtos de software testados. Essa técnica é utilizada para a identificação de funções incorretas que levam a saídas indesejada quando são executadas (KHAN; SADIQ, 2011). No entanto, esses critérios apresentam algumas limitações, uma vez que não garantem que as partes críticas ou essenciais do produto em teste sejam exercitadas. É importante ressaltar que a eficácia desta técnica depende da qualidade da especificação do produto (JUNIOR *et al.*, 2012).

Os testes exaustivos são, em geral, praticamente impossíveis de serem aplicados durante a atividade de teste de software, pois demandam muitos recursos, por exemplo, o tempo (WAZLAWICK, 2013). Desta forma, é necessário utilizar critérios para facilitar a aplicação da atividade de teste de software, pois eles apresentam diretrizes para auxiliar a criação de casos de teste com maior qualidade. Os critérios de Teste Funcional mais conhecidos são:

- **Particionamento em Classes de Equivalência:** Este critério divide o domínio de entrada em classes de equivalência, no qual os elementos das classes apresentam um comportamento similar, possuindo o mesmo impacto durante a execução do teste de software. Alguns autores também consideram o domínio de saída, identificando possíveis alternativas que poderiam determinar classes de equivalência. Esses casos de teste devem ser projetados para cobrirem pelo menos uma vez cada partição, possibilitando revelar erros com uma menor quantidade de casos de teste. As partições de equivalência são definidas a partir da especificação do software a ser testado ou por meio de experiências para a identificação de prováveis classes de equivalência para a detecção de erros. A principal vantagem desse critério é a redução no tamanho do domínio de entrada (WU, 2012). Para aplicação desse critério é necessário seguir apenas dois passos, sendo eles: i) identificar as classes de equivalência, observando as entradas e as saídas para realizar o particionamento do domínio de entrada; e ii) gerar pelo menos um caso de teste para cada classe de equiva-

lência. Assim, será possível obter um menor número de casos de teste (SOMMERVILLE, 2011; DELAMARO; MALDONADO; JINO, 2016; JUNIOR *et al.*, 2012);

- **Análise do Valor Limite:** Este critério é utilizado como complemento do critério de particionamento de em classes equivalência. A diferença deste critério é que o mesmo explora os valores limites das classes de equivalência em vez de qualquer elemento de uma partição (DELAMARO; MALDONADO; JINO, 2016). Esse critério é utilizado porque uma grande parte dos erros geralmente ocorrem nas fronteiras do domínio de entrada. Casos de testes que exploram condições limites têm maior probabilidade de encontrar defeitos (JUNIOR *et al.*, 2012; MYERS; SANDLER; BADGETT, 2011);
- **Teste Funcional Sistemático:** Este critério combina os critérios particionamento em classes de equivalência e análise do valor limite. Quando os domínios de entrada e saída já estão particionados, esse critério requer que pelo menos dois casos de teste de cada partição sejam exercitados. Assim, será possível minimizar os defeitos coincidentes. Este critério sugere algumas diretrizes para auxiliar na identificação dos casos de teste, tais como: valores numéricos, intervalos variáveis, texto, entre outras (DELAMARO; MALDONADO; JINO, 2016).
- **Grafo Causa-Efeito:** Os critérios particionamento em classes de equivalência e análise do valor limite não exploram combinações dos dados de entrada. Já o critério de causa-efeito ajuda na definição de casos de testes que exploram ambiguidade e incompletude nas especificações, que provavelmente não seriam consideradas pelos outros critérios. Porém, a eficiência desse critério, assim como os demais critérios do teste funcional, depende da experiência do testador e da qualidade da especificação, para que assim seja possível identificar as causas e os efeitos (DELAMARO; MALDONADO; JINO, 2016).

Uma vantagem do Teste Funcional é que ele requer apenas a especificação do software para derivar os requisitos de teste. Desta forma, essa técnica pode ser utilizada em qualquer paradigma, tais como: procedimental, orientado a objetos, entre outros. No entanto, uma desvantagem do Teste Funcional é que ele não pode assegurar que partes críticas do código foram testadas, pois essa técnica não analisa o código fonte (DELAMARO; MALDONADO; JINO, 2016; MYERS; SANDLER; BADGETT, 2011).

2.2.2.2 Teste Estrutural

O Teste Estrutural, ou teste de caixa branca, é baseado na implementação do software a ser testado, isso requer a execução de partes ou componentes do software. Os critérios pertencentes a esta técnica são classificada em três categorias que são: **critérios baseados em complexidade**, **critérios baseados em fluxo de controle** e **critérios baseados em fluxos de dados**. Geralmente, no Teste Estrutural utiliza uma representação do programa a ser testado, denominado “Grafo de

Fluxo de Controle” (GFC) ou “Gráfo de Programa” (MYERS; SANDLER; BADGETT, 2011). Na Figura 3 é apresentado o código do programa BubbleSort em java. O BubbleSort é um algoritmo muito conhecido na literatura para a ordenação de vetores. Esse código foi utilizado para exemplificar o desenvolvimento de um GFC conforme apresentado na Figura 4.

```
/******  
BubbleSort.java  
ESPECIFICACAO: O programa percorre todo o vetor comparando  
elementos adjacentes (dois a dois). Neste exemplo só serão con-  
siderados vetores com valores inteiros e com o tamanho igual a  
4. Desta forma, o programa fornece como saída o vetor ordenado.  
*****/  
  
public void bubbleSort (int a[]) {  
    int i, j, aux;  
    size = 4;  
    for (i = 0; i < 4; i++) {  
        for (j = size - 1; j > i; j--) {  
            if (a[j-1] > a[j] {  
                aux = a[j-1];  
                a[j-1] = a[j];  
                a[j] = aux;  
            }  
        }  
    }  
}
```

Figura 3 – Código do Programa BubbleSort em Java

No GFC cada nó representa uma execução indivisível do código que termina em uma instrução simples ou condicional (DELAMARO; MALDONADO; JINO, 2016). Os critérios baseados na complexidade utilizam informações sobre a complexidade do programa testado. Um dos critérios mais conhecidos dessa classe é o critério de McCabe (ou teste de caminho básico), que utiliza a métrica de complexidade ciclomática, que fornece uma medida quantitativa da dificuldade para conduzir esse tipo de teste (PRESSMAN, 2011).

Os critérios baseados em fluxos de controle utilizam características de controle da execução do programa, como comandos ou desvios (DELAMARO; MALDONADO; JINO, 2016; MYERS; SANDLER; BADGETT, 2011). Os principais critérios dessa categoria são:

- **Todos-Nós:** exige que cada comando do programa testado seja executado pelo menos uma vez, ou seja, exige que a execução do programa passe, ao menos uma vez, em cada no vértice do GFC;

Figura 4 – Grafo de Fluxo de Controle do programa Bubble Sort

- **Todas-Arestas:** exige que cada aresta (desvio de fluxo de controle) do programa seja exercitada pelo menos uma vez;
- **Todos-Caminhos:** exige que todos os caminhos possíveis do programa sejam executados.

Já os critérios baseados em fluxos de dados concentram-se nas definições de variáveis e seus usos para derivar requisitos de teste. Esses critérios são mais adequados para certas classes de defeitos, como os defeitos computacionais. A família mais conhecida dos critérios baseados em fluxo de dados são os critérios de Rapps e Weyuker. Os critérios de Rapps e Weyuker utilizam um GFC estendido denominado “Grafo Def-Uso” (*def-use graph*), o qual adiciona informações sobre o fluxo de dados, caracterizando associações entre pontos do programa relativos a definição de variável e suas utilizações (DELAMARO; MALDONADO; JINO, 2016; MYERS; SANDLER; BADGETT, 2011). Na Figura 5 é ilustrado um exemplo do Grafo Def-Uso do programa Bubble Sort.

Os critérios baseados em fluxo de dados caracterizam dois tipos de usos de variáveis, sendo eles:

- **Uso Predicativo** (p-uso): Uso da variável em condições. Por exemplo: $\text{if } (c \leq b)$;
- **Uso Computacional** (c-uso): Uso da variável em computações. Por exemplo: $c = b * 3$;

Os critérios de fluxo de dados procura caracterizar associações entre definições e usos de variáveis. A definição de uma variável ocorre quando ela recebe um valor, por exemplo: $b = 7$ e $c = 9$. A seguir, apresentam-se os principais critérios baseados no fluxo de dados.

Figura 5 – Grafo de Def-Uso do programa Bubble Sort

- **Todas-Definições:** exige que cada definição de uma variável seja executada pelo menos uma vez, não importa se é por c-uso ou p-uso;
- **Todos-Usos:** exige que toda associação entre uma definição de variável e todos seus usos (c-uso e p-uso) sejam exercitadas, por pelo menos um caminho livre de definição;
- **Todos-DU-Caminhos:** exige que todas as associações entre uma definição de variável e seus subsequentes usos (c-uso e p-uso) sejam exercitadas por todos os caminhos livre de definição e de laços.

Um caminho livre de definição é um caminho que não contém redefinição de uma variável ao longo do caminho. Por exemplo, o caminho (3, 4, 8, 2, 9) do Grafo-Def-Uso apresentado na Figura 5 é um caminho livre de definição, pois não há redefinição da variável *i*.

A técnica de Teste Estrutural é baseada no conhecimento interno do programa, ou seja, analisa-se o código do programa testado. Desta forma, é possível testar um conjunto básico de funcionalidades de um programa e garantir uma melhor qualidade do produto de software do que quando apenas testado apenas com a técnica funcional (DELAMARO; MALDONADO; JINO, 2016; MYERS; SANDLER; BADGETT, 2011).

2.2.2.3 Teste Baseado em Defeitos

O Teste de Mutação ou Análise de Mutantes é o critério de teste mais conhecido da técnica de Teste Baseado em Defeitos. Este critério adiciona possíveis defeitos ao programa

uniformemente. Para isso, utilizam-se os defeitos que são frequentemente cometidos pelos desenvolvedores. No teste de mutação, o programa testado é alterado várias vezes, criando um conjunto de programas alternativos (mutantes) (DELAMARO; MALDONADO; JINO, 2016; LOCHAB; SINGHAL; BANSAL, 2014).

O principal objetivo deste critério é mostrar que o programa em teste não possui determinados tipos de defeitos (JIA; HARMAN, 2011). Os operadores de mutação indicam quais alterações sintáticas devem ser realizadas para a criação dos mutantes. Porém, não existe uma forma direta de definir os operadores de mutação para uma determinada linguagem. Geralmente, esses operadores são criados a partir de experiências no uso da linguagem e enganos típicos. Um exemplo são os operadores de mutação para a linguagem C que possui um conjunto de 75 (setenta e cinco) operadores de mutação (DELAMARO; MALDONADO; JINO, 2016). Alguns deles são:

- **SSDL**: eliminação de comandos;
- **ORRN**: troca de operador relacional;
- **STRI**: armadilha em condição de comando if;
- **Vsrrr**: troca de variáveis escalares.

O teste de mutação é fundamentado em duas premissas “hipótese programador competente” e o “efeito do acoplamento”. Na primeira, “hipótese programador competente”, assume-se que o programa está correto ou próximo ao correto e qualquer alteração sintática não causará erro sintático, mas mudará a semântica do programa. Na segunda, “efeito do acoplamento”, assume-se que dados de teste que diferenciam erros simples também distinguirão os complexos (DELAMARO; MALDONADO; JINO, 2016).

A partir das três técnicas de teste de software apresentadas anteriormente, a saber: Teste Funcional, Teste Estrutural e Teste Baseado em Defeitos, é possível obter de forma sistemática um conjunto de casos de teste eficazes com relação à atividade de teste de software, e possivelmente revelar falhas nos programas testados. É importante ressaltar que essas técnicas não devem ser utilizadas separadamente, já que as mesmas são complementares na execução da atividade de teste de software. Sendo assim, essas técnicas devem ser utilizadas em uma estratégia incremental de teste (DELAMARO; MALDONADO; JINO, 2016).

2.3 Ensino de Teste de Software

Apesar do teste de software ser reconhecido como uma atividade importante na garantia de qualidade de produtos de software, muitos estudantes sentem-se desmotivados a aprender conteúdos relacionados com o teste de software (JIA; YANG, 2013). Essa desmotivação pode

estar relacionada com: i) a incoerência entre os conteúdos práticos e teóricos, causando um menor interesse entre os estudantes; ii) os conteúdos ensinados em sala de aula não serem os mesmos exigidos na indústria; iii) os estudantes possuírem dificuldades em entender o processo e as etapas de testes; e iv) a falta de experiência dos estudantes no desenvolvimento de software.

Como consequência dessa desmotivação, a indústria de teste de software defronta-se com a carência de mão-de-obra especializada em teste de software (SOUZA D. M.; MALDONADO; BARBOSA, 2012; VALLE; BARBOSA; MALDONADO, 2015b). Segundo Smith *et al.* (2012), isso é ocasionado pela dificuldade e ineficiência em ensinar teste de software por meio de abordagens tradicionais que utilizam apenas aulas teóricas, pois há uma carência de ambientes que motivem os estudantes a aprenderem atividades relacionadas com o teste de software. Além disso, outro fator que pode contribuir para a carência de profissionais qualificados é a desvalorização dos profissionais de teste de software, geralmente seus salários são menores se comparados com os demais cargos de empresas de desenvolvimento de software (VALLE; BARBOSA; MALDONADO, 2015a). Além disso, o profissional de teste de software não possui reconhecimento por seu trabalho, já que o teste de software é considerado por muitos desenvolvedores como uma atividade destrutiva (EDWARDS; SHAMS, 2014; SOUZA; ISOTANI; BARBOSA, 2015).

No entanto, esse cenário tem se modificado e algumas empresas têm alterado suas estratégias de alocação e responsabilização desses profissionais nas equipes de desenvolvimento e teste, uma vez que perceberam que essas atividades estão inteiramente relacionadas. Mesmo não havendo evidências científicas, pesquisadores afirmam que testadores com bons conhecimentos em programação, consequentemente, realizam testes com maior eficiência (SOUZA D. M.; MALDONADO; BARBOSA, 2012; VALLE; BARBOSA; MALDONADO, 2015b). Desta forma, muitas empresas começaram a entender que o teste de software é uma atividade importante para o desenvolvimento de produtos de software de alta qualidade.

Diante desse cenário, percebe-se que é imprescindível a elaboração de instrumentos que facilitem o ensino e o treinamento de teste de software, formando profissionais capacitados e motivados em trabalhar com o teste de software (WONG *et al.*, 2011). Para amenizar esses problemas, ferramentas de teste de software têm sido utilizadas para facilitar o processo de execução de teste de software, permitindo detectar o maior número de defeitos possíveis, para que assim, seja possível garantir a qualidade dos produtos de software desenvolvidos. Atualmente, existem diversas ferramentas de teste de software, tais como: JaBUTi (VINCENZI *et al.*, 2005) muJava (MA; OFFUTT; KWON, 2005), SPACES (BARBOSA *et al.*, 2004), entre outras. Essas ferramentas podem ser utilizadas como um instrumento de apoio ao ensino de teste de software. Porém, utilizar apenas essas ferramentas não é algo suficiente para motivar e facilitar o ensino de teste de software. Desta forma, pesquisas foram realizadas com o intuito investigar a utilização de outras abordagens para auxiliar o ensino de teste de software, conforme discutido na próxima subseção.

2.3.1 Mapeamento Sistemático sobre Abordagens para Auxiliar o Ensino de Teste de Software

Diante do cenário apresentado anteriormente, percebe-se que é imprescindível a elaboração de instrumentos que facilitem o ensino e o treinamento de teste de software, formando profissionais capacitados e motivados em trabalhar com o teste de software (WONG *et al.*, 2011). Para amenizar esses problemas, ferramentas de teste têm sido utilizadas para facilitar a realização da atividade de teste de software. Porém, utilizar apenas essas ferramentas, não é suficiente para motivar e facilitar o ensino de teste de software. É necessário a utilização de outras abordagens para auxiliar o ensino desse conteúdo. Nesse contexto, entende-se como abordagem as metodologias e os instrumentos utilizados para auxiliar o ensino de algum conteúdo, nesse caso o teste de software (VALLE; BARBOSA; MALDONADO, 2015b).

Para identificar quais são as principais abordagens utilizadas no ensino de teste de software, Valle, Barbosa e Maldonado (2015b) realizaram um mapeamento sistemático¹ para caracterizar o estado da arte referente ao ensino de teste de software. Neste trabalho foram identificadas as principais abordagens utilizadas para auxiliar o ensino de teste de software, as fases de teste de software contempladas no ensino dessa atividade, as tecnologias utilizadas para criação das abordagens, bem como as linguagens alvo e, por fim, verificar como foram realizadas as avaliações das abordagens propostas para auxiliar o ensino de teste de software.

Na Figura 6 são apresentadas as principais abordagens para auxiliar o ensino de teste de software e as porcentagens dos trabalhos que utilizaram cada uma dessas abordagens.

- **Módulos Educacionais:** São unidades concisas de estudo, compostas por conteúdos teóricos combinados com atividades práticas e avaliações, apoiadas por recursos tecnológicos e computacionais (BARBOSA; MALDONADO, 2011);
- **Jogos Educacionais:** São jogos que proporcionam práticas educacionais atrativas e inovadoras, nos quais os usuários podem aprender de forma mais ativa, dinâmica e motivadora (FARIAS *et al.*, 2012);
- **Ensino Conjunto de Teste com Programação:** É o ensino conjunto de conceitos básicos de programação e de teste de software. Muitas pesquisas demonstram que o ensino conjunto dessas disciplinas possui benefícios (SOUZA D. M.; MALDONADO; BARBOSA, 2012);
- **Quiz:** É uma forma interativa de avaliar o conhecimento de usuários, por meio de questionários (MUSTAKEROV; BORISSOVA, 2005);
- **Revisão por Pares:** Utilizar essa técnica para auxiliar o ensino de teste de software, permite um aprendizado lúdico e competitivo, no qual os estudantes aprendem uns com os outros (SMITH *et al.*, 2012);

¹ Mais detalhes no Anexo A

Figura 6 – Abordagens para Auxiliar o Ensino de Teste de Software (VALLE; BARBOSA; MALDONADO, 2015b)

- **Desenvolvimento Dirigido por Testes (TDD):** Nessa técnica o desenvolvedor escreve um caso de teste e em seguida produz um código que possa ser validado pelo teste (EDWARDS, 2003);
- **Tutorial:** É uma ferramenta de ensino e aprendizagem que pode ser um programa de computador ou um texto que contém ou não imagens que auxiliam no processo de aprendizagem, demonstrando o passo a passo para a realização de alguma atividade (LIU; KUO; CHEN, 2010);
- **Rede Social:** Essa abordagem permite que usuários comuniquem-se uns com outros trocando experiências no decorrer do processo de aprendizagem (CLARKE *et al.*, 2011);
- **Modelo de Residência de Software:** Essa abordagem inclui o ensino tradicional de conceitos relevantes de um determinado conteúdo e em seguida a realização de atividades

práticas com profundidade e/ou com especialização em algum assunto específico (SAM-PAIO *et al.*, 2005);

- **Aprendizagem Baseada em Problemas:** Essa abordagem permite que os estudantes trabalhem em equipe com o intuito de resolverem problemas, incentivando o desenvolvimento de habilidades como atitudes, auto iniciativa e cooperação (FIGUERÊDO *et al.*, 2011);
- **Aprendizagem Baseada em Desempenho:** Essa abordagem utiliza a medição do desempenho dos usuários para esclarecer os principais objetivos e demonstrar as necessidades da aprendizagem individual do usuário (WANG *et al.*, 2011);

Como pode ser observado na Figura 6, foram identificadas 11 abordagens diferentes para auxiliar o ensino de teste de software. As duas abordagens mais utilizadas são os **Jogos Educacionais** e o **Ensino de Teste com Programação**. Juntas essas abordagens foram utilizadas em mais de 50% dos trabalhos analisados. Os **Módulos Educacionais** e o **Desenvolvimento Dirigido por Testes** foram utilizados em aproximadamente 12% dos trabalhos analisados, já a abordagem **Tutorial** foi utilizada em 8% dos trabalhos. Por fim, algumas abordagens como **Revisão por Pares**, **Quiz**, **Aprendizagem Baseada em Problemas**, **Aprendizagem Baseada em Desempenho**, **Residência de Software** e **Rede Social** foram utilizadas aproximadamente em 4% dos trabalhos analisados. É importante ressaltar que alguns trabalhos utilizaram mais de uma abordagem para auxiliar o ensino de teste de software. Por exemplo, o trabalho proposto por Barbosa *et al.* (2008) que utilizou Módulos Educacionais e o Ensino Conjunto de Teste de Software com Programação.

Outra análise que pode ser feita observando a Figura 6 é com relação às técnicas do teste de software que foram contempladas pelas abordagens identificadas. Grande parte dos trabalhos contemplaram o Teste Funcional e Teste Estrutural. No entanto, apenas dois trabalhos abordaram a Técnica Baseado em Defeitos. Além disso, grande parte dos trabalhos analisados foram avaliados por meio de uma avaliação empírica para comprovar a efetividades das abordagens consideradas (VALLE; BARBOSA; MALDONADO, 2015b).

2.3.2 Currículos das Principais Universidades do Brasil e do Exterior

Para conhecer como são capacitados os profissionais de teste de software Valle, Barbosa e Maldonado (2015a) analisaram os currículos de referência propostos pela Sociedade Brasileira de Computação (SBC) e pela *Association for Computing Machinery* (ACM) para os cursos de graduação da área de Computação. O objetivo do currículo de referência é fornecer apoio e diretrizes para a definição, implantação e avaliação de cursos de graduação na área de computação. Esses currículos preconizam que os conteúdos relacionados com o teste sejam abordados em um dos tópicos da disciplina de Engenharia de Software, denominado Verificação, Validação e Teste de Software, no qual é atribuído poucas aulas para o ensino desse conteúdo (SBC, 2005; CASSEL *et al.*, 2008; LEBLANC *et al.*, 2006; MAO, 2008; TOPI *et al.*, 2010).

Para caracterizar essa situação, Valle, Barbosa e Maldonado (2015a) analisaram os currículos das melhores universidades do Brasil e do exterior, mais detalhes podem ser observados no Anexo A. Para a seleção das universidades brasileiras que tiveram seus currículos analisados os autores utilizaram o *ranking* disponibilizado pelo RUF (Ranking Universitário Folha)² do ano de 2014. O RUF é uma avaliação anual sobre o ensino superior do Brasil realizada pela Folha. Nessa avaliação estão classificadas as 192 melhores universidades brasileiras, incluindo públicas e privadas, analisando cinco indicadores, a saber: pesquisa, internacionalização, inovação, ensino e mercado (VALLE; BARBOSA; MALDONADO, 2015a). Os cursos que tiveram seus currículos analisados foram Ciência da Computação, Engenharia da Computação, Sistemas de Informação e Engenharia de Software. Na Tabela 1 são apresentadas as 25 melhores universidades brasileiras que tiveram seus currículos analisados.

Tabela 1 – Universidades brasileiras que tiveram seus currículos analisados (VALLE; BARBOSA; MALDONADO, 2015a)

Universidades Brasileiras	Sigla
Universidade de São Paulo	USP
Universidade Federal de Minas Gerais	UFMG
Universidade Federal do Rio de Janeiro	UFRJ
Universidade Federal do Rio Grande do Sul	UFRGS
Universidade Estadual de Campinas	UNICAMP
Universidade Estadual Paulista Júlio de Mesquita Filho	UNESP
Universidade Federal de Santa Catarina	UFSC
Universidade de Brasília	UNB
Universidade Federal do Paraná	UFPR
Universidade Federal de São Carlos	UFSCAR
Universidade Federal de Pernambuco	UFPE
Universidade Federal de São Paulo	UNIFESP
Universidade Federal do Ceará	UFC
Universidade Federal da Bahia	UFBA
Universidade Federal de Santa Maria	UFSM
Universidade Federal Fluminense	UFF
Universidade do Estado do Rio de Janeiro	UERJ
Pontifícia Universidade Católica do Rio Grande do Sul	PUCRS
Universidade Federal de Viçosa	UFV
Pontifícia Universidade Católica do Rio de Janeiro	PUC-RIO
Universidade Federal do Rio Grande do Norte	UFRN
Universidade Federal de Goiás	UFG
Universidade Estadual de Maringá	UEM
Universidade Estadual de Londrina	UEL
Universidade Federal da Paraíba	UFPB

Os currículos de referência propostos pela SBC para os cursos de graduação na área de Computação sugerem que o conteúdo de teste de software seja abordados em um dos tópicos da disciplina de Engenharia de Software, denominado Verificação, Validação e Teste de Software

² Disponível em: <http://ruf.folha.uol.com.br>

(SBC, 2005; SBC, 2002; SBC., 2003). Em consequência, em geral, as universidades brasileiras abordam o teste de software de forma isolada, conforme induzido pelos currículos de referência propostos pela SBC. A partir da análise realizada nos currículos das universidades brasileiras, observou-se que há algumas universidades que possuem uma disciplina específica e obrigatória para o ensino de teste de software, tais como: Universidade de Brasília (UNB), Universidade Federal do Ceará (UFC), Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Universidade Federal do Rio Grande do Norte (UFRN) e Universidade Federal de Goiás (UFG). Em geral, essa disciplina é ofertada no curso de Engenharia de Software (VALLE; BARBOSA; MALDONADO, 2015a).

Há também duas universidades brasileiras que possuem uma disciplina optativa para o ensino de teste de software, sendo elas: Universidade Federal do Rio de Janeiro (UFRJ) e o Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP). No entanto, de modo geral, no cenário nacional o teste de software é abordado de forma isolada durante o curso de graduação, sem apresentar aos alunos uma visão integrada do teste de software com outras disciplinas, tais como: Algoritmos, Programação Orientada a Objetos, Estrutura de Dados, Análise e Especificação de Requisitos, Reuso de Software, entre outras (VALLE; BARBOSA; MALDONADO, 2015a).

A *Association for Computing Machinery* (ACM) também propõe currículos de referência para cursos de graduação da área de computação. Para os cursos de graduação em Ciência da Computação e Engenharia de Software, a ACM recomenda que os conteúdos relacionados com teste de software sejam abordados em um dos tópicos das disciplinas de programação e na disciplina de verificação e validação de software (ACM; IEEE, 2013; LEBLANC *et al.*, 2006). Para os cursos de Engenharia de Computação, a ACM aconselha que os conteúdos relacionados com teste de software sejam abordados como um dos tópicos das disciplinas de Engenharia de Software e Engenharia de Sistemas de Computação (LEBLANC *et al.*, 2006). Por fim, para o curso de Sistemas de Informações, a ACM sugere que os conteúdos relacionados com teste de software sejam abordados em tópicos de disciplinas de programação (TOPI *et al.*, 2010). De modo geral, a ACM sugere que os conteúdos de teste de software sejam abordados de forma superficial e com poucas aulas (VALLE; BARBOSA; MALDONADO, 2015a).

Para verificar como o teste de software é abordado nos cursos de Ciência da Computação das melhores universidades do mundo, Valle, Barbosa e Maldonado (2015a) consideram o *ranking* 2014-2015 divulgado pela *Times Higher Education*³, que é uma revista inglesa que disponibiliza artigos e notícias sobre a educação de nível superior. Essa revista disponibiliza anualmente um *ranking* das melhores universidades do mundo. Na Tabela 2 são apresentadas as universidades do exterior que tiveram seus currículos analisados.

Os currículos das universidades citadas anteriormente propõem que os conteúdos relacionados com o teste de software sejam ensinados em tópicos dentro das disciplinas de Engenharia

³ Disponível em: <http://www.timeshighereducation.co.uk/>

de Software e/ou programação. Semelhante ao cenário nacional, as universidades do exterior atribuem poucas aulas para o ensino de Teste de Software em sala de aula, fornecendo apenas uma visão geral dos conteúdos. Além disso, não proporcionam uma visão integrada dos conteúdos de teste de software com outras disciplinas ao longo dos curso de graduação em Computação (VALLE; BARBOSA; MALDONADO, 2015a).

Tabela 2 – Universidades do exterior que tiveram seus currículos analisados (VALLE; BARBOSA; MALDONADO, 2015a)

Universidades do Exterior	País
Massachusetts Institute of Technology	Estados Unidos
Stanford University	Estados Unidos
California Institute of Technology	Estados Unidos
Princeton University	Estados Unidos
University of Cambridge	Reino Unido
Imperial College London	Reino Unido
University of Oxford	Reino Unido
Swiss Federal Institute of Technology Zürich	Suíça
University of California - Los Angeles	Estados Unidos
University of California - Berkeley	Estados Unidos
Georgia Institute of Technology	Estados Unidos
École Polytechnique Fédérale de Lausanne	Suíça
National University of Singapore	Singapura
Carnegie Mellon University	Estados Unidos
Cornell University	Estados Unidos
Northwestern University	Estados Unidos
University of Illinois at Urbana Champaign	Estados Unidos
Delft University of Technology	Holanda
Hong Kong University of Science and Technology	Japão
Harvard University	Estados Unidos
University of California - Santa Barbara	Estados Unidos

2.4 Jogos Educacionais

Os jogos eletrônicos começaram a ser desenvolvidos na década de 50, em consequência da evolução dos computadores da época. Os primeiros jogos eletrônicos para videogames e computadores eram fundamentados nos jogos de tabuleiros e de cartas (ZAMBIASI; PINHEIRO, 2010). No entanto, nos últimos anos houve um grande aumento no número de jogadores e empresas de desenvolvimento de jogos eletrônicos. Os desenvolvedores desses jogos têm explorado os diferentes estilos de jogos eletrônicos, e considerado as diversas áreas do conhecimento, como a educação (MACHADO; MORAES; NUNES, 2009).

Segundo Prensky (2012) os estudantes de hoje não são iguais aos estudantes de antigamente, uma vez que mudaram radicalmente nos últimos anos. No entanto, os sistemas educacionais vigentes não evoluíram na mesma proporção. Os estudantes de hoje cresceram

cercados por diversas tecnologias, tais como: computadores, vídeo games, câmeras digitais, celulares, entre outros. Os estudantes de hoje pensam e processam informações de modo diferente das gerações anteriores. Esses estudantes podem ser chamados de **nativos digitais**, pois desde que nasceram passam grande parte de seu tempo jogando vídeo games, enviando e-mails, navegando na internet e realizando outras atividades tecnológicas. As pessoas que não nasceram nesse mundo digital e adotaram alguma atividade tecnológica podem ser chamados de **imigrantes digitais** (PRENSKY, 2001). No entanto, os imigrantes digitais provavelmente possuem dificuldades na utilização de novas tecnologias, por isso é necessário um tempo de adaptação para que eles consigam utilizar essas ferramentas (COSTA; CUZZOCREA; NUZZACI, 2014).

Devido às mudanças ocorridas no comportamento dos estudantes de hoje, novas ferramentas devem ser utilizadas para auxiliar no processo de ensino e aprendizagem de conteúdos educacionais. Dentre elas, encontram-se os jogos educacionais, visto que são considerados importantes ferramentas que facilitam a aprendizagem de conceitos e ideias, uma vez que os mesmos motivam e proporcionam prazeres aos jogadores, permitindo que eles obtenham conhecimentos combinados com a diversão (BAKKER; HEUVEL-PANHUIZEN; ROBITZSCH, 2015). Outro benefício na utilização dos jogos educacionais é que eles podem fornecer um *feedback* imediato aos usuários, no qual os estudantes podem observar instantaneamente a consequência de suas ações no jogo (PRENSKY, 2003).

Os jogos educacionais fornecem motivações aos usuários para a aprendizagem de novas experiências, exercitando suas habilidades de resoluções de problemas. Essa abordagem proporciona o aumento da motivação para a realização das tarefas propostas e possibilita que os usuários participem ativamente na construção de seu próprio conhecimento. Para o desenvolvimento dos jogos educacionais, muitos projetistas consideram uma série de atividades que possibilitam os jogadores explorarem seus próprios conhecimentos e vivenciar novas experiências. Assim, os usuários podem enfrentar novos desafios e estimular suas próprias habilidades (HSU; LIN; SHIH, 2013).

Esses jogos podem ser considerados instrumentos de apoio à aprendizagem, no qual o professor pode utilizá-los em sala de aula como um recurso didático ou utilizá-los como um instrumento complementar para motivar os estudantes ou auxiliar na apresentação dos conteúdos propostos. Esses jogos, geralmente, são projetados especificadamente para auxiliar o ensino de algum conteúdo específico, por exemplo, teste de software, programação, entre outros (AMARAL; BRAGA; GALVAO, 2013).

Os jogos educacionais podem ser utilizados em diversas situações, tais como: reforçar os conhecimentos já adquiridos em sala de aula, facilitar o ensino de algum conteúdo, desenvolver habilidades dos estudantes, entre outras (AMARAL; BRAGA; GALVAO, 2013). A aprendizagem é vista por muitas pessoas como um “dever” do estudante, sendo entendida como uma forma de trabalho. Os jogos educacionais podem ser uma alternativa para amenizar esse problema, já que envolve o prazer, a diversão, a motivação, o interesse e a paixão, incentivando os estudantes

a dedicarem parte de seu tempo e esforços para vencerem obstáculos propostos, adquirindo conhecimentos combinados com a diversão (NETO; FONSECA, 2013).

Um dos papéis fundamentais dos jogos educacionais é que eles podem proporcionar a minimização do fracasso aos jogadores, uma vez que os estudantes podem recomeçar um jogo salvo, permitindo que eles se arrisquem e experimentem hipóteses que seriam muito difíceis de testar em situações reais, no qual o custo do fracasso é muito maior (MATTAR, 2010; MEDEIROS; SCHIMIGUEL, 2012; GEE, 2005). Um exemplo de minimização de fracasso seria utilizar um jogo educacional para simular um laboratório de química. Nesse jogo o usuário poderia realizar diversas soluções químicas e caso o jogador realizasse essa atividade de forma incorreta, o mesmo poderia retornar ao início do jogo e realizar corretamente essa atividade. Na realidade, se alguma solução química fosse realizada de forma incorreta, esse problema não seria tão simples, pois essa ação poderia ser prejudicial à saúde e à segurança do estudante.

Para motivar os estudantes a utilizarem esses jogos, aconselha-se que os mesmos apresentem boas características de jogabilidade, a saber: bons gráficos, sons de alta qualidade, interfaces avançadas, entre outras, semelhantes aos jogos modernos disponíveis no mercado. No entanto, há uma dificuldade em combinar os aspectos lúdicos com os conteúdos educacionais. Em geral, os jogos educacionais focam em apenas em um desses aspectos, o que permite a criação de jogos educacionais de baixa qualidade. Para resolver esse problema é necessário realizar um planejamento sistematizado para a criação de jogos educacionais, considerando tanto o aspecto lúdico quanto o educacional. Quando há uma relação equilibrada entre esses elementos podem-se desenvolver bons jogos educacionais (ZARRAONANDIA *et al.*, 2012).

2.4.1 Elementos Essenciais a Qualidade de Jogos Educacionais

Para a utilização dos jogos educacionais é necessário realizar uma verificação dos reais benefícios desses jogos antes de utilizá-los em um ambiente educacional. Essa análise é necessária porque geralmente a decisão de utilizá-los em um ambiente educacional é por meio de suposições de seus benefícios (ESCUDEIRO; ESCUDEIRO, 2012; CAGILTAY; OZCELIK; OZCELIK, 2015). No entanto, muitos professores escolhem esses jogos sem a realização de uma análise ou avaliação rigorosa para verificar a qualidade desses jogos. Isso é necessário, pois existem muitos jogos educacionais com características de jogabilidade e conteúdos educacionais de baixa qualidade (HAYS, 2005; VALLE *et al.*, 2013). Dessa forma, é imprescindível que jogos educacionais possuam elementos de qualidade.

Com o intuito de elencar esses elementos, Annetta (2010) publicou uma pesquisa que disponibilizava uma lista de seis componentes que todo jogo educacional deveria contemplar para ser considerado de boa qualidade. Essa pesquisa baseou-se em doze anos de estudos relacionados a jogos educacionais. Na Figura 7 são ilustrados os seis componentes propostos por Annetta (2010). Esses elementos foram alinhados conforme sua importância, o elemento de identidade é considerado o elemento mais importante em um jogo educacional.

Figura 7 – Elementos Essenciais à Qualidade de Jogos Educacionais. Adaptado de (ANNETTA, 2010).

- **Identidade:** A identidade é o primeiro componente para um jogo educacional ser considerado de boa qualidade. Esse componente diz que o estudante deve se sentir parte do jogo e acreditar que ele realmente está no ambiente. O sentimento de identidade pode permitir que os estudantes não limitem suas emoções e metas, para que assim eles possam realizar seus objetivos de forma prazerosa. Nos jogos modernos, a identidade do usuário pode ser representada por meio de um personagem único chamado *avatar*⁴. A presença de *avatars* pode influenciar a presença social dos jogadores e ajudar na interação com outros jogadores em um ambiente *multiplayer*;
- **Imersão:** Quando os estudantes estão imersos em um jogo educacional, eles ficam envolvidos com os conteúdos educacionais e motivados a vencerem os obstáculos propostos. Geralmente, quando os usuários possuem um sentimento de identidade e imersão combinados, eles perdem a noção de tempo e espaço enquanto jogam, ou seja, entram em um estado de fluxo⁵;
- **Interatividade:** Quando há uma boa interatividade nos jogos educacionais, os estudantes podem trocar experiências entre si em um ambiente *multiplayer* ou com os agentes computadorizados, nesse caso NPC (*Non Player Caracteres*)⁶. Quando os jogos educacionais são *multiplayers*, a distância geográfica para comunicação entre os jogadores reduz e

⁴ Personagem que representa o usuário e que é exibido em primeira pessoa (CUPERSCHMID; HILDEBRAND, 2013).

⁵ O estado fluxo é quando os jogadores estão tão envolvidos em uma atividade que nada mais parece importar, ou seja, os usuários perdem o sentido de tempo e espaço no jogo. A experiência é tão agradável que os jogadores vão fazer isso mesmo com um grande custo para que eles se sintam satisfeitos (CSIKSZENTMIHALYI, 1975)

⁶ Personagens dos jogos que não são controlados por usuário humano.

consequentemente, a comunicação torna-se uma fator importante para o sucesso do jogo educacional;

- **Maior Complexidade:** Bons jogos educacionais devem possuir níveis de complexidade diferentes para os obstáculos propostos. Esses níveis devem estar de acordo com o nível do usuário; caso contrário, os jogadores poderão se sentir desmotivados para a realização das tarefas. Por exemplo, um jogo educacional não pode ter o mesmo nível de complexidade para as diferentes faixas etárias, ou seja, um jogo criado para um usuário de cinco anos pode não ser tão interessante para um usuários de quinze anos, pois as expectativas desses usuários com relação ao jogo são diferentes.
- **Análise de Desempenho:** Bons jogos educacionais devem possuir instrumentos que permitam observar o desempenho de seus usuários. O *feedback* é considerado por alguns pesquisadores como um instrumento para a avaliação do desempenho dos usuários. O *feedback* pode ser obtido coletando informações dos servidores dos jogos, tais como: a identificação dos estudantes, o tempo de duração do jogo e a localização do estudante dentro do ambientes virtual. Verificar o aprendizado por meio de observações virtuais permite uma análise do ensino, observando o aprendizado que o estudante obteve com a utilização do jogo proposto;
- **Instrutivo:** Bons jogos educacionais devem oferecer ambientes desafiadores que se adaptem de acordo com o desempenho de seus jogadores. Os estudantes de hoje tornaram-se multitarefas, pois escutam músicas, trocam e-mails, jogam vídeo games, assistem TV e utilizam redes sociais tudo isso ao mesmo tempo sem nenhuma dificuldade. Sendo assim, os jogos educacionais devem oferecer a possibilidade dos usuários realizarem tarefas simultâneas. Uma característica imprescindível dos jogos educacionais é que eles podem capturar os pontos fracos e fortes dos estudantes e proporcionar um ambiente que se adapta de acordo com as necessidades dos estudantes. Logo, os estudantes estarão completamente imersos no ambiente de aprendizagem e não perceberão que estão aprendendo os conteúdos educacionais enquanto jogam, tornando a aprendizagem oculta.

Para que os jogos educacionais sejam inseridos em ambientes educacionais, eles devem possuir elementos de boa qualidade como os que foram apresentados anteriormente, pois geralmente os jogos educacionais são inseridos em ambientes educacionais apenas com suposições de seus benefícios, sem que haja uma verificação formal de sua qualidade (VALLE *et al.*, 2013). As principais fases desse processo são: i) concepção/especificação; ii) design; iii) teste e avaliação da qualidade; iv) codificação; v) versão final e suporte.

2.4.2 Processos de Desenvolvimento de Jogos Educacionais

Para auxiliar o desenvolvimento de jogos educacionais têm se utilizado metodologias específicas. Dentre elas, encontra-se o processo GWP (*Game Waterfall Process*) (FLOOD,

2003) que é uma adaptação do processo de desenvolvimento Cascata. As principais fases são: i) concepção/especificação; ii) design; iii) codificação; iv) teste e avaliação da qualidade; v) versão final e suporte. Essas fases ocorrem de forma sequencial sem a possibilidade de interações em fases anteriores. Para isso, os requisitos devem ser bem definidos antes da execução do processo.

Outra metodologia encontrada para auxiliar o desenvolvimento de jogos educacionais é a OriGame. Essa metodologia consiste no desenvolvimento de um diagrama que relaciona as etapas existentes na produção de jogos educacionais em uma visão geral do processo. No desenvolvimento dos jogos existem algumas tomadas de decisão que direcionam a produção dos jogos de acordo com o planejamento, apresentando o que pode ser feito e o que pode ser descartado. Para garantir o correto funcionamento do projeto há ciclos de teste que validam os protótipos físicos, digitais e funcionais até que o jogo seja aceito como um artefato completo e divertido (SANTOS; GÓES; ALMEIDA, 2012).

O processo *Game-Scrum* (GODOY; BARBOSA, 2010) é uma adaptação da metodologia Scrum e XP para o desenvolvimento de jogos educacionais. Essa metodologia é composta de três fases, sendo elas: i) Pré-Produção: que consiste na definição dos objetivos, *brainstorming*, prototipação e desenvolvimento do *game desing*; ii) Produção: que consiste na definição dos requisitos e no desenvolvimento do jogo; e iii) Pós-Produção: que consiste na descrição do documento de *postmortem*.

A metodologia DevJSTA foi proposta por Rocha e Araujo (2014) para o desenvolvimento e integração de jogos, simulações, treinamentos e avaliações. Essa metodologia tem por objetivo integrar jogo, conteúdo, simulação, treinamento das competências requeridas, medições, *feedback*, avaliações e validações para possibilitar a criação, reuso, interoperabilidade e extensão de novos jogos sérios de treinamentos. A metodologia DevJSTA consiste em três fases principais com processos, sendo elas: (1) **pré-produção**: planejamento; (2) **produção**: análise, projeto, implementação, integração e teste; e (3) **pós-produção**: execução e avaliação dos resultados. Além disso, há o processo de **verificação e validação** que compreende todo o ciclo de desenvolvimento.

No entanto, estudos recentes têm considerado os jogos educacionais como um tipo de REA (Recurso Educacional Aberto). Para apoiar o desenvolvimento de REAs foi desenvolvido uma metodologia que é uma evolução das metodologias propostas por Rocha e Araujo (2014) e Arimoto e Barbosa (2016), considerando características de processos ágeis. Os recursos educacionais desenvolvidos com seu auxílio incluem: jogos educacionais, jogos sérios, simulações interativas e animações para fins educacionais e de treinamento (ROCHA *et al.*, 2016). Essa metodologia foi utilizada para a descrição do jogo Testing Game conforme apresentado no Capítulo 3. As principais características de métodos ágeis consideradas são:

- **Indivíduos e interações**: participação colaborativa e integrada de diferentes profissionais;
- **Recurso educacional funcionando**: desenvolvimento incremental com entrega contínua;

- **Colaboração com o cliente:** participação colaborativa do cliente, com avaliação contínua dos artefatos entregáveis.
- **Adaptação a mudanças:** metodologia flexível e iterativa.

Na Figura 8 é ilustrado uma visão geral da metodologia utilizada para a descrição do jogo Testing Game.

Figura 8 – Visão Geral da Metodologia de Desenvolvimento de Recursos Educacionais (ROCHA *et al.*, 2016)

Conforme apresentado na Figura 8, a metodologia de desenvolvimento de recursos educacionais abertos consiste em cinco processos e 10 subprocessos, sendo eles: **I - Pré-produção:** (1) Planejamento Inicial; **II - Produção:** (2) Análise e Planejamento da Sprint, (3) Projeto Iterativo, (4) Implementação Incremental, (5) Integração, teste e revisão da Sprint; **III - Pós-produção:** (6) Ambiente e Manutenção, (7) Execução e (8) Avaliação da Aprendizagem; **IV - Avaliação, Verificação dos Artefatos** e **V - Gestão de Projeto:** (9): Licenciamento e Publicação (10). Cada subprocesso contém diferentes atividades que orientam seu desenvolvimento. Na fase de planejamento é necessário informar o tipo de recurso educacional que será desenvolvido.

Na Figura 9 é especificado cada processo e atividade da metodologia. O processo “Gestão de Projetos” tem por objetivo planejar o projeto, gerenciar seus riscos e monitorar seu progresso. Para isso, há atividades distribuídas ao longo do desenvolvimento do recurso educacional. No processo “Pré-Produção” apresentado na Figura 8 é realizado o planejamento inicial do recurso educacional. Esse processo tem por objetivo realizar um planejamento do recurso educacional que será desenvolvido. Para isso, é necessário realizar uma análise sobre as necessidades para se produzir recursos educacionais, tanto pedagógicas quanto técnicas (ROCHA *et al.*, 2016).

Figura 9 – Visão Geral das Atividades da Metodologia de Desenvolvimento de Recursos Educacionais (ROCHA *et al.*, 2016)

Na Figura 10 é apresentado o *workflow* do subprocesso de Planejamento Inicial. Os principais papéis que atuam nesse subprocesso são: (1) **Educador** que é responsável pelos aspectos pedagógicos do recurso educacional e (2) **Equipe de desenvolvimento** que é a equipe responsável pelos aspectos técnicos do recurso educacional. O processo “Produção” tem por objetivo produzir recursos educacionais de forma iterativa e incremental. As atividades desenvolvidas neste processo são: i) Análise e Planejamento de *Sprint*; ii) Projeto Iterativo; iii) Implementação Incremental; iv) e Integração, Testes e Revisão da *Sprint*.

Na Figura 11 é apresentado o *workflow* do subprocesso de Projeto Iterativo do processo “Produção”. Neste subprocesso deve-se elaborar e descrever o programação educacional de cada fase, bem como descrever os critérios para avaliação dos resultados. Essa atividade é de responsabilidade do Educador. Os analistas de software precisam analisar e projetar a arquitetura do REA que será desenvolvido, analisar e modelar as simulações, analisar e projetar a interface e analisar e especificar o projeto experimental/avaliação. Já os Designer de jogos devem elaborar o projeto do jogo e suas fases. Por fim, os testadores devem analisar e especificar os projetos de testes (ROCHA *et al.*, 2016).

No processo “Pós-Produção” são executados os seguintes subprocessos: i) Ambiente e Manutenção: que é responsável pela configuração e instalação dos itens necessários (por exemplo, Banco de Dados, Aplicativo, Plugin, entre outros) e pelo gerenciamento de requisições

Figura 10 – *Workflow* do subprocesso de planejamento inicial (ROCHA *et al.*, 2016)

de mudanças e backups; ii) Execução: responsável por transmitir as instruções aos alunos, ou seja, a forma de acesso, interação, objetivos e avaliação. Além de preencher um questionário de perfil do estudante. Além disso, são coletados e armazenados dados para o *feedback* do REA desenvolvido; e iii) Avaliação da Aprendizagem: responsável por gerar relatórios com a avaliação do desempenho dos estudantes. Esses relatórios devem ser apresentado aos alunos (ROCHA *et al.*, 2016).

Por fim, o processo “Avaliação, Verificação e Validação” avalia as saídas dos processos (Pré-Produção, Produção e Pós-Produção), ou seja, os artefatos gerados em cada processo. Assim, os artefatos gerados devem ser avaliados para que assim seja possível reusá-los em outras *sprints* (ROCHA *et al.*, 2016). É importante ressaltar que essa metodologia foi utilizada no Capítulo 3 para auxiliar a descrição do jogo desenvolvido neste projeto de mestrado.

Figura 11 – Workflow do subprocesso de projeto iterativo (ROCHA *et al.*, 2016)

2.4.3 Classificação dos Jogos Educacionais

De acordo com a literatura, os jogos educacionais podem ser classificados em diferentes categorias. Essas categorias são estabelecidas de acordo com os objetivos propostos pelos jogos educacionais (AMARAL; BRAGA; GALVAO, 2013; TAROUCO *et al.*, 2004; FERNANDES, 2012; CORRÊA *et al.*, 2013; ARSENAULT, 2009; BITTENCOURT, 2005). A seguir apresentam-se as principais categorias identificadas para a classificação dos jogos educacionais.

- **Ação:** Os jogos educacionais de ação são narrativas com movimentações e ações rápidas, deixando os jogadores sempre atentos para a realização de novas tarefas. Esses jogos podem auxiliar no desenvolvimento psicomotor do estudante, desenvolvendo diversas habilidades, a saber: reflexos, coordenação olho mão, agilidade de pensamento quando ocorre uma circunstância inesperada, entre outras. Essa categoria de jogos educacionais é muito utilizada pelos adolescentes e adultos, em especial os do sexo masculino (AMARAL; BRAGA; GALVAO, 2013). Nessa categoria, destaca-se o desenvolvimento do raciocínio e

incentivo a liberdade e senso de exploração dos estudantes. A interface desses jogos deve chamar a atenção dos usuários para que eles finalizem suas tarefas o mais rápido possível;

- **Aventura:** Os jogos educacionais de aventura, em geral, são narrativas heroicas que envolvem mistérios e a exploração de novas situações. Essa categoria de jogos educacionais é caracterizada pelo controle, por parte do usuário, do ambiente a ser descoberto. Quando esses jogos são bem modelados pedagogicamente, eles podem auxiliar na simulação de atividades impossíveis de serem vivenciadas em sala de aula, tais como um desastre ecológico ou um experimento químico;
- **Lógico:** Os jogos educacionais lógicos têm como objetivo desafiar a mente dos estudante ao invés de seus reflexos. Esses jogos geralmente possuem temporizadores, os quais oferecem um limite de tempo para que os estudantes possam realizar as tarefas propostas dentro do prazo estipulado. Alguns exemplos de jogos educacionais lógicos são: xadrez, damas, caça-palavras, palavras-cruzadas e jogos que exigem resoluções matemáticas;
- **Role-playing game (RPG):** Os jogos educacionais de RPG propõem aos estudantes um ambiente cativante e motivador. Nos jogos dessa categoria, os estudantes assumem o papel de um personagem fictício, controlando suas ações em diversos cenários e interagindo com outros personagens que são encontrados ao longo do jogo. Os jogadores podem construir sua história dinamicamente de acordo com as ações e escolhas que eles realizam. Os jogos dessa categoria são complexos e difíceis de serem desenvolvidos. Eles estimulam a socialização e auxilia no desenvolvimento das habilidade comunicativas;
- **Estratégicos:** Os jogos educacionais de estratégia têm como principal objetivo estimular a sabedoria e habilidade de negócios dos estudantes, principalmente quando relacionados a construção e administração de negócios. Esses jogos podem ser utilizados como ferramenta de simulação, em que os estudantes põem em prática os conhecimentos adquiridos na sala de aula. Os jogos educacionais de estratégia desenvolvem as habilidade de planejamento e persistência em longo prazo.

Geralmente, para o desenvolvimento dos jogos educacionais são considerados diferentes categorias (tais como: ação, aventura, lógicos, entre outras). No entanto, um mesmo jogo educacional pode ser classificado em diferentes categorias, por exemplo, um jogo educacional que pertence à classificação dos jogos estratégico e lógico ao mesmo tempo. Isso acontece porque diferentes jogos possuem diferentes objetivos. Além disso, os jogos podem apresentar diferentes níveis para as diferentes categoria consideradas (CORRÊA *et al.*, 2013).

2.4.4 Jogos Educacionais no Domínio de Teste de Software

Na Subseção 2.3 foi apresentada uma visão geral sobre como o teste de software tem sido ensinado. A partir dessa análise, observou-se que o teste de software é considerado um

conteúdo difícil de ser ensinado por meio de abordagens tradicionais que utilizam apenas aula teóricas e ferramentas tradicionais. Outro ponto observado foi que os estudantes apresentam uma desmotivação em aprender conteúdos relacionados com o teste de software (SOUZA D. M.; MALDONADO; BARBOSA, 2012; SMITH *et al.*, 2012).

Para amenizar esses problemas, alguns pesquisadores propuseram jogos educacionais de apoio ao ensino de teste de software. Esses jogos foram identificados por meio de um mapeamento sistemático realizado por Valle, Barbosa e Maldonado (2015b), no qual identificaram os jogos educacionais como uma abordagem de apoio ao ensino de teste de software. Na Figura 12 é apresentada uma linha do tempo com os jogos educacionais no domínio de teste de software identificados.

Figura 12 – Linha do Tempo com os Jogos Educacionais no Domínio de Teste de Software

O primeiro jogo identificado foi o U-TEST (THIRY; ZOUCAS; SILVA, 2011). Na Figura 13 é ilustrada a tela inicial desse jogo de apoio ao ensino de teste de software. O principal objetivo do jogo U-TEST é auxiliar o ensino de teste de software, especificamente proporcionar aos estudantes uma forma atrativa para entender e aplicar os critérios de particionamento em classes de equivalência e análise de valor limite. Desta forma, os estudantes podem revisar os principais conceitos relacionados com o teste de software (SILVA; THIRY, 2010; THIRY; ZOUCAS; SILVA, 2011).

O segundo jogo identificado foi o Jogo das 7 Falhas. Ele foi proposto para auxiliar o ensino de teste de software, especificamente o teste funcional. O objetivo desse jogo é encontrar as sete falhas existentes nas funcionalidades de um sistema no menor tempo possível (DINIZ; DAZZI, 2011b). Na Figura 14 é apresentada a tela inicial do Jogo das 7 Falhas.

Para a utilização do Jogo das 7 Falhas, os jogadores devem possuir conhecimentos básicos de teste de software, tais como: diferenciar erros, defeitos e falhas, e conhecer teste funcional e testes de sistemas (DINIZ; DAZZI, 2011a). No Jogo das 7 Falhas, o jogador assume o papel de testador em uma equipe de teste de software de uma empresa fictícia denominada “Diniz Quality Assurance”, com o intuito de encontrar as sete falhas existentes em cada funcionalidade

Figura 13 – Tela inicial do jogo U-TEST (SILVA; THIRY, 2010)

Figura 14 – Tela inicial do Jogo das 7 Falhas (DINIZ; DAZZI, 2011b)

testada, utilizando os critérios de particionamento de classe de equivalência e/ou valor limite (DINIZ; DAZZI, 2011c).

Os jogadores passarão de fase somente quando encontrar as sete falhas existentes dentro do tempo estimado. Na avaliação do Jogo das 7 Falhas, os avaliadores comentaram que é necessário criar erros mais fáceis de serem identificados (DINIZ; DAZZI, 2011a). Outra limitação identificada nesse jogo é que ele contempla apenas os conteúdos superficiais do teste funcional.

O terceiro jogo identificado foi o JETS (Jogo da Equipe de Teste de Software). Ele é um *serious game* que simula o setor de teste de uma empresa fictícia. Esse jogo tem como principal objetivo fornecer uma visão geral de como o teste de software pode ser inserido em uma empresa de desenvolvimento de software. Neste jogo, o professor assume o papel de gerente de projeto, podendo auxiliar os estudantes no desafios encontrados (SILVA; MULLER, 2012). Na Figura 15 é ilustrado o cenário da fase de Analista de Teste de Software.

Figura 15 – Jogo JETS (Jogo da Equipe de Teste de Software) (SILVA; MULLER, 2012)

O jogo JETS contém quatro fases que correspondem aos quatro cargos de uma equipe de teste. Essas fases são sequenciais e para avançá-las os estudantes devem atingir uma determinada pontuação no jogo (SILVA; MULLER, 2012). As quatro fases do jogo são: i) Testador de Software, ii) Analista de Teste de Software; iii) Arquiteto de Teste de Software; e iv) Líder de Equipe de Teste.

O quarto jogo identificado foi o TestEG, que é um jogo educacional que tem por objetivo apoiar o ensino de teste de software. O TestEG é um jogo *single-player*, no qual o jogador assume o papel de gerente de teste em uma empresa de desenvolvimento de software (OLIVEIRA, 2013). Esse jogo foi avaliado por meio de uma metodologia específica para avaliação de jogos educacionais proposta por Savi, Wangenheim e Borgatto (2011). Os avaliadores comentaram que é necessário uma evolução do jogo para tornar sua interface mais intuitiva. O jogo TestEG é focado principalmente no papel de gerente de teste.

O quinto jogo identificado foi o *iTest Learning*, que é um jogo educacional para auxiliar o ensino de teste de software, especificamente a fase de planejamento e projeto de teste de software (FARIAS *et al.*, 2012; BEZERRA *et al.*, 2014). Na Figura 16 é ilustrada a primeira fase do jogo *iTest Learning*.

Para começar a jogar o *iTest Learning*, o jogador deve escolher um nível de dificuldade (fácil, médio, difícil), e em seguida, deve escolher um projeto (Scolar, SAPPE, NaDATA, PraViagem, GEDAI, SGEN, CoffeeSys) referente ao nível de dificuldade escolhido. Em seguida, o jogo fornece ao jogador uma descrição do projeto, para que os jogadores possam realizar o planejamento e o projeto dos testes. Segundo Farias *et al.* (2012), o público alvo deste jogo são os estudantes de graduação dos cursos da área de Computação/Informática que possuem disciplinas que contemplem conteúdos relacionados com o teste de software.

O jogo *iTest Learning* foi avaliado utilizando uma metodologia específica para avaliação de jogos educacionais, proposta por Savi, Wangenheim e Borgatto (2011). Nessa avaliação, os estudantes comentaram que a interface é intuitiva e de fácil interação. No entanto, os estudantes

Figura 16 – Primeira fase do jogo *iTest Learning* (FARIAS *et al.*, 2012)

também apontaram algumas sugestões de melhorias para o jogo, entre elas: a necessidade de uma melhor distribuição entre as questões do jogo, a inclusão de animações, a melhoria no *design* da tela, tornar o jogo mais dinâmico, disponibilizar um tutorial, incluir personagens e sons (BEZERRA *et al.*, 2014; MOREIRA; COUTINHO, 2013).

O sexto jogo identificado foi o *iLearnTest*⁷, o qual foi desenvolvido com o propósito de auxiliar no ensino de teste de software, especificamente treinar os usuários para a obtenção da certificação Base do ISTQB (*International Software Qualifications Board*) baseado na estrutura dos capítulos do programa de certificação da versão 2011 (RIBEIRO; PAIVA, 2015). Na Figura 17 é apresentado um cenário do jogo *iLearnTest*.

Figura 17 – Fase de Separar Conceitos do Jogo *iLearnTest* (RIBEIRO; PAIVA, 2015)

⁷ Disponível em: <https://web.fe.up.pt/~apaiva/iLearnTest/>

O jogo iLearnTest é baseado em jogos de plataformas, no qual o personagem é controlado pelas setas do teclado. O jogo está dividido em seis plataformas coloridas, na qual cada uma dessas plataformas representam um conteúdo. Os conteúdos abordados no jogo são: i) fundamentos de teste; ii) testes por meio do ciclo de vida de software; iii) técnicas estáticas; iv) técnicas de concepção de testes; v) gestão de testes; e vi) ferramentas de suporte aos testes (RIBEIRO; PAIVA, 2015).

Para avaliar a eficácia do jogo no processo de ensino de aprendizagem de conteúdos relacionados com o teste de software, realizou-se um experimento com oito estudantes universitários. Os sujeitos do experimento foram divididos em dois grupos, no qual o grupo A não utilizou o jogo iLearnTest e o grupo B utilizou o jogo iLearnTest. No final do experimento os estudantes realizaram um teste para analisar os resultados obtidos. A partir das análises realizadas, observou-se que os estudantes do grupo B obtiveram uma maior motivação para aprendizagem dos conteúdos de teste de software do que os estudantes do grupo A. Quanto à aprendizagem de conteúdos de teste de software, os estudantes do grupo B obtiveram uma pequena melhora com relação ao grupo A (RIBEIRO; PAIVA, 2015).

Por fim, o sétimo jogo identificado foi o JoVeTest⁸, que é um jogo de apoio ao ensino de teste de software. Esse jogo foi desenvolvido para suprir as necessidades relacionados com a formação de novos profissionais, apresentando os conteúdos de teste de software de forma mais efetiva e divertida. A dinâmica do JoVeTest é a mesma do popular jogo da velha, no qual são dois jogadores e cada um deles possui um símbolo (X ou O) e cada jogador joga de forma alternada (BARBOSA; NEVES; NETO, 2016). Na Figura 18 é apresentado um cenário do jogo JoVeTest.

Figura 18 – Tabuleiro do Jogo JoVeTest (BARBOSA; NEVES; NETO, 2016)

Para que o jogador possa marcar seu símbolo no tabuleiro, ele deve responder corretamente uma questão de múltipla escolha sobre o teste de software. Essas questões foram

⁸ Disponível em: <http://experts.icomp.ufam.edu.br/jovetest>

selecionadas das provas do CTFL (*Certified Tester Foundation Level*) que foram realizadas entre os anos 2011 a 2015. Esse exame é realizado pelo ISTQB (*International Software Testing Qualifications Board*) para certificação de profissionais na área de teste de software (BARBOSA; NEVES; NETO, 2016).

A avaliação do jogo foi realizada com estudantes da disciplina de Verificação e Validação do curso de Sistemas de Informação da Universidade do Amazonas. Os estudantes avaliaram o jogo de forma positiva, quanto a facilidade de consolidação dos conteúdos da aula, motivação para revisar os conteúdos da aula e afirmaram que o jogo tornou o ensino/aprendizado mais satisfatório. No entanto, os estudantes fizeram algumas sugestões para o jogo, dentre elas tornar o jogo um aplicativo móvel para ser jogado em dupla pela internet (BARBOSA; NEVES; NETO, 2016).

Na Tabela 3 é apresentado um resumo das características dos jogos educacionais no domínio de teste de software identificados por meio de um mapeamento sistemático realizado por Valle, Barbosa e Maldonado (2015b). As características analisadas foram as técnicas de teste de software, a classificação do jogo e o tipo de avaliação realizada.

Tabela 3 – Características dos Jogos Educacionais no Domínio de Teste de Software

Jogo	Técnica de Teste / Atividade de Teste	Classificação	Avaliação
U-Test	Funcional	Estratégico	Experimento Controlado
Jogo das 7 Falhas	Funcional	Lógico	Experimento Controlado
JETS	Funcional	Estratégico	Experimento Controlado
TestEG	Funcional	Estratégico	Estudo de Caso
iTestLearning	Planejamento e Projeto	Lógico	Estudo de Caso
iLearnTest	Funcional e Estrutural	Aventura	Experimento Controlado
JoVeTest	Funcional e Estrutural	Lógico	Estudo de Caso

É importante ressaltar que as informações da Tabela 3 foram obtidas por meio do entendimento das leituras dos artigos, uma vez que esses artigos não apresentam de forma clara as informações relacionadas com essas características. Como pode ser observado na Tabela 3, nenhum dos jogos identificados abordam o Teste de Mutação. Além disso, foram identificados jogos do gênero estratégico, lógico e aventura. Por fim, validações realizadas foram feitas por meio de experimentos controlado e estudos de caso.

2.5 Considerações Finais

Neste capítulo foram apresentados os principais conceitos relacionados com o tema deste projeto de mestrado. Os conteúdos abordados neste capítulo mostraram que o teste de software é uma atividade essencial no processo de desenvolvimento de software, pois garante uma qualidade mínima para o sistema testado. Uma atividade de teste de software bem sucedida é aquela que

revela a presença de defeitos no software testado. Para auxiliar a execução da atividade de teste existem diferentes técnicas e critérios.

No entanto, apesar do teste de software ser considerado uma atividade essencial no processo de desenvolvimento de software, há uma carência de profissionais qualificados nessa área e uma desmotivação por partes dos estudantes para aprenderem esses conteúdos. Além disso, há uma carência de ambientes que motivem os estudantes a realizar atividades de teste de software. Para resolver esse problemas algumas abordagens têm sido propostas para facilitar o ensino de teste de software, por exemplo, jogos educacionais.

Por meio de uma revisão de literatura foram identificados sete jogos educacionais no domínio de teste de software. Em geral, esses jogos abordam apenas a técnica do Teste Funcional, e não abordam conteúdos relacionados com as técnicas de Teste Estrutural e Baseado em Defeitos. Desta forma, percebe-se que é imprescindível o desenvolvimento de jogos educacionais que abordem outras técnicas e critérios de teste de software. No próximo capítulo é apresentado o jogo Testing Game desenvolvido neste projeto de mestrado.

UM JOGO EDUCACIONAL PARA O ENSINO DE TESTE DE SOFTWARE

3.1 Considerações Iniciais

O teste de software é considerado uma importante atividade para a garantia da qualidade de produtos de software (DELAMARO; MALDONADO; JINO, 2016). No entanto, há uma carência de profissionais qualificados nessa área (SOUZA *et al.*, 2013). Isso pode estar relacionado com a dificuldade de ensinar teste de software por meio de abordagens tradicionais que utilizam apenas aulas teóricas e ferramentas tradicionais (VALLE; BARBOSA; MALDONADO, 2015b). Desta forma, percebe-se que há uma necessidade em utilizar novas abordagens para auxiliar o ensino de teste de software (VALLE; BARBOSA; MALDONADO, 2015a). Sendo assim, foi desenvolvido um jogo educacional de apoio ao ensino de teste de software com programação conforme apresentado neste capítulo.

Na Seção 3.2 é apresentado a metodologia utilizada para a condução deste projeto de mestrado. Na Seção 3.3 é apresentada uma descrição sucinta das duas versões do jogo Testing Game, a primeira versão utilizando o motor de jogos Cocos2D e a segunda versão utilizando o Construct 2 para auxiliar o seu desenvolvimento. Na Seção 3.4 é apresentada a descrição da segunda versão do jogo considerada neste trabalho. A descrição do jogo foi feita por utilizando a metodologia de desenvolvimento de recursos educacionais discutida na Subseção 2.4.2, a qual contém: Planejamento Inicial, Produção, Pós-Produção, Avaliação e Verificação dos Artefatos e Gestão de Projetos. Na Seção 3.5 é demonstrado um exemplo de aplicação do jogo Testing, utilizando a Subfase 2 do Teste Estrutural. Na Seção 3.6 são comparadas as características dos jogos educacionais de teste de software com as características do jogo Testing Game. Por fim, na Seção 3.7 são apresentadas as considerações finais deste capítulo.

3.2 Metodologia

O desenvolvimento deste projeto de mestrado foi dividido em três partes conforme apresentado na Figura 19, sendo elas: Revisão da Literatura e Mapeamento Sistemático, Implementação do Jogo Testing Game e Avaliação do Jogo Testing Game.

Figura 19 – Metodologia utilizada para o desenvolvimento do jogo Testing Game

Na primeira etapa foi conduzida uma **Revisão de Literatura** para identificar os principais conceitos e pesquisas relacionados aos temas deste trabalho. Para isso, foram identificados os principais conceitos e terminologias relacionados com o teste de software, bem como as principais técnicas e critérios de teste, mais detalhes podem encontrados na Seção 2.2. Além disso, foi realizada uma pesquisa sobre os principais conceitos de jogos educacionais, bem como aplicações, classificações e processos de desenvolvimento, conforme apresentado na Seção 2.4.

Para caracterizar o estado da arte referente ao ensino de teste de software foi realizado um **Mapeamento Sistemático** sobre as principais abordagens de apoio ao ensino de teste de software (VALLE; BARBOSA; MALDONADO, 2015b), mais detalhes podem ser observados no Anexo A. Por meio deste mapeamento sistemático foram identificadas 11 abordagens de apoio ao ensino de teste de software. Dentre elas, encontram-se os jogos educacionais e o ensino integrado de teste de software com programação. Juntas essas abordagens foram utilizadas em mais de 50% dos trabalhos analisados. Em especial, muitas pesquisas têm sido realizadas sobre

jogos educacionais, demonstrando que, em geral, eles são ferramentas que podem aumentar a motivação dos estudantes e o aprendizado. Desta forma, este projeto de mestrado considerou a combinação dessas duas abordagens. Logo, o jogo desenvolvido tem por objetivo auxiliar o ensino de teste de software com programação.

Uma segunda pesquisa foi realizada para analisar como as universidades do Brasil e do exterior abordam o ensino de teste de software nas universidades (VALLE; BARBOSA; MALDONADO, 2015a), mais detalhes podem ser observados no Anexo A. Como resultado, observou-se que grande parte das universidades abordam o teste de software em um dos tópicos da disciplina de Engenharia de Software, atribuindo poucas aulas a esse conteúdo e sem uma visão integrada com outras disciplinas. Por meio dessa caracterização do estado da arte com relação ao ensino de teste de software, observou-se que há uma necessidade de desenvolver ferramentas de apoio ao ensino de teste de software considerando uma visão integrada com outras disciplinas, neste caso, programação.

Desta forma, foi realizada a segunda etapa da metodologia que é a **Implementação do Jogo Testing Game**. Para auxiliar essa implementação foi realizado um mapeamento sistemático que teve por objetivo identificar motores de jogos disponíveis na literatura. Como resultado, identificaram-se 40 motores de jogos, os quais estão disponíveis sob licenças proprietárias, gratuitas e software livre. Este mapeamento sistemático teve como principal objetivo auxiliar a escolha de um motor de jogos para apoiar o desenvolvimento do jogo Testing Game. Para a seleção do motor de jogos, desejava-se que ele estivesse disponível sob licença software livre. Desta forma, o motor de jogos Cocos2D foi selecionado.

O Cocos2D apoiou o desenvolvimento do jogo Testing Game em sua versão inicial para a plataforma móvel. No entanto, houveram algumas dificuldades na utilização desse motor de jogos. Dentre elas: poucas funcionalidades que são comuns a jogos eletrônicos e pouco material disponível na literatura sobre seu funcionamento. Consequentemente, o resultado obtido não foi um resultado satisfatório, uma vez que a dinâmica do jogo se limitava apenas em arrastar objetos na tela e essa versão do jogo considerava poucas características essenciais a qualidade dos jogos educacionais conforme apresentado na Seção 2.4.1.

Desta forma, uma nova versão do jogo Testing Game foi desenvolvida utilizando o motor de jogos Construct 2. Esse motor de jogos possui licença proprietária, no entanto, os jogos desenvolvidos com seu auxílio podem ser disponibilizados sob licença software livre. A nova versão do jogo está disponibilizada para a plataforma Web, mais detalhes podem ser observados na descrição do jogo na Seção 3.4. Para a descrição do Testing Game foi utilizada a metodologia de desenvolvimento de recursos educacionais focada no desenvolvimento de jogos educacionais e artefatos gamificados (ROCHA *et al.*, 2016). Essa metodologia consiste em cinco processos, sendo eles: Pré-Produção, Produção, Pós-Produção, Avaliação e Verificação dos artefatos e Gestão de Projeto. Esses processos são discutidos com mais detalhes na próxima subseção.

Por fim, foi realizada a terceira etapa da metodologia que foi a **Avaliação do Jogo Testing Game**. Essa avaliação foi realizada utilizando uma metodologia experimental definida por Shull, Carver e Travassos (2001) para introdução de produtos de software na indústria. Essa metodologia é composta por quatro etapas que avaliam os produtos de software desde sua primeira versão até a sua utilização na indústria.

Essas etapas correspondem a diferentes estudos, sendo eles: i) estudo de viabilidade: o principal objetivo é avaliar se a tecnologia avaliada é viável, ou seja, se atende aos objetivos definidos de forma razoável. Desta forma, é analisada a possibilidade da continuidade do desenvolvimento da tecnologia avaliada; ii) estudo de observação: geralmente, é realizado no ambiente em que a aplicação prática da tecnologia pode ser observada por pesquisadores. Sendo assim, pode-se identificar as principais dificuldades que os participantes apresentam, assim a tecnologia avaliada pode ser refinada; iii) estudo de caso (ciclo de vida): geralmente, é realizado quando há indícios de que a tecnologia avaliada é efetiva. No entanto, essa tecnologia é avaliada de forma isolada, ou seja, não é possível verificar qual o efeito de sua aplicação em conjunto com outras tecnologias.; e iv) estudo de caso (indústria): tem por objetivo analisar a aplicação da tecnologia avaliada em um ambiente industrial.

O mapeamento sistemático desenvolvido por Valle, Barbosa e Maldonado (2015b) e a análise dos currículos das melhores universidades realizada por Valle, Barbosa e Maldonado (2015a) foram utilizados como motivação para o desenvolvimento deste projeto de mestrado. Para a condução da avaliação do Testing Game foi utilizado apenas o primeiro estudo da metodologia experimental mencionada anteriormente, a saber: estudo de viabilidade. Os próprios autores deste trabalho realizaram um estudo de viabilidade preliminar da versão inicial do jogo desenvolvida com auxílio do motor de jogos Cocos2D para analisar a viabilidade dessa ferramenta. Esse estudo motivou o desenvolvimento de uma segunda versão do jogo utilizando outro motor de jogos, a saber: Construct 2, para atingir os objetivos esperados com a condução deste projeto.

Após o desenvolvimento da versão atual do jogo foi realizado o estudo de viabilidade para verificar se o jogo Testing Game é viável e se há possibilidade da continuidade do desenvolvimento dessa ferramenta. O estudo de viabilidade foi dividido em duas etapas: a primeira para avaliar a qualidade do jogo com relação à motivação, experiência do usuário e aprendizagem, sob o ponto de vista dos estudantes. Para isso, foi utilizado o modelo de avaliação de qualidade de jogos educacionais na Engenharia de Software definido por Savi, Wangenheim e Borgatto (2011). Uma descrição desse modelo pode ser encontrada no Apêndice B. A segunda etapa do estudo de viabilidade teve como objetivo avaliar a usabilidade do jogo Testing Game. Para isso, foi utilizado o conjunto de heurísticas proposto por Nielsen (1994). Uma explicação sucinta sobre avaliação heurísticas e o conjunto de heurística utilizado nesse estudo estão disponíveis no Apêndice B. Assim, após a realização das três etapas da metodologia foi desenvolvido este projeto de mestrado.

3.3 Jogo Testing Game

A partir dos cenários apresentados por Valle, Barbosa e Maldonado (2015a) e Valle, Barbosa e Maldonado (2015b) na Subseção 2.3, observou-se a necessidade de desenvolverem novas ferramentas que auxiliassem o ensino de teste de software. Nesse contexto, este trabalho teve por objetivo desenvolver um jogo educacional de apoio ao ensino de teste de software com programação. Desta forma, este trabalho considerou as duas abordagens mais utilizadas para auxiliar o ensino de teste de software, a saber: jogos educacionais e ensino de teste com programação, de acordo com o mapeamento sistemático proposto por Valle, Barbosa e Maldonado (2015b).

Para isso, o jogo propõe situações em que o jogador seleciona casos de teste para programas com estrutura sequencial, condicional e de repetição. Assim, os jogadores podem observar que a atividade de teste de software pode ser considerada desde os seus primeiros programas desenvolvidos. Logo, os estudantes podem criar uma “cultura de teste de software” adequada, pois eles podem observar quais são os critérios de teste necessários para cada estrutura do programa e podem utilizar uma estratégia de teste incremental conforme comentado no Capítulo 2. A seguir, apresentam-se as duas versões desenvolvidas do jogo Testing Game.

3.3.1 Versão Inicial - Cocos2D

A versão inicial do jogo Testing Game foi desenvolvida utilizando o motor de jogos Cocos2D. Esse motor de jogos foi selecionado a partir de um mapeamento sistemático descrito na subseção 3.4.1.7.1. A ideia inicial foi utilizar um motor de jogos *open source* para auxiliar o desenvolvimento do jogo para as plataformas Web e Móvel. No entanto, nesta primeira versão, foram identificadas algumas dificuldades, dentre elas: poucos tutoriais disponíveis na literatura e uma comunidade pouco ativa. Além disso, a ferramenta Cocos2D apresentou algumas limitações, tais como: i) fornecer poucas funcionalidades comuns aos jogos eletrônicos; e ii) funcionalidades pouco intuitivas. Na Figura 20 é apresentada uma fase do jogo Testing Game na versão inicial.

A versão inicial foi desenvolvida para a plataforma Móvel. Nesta fase, os jogadores precisam identificar os vetores que tenham o tamanho válido de acordo com a especificação disponibilizada no início do jogo, ou seja, os vetores que tenham o tamanho igual a 4. Como pode ser observado, os jogadores podem visualizar a pontuação e o tempo no topo da tela, além de poderem voltar para a fase anterior a qualquer momento. É importante ressaltar que os jogadores só avançarão de fase se conseguirem concluir o desafio proposto. Outra fase do jogo na versão inicial pode ser observada na Figura 21.

Nesta fase, os jogadores precisam encontrar casos de teste para satisfazer o critério “Todos-Nós”. Quando casos de teste válidos são selecionados, o GFC é pintado com a cor vermelha para mostrar quais foram os nós cobertos do grafo. É importante ressaltar que no topo da tela há um módulo educacional representado pela imagem de um livro sobre o critério

Figura 20 – Fase do Testing Game - Versão Inicial - sobre vetores com o tamanho válido

Figura 21 – Fase do Testing Game - Versão Inicial - sobre o critério Todos-Nós

“Todos-Nós”.

O resultado obtido na versão inicial foi considerado insatisfatório, pois não foram considerados grande parte das elementos essenciais à qualidade dos jogos educacionais conforme apresentado na Seção 2.4. Esses elementos não foram considerados, pois houve uma grande dificuldade para implementá-los utilizando o Cocos2D, uma vez que há pouco material e

exemplos disponíveis na literatura sobre esse motor de jogos. Logo, na versão inicial do jogo, os usuários precisam apenas arrastar os elementos da tela. Diante desse resultado, foi necessário desenvolver uma segunda versão do jogo, na qual foi escolhido um novo motor de jogos, a saber: Construct 2, conforme apresentada na subseção a seguir.

3.3.2 Segunda Versão - Construct 2

A segunda versão do jogo Testing Game foi desenvolvida utilizando o motor de jogos Construct 2. Esse motor de jogos será discutido com mais detalhes na Subseção 3.4.1.7.2. No entanto, diferentemente do motor de jogos Cocos2D, o Construct 2 possibilita a implementação dos principais elementos essenciais à qualidade dos jogos educacionais, conforme apresentado na Seção 2.4, de forma simples e intuitiva.

A segunda versão do jogo Testing Game é a versão considerada como resultado deste projeto de mestrado. O jogo foi desenvolvido para a plataforma Web e sua descrição está detalhada na próxima seção deste capítulo.

3.4 Descrição do Jogo Testing Game

Em sua primeira versão, não foram utilizadas metodologias específicas de desenvolvimento de jogos para auxiliar o desenvolvimento do jogo Testing Game. O jogo foi desenvolvido utilizando um processo ad-hoc. No entanto, foi utilizada uma metodologia de desenvolvimento de recursos educacionais focada no desenvolvimento de jogos educacionais e artefatos gamificação para auxiliar a descrição do jogo Testing Game. Essa metodologia é uma evolução das metodologias propostas por Rocha e Araujo (2014) e Arimoto e Barbosa (2016) e está disponível por meio de um relatório técnico publicado na biblioteca do ICMC/USP (ROCHA *et al.*, 2016).

Além de auxiliar a descrição do jogo Testing Game, essa descrição contribuiu para verificar se o jogo considerou particularidades necessárias para o desenvolvimento de jogos educacionais. Os aspectos que não foram considerados no desenvolvimento do jogo poderão ser utilizados como trabalhos futuros para a evolução do jogo. Na Figura 22 é ilustrado uma visão geral da metodologia utilizada para a descrição do jogo Testing Game.

Essa metodologia é fundamentada em práticas ágeis e baseada em práticas de projeto pedagógico, avaliação, treinamento e jogos. Ela tem por objetivo apoiar o desenvolvimento de recursos educacionais abertos para que eles sejam eficientes e eficazes (ROCHA *et al.*, 2016). Os princípios de métodos ágeis considerados nessa metodologia são:

- **Indivíduos e interações:** participação colaborativa e integrada de diferentes profissionais;
- **Recurso educacional funcionando:** desenvolvimento incremental com entrega contínua;

Figura 22 – Visão Geral da Metodologia de Desenvolvimento de Recursos Educacionais (ROCHA *et al.*, 2016)

- **Colaboração com o cliente:** participação colaborativa do cliente, com avaliação contínua dos artefatos entregáveis.

A metodologia de desenvolvimento de recursos educacionais abertos consiste em 5 processos e 10 subprocessos (ROCHA *et al.*, 2016), conforme apresentado a seguir:

- **I - Pré-produção:** (1) Planejamento Inicial;
- **II - Produção:** (2) Análise e Planejamento da Sprint, (3) Projeto Iterativo, (4) Implementação Incremental, (5) Integração, teste e revisão da Sprint;
- **III - Pós-produção:** (6) Ambiente e Manutenção, (7) Execução e (8) Avaliação da Aprendizagem.
- **V - Avaliação, Verificação dos Artefatos**
- **VI - Gestão de Projeto:** (9): Licenciamento e Publicação (10);

Cada subprocesso contém diferentes atividades que orientam seu desenvolvimento. Essa metodologia auxilia o desenvolvimento de diferentes recursos educacionais, tais como: jogos educacionais, simulações, artefatos gamificados, entre outros. Desta forma, na fase de planejamento é necessário informar o tipo de recurso educacional que será desenvolvido. De acordo com o tipo do recurso educacional desejado, alguns grupos de atividades podem ser consideradas ou não. Além disso, algumas atividades podem ocorrer de forma paralela durante a execução do subprocesso (ROCHA *et al.*, 2016).

A metodologia é baseada em métodos ágeis quanto à produção rápida e contínua de artefatos. Assim, os artefatos que serão entregues deverão ser planejados, produzidos e avaliados em cada Iteração (entre uma a três semanas) (ROCHA *et al.*, 2016).

3.4.1 Planejamento Inicial

O objetivo desse subprocesso é realizar um planejamento do recurso educacional que será produzido, contendo as necessidades de produzir o recurso educacional, tanto pedagógicas quanto técnicas.

3.4.1.1 Contexto e Problema

O teste de software é reconhecido como uma atividade importante para a garantia da qualidade de produtos de software (DELAMARO; MALDONADO; JINO, 2016). No entanto, alguns estudos demonstram que há uma carência de profissionais qualificados nessa área (SOUZA D. M.; MALDONADO; BARBOSA, 2012; VALLE; BARBOSA; MALDONADO, 2015b). Diante desse cenário Valle, Barbosa e Maldonado (2015a) realizaram uma pesquisa para conhecer como o teste de software é abordado nas universidades. Para isso, foram selecionadas as melhores universidades nacionais e internacionais para analisar os currículos de referência. O resultado obtido foi que grande parte dessas universidades abordam o teste de software como um tópico da disciplina de Engenharia de Software com poucas aulas para esse conteúdo e não há uma relação dos conteúdos de teste de software com outras disciplinas.

Desta forma, percebeu-se que há a necessidade de utilizar outras abordagens para apoiar abordagens tradicionais que utilizam apenas aulas teóricas com ferramentas tradicionais. Para identificar outras abordagens Valle, Barbosa e Maldonado (2015b) realizaram um mapeamento sistemático com o objetivo de identificar abordagens de apoio ao ensino de teste. Os jogos educacionais e o ensino de teste com programação foram as duas abordagens mais utilizadas pelos trabalhos analisados. Desta forma, este trabalho teve por objetivo desenvolver um jogo educacional de apoio ao ensino integrado de teste de software com programação.

3.4.1.2 Público-Alvo

O público-alvo do jogo Testing Game são estudantes de graduação que cursam disciplinas que contém conteúdos relacionados com o teste de software. O jogo pode ser utilizado como um material de apoio às aulas de teste de software, pois os estudantes podem treinar suas habilidades adquiridas em sala de aula. Desta forma, o jogo pode ser considerado um material complementar à aprendizagem dos conteúdos de teste de software.

3.4.1.3 Conteúdo

O livro “Introdução ao Teste de Software” escrito por Delamaro, Maldonado e Jino (2016) e a nota didática “Introdução ao Teste de Software com Ferramentas Java” (VINCENZI *et al.*, 2016) foram utilizados para a elaboração dos conteúdos abordados no jogo e também como literatura complementar. O jogo aborda as três técnicas do teste de software, a saber: Funcional, Estrutural e Baseado em Defeitos. Para demonstrar a aplicação dos critérios considerados no jogo foi utilizado o programa BubbleSort escrito em Java. Na Tabela 4 são apresentados os conteúdos abordados no jogo para cada técnica de teste considerada.

Tabela 4 – Conteúdos abordados no Jogo Testing Game

Técnicas de Teste de Software	Conteúdo
Teste Funcional	Critérios de Teste de Software, Especificação do Programa BubbleSort, Critério de Particionamento em Classes de Equivalência, Tabela de Classe de Equivalência, Critério de Análise do Valor Limite e Conjunto de Caso de Teste Mínimo.
Teste Estrutural	Grafo de Fluxo de Controle, Critério Todos-Nós, Critério Todas-Arestas, Usos de Variável (definição, uso computacional, uso predicativo), Grafo Def-Uso, Critério Todas-Definições, Critério Todos-Usos, Conjunto Mínimo de Casos de Teste e Caminho Não-Executável.
Teste Baseado em Defeitos	Operadores de Mutação em Java, Programas Mutantes, Mutantes Equivalentes, Escore de Mutação e Conjunto de Caso de Teste Mínimo.

Como pode ser observado na Tabela 4, o jogo Testing Game utiliza uma estratégia de teste incremental. Nessa estratégia são aplicados inicialmente critérios “mais fracos”, que podem ser menos eficazes para a avaliação da adequação do conjunto de casos de teste. Em seguida, são utilizados critérios “mais fortes” de forma incremental e eventualmente mais eficazes, porém, em geral, mais caros.

No entanto, o jogo Testing Game apresenta algumas limitações, tais como: i) utiliza apenas um único programa no jogo; ii) não possui acoplamento com ferramentas de teste de software; e iii) o programa BubbleSort, que foi utilizado no jogo, não contém defeitos conhecidos.

3.4.1.4 Avaliação

Para a avaliação do jogo Testing Game foi realizado um estudo de viabilidade que está descrito no Capítulo 4. Para a execução desse estudo foi utilizado o modelo de avaliação de qualidade de jogos educacionais proposto por Savi, Wangenheim e Borgatto (2011) e o conjunto de heurísticas proposto por Nielsen (1994). A avaliação ocorreu no mês de outubro de 2016 com alunos do Laboratório de Engenharia de Software do Programa de Pós-Graduação em

Ciência de Computação e Matemática Computacional do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP).

No jogo, os estudantes também podem visualizar seu desempenho por meio da pontuação e tempo disponibilizados em tempo real no jogo. Além disso, o jogo fornece uma classificação final para cada jogador, podendo ser: péssima, ruim, regular, boa e excelente. Assim, para considerar que os jogadores possuem um bom conhecimento em teste de software é esperado que eles obtenham uma classificação final **boa** ou **excelente**.

3.4.1.5 Objetivos de Design

Após a definição das necessidades pedagógicas do jogo, ou seja, definir o contexto, problema, público-alvo, conteúdo e avaliação, foi necessário definir as necessidades técnicas, ou seja, definir as particularidades exigidas em jogos educacionais, por exemplo, programação e banco de dados. Do ponto de vista do *design*, o Testing Game é um jogo para a plataforma Web com dimensão gráfica 2D. Nesse jogo, o estudante é representado por um *avatar*¹ que pode realizar diversas ações durante o jogo, tais como: andar, correr, pular, desviar e eliminar os inimigos, entre os outras ações. O jogo contém três níveis representados por três portas diferentes. Cada nível corresponde a uma técnica de teste de software conforme apresentado na Figura 23.

Figura 23 – Níveis do Jogo Testing Game

Para abrir as portas é necessário utilizar as chaves que são disponibilizadas durante o jogo. Cada nível do jogo é representado por uma técnica de teste que contém diferentes subfases. O teste funcional contém 8 subfases, o teste estrutural 10 subfases e o teste de mutação 5 subfases. Essas subfases estão representadas no jogo por cadeados conforme apresentado na Figura 24. Para liberar um cadeado, ou seja, uma subfase, é necessário executar a fase anterior com sucesso.

Em cada subfase há um desafio diferente com relação aos conteúdos abordados. No início de cada subfase há um enunciado que informa ao jogador o que ele deve fazer para obter sucesso na subfase. Outra funcionalidade é com relação ao conteúdo teórico do teste de software, a cada novo conceito abordado no jogo, há um módulo no qual o usuário pode acessar e ler sobre o conteúdo. Para ter acesso a essa funcionalidade, o jogador deve clicar em um ícone representado

¹ personagem controlado pelo jogador

Figura 24 – SubFases do Teste Funcional

por um livro que está localizado na parte superior da tela no lado direito. Nas subfases do teste estrutural os jogadores podem ter acesso ao código do programa testado clicando no ícone representado por uma lupa.

No jogo há um menu fixo na parte superior da tela que disponibiliza a pontuação total do jogador juntamente com o tempo de jogo conforme apresentado na Figura 25. Essas informações podem ser utilizadas como uma forma de *feedback* para o usuário verificar seu desempenho no jogo. Além disso, há uma funcionalidade no menu que permite o jogador desativar a música de fundo. É importante lembrar que por *default* o jogo começa com a música de fundo ativada. No entanto, o jogador pode ativar ou desativar em qualquer momento do jogo.

Figura 25 – Menu Superior do Jogo Testing Game

Quando os usuários jogarem os três níveis do jogo, eles terão treinado suas habilidades nas três técnicas do teste. Desta forma, eles utilizaram a estratégia de teste incremental, analisando cada estrutura do programa e observando quais são os critérios de teste que podem ser aplicados naquele contexto. Por fim, quando o jogador finalizar o jogo, eles terão acesso a uma classificação final com relação ao seu desempenho no jogo.

3.4.1.6 Objetivos de Arte

Do ponto de vista da arte, o Testing Game é um jogo com dimensão gráfica 2D que possui diversos cenários. Ele é considerado um jogo de plataforma, pois o jogador pode executar diversas ações, tais como: correr, pular entre diferentes plataformas e obstáculos, enfrentar inimigos e coletar objetos bônus. Essa categoria de jogos é muito conhecida entre os jogadores, uma vez que existem diversos jogos famosos pertencentes a essa categoria, tais como: Super Mario Bros, Mega Man, Sonic, Donkey Kong, entre outros.

Para a definição do *avatar*, inimigos, objetos e cenários do jogo foram selecionados imagens obtidas por meio de páginas da internet (por exemplo: <http://www.gameart2d.com> e <http://opengameart.org>) que disponibilizam as imagens com licenças gratuitas. Alguns exemplos

de imagens utilizadas no jogo são ilustradas na Figura 26. As imagens desta figura representam o *avatar* utilizado no jogo, juntamente com os inimigos que os jogadores devem eliminar durante o jogo para alcançar os objetivos propostos. Além disso, na figura há alguns objetos que podem ser encontrados durante o jogo.

Figura 26 – Exemplos de imagens utilizadas no jogo Testing Game

3.4.1.7 Objetivos Técnicos

Do ponto de vista técnico, utilizou-se o motor de jogos Construct 2 conforme comentado anteriormente para auxiliar o desenvolvimento do Testing Game. A escolha desse motor de jogos foi apoiada pelo resultado do mapeamento sistemático apresentado na próxima subseção. O Construct 2 fornece recursos pré-estabelecidos quanto a mecanismos gráficos, som e inteligência artificial. O Testing Game considerou a dinâmica *single player*, ou seja, possibilita a participação de apenas um usuário no jogo.

3.4.1.7.1 Game Engine

Os jogos eletrônicos têm ganhado grande destaque nos últimos anos. Atualmente, têm-se produzido jogos cada vez mais realistas e com narrativas mais complexas, exigindo dos jogadores um maior esforço reflexivo e cognitivo (COUTINHO *et al.*, 2013). Esses jogos têm sido utilizados em diversas áreas (FLORES *et al.*, 2008; LIMA; AMORIM, 2014) e têm conquistado um bom espaço no mercado internacional. Segundo o relatório elaborado pela empresa *Entertainment Software Association* (ESA) em 2012, as vendas da indústria de jogos eletrônicos atingiu 16 bilhões de dólares nos Estados Unidos (ASSOCIATION *et al.*, 2012; YANG; JIAO, 2011).

No entanto, desenvolver jogos eletrônicos não é uma tarefa trivial. É necessário realizar um bom planejamento e escolher boas tecnologias para auxiliar o desenvolvimento desses jogos. Dentre essas tecnologias estão os motores de jogos (em inglês, *Game Engine*) que são softwares

ou bibliotecas que executam funcionalidades padrões de jogos eletrônicos. Os motores de jogos oferecem recursos que dão suporte à renderização gráfica, áudio, animação, luzes, detecção de colisões, inteligência artificial, entre outros.

Os motores de jogos têm como principal objetivo facilitar o desenvolvimento de jogos eletrônicos para que não seja necessário desenvolvê-los a partir do zero (RANKIN; VARGAS *et al.*, 2008). Para escolher o motor de jogos utilizado no desenvolvimento do jogo Testing Game foi realizado um mapeamento sistemático que teve por objetivo de identificar os principais motores de jogos disponíveis na literatura. As questões de pesquisa consideradas neste mapeamento sistemático foram:

- Q1: Quais são os principais motores de jogos para auxiliar o desenvolvimento de jogos eletrônicos?
- Q2: Quais são as principais características dos motores de jogos identificados na Q1?
 1. Quais são os tipos de licenças que os motores de jogos possuem?
 2. Quais são as dimensões gráficas que os jogos desenvolvidos pelos motores de jogos suportam?
 3. Quais são as plataformas que os jogos desenvolvidos pelos motores de jogos suportam?
 4. Qual é a qualidade da documentação dos motores de jogos identificados?

Neste mapeamento sistemático foram selecionados 28 trabalhos. Após a análise desses trabalhos foram identificados 40 motores de jogos diferentes. Para coletar informações adicionais foram consultados manuais e páginas específicas sobre motores de jogos. Na Tabela 5 são apresentados os motores de jogos identificados e suas respectivas características.

As características consideradas neste mapeamento sistemático são: licença (P - Privada, G - Gratuita e SL - Software Livre), dimensões gráficas (2D e 3D), plataformas que os jogos desenvolvidos com seu auxílio podem ser exportados e qualidade da documentação (Boa, Regular e Ruim (Web, Desktop, Móvel e Console). Este mapeamento sistemático poderá ser utilizado em outros trabalhos, uma vez que forneceu uma visão geral dos motores de jogos disponíveis na literatura, possibilitando que os desenvolvedores de jogos eletrônicos escolham o motor de jogo que melhor se encaixe aos requisitos do seu jogo, ou seja, esse mapeamento sistemático pode ser considerado um instrumento de apoio à tomada de decisão no escopo de desenvolvimento de jogos.

A partir dos resultados obtidos foi selecionado o motor de jogos utilizado no desenvolvimento do jogo Testing Game. A primeira versão do jogo foi desenvolvida utilizando o motor de jogos Cocos 2D. Essa ferramenta foi selecionada, pois a ideia inicial era utilizar um motor de jogos que fosse *open source* e o jogo fosse exportado para a plataforma móvel. Porém,

Tabela 5 – Motores de Jogos Identificados no Mapeamento Sistemático

Motores	Licença			Dimensões		Plataforma				Documentação		
	P	G	SL	2D	3D	Web	Desktop	Móvel	Console	Boa	Regular	Ruim
3D Game Studio	X			X	X		X				X	
Torque Game Engine	X			X	X		X					X
Construct 2	X			X		X	X	X		X		
SIO2	X			X	X		X	X				X
RPG Maker	X			X			X		X			X
Stagecast Creator	X			X		X						X
ImpactJS	X			X		X		X				X
C4	X				X		X		X	X		
Deep Creator	X				X		X					X
Dx Studio	X			X	X							X
Neo Axis	X				X							X
SAGE	X				X		X	X			X	
RAGE	X				X		X		X		X	
Unity	X	X		X	X	X	X	X	X	X		
UDK	X	X			X	X	X	X	X	X		
Dark Basic Pro	X	X		X	X		X		X			X
TV3D	X	X			X		X					X
Game Maker	X	X		X		X	X	X	X	X		
Stencyl	X	X		X		X	X	X		X		
Microsoft XNA 2D		X		X			X	X	X			X
CAGE				X			X	X				X
Gamex			X	X			X	X				X
GreenFoot			X	X			X			X		
PyGame			X	X			X	X			X	
Cocos2D			X	X		X	X	X	X		X	
Crystal Space			X	X	X		X			X		
Fabula			X	X								X
id Tech 2			X		X		X		X			X
ZFXCE			X		X		X					X
e-Adventure			X	X			X			X		
LOVE 2D			X	X			X			X		
Allegro			X	X	X		X	X				X
Delta 3D			X		X							X
jMonkey			X		X		X	X	X			X
Lightfeather 3D			X		X		X					X
Nebula Device 2			X		X							X
Ogre 3D			X		X		X	X			X	
Open Dynamics Engine			X		X							X
Panda 3D			X		X		X			X		
Realm Forge Soya 3D			X		X		X					X

após sua utilização observou-se que esse motor de jogos apresentava algumas limitações, e consequentemente o resultado obtido no desenvolvimento do jogo não foi satisfatório conforme apresentado na Subseção 3.3.1.

Portanto, foi necessário desenvolver uma segunda versão do jogo utilizando outro motor de jogos, a saber: Construct 2. Para isso, na segunda seleção do motor de jogos foram considerados motores de jogos que possuem licença proprietária mas que possibilitam a publicação dos jogos utilizando uma licença *open source*. O Construct 2, o motor de jogos selecionado para auxiliar o desenvolvimento da segunda versão do jogo Testing Game, é fundamentado em uma licença proprietária e tem por objetivo auxiliar o desenvolvimento de jogos eletrônicos para as

plataformas Web, Desktop e Móvel. Um ponto positivo do Construct 2 com relação ao Cocos2D é que ele tem uma comunidade crescente e ativa, e consequentemente possui uma documentação de boa qualidade. A seguir, apresenta-se uma descrição sucinta do Construct 2.

3.4.1.7.2 Construct 2

O Construct 2² é um motor de jogos que auxilia o desenvolvimento de jogos eletrônicos 2D em HTML5. Essa ferramenta possibilita que desenvolvedores com pouco conhecimento em programação consiga desenvolver jogos eletrônicos sem grandes esforços, uma vez que ele contém um sistema *drag-and-drop*, um editor visual e um sistema de lógica fundamentado em eventos/comportamentos que facilitam o desenvolvimento de jogos (RIBEIRO; PAIVA, 2015). Na Figura 27 é ilustrada a interface do motor de jogos Construct 2.

Figura 27 – Interface do Ambiente de Desenvolvimento do Construct 2

Um dos grandes benefícios do Construct 2 é facilidade com que os jogos desenvolvidos com seu auxílio são compilados, pois essa atividade não requer muito tempo. Assim, pode-se visualizar os jogos de forma rápida e a qualquer momento (MANZO; MANZO, 2015). Logo, é possível realizar uma prototipagem rápida dos jogos, tornando o processo de teste, detecção e correção de problemas algo simples durante todo o processo de desenvolvimento dos jogos. Outro benefício do Construct 2 é que ele permite que os jogos desenvolvidos com seu auxílio sejam publicados em diferentes plataformas, tais como: Web, Mobile e Desktop (FRYDENBERG, 2015).

² Disponível em: <https://www.scirra.com>

3.4.2 Produção

Para a descrição do jogo Testing Game foram utilizadas 3 iterações que serão detalhadas a seguir:

3.4.2.1 Iteração 1 - Teste Funcional

A Iteração 1 representa o primeiro nível do jogo Testing Game. Nesse nível são abordados os conteúdos relacionados com a técnica de teste funcional. Nesse nível há 8 subfases conforme mencionado anteriormente.

3.4.2.1.1 Análise e Planejamento - Teste Funcional

No primeiro subprocesso da metodologia de desenvolvimento de recurso educacionais, foi necessário identificar os artefatos que foram utilizados para o desenvolvimento da Iteração. Como essa foi a primeira Iteração, foi necessário criar os artefatos que foram utilizados. Alguns objetos utilizados foram: chave para abrir as portas, 3 portas que representam os níveis do jogo, cadeados, ícones de som, ícone do conteúdo teórico, ícone com a imagem de um coração para representar a vida do personagem e moeda para atribuir bônus ao jogador. Esses objetos foram obtidos por meio de licença gratuita no *website*: <http://www.flaticon.com/free-icon/>.

O personagem do jogo, o tiro e seus inimigos foram obtidos por meio de licença gratuita no *website*: <http://www.gameart2d.com/>. As quatro imagens utilizadas como *background* nessa Iteração foram obtidas de forma gratuita no *website*: <http://opengameart.org/>. Já a tabela de equivalência, vetores e retângulos que representam o tipo de vetor, a saber: ordenado, desordenado e parcialmente ordenado, foram criados utilizando o programa Microsoft PowerPoint 2013 do pacote Microsoft Office 2013. Por fim, os casos de teste, enunciados, botão de continuar, *labels* de tempo e pontuação foram criados a partir dos recursos disponibilizados pelo motor de jogos Construct 2.

3.4.2.1.2 Projeto Interativo - Teste Funcional

Neste subprocesso, foi definido o programa educacional para cada subfase do jogo com relação aos conteúdos de teste funcional. Os jogadores só conseguem avançar de subfase e liberar a próxima subfase se conseguirem concluir a subfase atual com sucesso. A seguir, apresenta-se uma breve descrição das subfases do teste funcional.

- **Subfase 1:** Na primeira subfase, o jogador deve arrastar os critérios para a técnica de teste que os correspondem. As técnicas consideradas estão representadas por dois retângulos, o teste funcional como uma caixa preta e o teste estrutural como uma caixa branca. Quando o jogador arrastar um critério para a técnica correta, um sinal sonoro é realizado indicando que o jogador acertou, caso contrário, um sinal sonoro é emitido para indicar que a ação

realizada é incorreta. Nesta subfase está disponível um módulo de conteúdo teórico sobre o teste funcional. Esse conteúdo pode ser acessado clicando no ícone que representa um livro, o qual está fixo no topo da tela do lado direito. Na Figura 28, pode-se visualizar a interface da subfase 1.

Figura 28 – Primeira subfase do Teste Funcional no Testing Game

- **Subfase 2:** Na segunda subfase, é apresentado ao jogador a especificação do programa utilizado, a saber: BubbleSort, para aplicar os conceitos de teste de software durante a execução do jogo. A especificação diz que o programa BubbleSort só aceitará vetores com o tamanho igual a 4. Em seguida, é apresentado um enunciado para o jogador que diz que ele deve encontrar vetores com o tamanho igual a 4. Para isso, o jogador deve eliminar os inimigos e os vetores inválidos. Após o jogador completar o desafio é apresentado uma lista com os vetores válidos e inválidos da subfase. Na Figura 29 é apresentado a interface da subfase 2.

Figura 29 – Segunda subfase do Teste Funcional no Testing Game

- **Subfase 3:** A terceira subfase é semelhante a subfase 2. A única diferença é que o jogador deve encontrar vetores com tamanho igual a 4 e que eles contenham apenas números inteiros conforme a especificação do programa BubbleSort apresentada no jogo.
- **Subfase 4:** Nessa subfase, o jogador deve preencher uma tabela de equivalência a partir da especificação do programa BubbleSort. Nessa subfase há um módulo com conteúdo teórico sobre o critério “Particionamento em Classes de Equivalência”.
- **Subfase 5:** Nessa subfase, o jogador deve encontrar casos de teste válidos para satisfazer o critério de “Particionamento em Classes de Equivalência”. Para isso, ele deve eliminar os inimigos e os casos de teste que são inválidos. Na Figura 30 é apresentada a interface da Subfase 5.

Figura 30 – Quinta subfase do Teste Funcional no Testing Game

- **Subfase 6:** Nessa subfase, o jogador deve selecionar os casos de teste disponíveis na tela para as classes de equivalência do programa BubbleSort.
- **Subfase 7:** Nessa subfase, o jogador deve analisar a classe de equivalência Tamanho do Vetor para encontrar todos os casos de teste disponíveis na subfase para satisfazer o critério “Análise do Valor Limite”. Para isso, o jogador deve eliminar todos os inimigos e os casos de teste que são inválidos.
- **Subfase 8:** Por fim, nessa subfase, o jogador tem acesso a chave que abre a porta do próximo nível do jogo. O próximo nível corresponde aos conteúdos relacionados com o teste estrutural.

3.4.2.1.3 Implementação Incremental - Teste Funcional

Após a seleção dos conteúdos que seriam abordados no teste funcional, foram considerados os recursos de arte e áudio. Os cenários utilizados nas fases do teste funcional foram obtidos

por meio de um *website* conforme comentado anterior. É importante ressaltar que essas imagens estão disponíveis sob licença gratuita. Com relação aos recursos de áudio, foram utilizados apenas três artefatos, sendo eles: música de fundo, som de acerto e som de erro. Esses áudios foram obtidos por meio de uma biblioteca de áudio com licença gratuita disponível no YouTube.

Nesta Iteração não foram considerados recursos de vídeos. A programação das subfases nesta Iteração foi realizada utilizando o motor de jogos Construct 2 por meio de programação de blocos. Além disso, não foram considerados recursos de banco de dados, pois o jogo ainda não possuía essa funcionalidade.

3.4.2.1.4 Integração, Testes e Revisão - Teste Funcional

Após a definição, criação e reutilização dos artefatos, realizou-se a integração dos recursos de arte, multimídia e conteúdo. Assim, foi possível o desenvolvimento de cada subfase. Em seguida, houve a integração das subfases para que juntas formassem uma nível com conteúdos relacionados com o teste funcional. Para verificar a eficiência das subfases foram testadas suas principais funcionalidades. Logo, uma combinação entre as funcionalidade e uma sequência de passos foram utilizadas para testar essa Iteração. Após os testes realizados, a Iteração 1, que representa os conteúdos do teste funcional, foi aprovada como um recurso educacional pronto.

3.4.2.2 Iteração 2 - Teste Estrutural

A Iteração 2 representa o segundo nível do jogo Testing Game. Nesse nível são abordados os conteúdos relacionados com a técnica do teste estrutural. Nesse nível há 10 subfases conforme mencionado anteriormente.

3.4.2.2.1 Análise e Planejamento da Iteração - Teste Estrutural

No subprocesso “Análise e Planejamento” foram identificados os artefatos utilizados para o desenvolvimento da Iteração 2. Para isso, foram criados artefatos que ainda não haviam sido utilizados no projeto. Esses artefatos foram: um sinal de interrogação para representar um desafio, uma bandeira para representar o final da subfase e uma lupa para representar a opção de visualizar código fonte do programa Bubblesort. Esses objetos foram obtidos utilizando uma licença gratuita no *website*: <http://www.flaticon.com/free-icon/>. Além desses artefatos, foram utilizadas oito imagens diferentes para o *background* das subfases. Essas imagens também foram obtidas por meio de licença gratuita no *website*: <http://opengameart.org/>.

Os códigos do programa BubbleSort e os Grafos de Fluxo de Controle foram criados utilizando o programa Microsoft PowerPoint 2013 do pacote Microsoft Office 2013. Já os casos de teste, caminhos não-executáveis, enunciados, botão de continuar, *labels* de tempo e pontuação foram reusados da Iteração 1. Outros artefatos também foram reusados da Iteração 1, sendo eles: chave para abrir a porta do teste estrutural, ícones de som, ícone de conteúdo teórico, ícone com

a imagem de um coração para representar a vida do personagem, moeda para atribuir bônus ao jogador, personagem, tiro e os três inimigos. É importante ressaltar que todos esses objetos foram obtidos por meio de licenças gratuitas.

3.4.2.2.2 Projeto Interativo - Teste Estrutural

Neste subprocesso, foi definido o programa educacional de cada subfase com relação aos conteúdos do teste estrutural. É importante ressaltar que assim como na Iteração 1, os jogadores só têm acesso a próxima subfase se conseguirem concluir a subfase atual com sucesso. A seguir, apresenta-se uma breve descrição das subfases do teste estrutural.

- **Subfase 1:** Na primeira subfase, é apresentado ao jogador o código do programa BubbleSort. Em seguida, o jogador deve encontrar o GFC que corresponde ao código do programa BubbleSort. Para isso, o jogador deve eliminar os GFCs que não correspondem ao programa e eliminar os inimigos durante os desafios que são propostos na subfase. Nesta subfase está disponível um módulo de conteúdo teórico sobre o teste estrutural. Esse conteúdo pode ser acessado clicando no ícone que representa um livro, o qual está fixo no topo da tela do lado direito. Na Figura 31 é possível visualizar a interface da Subfase 1.

Figura 31 – Primeira subfase do Teste Estrutural no Testing Game

- **Subfase 2:** Na segunda subfase, o jogador deve encontrar casos de teste para satisfazer o critério “Todos-Nós” para o programa BubbleSort. Quando o jogador encontra um caso de teste válido, o GFC é atualizado na tela para demonstrar quais os nós do GFC foram cobertos com o respectivo caso de teste. É importante ressaltar que o jogador deve eliminar os casos de teste inválidos e os inimigos.
- **Subfase 3:** A terceira subfase é semelhante a Subfase 2 desta Iteração. No entanto, ao invés de encontrar casos de teste para satisfazer o critério “Todos-Nós”, o jogador deve encontrar casos de teste para satisfazer o critério “Todas-Arestas”.

- **Subfase 4:** Na quarta subfase, o jogador deve encontrar o conjunto mínimo de casos de teste para satisfazer os critérios “Todos-Nós” e “Todas-Arestas”. Para isso, o jogador deve eliminar os casos de teste que são inválidos ou que fornecem a mesma cobertura dos casos de teste que foram selecionados.
- **Subfase 5:** Na quinta subfase, o jogador deve apenas observar os usos das variáveis do programa Max. Esse programa foi utilizado para que os jogadores pudessem entender quais são os tipos de usos de variáveis para que nas próximas subfases eles utilizem esses conceitos no programa BubbleSort. Na Figura 32 é possível visualizar a interface da Subfase 5.

Figura 32 – Quinta subfase do Teste Estrutural no Testing Game

- **Subfase 6:** Na sexta subfase, o jogador deve encontrar todas as variáveis do programa BubbleSort que tiveram uso predicativo. Para isso, os jogadores deverão eliminar os inimigos e as variáveis que não tiveram uso predicativo. No final dessa subfase é apresentado ao jogador o GFC com as variáveis que tiveram uso predicativo. Nesta subfase está disponível um módulo de conteúdo teórico sobre os critérios da família de fluxo de dados. Esse conteúdo pode ser acessado clicando no ícone que representa um livro, o qual está fixo no topo da tela do lado direito.
- **Subfase 7:** A sétima subfase é semelhante a Subfase 6. No entanto, ao invés de encontrar as variáveis que tiveram uso predicativo, o jogador deve encontrar todas as variáveis do programa BubbleSort que tiveram uso computacional. Na Figura 33, pode-se visualizar a interface da Subfase 7.
- **Subfase 8:** Na oitava subfase, o jogador deve observar o grafo Def-Usos do programa BubbleSort e preencher uma tabela que demonstra em quais nós do grafo as variáveis foram definidas. Na Figura 34, pode-se visualizar a interface da Subfase 8.

Figura 33 – Sétima subfase do Teste Estrutural no Testing Game

Points: 0 Time: 7

Fill the table with the respective nodes of definition of variables from Bubble Sort program.

Associations			
Variable	Definition Node		
size			
i			
j			
aux			
a			

6 1 1 3
7 6 8

Def-Use Graph - Bubble Sort

Figura 34 – Oitava subfase do Teste Estrutural no Testing Game

- Subfase 9:** Na nona subfase, o jogador deve encontrar as associações requeridas para satisfazer o critério “Todos-Usos” para o programa BubbleSort. Para isso, o jogador deve eliminar as associações incorretas que forem encontradas durante a subfase.
- Subfase 10:** Por fim, na décima subfase, o jogador deve encontrar o caminho do GFC do programa BubbleSort que é um caminho não-executável. Para isso, o jogador deve eliminar os caminhos executáveis do GFC. Nesta subfase está disponível um módulo de conteúdo teórico que explica o que é um caminho não-executável. Esse conteúdo pode ser acessado clicando no ícone que representa um livro, o qual está fixo no topo da tela do lado direito.

3.4.2.2.3 Implementação Incremental - Teste Estrutural

Após a definição dos conteúdos abordados nesta Iteração, foram considerados para o desenvolvimento os recursos de arte e áudio. Os oito cenários da Iteração 2 que contém imagens de fundo foram obtidos por meio do *website* comentado anteriormente. É importante ressaltar que todas as imagens utilizadas estão sob licença gratuita. Com relação aos recursos de áudio, eles foram reusados da Iteração 1.

Na Iteração 2 não foram considerados recursos de vídeos. A programação das 10 subfases do teste estrutural foram desenvolvidas com o auxiliar do motor de jogos Construct 2. Também não foram considerados nessa Iteração recursos de banco de dados, uma vez que os dados coletados durante o jogo não são armazenados.

3.4.2.2.4 Integração, Testes e Revisão - Teste Estrutural

Após a execução dos subprocessos anteriores, realizou-se a integração dos artefatos considerados, a saber: recursos de arte, multimídia e conteúdo. Após a integração desses artefatos foram integradas as 10 subfases da Iteração. Assim, as 10 subfases formaram uma fase/nível sobre conteúdos do teste estrutural no jogo Testing Game. É importante ressaltar que foram testadas as principais funcionalidades desta Iteração. Para isso, foram geradas combinação entre as funcionalidades e uma sequências de passos que representam a execução do jogo. Após os testes realizados, a *Iteração 2* foi aprovada como pronto o recurso educacional de apoio ao ensino de teste estrutural.

3.4.2.3 Iteração 3 - Teste de Mutação

A Iteração 3 representa o terceiro e último nível do jogo Testing Game. Neste nível são abordados os conteúdos relacionados com o teste de mutação. Neste nível há 5 subfases conforme mencionado anteriormente.

3.4.2.3.1 Análise e Planejamento - Teste de Mutação

Neste subprocesso foram identificados os artefatos que foram necessários para o desenvolvido da *Iteração 3*. Para esta *Iteração* foram criados poucos artefatos, pois grande parte deles foram reusados das Iterações 1 e 2. Todas as subfases dessa Iteração utilizaram imagens de *background*. Essas imagens foram obtidas por meio de licença gratuita no *website*: <http://opengameart.org/>. Os códigos dos programa com mutação foram criados por meio do programa Microsoft PowerPoint 2013 do pacote Microsoft Office 2013.

Já os enunciados, botão de continuar, *labels* de tempo e pontuação, chave para abrir a porta do teste de mutação, ícones de som, ícone de conteúdo teórico, ícone com a imagem de um coração para representar a vida do personagem, moeda para atribuir bônus ao jogador,

personagem, tiro e os três inimigos foram reusados das *sprints* 1 e 2. É importante ressaltar que todos esses objetos foram obtidos por meio de licenças gratuitas (<http://www.gameart2d.com/>).

3.4.2.3.2 Projeto Interativo - Teste de Mutação

Neste subprocesso, foi definido o programa educacional de cada subfase com relação aos conteúdo do teste de mutação. Assim como nas *sprints* 1 e 2, os jogadores só têm acesso a próxima subfase se conseguirem concluir a subfase atual com sucesso. A seguir, apresenta-se uma breve descrição das subfases do teste de mutação.

- **Subfase 1:** Na primeira subfase, o jogador deve encontrar o programa mutante gerado a partir dos operadores de mutação: aritmético, atribuição, relacional e lógico. Para isso, o jogador deve eliminar todos os programas mutantes que não foram gerados por essas classes de operadores de mutação. Nesta subfase está disponível um módulo de conteúdo teórico sobre teste de mutação e operadores de mutação. Esse conteúdo pode ser acessado clicando no ícone que representa um livro, o qual está fixo no topo da tela do lado direito. Na Figura 35 é apresentada a interface da subfase 1 do teste de mutação do jogo Testing Game, na qual o jogador deve encontrar o programa mutante gerado a partir do operador de mutação relacional.

Figura 35 – Primeira subfase do Teste de Mutação no Testing Game

- **Subfase 2:** Na segunda subfase, o jogador deve encontrar casos de teste que eliminem os programas mutantes do programa Op. Para isso, o jogador deve eliminar os inimigos e os casos de teste inválidos.
- **Subfase 3:** A subfase 3 é semelhante a subfase 2. No entanto, ao invés do jogador encontrar casos de teste para eliminar os programas mutantes do programa Op ele deve encontrar casos de teste para eliminar os mutantes do programa BubbleSort. O jogador

pode visualizar no código quais as mutações foram eliminados com a seleção do caso de teste. Além disso, o jogador pode visualizar seu score de mutação que está disponível no topo da tela no lado direito de forma fixa.

- **Subfase 4:** Nessa subfase, o jogador deve eliminar os programas mutantes que são considerados mutantes equivalentes. Para isso, o jogador deve eliminar os inimigos, barreiras e mutantes que não são equivalentes. Para facilitar o entendimento do conteúdo abordado está disponível um módulo de conteúdo teórico sobre mutantes equivalentes. Esse conteúdo pode ser acessado clicando no ícone que representa um livro, o qual está fixo no topo da tela do lado direito. O local que houve mutação é marcado com cor vermelha no código do programa BubbleSort.

Figura 36 – Quarta subfase do Teste de Mutação no Testing Game

- **Subfase 5:** Por fim, na quinta e última subfase, o jogador deve selecionar casos de teste para eliminar os programas mutantes. Nessa subfase são apresentados aos jogadores 3 códigos de programas diferentes, o primeiro contém uma estrutura apenas sequencial, o segundo contém uma estrutura condicional e o último uma estrutura de repetição. Desta forma, os jogadores podem observar que pode-se aplicar o teste de mutação em qualquer estrutura de programa.

3.4.2.3.3 Implementação Incremental - Teste de Mutação

Após a definição dos conteúdos que foram abordados no teste de mutação, foram considerados os recursos de arte e áudio. Assim como nas *sprints* 1 e 2, os cenários utilizados foram obtidos por meio de um *website* conforme comentado anteriormente. Com relação aos recursos de áudio, esses foram reusados da Iteração 1. Na Iteração 3 não foram considerados recursos de vídeo. Para auxiliar o desenvolvimento dessa Iteração foi utilizado o motor de jogos Construct 2. É importante lembrar que não foram considerados recursos de banco de dados, pois o jogo não armazena informações coletadas durante o jogo.

3.4.2.3.4 Integração, Testes e Revisão da Iteração - Teste de Mutação

Após a definição dos artefatos e conteúdo, realizaram-se as integrações dos recursos de arte, multimídia e conteúdo. Desta forma, foram criadas as subfases desta Iteração. Após a integração das subfases foram realizados testes para verificar a eficiência do jogo Testing Game. Para realizar os testes foram criados um conjunto de passos e combinações entre as funcionalidades. Após realizar os testes, a Iteração 3 foi aprovada como recurso educacional pronto para auxiliar o ensino de teste de mutação.

3.4.3 Pós-Produção

Nesta etapa, o recurso educacional já foi desenvolvido e pode ser utilizado como instrumento de apoio à aprendizagem de conteúdos educacionais. Neste processo é necessário considerar a execução, ambiente e avaliação da aprendizagem conforme apresentado a seguir:

3.4.3.1 Execução

Neste subprocesso, os estudantes já podem utilizar o jogo como um recurso educacional de apoio ao ensino de teste de software. Até o momento, o jogo Testing Game foi utilizado apenas no estudo de viabilidade realizado para avaliar a qualidade e usabilidade do jogo, conforme detalhado no Capítulo 4.

3.4.3.2 Ambiente

O jogo Testing Game está disponibilizado para a plataforma Web no servidor do Scirra Arcade do Construct 2 e pode ser acessado por meio do link: <https://goo.gl/50pmWP>. Após acessar esse *link* é necessário esperar alguns segundos para que o jogo seja carregado no navegador. É importante ressaltar que para ter acesso ao jogo não é necessário instalar recursos, tais como: banco de dados, aplicativos, *plugins*, entre outros.

3.4.3.3 Avaliação da Aprendizagem

A avaliação do jogo foi realizada por meio da metodologia experimental definida por Shull, Carver e Travassos (2001), utilizando o estudo de viabilidade que é detalhado no Capítulo 4. Esse estudo teve por objetivo avaliar a usabilidade e a qualidade do jogo sob o ponto de vista dos estudantes com relação à motivação, experiência do usuário e aprendizagem. No entanto, não foram realizados experimentos com estudantes para verificar a aprendizagem adquirida com auxílio do jogo Testing Game. Esse tipo de avaliação será considerada em trabalhos futuros. Desta forma, percebe-se que os alunos podem obter um *feedback* do jogo com relação ao seu desempenho, verificando a pontuação, tempo e classificação final.

3.4.4 Avaliação, Verificação e Validação

Este é um processo transversal na metodologia de desenvolvimento de recursos educacionais. Ele avalia as saídas dos processos (Pré-Produção, Produção e Pós-Produção), ou seja, os artefatos gerados em cada processo. Desta forma, a cada artefato gerado foi avaliado sua qualidade para que assim fosse possível reusá-los em outras iterações.

3.4.5 Gestão de Projetos

Este é um processo que é transversal na metodologia de desenvolvimento de recursos educacionais. Este processo considera desde a concepção do recurso educacional no planejamento inicial até o processo de Pós-Produção com a avaliação da aprendizagem. Dentre os subprocessos considerados, o licenciamento e a publicação são descritos a seguir:

3.4.5.1 Licenciamento

O Construct 2 é um motor de jogos com três tipos de licença, a saber: Free Edition, Personal License e Business License. A licença Free Edition permite o desenvolvimento de projetos com até 100 blocos. O jogo Testing Game contém 897 blocos. Desta forma, utilizou-se neste projeto a licença Personal que é limitada quanto ao uso comercial, pois se o projeto desenvolvido alcançar mais do que 5000 dólares é necessário utilizar a licença Business License.

No entanto, é importante ressaltar que utilizando a licença Personal License todos os direitos dos produtos gerado no Construct 2 são dos desenvolvedores, ou seja, não é necessário pagar *royalties* por qualquer produto gerado ou publicado no Scirra. Desta forma, o jogo Testing Game foi disponibilizado utilizando uma licença *open source*.

3.4.5.2 Publicação

Após a definição da licença do recurso educacional, considerou-se a atividade publicação do recurso gerado. No jogo Testing Game não foram desenvolvidos tutoriais, pois para ter o acesso ao jogo não é necessário instalação de bibliotecas e softwares. O jogo é intuitivo e consequentemente apresenta algumas dicas nos enunciados das subfases, além de possuir módulos com conteúdo teórico. Sendo assim, não foram considerados tutoriais pedagógicos. No entanto, na avaliação realizada e apresentada no Capítulo 4, os usuários sugeriram que houvesse um tutorial mostrando os principais controles do jogo, mesmo que os controles utilizados no jogo Testing Game sejam os comandos padrões da área de jogos. Esse tipo de tutorial será considerado na próxima evolução do jogo.

O código do Testing Game está disponibilizado no Bitbucket³ que é um serviço de hospedagem de projetos similar ao GitHub e GitLab. No entanto, esse sistema utiliza apenas o Git. O Bitbucket é considerado um sistema de controle de versões distribuído. Para este projeto

³ Disponível: <https://bitbucket.org/>

de mestrado foram criados dois repositórios no Bitbucket. O primeiro com o código do Testing Game em HTML5 ⁴ e o segundo com o arquivo do jogo da plataforma Construct 2 ⁵.

3.5 Exemplo de Aplicação do Jogo Testing Game

Esta seção apresenta um exemplo da dinâmica do jogo Testing Game. Para isso, foi selecionado a Subfase 2 do Teste de Estrutural para demonstrar as principais funcionalidades, desafios e obstáculos. Na Figura 37 é apresentada a tela inicial da Subfase 2.

Figura 37 – Desafios da Subfase 2 do Teste Estrutural no Testing Game

Nesta Subfase, o jogador deve encontrar casos de teste para satisfazer o critério “Todos-Nós” para o programa BubbleSort. Para isso ele deve eliminar todos os inimigos que aparecerem durante a subfase. Quando casos de teste válidos são encontrados o GFC do programa que está fixo no lado direito da tela é atualizado, ou seja, os nós que foram cobertos pelo caso de teste são pintados com a cor vermelha. Na Figura 38 é possível observar que o personagem encontrou um caso de teste válido ([7,7,7,7] [7,7,7,7]).

Figura 38 – Caso de teste válido na Subfase 2 do Teste Estrutural no Testing Game

Após a identificação de um caso de teste válido, é necessário que o personagem “pegue” o caso de teste. Para isso, o personagem precisa tocar no caso de teste. Como pode ser observado,

⁴ Disponível em: <https://bitbucket.org/pedrohdvalle/testinggamehtml5>

⁵ Disponível em: https://bitbucket.org/pedrohdvalle/testinggame_construct2/overview

o caso de teste está localizado em um ponto alto do jogo. Para que o personagem consiga tocar no caso de teste válido, é preciso pular utilizando a tecla “backspace” do teclado e em seguida o caso de teste será transformado em uma moeda que representa a pontuação do jogador como pode ser observado na Figura 39.

Figura 39 – Moeda que representa a pontuação no Testing Game

Quando um caso de teste válido é coletado, adiciona-se 5 pontos no score total do jogo e quando um caso de teste inválido é coletado, subtrai-se 5 pontos no score total. No entanto, o jogador pode encontrar casos de teste que são válidos, mas que fornecem a mesma cobertura dos casos de teste que já foram coletados. Na Figura 40 é possível observar que o jogador encontrou um caso de teste válido ([3,3,3,3] [3,3,3,3]). Este caso de teste fornece a mesma cobertura de nós que o caso de teste coletado anteriormente ([7,7,7,7] [7,7,7,7]). No entanto, o jogador ganhará 5 pontos no score total, no entanto, o GFC permanecerá da mesma forma, uma vez que esse caso de teste fornece a mesma cobertura.

Figura 40 – Segundo caso de teste válido na Subfase 2 do Teste Estrutural no Testing Game

Na Figura 41 o jogador encontra um caso de teste inválido ([9,7,5][5,7,9]). Quando o jogador coleta esse caso de teste, subtrai-se 5 pontos da pontuação total do jogo, conforme apresentado na Figura 42, pois esse é um caso de teste inválido.

Para satisfazer o critério “Todos-Nós” é necessário selecionar outro caso de teste para que todos os nós do GFC sejam cobertos. Para isso, o jogador pode selecionar o caso de teste

Figura 41 – Caso de teste inválido na Subfase 2 do Teste Estrutural no Testing Game

Figura 42 – Pontuação subtraída na Subfase do Teste Estrutural no Testing Game

[4,3,2,1] [1,2,3,4] conforme apresentado na Figura 43.

Figura 43 – Terceiro caso de teste válido na Subfase 2 do Teste Estrutural no Testing Game

Quando o jogador coleta o caso de teste [4,3,2,1] [1,2,3,4], os Nós do GFC são pintados com cor vermelha conforme apresentado na Figura 44. Como pode ser observado, todos os nós do GFC foram cobertos e desta forma, o critério “Todos-Nós” foi satisfeito.

Após o jogador tocar no caso de teste válido, o caso de teste é transformado em uma moeda, na qual o jogador poderá pegá-la para adicionar 5 pontos no score total, conforme apresentado na Figura 45.

Figura 44 – Quarta subfase do Teste de Mutação no Testing Game

Figura 45 – Quarta subfase do Teste de Mutação no Testing Game

Após satisfazer o critério “Todos-Nós” o jogador pode ir até uma bandeira que está no fim da subfase e quando ele tocá-la, ele passará para a próxima subfase do jogo Testing Game.

3.6 Comparação dos Jogos Educacionais no Domínio de Teste de Software

A partir do mapeamento sistemático realizado por Valle, Barbosa e Maldonado (2015b) foram identificados sete jogos no domínio de teste de software, conforme apresentado na Subseção 2.4.4. Na Tabela 6 há uma comparação dos jogos identificados com o jogo Testing Game, analisando algumas características, sendo elas: técnica de teste, classificação e avaliação.

É importante ressaltar, que essas características foram obtidas por meio do entendimento da leitura dos artigos que correspondem a esses jogos. Uma vez que, grande parte desses jogos não fornecem claramente as informações exatas sobre as características analisadas. Por meio da Tabela 6, observa-se que seis jogos abordam apenas a técnica de Teste Funcional e que apenas o jogo Testing Game aborda as três técnicas. Além disso, o Testing Game foi o único jogo classificado no gênero Ação.

Tabela 6 – Comparação do jogo Testing Game com jogos no domínio de teste de software

Jogo	Técnica de Teste / Atividade de Teste	Classificação	Avaliação
U-Test	Funcional	Estratégico	Experimento Controlado
Jogo das 7 Falhas	Funcional	Lógico	Experimento Controlado
JETS	Funcional	Estratégico	Experimento Controlado
TestEG	Funcional	Estratégico	Estudo de Caso
iTestLearning	Planejamento e Projeto	Lógico	Estudo de Caso
iLearnTest	Funcional e Estrutural	Aventura	Experimento Controlado
JoVeTest	Funcional e Estrutural	Lógico	Estudo de Caso
Testing Game	Funcional, Estrutural e Mutação	Ação	Estudo de Viabilidade

3.7 Considerações Finais

Neste capítulo foi apresentado a descrição do jogo Testing Game. Essa descrição foi realizada utilizando uma metodologia de desenvolvimento de recursos educacionais que é uma evolução das metodologias propostas por Rocha e Araujo (2014) e Arimoto e Barbosa (2016). Para a descrição da metodologia, o jogo foi dividido em três *sprints*, a saber: teste funcional, teste estrutural e teste de mutação, essas *sprints* representam os níveis que o possui. Para auxiliar o desenvolvimento do jogo, utilizou-se o motor Construct 2, pois ele possibilita que se desenvolva jogos eletrônicos sem grandes esforços. Esse motor de jogos foi selecionado a partir de um mapeamento sistemático que teve por objetivo identificar motores de jogos disponíveis na literatura. Além disso, foi apresentado um exemplo de aplicação do jogo Testing Game e uma comparação desse jogo com os demais jogos do domínio de teste de software identificados na literatura. No próximo capítulo é apresentado o estudo de viabilidade realizado para avaliar a qualidade e usabilidade do jogo Testing Game.

AVALIAÇÃO DA QUALIDADE E USABILIDADE DO JOGO TESTING GAME

4.1 Considerações Iniciais

Segundo Hays (2005), Escudeiro e Escudeiro (2012) e Cagiltay, Ozcelik e Ozcelik (2015), é recomendado que se produzam evidências dos benefícios dos jogos educacionais antes que eles sejam utilizados como ferramentas de apoio à aprendizagem de conteúdos educacionais. No entanto, a avaliação da qualidade dos jogos educacionais é ainda, em geral, limitada ou inexistente (CONNOLLY; STANSFIELD; HAINEY, 2007; PETRI; WANGENHEIM, 2016). Desta forma, estudos têm sido realizados para propor modelos/métodos para auxiliar a avaliação de jogos educacionais. Desta forma, é possível avaliar se esses jogos estão adequados a serem inseridos em ambientes educacionais.

Para avaliar a qualidade do jogo Testing Game, desenvolvido neste projeto, foi realizado um estudo de viabilidade conforme proposto por Shull, Carver e Travassos (2001). Esse tipo de estudo permite identificar os principais pontos positivos e as limitações do produto de software avaliado antes que ele seja entregue aos usuários finais. Para a condução desse estudo, utilizou-se o modelo de avaliação da qualidade de jogos educacionais proposto por Savi, Wangenheim e Borgatto (2011). Esse modelo é focado em avaliar jogos educacionais no contexto do ensino de engenharia de software. Quanto à usabilidade do jogo, foi utilizado o conjunto de heurísticas proposto por Nielsen (1994).

Neste capítulo, apresenta-se o estudo de viabilidade realizado para verificar a qualidade e a usabilidade do Testing Game. Na Seção 4.2 é apresentado o planejamento do estudo de viabilidade. Na Seção 4.3 é demonstrada a execução da avaliação do Testing Game. Na Seção 4.4 são discutidos os resultados obtidos na avaliação do jogo. Por fim, na Seção 4.5, apresentam-se as considerações finais sobre a avaliação realizada.

4.2 Planejamento do Estudo de Viabilidade

Para verificar a qualidade e a usabilidade do jogo Testing Game utilizou-se o planejamento do estudo de viabilidade proposto a seguir. O planejamento desse estudo contém as questões de pesquisa, objetivos, seleção dos sujeitos, instrumentação e ameaças à validade.

4.2.1 Questões de Pesquisa

A seguir, apresentam-se as questões de pesquisa proposta no estudo de viabilidade para verificar a qualidade e a usabilidade do jogo desenvolvido neste projeto de mestrado.

- QP_1 : O jogo Testing Game apresenta uma boa qualidade, com relação à motivação, experiência do usuário e aprendizagem sob o ponto de vista dos estudantes?
- QP_2 : O jogo Testing Game apresenta uma boa usabilidade sob o ponto de vista dos estudantes?

É importante ressaltar que neste estudo de viabilidade não está sendo avaliado a aprendizagem adquirida pelos alunos na utilização do jogo Testing Game, mas sim opinião dos estudantes com relação a contribuição do jogo para sua aprendizagem.

4.2.2 Objetivo

A seguir, apresenta-se o objetivo do estudo de viabilidade realizado neste projeto de mestrado de acordo com o método GQM (SOLINGEN *et al.*, 2002; BASILI *et al.*, 2007):

Analisar o jogo Testing Game

Com propósito de avaliá-lo

Com respeito a sua qualidade e usabilidade

Do ponto de vista dos estudantes

No contexto de estudantes de pós-graduação em Engenharia de Software, especificamente teste de software.

4.2.3 Seleção dos Sujeitos

Para a realização deste estudo, foram convidados estudantes de pós-graduação (mestrado e doutorado) do Laboratório de Engenharia de Software (Labes) do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP). Os sujeitos que participaram do estudo de viabilidade possuem conhecimento na área de teste de software, uma vez que seus projetos de pesquisa estão estritamente relacionados com o teste de software. Além disso, os sujeitos cursaram a disciplina “SSC5877-5/2 Validação e Teste de Software” oferecida na pós-graduação.

É importante ressaltar que todos os sujeitos aceitaram participar do estudo de forma voluntária. Para participação deste estudo, os estudantes tiveram que:

- Manifestar interesse em participar do estudo, assinando um formulário de consentimento;
- Preencher um formulário de caracterização de perfil para verificar o grau de conhecimento de cada sujeito;
- Participar do treinamento sobre os conceitos de teste de software, jogos educacionais e avaliação heurística; e
- Responder o questionário de *Feedback*.

4.2.4 Instrumentação

A seguir, apresenta-se uma breve descrição dos materiais utilizados no estudo de viabilidade, os quais estão disponíveis no pacote de experimento no Apêndice A:

- Materiais utilizados nas duas etapas do estudo de viabilidade:
 - **Formulário de Consentimento:** Os sujeitos assinaram um Formulário que os sujeitos manifestaram seu interesse em participar do estudo de viabilidade de forma voluntária. Esse formulário é dividido em três partes: i) visão geral; ii) confidencialidade; e iii) benefícios e liberdade de desistência.
 - **Jogo Educacional:** Foi utilizado o jogo Testing Game para realização do estudo de viabilidade. Esse jogo está disponível para a plataforma Web e aborda três técnicas de teste de software, sendo elas: Funcional, Estrutural e Baseado em Defeitos (Mutação);
 - **Questionário de Feedback:** Questionário de *feedback* dos sujeitos sobre o estudo de viabilidade. Nesse questionário tem uma questão em que os sujeitos autoriza ou não a utilização dos seus dados no estudo de viabilidade. Além disso, existem haviam questões sobre a qualidade do treinamento, estrutura do estudo de viabilidade e críticas/sugestões ao experimento.
- Materiais utilizados na primeira etapa do estudo de viabilidade (Avaliação da Qualidade):
 - **Formulário de Caracterização:** Trata-se de um questionário para caracterizar o nível de conhecimento de cada sujeito. Nesse questionário existem perguntas sobre o nível de conhecimento de cada sujeito em: teste de software, linguagem de programação Java, jogos educacionais e avaliação de jogos educacionais;
 - **Material de treinamento:** Material de treinamento para os sujeitos que participaram do estudo de viabilidade. Nesse material existem conteúdos relacionados com teste

de software e jogos educacionais. Além disso, foi utilizado uma nota didática sobre teste de software com ferramentas para auxiliar o teste de software de programação em Java (VINCENZI *et al.*, 2016);

- **Modelo de Avaliação de Jogos Educacionais:** Foi utilizado o modelo de avaliação de jogos educacionais proposto por Savi, Wangenheim e Borgatto (2011). Esse modelo contém 37 perguntas que avaliam três dimensões, sendo elas: a motivação, experiência dos usuários e a aprendizagem.
- Materiais utilizados na segunda etapa do estudo de viabilidade (Avaliação da Usabilidade):
 - **Formulário de Caracterização:** Trata-se de um questionário para caracterizar o nível de conhecimento de cada sujeito. Nesse questionário existem perguntas sobre o nível de conhecimento de cada sujeito em: teste de software, linguagem de programação Java, avaliação de jogos educacionais e avaliação heurística;
 - **Material de treinamento:** Material de treinamento para os sujeitos que participaram do estudo de viabilidade. Nesse material existem conteúdos relacionados com teste de software e jogos educacionais. Além disso, foi utilizado uma nota didática sobre teste de software com ferramentas para auxiliar o teste de software de programação em Java (VINCENZI *et al.*, 2016);
 - **Heurísticas de Usabilidade:** Foi utilizado o conjunto de heurísticas proposto por Nielsen (1994) para avaliação da usabilidade do jogo Testing Game. Esse conjunto contém 10 heurísticas que pode receber diferentes conceitos, que variam de 0 à 4, para os problemas identificados.

4.2.5 Ameaças à Validade

Uma questão fundamental em um estudo experimental é com relação à validade de seus resultados. Sendo assim, a validade dos resultados deve ser considerada na fase de planejamento para que os resultados sejam adequados ao grupo de interesse (WOHLIN *et al.*, 2012). Há quatro tipos de validade de resultados de estudos conforme apresentado a seguir:

- **Validade de Conclusão:** está relacionada com a capacidade de inferir conclusões corretas a respeito do tratamento e o resultado do estudo. Sendo assim, é necessário considerar a confiabilidade das medidas. Os formulários poderiam ser uma ameaça desse tipo, pois eles poderiam está mal elaborados e não permitirem a coleta de informações relevantes. Para resolver esse problema, foram utilizados exemplos de formulários de experimentos já executados com sucesso. Outro fator pode ser a descrição das perguntas do modelo de avaliação de jogos educacionais e do conjunto de heurísticas. No entanto, esses dois métodos de avaliação têm sido muito utilizados na literatura e já estão validados.

- **Validade de Construção:** está relacionado com a relação entre a teoria e a observação de um estudo, ou seja, está relacionada com influência dos tratamentos nos resultados observados. Uma vez que o jogo Testing Game foi desenvolvido neste projeto de mestrado, e a escolha dos métodos de avaliação poderia ser uma ameaça desse tipo, pois poderiam ser escolhidos métodos de avaliação que favorecessem o jogo desenvolvido. Porém, os métodos de avaliação selecionados são muito conhecidos e bem avaliados na literatura com um bom nível de validação.
- **Validade Interna:** está relacionada com as influências que podem afetar a casualidade, das quais o pesquisador não possui conhecimento ou controle. Para isso, é necessário que haja uma maior atenção quanto aos participantes, com relação à seleção da população, divisão de classes, aspectos sociais, entre outros. Um ponto que pode ter influenciado os resultados foi a utilização de alunos de pós-graduação do ICMC/USP como participantes do estudo. No entanto, os estudantes participaram do estudo de forma anônima. Além disso, os participantes não receberam nenhum tipo de recompensa ou favorecimento para não criarem expectativas e se comportassem de forma anormal durante o estudo com o objetivo de obter vantagens.
- **Validade Externa:** está relacionada com a generalização dos resultados obtidos no estudo para outros contextos. Desta forma deve-se observar a interação entre a seleção do sujeito e a interação do ambiente de execução. A experiência dos participantes pode ser considerada como uma ameaça à validade externa. No entanto, para amenizar esse problema foram realizados treinamentos para que os participantes tivessem um conhecimento mínimo para a execução do estudo. Além disso, o estudo foi realizado em um ambiente acadêmico e os resultados podem não ser semelhantes se esse estudo tivesse sido realizado com participantes da indústria.

4.3 Execução do Estudo de Viabilidade

Após a definição do planejamento do estudo de viabilidade, foi realizada a execução do estudo em outubro/2016 no Instituto de Ciências Matemáticas e de Computação (ICMC/USP). Conforme comentado anteriormente, o estudo de viabilidade foi realizado em duas etapas. A primeira etapa teve por objetivo avaliar a qualidade do jogo Testing Game, analisando a capacidade de motivação do jogo, a experiências dos jogadores e a aprendizagem e a segunda etapa teve por objetivo avaliar a usabilidade do jogo. A seguir, apresenta-se a execução das duas etapas do estudo de viabilidade realizado.

4.3.1 Execução da Primeira Etapa - Avaliação da Qualidade do Jogo Testing Game

Nessa etapa, foram convidados 15 estudantes para participar da avaliação de forma voluntária. Grande parte dos estudantes são do sexo masculino (86%) e todos os estudantes são alunos de pós-graduação (8 participantes são alunos de mestrado e 7 participantes são alunos de doutorado). Para participar do estudo de viabilidade, os estudantes assinaram um termo de consentimento que apresenta uma visão geral sobre estudo e a confidencialidade dos dados coletados. Além disso, tem uma seção no documento que discute sobre os benefícios da participação no estudo de viabilidade e sobre a possibilidade que os estudantes tinham em desistir do estudo. Em seguida, os estudantes responderam o formulário de caracterização de perfil para identificar o perfil dos participantes, por exemplo: nível de conhecimento em jogos educacionais, teste de software, entre outros.

A partir do formulário de caracterização de perfil, foi identificado o perfil dos sujeitos. Na Tabela 7 são apresentadas as porcentagens (%) do níveis de conhecimento d teste de software dos sujeitos e jogos educacionais na primeira etapa do estudo de viabilidade.

Tabela 7 – Perfil dos sujeitos que participaram da primeira etapa do estudo de viabilidade

Nível de Conhecimento	Teste de Software (%)	Jogos Educacionais (%)
Sem Conhecimento	0%	6,7%
Básico	26,7%	47,6%
Médio	46,7%	20%
Avançado	26,7%	26,7%

Com relação ao conhecimento na área de jogos educacionais, 47,6% dos estudantes consideram que possuem conhecimento básico, 26,7% avançado, 20% médio e 6,7% disseram que não possuem conhecimento na área de jogos educacionais. Com relação ao conhecimento na área de teste de software, 46,7% dos estudantes consideram que possuem conhecimento médio, 26,7% avançado e 26,7% disseram que possuem conhecimento básico na área de teste de software. Apenas 13,3% dos estudantes cursaram alguma disciplina específica para o ensino de teste de software na graduação. Além disso, apenas 6,7% dos estudantes possuem experiência em teste de software na indústria, especificamente, mais de 3 anos de experiência.

É importante ressaltar que apenas 13,7% dos estudantes já tinham utilizado o modelo de avaliação de jogos educacionais proposto por Savi, Wangenheim e Borgatto (2011). No entanto, grande parte dos estudantes já tinha participado de avaliações de jogos educacionais, especificamente, 20% dos estudantes já participaram mais de 3 vezes, 6,7% participaram 3 vezes, 26,7% participaram 2 vezes, 26,7% participaram apenas 1 vez e apenas 20% nunca participaram de avaliações de jogos educacionais.

Após os estudantes responderem o questionário de caracterização de perfil, foi realizado um treinamento para que eles estivessem aptos a participar do estudo de viabilidade de forma

anônima. No treinamento foram apresentados conteúdos relacionados com o teste de software, especificamente sobre técnicas e critérios de teste. Para a elaboração desse conteúdo foi utilizado uma versão atualizada da nota didática "Introdução ao Teste de Software" (MALDONADO *et al.*, 2004) com ferramentas para auxiliar o teste de software de programação em Java. Essa nota didática apresenta uma estratégia incremental de teste de software assim como o jogo Testing Game. Além disso, foram apresentadas as principais características de jogos educacionais disponibilizando exemplos com outros jogos. Por fim, no treinamento foi apresentado o modelo de avaliação de qualidade jogos educacionais na área de engenharia de software proposto por Savi, Wangenheim e Borgatto (2011), juntamente com um exemplo de avaliação de um jogo educacional que utilizou esse modelo. Assim, os estudantes ficaram aptos a participar do estudo de viabilidade.

Em seguida, os participantes jogaram o jogo Testing Game e utilizaram o modelo de avaliação proposto por Savi, Wangenheim e Borgatto (2011) para avaliar a qualidade do jogo. Por fim, os estudantes responderam um questionário de *feedback* para autorizar a utilização de seus dados no estudo de viabilidade. É importante ressaltar que todos os estudantes autorizaram a utilização dos dados coletados no estudo de viabilidade. Além disso, haviam questões sobre a qualidade do treinamento, estrutura do estudo de viabilidade e críticas e sugestões do experimento. Com relação ao treinamento do experimento, 84,6% dos estudantes afirmaram que o treinamento foi de fácil entendimento e objetivo. Com relação à estrutura do experimento, 38,5% dos estudantes disseram que o experimento não foi cansativo e 23,1% dos estudantes foram indiferentes a essa pergunta do questionário. No entanto, é importante ressaltar que 92,3% dos estudantes disseram que o experimento foi bem estruturado e apenas 6,7% dos estudantes foram indiferentes a essa pergunta. Além disso, foram coletadas informações sobre as críticas e sugestões do jogo que serão discutidas na seção 4.4.3.

4.3.2 Execução da Segunda Etapa - Avaliação Heurística do Jogo Testing Game

Nesta etapa, foram convidados 5 estudantes para participar da avaliação heurística de forma voluntária. Esses estudantes foram selecionados do conjunto de sujeitos que participaram do estudo de viabilidade da primeira etapa. É importante ressaltar que o fato de selecionar avaliadores que já participaram da avaliação da qualidade do jogo na primeira etapa não será um viés para os resultados, uma vez que a avaliação heurística é recomendada para avaliadores especialistas. Essa quantidade de avaliadores é a quantidade máxima recomendada em uma avaliação heurística. Grande parte dos estudantes são do sexo masculino (80%) e todos são estudantes de pós-graduação (3 de mestrado e 2 de doutorado). Para participar da avaliação heurística, os estudantes assinaram um termo de consentimento que apresentava uma visão geral do estudo realizado e da confidencialidade das informações coletadas. No questionário de caracterização de perfil havia também uma seção sobre a possibilidade dos estudantes desistirem

do estudo. Em seguida, os estudantes responderam um questionário de caracterização de perfil semelhante ao questionário de caracterização de perfil da primeira etapa do estudo de viabilidade.

Por meio desse questionário, foram identificados os perfis dos participantes da avaliação heurística. Na Tabela 8 são apresentadas as porcentagens (%) do níveis de conhecimento dos sujeitos em teste de software e jogos educacionais na segunda etapa do estudo de viabilidade.

Tabela 8 – Perfil dos sujeitos que participaram da segunda etapa do estudo de viabilidade

Nível de Conhecimento	Teste de Software (%)	Jogos Educacionais (%)
Básico	60%	0%
Médio	20%	60%
Avançado	20%	40%

Com relação ao nível de conhecimento sobre a área de jogos educacionais, 60% dos participantes têm nível médio e os outros 40% avançado. No entanto, 60% dos participantes já participaram 1 vez de avaliações de jogos educacionais e 40% já participaram 2 vezes. Com relação à avaliação heurística, 80% dos participantes já realizaram alguma avaliação heurística. Com relação ao teste de software, 60% classificaram seu conhecimento em teste de software como básico, 20% médio e 20% como avançado. No entanto, todos os participantes já cursaram uma disciplina específica de teste de software na pós-graduação.

Após os estudantes responderem o questionário de caracterização de perfil, foi realizado um treinamento sobre teste de software, especificamente técnicas e critérios de teste de software, utilizando a mesma nota didática utilizada na primeira etapa do estudo de viabilidade. Além disso, no treinamento havia conteúdos sobre jogos educacionais e suas principais características de boa qualidade e sobre avaliação heurística. No treinamento, os participantes fizeram um exemplo de avaliação heurística em outro jogo educacional.

Em seguida, os participantes jogaram o Testing Game e utilizaram o conjunto de heurísticas proposto por Nielsen (1994) para identificar eventuais problemas de usabilidade. Por fim, os participantes responderam um formulário de *feedback* sobre o treinamento, estrutura do experimento e críticas e sugestões para o jogo. De forma geral, os participantes acharam que o treinamento estava adequado e o experimento estava bem estruturado.

4.4 Análise dos Resultados

Nesta seção é apresentado os resultados obtidos no estudo de viabilidade realizado. Assim como o planejamento e a execução do estudo de viabilidade, os resultados foram divididos em duas etapas, a primeira está relacionada com a avaliação da qualidade do jogo Testing Game e a segunda etapa com a avaliação da usabilidade do jogo desenvolvido.

4.4.1 Resultado da Avaliação da Qualidade do Jogo Testing Game

Para avaliação da qualidade do jogo Testing Game, analisando a capacidade de motivação do jogo, a experiências dos jogadores e a aprendizagem foi utilizado o modelo de avaliação proposto por Savi, Wangenheim e Borgatto (2011) conforme mencionado anteriormente.

Os participantes pontuaram cada afirmação do questionário utilizando a escala *likert* de -2 a +2, ou seja de “Discordo Totalmente” a “Concordo Totalmente”. Quanto maior a quantidade de notas +1 e +2 atribuídas a uma afirmação maior é o nível de concordância dos participantes com a afirmação sobre o jogo. Desta forma, é possível identificar os principais pontos positivos e negativos sob o ponto de vista dos estudantes.

Na Figura 46 são apresentados os resultados relacionados com a **Motivação**. O jogo obteve um resultado positivo em grande parte dos itens avaliados, no qual apresentou 86,69% de pontuação nas afirmações “Concordo Totalmente” e “Concordo Parcialmente”, ou seja, grande parte dos estudantes concordaram com as características avaliadas no subcomponente de **Motivação**.

Figura 46 – Avaliação do Jogo com Relação ao Subcomponente Motivação

Por meio da Figura 46, pode-se observar que na categoria **satisfação**, 80% dos estudantes avaliaram de forma positiva, ou seja, demonstraram satisfação em jogar. No entanto, 20% dos estudantes tiveram dificuldades em avançar no jogo por meio de seus próprios conhecimentos. Com relação à **confiança**, 90% dos estudantes avaliaram o jogo como um recurso educacional de fácil compreensão e que promove a confiança. Dentre os subcomponentes da **Motivação**, a categoria **relevância** obteve melhor o resultado, aproximadamente 91,13% estudantes avaliaram de forma positiva essa característica, ou seja, o jogo está adequado para o aprendizado de conteúdos relacionados com o teste de software.

Por fim, a categoria **atenção** também foi avaliada de forma positiva, 84,5% dos estudantes permaneceram atentos aos desafios propostos no jogo, uma vez que eles afirmaram que o *design* do jogo é muito atraente. Desta forma, percebe-se que o jogo Testing Game obteve uma boa avaliação quanto à motivação dos estudantes. Outro subcomponente avaliado foi a **Experiência dos Usuários**. Os resultado da avaliação desse subcomponente é ilustrado na Figura 47, no qual 87,7% dos estudantes a avaliaram de forma positiva.

Figura 47 – Avaliação do Jogo com Relação ao Subcomponente Experiência do Usuário

Nas afirmações relacionados com a **competência**, 83,35% estudantes avaliaram de forma positiva, uma vez que eles conseguiram atingir o objetivo do jogo por meio de suas próprias habilidades. Com relação à categoria **divertimento**, 88,32% dos estudantes avaliaram de forma positiva. Eles afirmaram que recomendariam o jogo aos seus colegas e que ficaram desapontados quando o jogo acabou. É importante ressaltar que todos os estudantes avaliaram de forma positiva o item “Me diverti com o jogo”, no qual todos os estudantes avaliaram essa afirmação com os conceitos “Concordo Totalmente” e “Concordo Parcialmente”.

Com relação à categoria **desafio**, 93,15% dos estudantes avaliaram de forma positiva. Essa foi a categoria do subcomponente **Experiência dos Usuários** que obteve melhor resultado. Os estudantes afirmaram que o jogo foi desafiador e que os desafios propostos no jogo estão de acordo com o seu nível conhecimento. A categoria **interação social** não foi considerada nessa avaliação, pois o jogo Testing Game é um jogo *single player*. Por fim, a categoria **imersão** também obteve uma boa avaliação, na qual 84,46% dos estudantes avaliaram de forma positiva essa característica. Os estudantes sentiram-se parte do ambiente do jogo e esqueceram temporariamente suas preocupações do dia-a-dia enquanto eles estavam totalmente concentrados no jogo.

Por fim, o subcomponente **Aprendizagem** obteve uma avaliação positiva. Na Figura 48 é possível observar os resultados da avaliação do estudantes com relação ao aprendizado. Aproximadamente 82,23% dos estudantes avaliaram as afirmações desse subcomponente com os conceitos “Concordo Totalmente” e “Concordo Parcialmente”.

Figura 48 – Avaliação do Jogo com Relação ao Subcomponente Aprendizagem

É importante ressaltar que o subcomponente **Aprendizagem** não obteve avaliações negativas. Os estudantes afirmaram que o Testing Game poderá contribuir para suas experiências na vida profissional. Além disso, afirmaram que o jogo contribuiu para aprendizagem deles em teste de software e foi mais eficiente que outras atividades tradicionais já utilizadas para o ensino desse conteúdo.

4.4.2 Resultado da Avaliação Heurística do Jogo Testing Game

A avaliação heurística é fundamentada em um conjunto de critérios para identificar problemas de usabilidade em interfaces de usuários. Esses critérios são utilizados pelos especialistas para verificar visualmente uma interface e analisar se ela está em conformidade com os critérios heurísticos (NIELSEN, 1994).

Desta forma, por meio da avaliação heurística realizada na segunda etapa do estudo de viabilidade, foram identificados 7 problemas de usabilidade no jogo Testing Game. Esses problemas estão listados na Tabela 9, juntamente com seus respectivos graus de severidade, os quais foram sugeridos pelos avaliadores.

Conforme apresentado na Figura 49, a H03 - “Controle do usuário e liberdade: o sistema deve tornar disponíveis funções que possibilitem saídas de funções indesejadas” foi a heurística que auxiliou a identificação da maior quantidade de problemas de usabilidade. Os avaliadores indicaram a necessidade de haver uma opção para sair do jogo. Atualmente, os jogadores só conseguem sair do jogo quando finalizam uma fase. Outro ponto indicado pelos avaliadores é com relação à possibilidade dos jogadores poderem consultar a especificação do programa

Tabela 9 – Lista de problemas encontradas na avaliação heurística do Jogo Testing Game

Heurística	Problemas	Severidade
H01	Não há uma indicação que difere as fases que foram completadas das fases que estão abertas e não foram completadas	2
H02	É difícil identificar que a chave deve ser arrastada à porta	4
H03	Não é possível sair de uma fase	2
H03	Não é possível consultar a especificação do programa e a descrição da atividade no meio das fases	3
H03	Não é possível navegar entre os menus de fases. Não há opções para voltar	3
H06	Não é apresentado ao usuário os seus principais erros e acertos em cada fase	3
H10	Não há uma explicação sobre quais são os comandos do jogo	4

testado no jogo, a saber: BubbleSort, e a descrição das atividades em tempo real. No jogo, essa ação só é possível ser realizada no início de cada subfase.

Figura 49 – Frequência de problemas encontrados por heurística no jogo Testing Game

Por meio da H01 - “*Visibilidade do status do sistema: o usuário deve ser informado pelo sistema em tempo razoável sobre o que está acontecendo*”, os avaliadores indicaram que há necessidade de mostrar para os jogadores quais são as fases do jogo que foram completadas e quais estão disponíveis. No jogo essa ação foi implementada com a ideia de que os cadeados abertos são as fases disponíveis e os cadeados fechados são as fases indisponíveis no momento. Porém, segundo os avaliadores essa ação não é sugestiva. Sendo assim, foi sugerido que os cadeados que representam as fases disponíveis ou completadas estejam com outras cores.

Outro problema de usabilidade identificado foi por meio da H02 - “*Compatibilidade do sistema com o mundo real*: o modelo lógico do sistema deve ser compatível com o modelo lógico do usuário”. Para acessar as fase do jogo, a saber: Teste Funcional, Teste Estrutural e Teste de Mutação, os jogadores precisam pegar a chave que está disponível e arrasta-lá para a porta correspondente à fase do jogo. No entanto, os avaliadores disseram que é difícil identificar que a chave deve ser arrastada à porta. Então, sugeriram que a chave seja alterada para um botão com o nome “Abrir”. No entanto, acredita-se que essa sugestão não seja uma estratégia, pois o botão pode ser menos acessível que a chave.

Por meio da H06 - “*Reconhecimento ao invés de lembrança*: as instruções para o bom funcionamento do sistema devem estar visíveis no contexto em que o usuário se encontra”, os avaliadores indicaram que há necessidade do jogo apresentar ao usuário quais são os seus principais erros e acertos em cada fase. No jogo, é apresentada apenas informações sobre quais são os vetores corretos e incorretos, casos de teste que são corretos e incorretos e outras informações esperadas pelos jogadores. Porém, não é apresentado quais são os principais erros cometidos pelos jogadores.

Por fim, a partir da H10 - “*Ajuda aos usuários no reconhecimento, diagnóstico e correção de erros*: as mensagens devem ser expressas em linguagem clara, indicando as possíveis soluções”, os avaliadores indicaram a necessidade de criar uma explicação sobre quais são os comandos do jogo, mesmo no jogo Testing Game foi utilizado comandos que são padrões de jogos eletrônicos, tais como: pular, correr e atirar.

4.4.3 Discussão dos Resultados

A partir dos resultados obtidos na execução do estudo de viabilidade, foi possível indicar possíveis respostas para as questões de pesquisa proposta no planejamento do estudo apresentado na seção 4.2. A QP₁ está relacionada com a qualidade do jogo Testing Game, com relação à motivação, experiência do usuário e aprendizagem sob o ponto de vista dos estudantes. A partir dos resultados obtidos e apresentados na subseção 4.4.1, pode-se perceber que o jogo Testing Game apresenta uma boa qualidade do ponto de vista dos estudantes.

Aproximadamente 86,9% dos estudantes avaliaram a **Motivação** de forma positiva. Os estudantes afirmaram que sentiram satisfação em jogar e disseram que o jogo é de fácil compreensão e ele está adequado para o aprendizado de teste de software. Além disso, durante a execução do estudo, percebeu-se que os estudantes permaneceram atentos aos desafios propostos no jogo.

A **Experiência dos Usuários** também obteve uma avaliação positiva entre os 87,7% dos estudantes que participaram do estudo. Os estudantes conseguiram atingir o objetivo do jogo com aspectos de diversão. Esses estudantes disseram que jogariam novamente o jogo Testing e recomendaria o jogo entre seus colegas, pois o jogo apresentou uma dinâmica desafiadora o

que contribui para eles se sentissem parte do ambiente, ou seja, se sentissem imersos no jogo. Por fim, 82,33% dos estudantes avaliaram a **Aprendizagem** de forma positiva, uma vez que eles afirmaram que o jogo contribui para a aprendizagem de conteúdos relacionados com o teste de software. De modo geral, a média aritmética dos subcomponentes considerados, a saber: Motivação, Experiência do Usuário e Aprendizagem, indica que 85,64% dos estudantes avaliam a qualidade do jogo de forma positiva.

Outra característica avaliada no estudo de viabilidade foi com relação à usabilidade conforme proposto em QP_2 . Na avaliação realizada, os estudantes identificaram 7 problemas de usabilidade por meio de 5 heurísticas. Do total de 10 heurísticas do conjunto utilizado, 5 não identificaram problemas de usabilidade no jogo Testing Game. No entanto, foram identificados alguns problemas nessa avaliação, tais como: falta de documentação sobre os comandos no jogo, ausência de indicação dos principais erros e acertos, falta de menus para navegação entre as telas, entre outros problemas, conforme apresentado na subseção 4.4.2.

Porém, durante a execução da avaliação do jogo e por meio do questionário de *feedback* foram coletadas sugestões de melhorias para o jogo. No jogo Testing Game quando um jogador perde, ele é redimensionado para a tela inicial. Os estudantes disseram que seria interessante que o jogador voltasse para o início da fase que ele perdeu, uma vez que o objetivo do jogo é a aprendizagem, e desta forma, os estudantes poderiam refazer os desafios de onde pararam até eles conseguirem atingir o objetivo do jogo. Além disso, os estudantes sentiram a necessidade de um manual que explicasse os comandos do jogo, apesar dos comandos do jogo serem os comandos padrões de jogos eletrônicos.

Os estudantes sugeriram ainda que o jogo fosse *multiplayer*, pois assim eles poderiam exercitar a característica de interação social e trocar experiências com os colegas. Por fim, os jogadores indicaram a necessidade de uma opção de fechar a fase em tempo real e voltar para o menu com as fases que estavam disponíveis. Apesar das sugestões feitas pelos estudantes para a evolução do jogo, grande parte deles disseram que se sentiram motivados e se divertiram quando utilizaram o jogo. Além disso, eles afirmaram que o jogo contribuiu para sua aprendizagem, uma vez que eles puderam revisar os conteúdos relacionados com o teste de software.

4.5 Considerações Finais

Neste capítulo, apresentou-se a avaliação do jogo Testing Game analisando sua qualidade e usabilidade. A avaliação do jogo foi realizada por meio de um estudo de viabilidade. Os resultados obtidos no estudo indicam que o jogo Testing Game apresenta uma boa qualidade, com relação à motivação, experiência do usuário e aprendizagem do ponto de vista dos estudantes, uma vez que 85,64% dos estudantes avaliam o jogo de forma positiva. Com relação à usabilidade foram identificados apenas 7 problemas por meio de 5 heurísticas. Esses problemas de usabilidade serão considerados na evolução do jogo. No entanto, esses problemas não afetam os benefícios

proporcionados na utilização do jogo. Desta forma, pode-se dizer que o jogo Testing Game está apto a ser inserido em um ambiente educacional para auxiliar o ensino de teste de software.

CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

5.1 Visão Geral da Pesquisa

O teste de software é reconhecido como uma atividade importante na garantia da qualidade de produtos de software. No entanto, muitos estudantes sentem-se desmotivados à aprender conteúdos relacionados com o teste de software (JIA; YANG, 2013). Para amenizar esses problemas, algumas abordagens têm sido utilizadas para motivar os estudantes (VALLE; BARBOSA; MALDONADO, 2015b). Dentre, essas abordagens destaca-se os jogos educacionais. Desta forma, este projeto de mestrado teve como objetivo o desenvolvimento de um jogo educacional de apoio ao ensino de teste de software. Para auxiliar o desenvolvimento desse jogo educacionais foi utilizado motor de jogo Construct 2.

O jogo Testing Game utiliza a estratégia do teste incremental, considerando o Teste Funcional, Teste Estrutural e o Teste de Mutação. Para avaliar a efetividade do jogo desenvolvido, foi realizado um estudo de viabilidade em duas etapas. A primeira etapa teve por objetivo avaliar a qualidade do jogo Testing com relação à motivação, experiência do usuários e aprendizagem do ponto de vista dos estudantes. A segunda etapa teve por objetivo avaliar a usabilidade do jogo utilizando um conjunto de heurísticas. Os resultados obtidos demonstraram que o Jogo Testing Game apresenta uma boa qualidade e usabilidade sob o ponto de vista dos estudantes. Sendo assim, acredita-se que esse jogo poderá ser utilizado como instrumento de apoio ao teste de software e está apto a ser inserido em um ambiente educacional.

5.2 Contribuições

Dentre as contribuições identificadas com a realização deste projeto de mestrado, pode-se destacar:

- Identificação do estado da arte referente ao ensino de teste de software, especificamente sobre as principais abordagens utilizadas para auxiliar o ensino de teste de software;
- Um panorama sobre como o teste de software tem sido abordado nos cursos de graduação da área de Computação nas universidades do Brasil e do exterior, motivando uma reflexão sobre o ensino integrado de teste de software com outras disciplinas;
- Identificação dos principais motores de jogos disponíveis na literatura. Os resultados obtidos podem ser considerados como um instrumento de apoio à tomada de decisão;
- Desenvolvimento do jogo Testing Game para a plataforma Web abordando o Teste Funcional, Teste Estrutural e Teste de Mutação;
- Motivação da criação de uma “cultura de teste de software” adequada entre os estudantes;
- O aumento do interesse dos estudantes em aprender conteúdos educacionais relacionados com o teste de software;
- Validação do jogo Testing Game, identificando os principais pontos positivos, limitações e pontos de melhoria.

5.3 Trabalhos Futuros

A partir do estudo de viabilidade realizado no Capítulo 4, foram identificados diversos aspectos do jogo Testing Game que podem ser investigados como trabalhos futuros. Dentre esses aspectos, destacam-se:

- Desenvolvimento de um manual de ajuda com os principais comandos do jogo;
- Desenvolvimento de menus no jogo para permitir a navegação entre fases do jogo e o acesso a especificação do programa testado;
- Desenvolvimento de uma funcionalidade que permita os jogadores retornarem para fase atual quando perderem uma partida;
- Disponibilização dos principais erros dos jogadores e orientações (bibliografia, materiais adicionais sobre os tópicos com dificuldade); e
- Resolver problemas de usabilidade identificados;

Além dos aspectos de melhorias identificados no estudo de viabilidade, pretende-se evoluir o jogo Testing Game para contemplar os seguintes aspectos:

- Realizar um estudo de caso para avaliar a aprendizagem dos estudantes;

- Aplicar o jogo Testing Game em uma disciplina;
- Contemplar a característica de *multiplayer*, para que os jogadores possam trocar informações entre si e desenvolver habilidades de interação social;
- Integrar ferramentas de teste de software para que os jogadores possam utilizá-las em tempo real para testar os programas no jogo;
- Combinar características de outros jogos. Por exemplo, no final de cada técnica de teste existir um módulo com o jogo da velha para testar os conhecimentos dos estudantes, semelhante ao jogo JoVeTest.

5.4 Produção Científica

A seguir, apresentam-se as publicações já finalizadas e as publicações que estão em elaboração.

5.4.1 Publicações Elaboradas

Como parte dos resultados obtidos na realização deste projeto de mestrado, destacam-se as seguintes publicações:

- VALLE, PEDRO HENRIQUE DIAS ; BARBOSA, ELLEN FRANCINE ; MALDONADO, JOSÉ CARLOS . **Um Mapeamento Sistemático sobre Ensino de Teste de Software**. In: XXVI Simpósio Brasileiro de Informática na Educação, 2015, Maceió, Brasil, 2015. p. 71-80.
- VALLE, PEDRO HENRIQUE DIAS ; BARBOSA, ELLEN FRANCINE ; MALDONADO, JOSÉ CARLOS . **CS curricula of the most relevant universities in Brazil and abroad: Perspective of software testing education**. In: XVII International Symposium on Computers in Education (SIIE), Setubal, Portugal. IEEE, 2015. p. 62-68.

5.4.2 Publicações em Elaboração

- VALLE, P.H.D; MALDONADO, J.C; Um Mapeamento Sistemático sobre Motores de Jogos: Apoio à tomada de decisão.
- VALLE, P.H.D; MALDONADO, J.C; Testing Game: Um Jogo Educacional Para o Ensino de Teste de Software.
- VALLE, P.H.D; MALDONADO, J.C; Uma Avaliação Experimental do Jogo Testing Game.

REFERÊNCIAS

ACM; IEEE. **Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science**. New York, USA: ACM, 2013. Citado nas páginas 22 e 42.

AMARAL, H.; BRAGA, J. L.; GALVAO, A. Game architecture for teaching-learning process: An application on an undergraduate course. In: **IEEE International Games Innovation Conference (IGIC)**. Vancouver, Canada: IEEE, 2013. p. 1–6. Citado nas páginas 44 e 52.

ANDERSON, L. W.; KRATHWOHL, D. R.; BLOOM, B. S. **A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives**. USA: Allyn & Bacon, 2001. Citado na página 144.

ANNETTA, L. A. The “i’s” have it: A framework for serious educational game design. **Review of General Psychology**, Educational Publishing Foundation, v. 14, n. 2, p. 105 – 112, 2010. Citado nas páginas 13, 45 e 46.

ARIMOTO, M. M.; BARBOSA, E. F. **Agil Development of Open Educational Resources**. Tese (Doutorado) — Universidade de São Paulo. Instituto de Ciências Matemáticas e de Computação, 2016. Citado nas páginas 48, 67 e 93.

ARSENAULT, D. Video game genre, evolution and innovation. **Eludamos. Journal for Computer Game Culture**, v. 3, n. 2, p. 149–176, 2009. Citado na página 52.

ASSOCIATION, E. S. *et al.* **Essential facts about the computer and video game industry**. Retrieved March 14. 2012. Citado na página 73.

BAKKER, M.; HEUVEL-PANHUIZEN, M. van den; ROBITZSCH, A. Effects of playing mathematics computer games on primary school students multiplicative reasoning ability. **Contemporary Educational Psychology**, v. 40, n. 0, p. 55 – 71, 2015. Citado na página 44.

BARBOSA, A. K. T.; NEVES, L. L. E.; NETO, A. C. D. Jovetest - jogo da velha para auxiliar no ensino e estudo de teste de software. In: **IX Fórum de Educação em Engenharia de Software**. Maringá, Brasil: SBC, 2016. p. 65–76. Citado nas páginas 13, 58 e 59.

BARBOSA, D.; ANDRADE, W.; MACHADO, P.; FIGUEIREDO, J. Spaces—uma ferramenta para teste funcional de componentes. In: **XVIII Simpósio Brasileiro de Engenharia de Software (SBES)**. Brasília, Brasil: SBC, 2004. p. 55–60. Citado na página 37.

BARBOSA, E.; MALDONADO, J. E-infrastructures and technologies for lifelong learning: Next generation environments. In: _____. London, UK: Information Science Reference, 2011. cap. Collaborative development of educational modules: a need for lifelong learning. Citado na página 38.

BARBOSA, E. F.; SILVA, M. A.; CORTE, C. K.; MALDONADO, J. C. Integrated teaching of programming foundations and software testing. In: **XXXVIII Frontiers in Education Conference**. Saratoga Springs, USA: IEEE, 2008. p. S1H5–S1H10. Citado nas páginas 23 e 40.

- BARRIOCANAL, E. G.; URBÁN, M.-Á. S.; CUEVAS, I. A.; PÉREZ, P. D. An experience in integrating automated unit testing practices in an introductory programming course. **SIGCSE Bulletin**, ACM, v. 34, n. 4, p. 125–128, 2002. Citado na página 23.
- BASILI, V.; HEIDRICH, J.; LINDVALL, M.; MÜNCH, J.; REGARDIE, M.; ROMBACH, D.; SEAMAN, C.; TRENDOWICZ, A. Gqm+ strategies: A comprehensive methodology for aligning business strategies with software measurement. **Software Metrik Kongress**, 2007. Citado na página 96.
- BATTISTELLA, P. E.; WANGENHEIM, A. von; WANGENHEIM, C. G. von. Sortia-um jogo para ensino de algoritmo de ordenação: estudo de caso na disciplina de estrutura de dados. In: **XXIII Simpósio Brasileiro de Informática na Educação**. Rio de Janeiro, Brasil: SBC, 2012. Citado na página 143.
- BEZERRA, C. I.; COUTINHO, E. F.; SANTOS, I. S.; MONTEIRO, J. M.; ANDRADE, R. M. Evolução do jogo itest learning para o ensino de testes de software: Do planejamento ao projeto. In: **XIX Conferência Internacional sobre Informática na Educação (TISE)**. Fortaleza, Brasil: Nuevas Ideas En Informática Educativa, 2014. Citado nas páginas 23, 56, 57 e 143.
- BITTENCOURT, J. R. Promovendo a ludicidade através de jogos livres. In: **XVI Simpósio Brasileiro de Informática na Educação**. Juiz de Fora, Brasil: SBC, 2005. p. 43–63. Citado na página 52.
- CAGILTAY, N. E.; OZCELIK, E.; OZCELIK, N. S. The effect of competition on learning in games. **Computers Education**, v. 87, p. 35 – 41, 2015. Citado nas páginas 45 e 95.
- CASSEL, L.; CLEMENTS, A.; DAVIES, G.; GUZDIAL, M.; MCCAULEY, R.; MCGETTRICK, A.; SLOAN, B.; SNYDER, L.; TYMANN, P.; WEIDE, B. W. Computer science curriculum 2008: An interim revision of cs 2001. In: **ATechnical Report**. New York, USA: ACM, 2008. Citado na página 40.
- CLARKE, P. J.; ALLEN, A. A.; KING, T. M.; JONES, E. L.; NATESAN, P. Using a web-based repository to integrate testing tools into programming courses. In: **X ACM international conference companion on Object oriented programming systems languages and applications companion**. Amsterdam, Netherlands: ACM, 2010. p. 193–200. Citado na página 23.
- CLARKE, P. J.; PAVA, J.; WU, Y.; KING, T. M. Collaborative web-based learning of testing tools in se courses. In: **XLII Technical symposium on Computer science education**. Dallas, USA: ACM, 2011. Citado na página 39.
- COLLOFELLO, J.; VEHATHIRI, K. An environment for training computer science students on software testing. In: **XXXV Annual Conference Frontiers in Education**. Indianapolis, USA: IEEE, 2005. p. T3E–6. Citado na página 23.
- CONNOLLY, T. M.; STANSFIELD, M.; HAINEY, T. An application of games-based learning within software engineering. **British Journal of Educational Technology**, Wiley Online Library, v. 38, n. 3, p. 416–428, 2007. Citado na página 95.
- CORRÊA, A. G. D.; MARTINAZZO, A. A. G.; BIAZON, L. C.; ARCHANJO, M.; VENÂNCIO, V.; FICHEMAN, I. K.; LOPES, R. de D. Jogos educacionais para tv digital interativa. **Revista Trilha Digital**, v. 1, n. 1, 2013. Citado nas páginas 52 e 53.

CORTE, C. K. D.; MALDONADO, J. C. **Ensino integrado de fundamentos de programação e teste de software**. Dissertação (Mestrado) — Universidade de São Paulo. Instituto de Ciências Matemáticas e de Computação, 2006. Citado na página 23.

COSTA, S.; CUZZOCREA, F.; NUZZACI, A. Use of the internet in educative informal contexts. implication for formal education. **Comunicar**, v. 22, n. 43, p. 163–171, 2014. Citado na página 44.

COUTINHO, I.; RODRIGUES, P.; CARNEIRO, Y.; GUIMARÃES, J.; LIMA, L.; QUINTO, C.; ALVES, L. Jogos eletrônicos e tecnologia assistiva. In: **IX Seminário Jogos Eletrônicos Educação Comunicação**. Salvador, Brasil: Universidade do Estado da Bahia - UNEB, 2013. p. 163–171. Citado na página 73.

CSIKSZENTMIHALYI, M. **Beyond boredom and anxiety: The experience of play in work and games**. San Francisco, USA: Jossey-Bass, 1975. Citado na página 46.

CUPERSCHMID, A. R. M.; HILDEBRAND, H. R. **Heurísticas de Jogabilidade: usabilidade e entretenimento em jogos digitais**. Campinas, Brasil: Marketing Aumentado, 2013. Citado na página 46.

DELAMARO, M.; MALDONADO, J.; JINO, M. **Introdução ao teste de software**. 2 edição. ed. Rio de Janeiro: Elsevier, 2016. Citado nas páginas 13, 21, 22, 27, 28, 29, 30, 32, 33, 34, 35, 36, 61, 69 e 70.

DINIZ, L. L.; DAZZI, R. L. Jogo para o apoio ao ensino do teste de caixa-preta. In: **XXII Simpósio Brasileiro de Informática na Educação**. Aracaju, Brasil: SBC, 2011. p. 426–435. Citado nas páginas 54 e 55.

DINIZ, L. L.; DAZZI, R. L. S. Jogo das sete falhas: Um jogo educacional para apoio ao ensino do teste caixa preta. In: **Computer on the Beach**. Florianópolis: Universidade do Vale do Itajaí, 2011. p. 1–10. Citado nas páginas 13, 23, 54 e 55.

_____. Jogo digital para o apoio ao ensino do teste de caixa-preta. In: **X Simpósio Brasileiro de Qualidade de Software (SBQS)**. Curitiba, Brasil: SBC, 2011. p. 1–15. Citado na página 55.

DVORNIK, T.; JANZEN, D. S.; CLEMENTS, J.; DEKHTYAR, O. Supporting introductory test-driven labs with webide. In: **XIV Conference on Software Engineering Education and Training (CSEE&T)**. Honolulu, USA: IEEE, 2011. p. 51–60. Citado na página 23.

EDWARDS, S. H. Improving student performance by evaluating how well students test their own programs. **Journal on Educational Resources in Computing (JERIC)**, ACM, v. 3, n. 3, p. 1, 2003. Citado nas páginas 23 e 39.

_____. Using software testing to move students from trial-and-error to reflection-in-action. **ACM SIGCSE Bulletin**, ACM, Norfolk, Virginia, USA, v. 36, n. 1, p. 26–30, 2004. Citado na página 23.

EDWARDS, S. H.; SHAMS, Z. Do student programmers all tend to write the same software tests? In: **XXXVIII Conference on Innovation and Technology in Computer Science Education**. Uppsala, Sweden: ACM, 2014. p. 171–176. Citado nas páginas 24 e 37.

- ESCUDEIRO, P.; ESCUDEIRO, N. Evaluation of serious games in mobile platforms with qef: Qef (quantitative evaluation framework). In: **VII International Conference on Wireless, Mobile and Ubiquitous Technology in Education (WMUTE)**. Takamatsu, Japan: IEEE, 2012. p. 268–271. Citado nas páginas 45 e 95.
- FARIAS, V.; MOREIRA, C.; COUTINHO, E.; SANTOS, I. S. itest learning: Um jogo para o ensino do planejamento de testes de software. In: **V Fórum de Educação em Engenharia de Software**. Natal: XXVI Simpósio Brasileiro de Engenharia de Software, 2012. Citado nas páginas 13, 38, 56 e 57.
- FEBBRARO, O.; LEONE, N.; REALE, K.; RICCA, F. Unit testing in aspide. In: **Applications of Declarative Programming and Knowledge Management**. Vienna, Austria: Springer, 2013. p. 345–364. Citado na página 29.
- FERNANDES, J. C. L. Educação digital: Utilização dos jogos de computador como ferramenta de auxílio à aprendizagem. **FaSci-Tech**, v. 1, n. 3, 2012. Citado na página 52.
- FIGUERÊDO, C. d. O.; SANTOS, S. C. dos; BORBA, P. H.; ALEXANDRE, G. H. Using pbl to develop software test engineers. In: **XIV International Conference on Computers and Advanced Technology in Education**. Cambridge, United Kingdom: IASTED Conferences, 2011. p. 1–7. Citado na página 40.
- FLOOD, K. **Game Unified Process (GUP)**. 2003. GameDev.net. Acessado em setembro de 2016. Citado na página 48.
- FLORES, E.; TOBON, G.; CAVALLARO, E.; CAVALLARO, F. I.; PERRY, J. C.; KELLER, T. Improving patient motivation in game development for motor deficit rehabilitation. In: **International Conference on Advances in Computer Entertainment Technology**. Yokohama, Japan: ACM, 2008. Citado na página 73.
- FRASER, G.; ARCURI, A. Whole test suite generation. **IEEE Transactions on Software Engineering**, IEEE, v. 39, n. 2, p. 276–291, 2013. Citado na página 27.
- FREITAS, S. D.; MAHARG, P. **Digital games and learning**. New York: Bloomsbury Publishing, 2011. Citado na página 23.
- FRYDENBERG, M. Game development as a pathway to information technology literacy. In: **Proceedings of the EDSIG Conference**. Wilmington, USA: ISEDJ, 2015. p. 1–15. Citado na página 76.
- GEE, J. P. **Good video games and good learning**. New York, USA: Peter Lang, 2005. Citado na página 45.
- GODOY, A.; BARBOSA, E. F. Game-scrum: An approach to agile game development. In: **IX Simpósio Brasileiro de Jogos e Entretenimento Digital**. Florianópolis, Brasil: SBC, 2010. p. 292–295. Citado na página 48.
- HAYS, R. T. **The effectiveness of instructional games: A literature review and discussion**. Orlando, USA, 2005. Citado nas páginas 45 e 95.
- HSU, Y.-J.; LIN, C.-H.; SHIH, J.-L. Developing multi-player digital adventure education game with motion sensing technologies. In: **XIII International Conference on Advanced Learning Technologies (ICALT)**. Beijing, China: IEEE, 2013. p. 207–209. Citado na página 44.

ISO/IEC. **ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models**. [S.l.], 2010. Citado na página 145.

JIA, S.; YANG, C. Teaching software testing based on cdio. **World Transactions on Engineering and Technology Education**, v. 11, n. 4, 2013. Citado nas páginas 36 e 111.

JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. **IEEE Transactions on Software Engineering**, v. 37, n. 5, p. 649–678, Sept 2011. Citado na página 36.

JUNIOR, E. M. B.; SILVEIRA, D. S.; CRUZ, M. L. P. M.; WANDERLEY, F. A. A method for generation of tests instances of models from business rules expressed in ocl. **Latin America Transactions**, IEEE, v. 10, n. 5, p. 2105–2111, 2012. Citado nas páginas 30, 31 e 32.

KELLER, J. M. Development and use of the arcs model of instructional design. **Journal of instructional development**, v. 10, n. 3, p. 2–10, 1987. Citado na página 144.

KHAN, M.; SADIQ, M. Analysis of black box software testing techniques: A case study. In: **International Conference and Workshop on Current Trends in Information Technology (CTIT)**. Dubai, United Arab Emirates: IEEE, 2011. p. 1–5. Citado nas páginas 30 e 31.

LEBLANC, R. J.; SOBEL, A.; DIAZ-HERRERA, J. L.; HILBURN, T. B. *et al.* **Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering**. 2006. Citado nas páginas 22, 40 e 42.

LIMA, P. A. L.; AMORIM, D. G. Jogos que auxiliam no tratamento de enfermidades: Serious games. **Revista Opara**, v. 4, n. 1, p. 57–70, 2014. Citado na página 73.

LIU, H.; KUO, F.-C.; CHEN, T. Y. Teaching an end-user testing methodology. In: **XXIII Software Engineering Education and Training**. Pittsburgh, USA: IEEE, 2010. p. 1–8. Citado na página 39.

LLING MICHAEL E MCKAY, F. K. . o. Avaliação heurística para a programação iniciante sistemas. **Trans. Comput. Educ.**, v. 16, n. 3, jun. 2016. Citado na página 145.

LOCHAB, P.; SINGHAL, A.; BANSAL, A. Generation of mutation operators for aspect-oriented software systems. In: **V International Conference Confluence The Next Generation Information Technology Summit**. Noida, India: IEEE, 2014. p. 748–752. Citado na página 36.

LOPES, A. C.; MARQUES, A. B.; CONTE, T. Avaliação do jogo inspssoft: Um jogo para o ensino de inspeção de software. **XII Simpósio Brasileiro de Qualidade de Software**, 2013. Citado na página 143.

MA, Y.-S.; OFFUTT, J.; KWON, Y. R. Mujava: an automated class mutation system. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 15, n. 2, p. 97–133, 2005. Citado na página 37.

MACHADO, L. S.; MORAES, R. M.; NUNES, F. L. Serious games para saúde e treinamento imersivo. **Abordagens Práticas de Realidade Virtual e Aumentada**, SBC Porto Alegre, Brazil, v. 1, p. 31–60, 2009. Citado na página 43.

- MALDONADO, J. C.; BARBOSA, E. F.; VINCENZI, A. M. R.; DELAMARO, M. E.; SOUZA, S.; JINO, M. Introdução ao teste de software. **Notas Didáticas do Instituto de Ciências Matemáticas e de Computação (ICMC)**, 2004. Citado na página 101.
- MANZO, V.; MANZO, D. Interactive music systems within multimedia game development environments. **Journal of Technology in Music Learning**, v. 5, n. 2, 2015. Citado na página 76.
- MAO, C. Towards a question-driven teaching method for software testing course. In: **International Conference on Computer Science and Software Engineering**. Wuhan, China: IEEE, 2008. p. 645–648. Citado nas páginas 22 e 40.
- MATTAR, J. **Games em educação: como os nativos digitais aprendem**. São Paulo: Person Prentice Hall, 2010. Citado na página 45.
- MEDEIROS, M. d. O.; SCHIMIGUEL, J. Uma abordagem para avaliação de jogos educativos: Ênfase no ensino fundamental. In: **XXIII Simpósio Brasileiro de Informática na Educação**. Rio de Janeiro, Brasil: SBC, 2012. p. 1–5. Citado na página 45.
- MESQUITA, L.; MONTEIRO, M. A. A.; SENA, G. J. de; NINOMIYA, M. P.; COSTA, C. A. da. Xxxviii education for energy efficiency through an educational game. In: **Frontiers in Education Conference (FIE)**. Oklahoma City, USA: IEEE, 2013. p. 535–540. Citado na página 143.
- MESZAROS, G. **xUnit test patterns: Refactoring test code**. Boston, USA: Pearson Education, 2007. Citado nas páginas 29 e 30.
- MOHAMED, H.; JAAFAR, A. Analyzing critical usability problems in educational computer game (usaecg). In: **VII International Conference on Human-Computer Interaction (IAS-TED), Baltimore**. Baltimore, USA: IEEE, 2012. p. 162–168. Citado nas páginas 145 e 146.
- MOODY, D. L.; SINDRE, G. Evaluating the effectiveness of learning interventions: an information systems case study. **European Conference on Information Systems (ECIS)**, p. 1–18, 2003. Citado na página 144.
- MOREIRA, C.; COUTINHO, E. Avaliação do jogo itestlearning: Um jogo para o ensino de planejamento de testes de software. In: **XXI Workshop sobre Educação em Computação (WEI)**. Maceió, Brasil: SBC, 2013. p. 611,620. Citado na página 57.
- MUSTAKEROV, I.; BORISSOVA, D. A conceptual approach for development of educational web-based e-testing system. **Expert Systems with Applications**, Elsevier, 2005. Citado na página 38.
- MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. New Jersey, USA: John Wiley & Sons, 2011. Citado nas páginas 32, 33, 34 e 35.
- NETO, J. F. B.; FONSECA, F. d. S. da. Jogos educativos em dispositivos móveis como auxílio ao ensino da matemática. **RENOTE**, v. 11, n. 1, 2013. Citado nas páginas 23 e 45.
- NIELSEN, J. Heuristic evaluation. **Usability inspection methods**, v. 17, n. 1, p. 25–62, 1994. Citado nas páginas 15, 25, 64, 70, 95, 98, 102, 105, 138, 142, 143, 145, 146 e 147.

- OLIVEIRA, B. C. de. Testeg - um software educacional para o ensino de teste de software. In: **Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras**. Lavras, Brasil: UFLA, 2013. p. 1–92. Citado nas páginas 23 e 56.
- PATTERSON, A.; KÖLLING, M.; ROSENBERG, J. Introducing unit testing with bluej. **ACM SIGCSE Bulletin**, ACM, v. 35, n. 3, p. 11–15, 2003. Citado na página 23.
- PETRI, G.; WANGENHEIM, C. G. von. How to evaluate educational games: a systematic. **Journal of Universal Computer Science**, v. 22, n. 7, p. 992–1021, 2016. Citado na página 95.
- PINELLE, D.; WONG, N.; STACH, T. Heuristic evaluation for games: Usability principles for video game design. In: **Conference on Human Factors in Computing Systems (SIGCHI)**. Florence, Italy: ACM, 2008. p. 1453–1462. Citado na página 145.
- PRENSKY, M. Digital natives, digital immigrants. **On the horizon**, MCB UP Ltd, v. 9, n. 5, 2001. Citado na página 44.
- _____. Digital game-based learning. **Computers in Entertainment (CIE)**, ACM, v. 1, n. 1, 2003. Citado na página 44.
- PRENSKY, M. R. **From digital natives to digital wisdom: Hopeful essays for 21st century learning**. California, USA: Corwin Press, 2012. Citado na página 43.
- PRESSMAN, R. S. **Engenharia de software**. 7. ed. São Paulo, Brasil: McGraw Hill Brasil, 2011. Citado nas páginas 21, 28, 30 e 33.
- RADATZ, J.; GERACI, A.; KATKI, F. Ieee standard glossary of software engineering terminology. **IEEE Std**, New York, USA, 1990. Citado na página 28.
- RAMAKRISHNAN, S. Visualizing oo testing in virtual communities-distributed teaching and learning. In: IEEE. **XXX Technology of Object-Oriented Languages and Systems**. California, USA, 1999. p. 300–310. Citado na página 23.
- RANKIN, J. R.; VARGAS, S. S. *et al.* Survey of development tools for low-end serious games. Australia: ICITA,, 2008. Citado na página 74.
- RIBEIRO, T. P. B.; PAIVA, A. C. R. ilearntest: Educational game for learning software testing. In: **X Iberian Conference on Information Systems and Technologies (CISTI)**. Aveiro, Portugal: IEEE, 2015. p. 1–6. Citado nas páginas 13, 57, 58 e 76.
- ROCHA, R.; VALLE, P.; MALDONADO, J.; BITTENCOURT, I.; ISOTANI, S. Metodologia de desenvolvimento de recursos educacionais: jogos, simulações e ambiente virtuais com objetivos educacionais e de avaliação. In: **Relatório Técnico Interno – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo**. São Carlos, Brasil: ICMC/USP, 2016. p. v.1. Citado nas páginas 13, 25, 48, 49, 50, 51, 52, 63, 67, 68 e 69.
- ROCHA, R. V. d.; ARAUJO, R. B. d. **Metodologia Interativa e Modelos Integradores para Desenvolvimento de Jogos Sérios de Treinamento e Avaliação de Desempenho Humano**. Tese (Doutorado) — Universidade Federal de São Carlos. Departamento de Computação, 2014. Citado nas páginas 48, 67 e 93.

- SAMPAIO, A.; ALBUQUERQUE, C.; VASCONCELOS, J.; CRUZ, L.; FIGUEIREDO, L.; CAVALCANTE, S. Software test program: a software residency experience. In: **XXVII International Conference on Software Engineering**. St. Louis, USA: ACM, 2005. p. 1–2. Citado na página 40.
- SANTOS, R.; GÓES, V.; ALMEIDA, L. Metodologia origame: um processo de desenvolvimento de jogos. In: **XI Simpósio Brasileiro de Jogos e Entretenimento Digital**. Brasília, Brasil: SBC, 2012. p. 125–131. Citado na página 48.
- SAVI, R.; WANGENHEIM, C.; BORGATTO, A. Um modelo de avaliação de jogos educacionais na engenharia de software. In: **XXV Simpósio Brasileiro de Engenharia de Software (SBES)**. São Paulo: SBC, 2011. p. 1–10. Citado nas páginas 14, 25, 56, 64, 70, 95, 98, 100, 101, 103, 130, 143 e 144.
- SBC. **Currículo de Referência para Cursos de Licenciatura em Computação**. 2002. Citado na página 42.
- SBC. **Currículo de Referência para Cursos de Graduação em Bacharelado em Sistemas de Informação**. 2003. Citado na página 42.
- SBC. **Currículo de Referência para Cursos de Graduação em Bacharelado em Ciência da Computação e Engenharia de Computação**. 2005. Citado nas páginas 22, 40 e 42.
- SHULL, F.; CARVER, J.; TRAVASSOS, G. H. An empirical methodology for introducing software processes. In: **VIII European Software Engineering Conference Held Jointly and IX International Symposium on Foundations of Software Engineering (ACM SIGSOFT)**. New York, USA: ACM, 2001. p. 288–296. Citado nas páginas 25, 64, 87 e 95.
- SILVA, A. C.; THIRY, M. **Jogo educacional para apoiar o ensino de técnicas para elaboração de testes de unidade**. Dissertação (Mestrado) — Universidade do Vale do Itajaí, 2010. Citado nas páginas 13, 54 e 55.
- SILVA, R. A.; GOMES, E. W. C.; MATOS, S. N. Plano de teste para validação do subframework de análise semântica de fórmulas. In: **IX International Conference on Information Systems and Technology Management (CONTECSI)**. USP: São Paulo, Brasil, 2012. p. 4182–4208. Citado na página 22.
- SILVA, T. G. da; MULLER, F. M. **Jogos Sérios em Mundos Virtuais: uma abordagem para o ensino-aprendizagem de teste de Software**. Dissertação (Mestrado) — Universidade Federal de Santa Maria, 2012. Citado nas páginas 13, 23, 55 e 56.
- SMITH, J.; TESSLER, J.; KRAMER, E.; LIN, C. Using peer review to teach software testing. In: **IX annual international conference on International computing education research**. Melbourne, Australia: ACM, 2012. p. 93–98. Citado nas páginas 23, 37, 38 e 54.
- SOLINGEN, R. van; BASILI, V.; CALDIERA, G.; ROMBACH, H. D. **Goal Question Metric (GQM) Approach**. Maidenhead, England: John Wiley Sons, Inc., 2002. Citado na página 96.
- SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011. Citado nas páginas 21, 27, 28, 30 e 32.
- SOUZA, D. M. D.; ISOTANI, S.; BARBOSA, E. F. Teaching novice programmers using progttest. **International Journal of Knowledge and Learning**, Inderscience Publishers (IEL), v. 10, n. 1, p. 60–77, 2015. Citado nas páginas 24 e 37.

SOUZA, D. M. de; COSTA, S. L. da; FILHO, N. F. D.; BARBOSA, E. F. Um estudo experimental do ambiente progest no ensino de programação. In: **X Workshop Latinoamericano Ingeniería de Software Experimental (ESELAW)**. Montivideo, Uruguay: CIbSE, 2013. p. 75–88. Citado nas páginas 23 e 61.

SOUZA D. M.; MALDONADO, J. C. de; BARBOSA, E. F. Aspectos de desenvolvimento e evolução de um ambiente de apoio ao ensino de programação e teste de software. In: **XXIII Simpósio Brasileiro de Informática na Educação**. Rio de Janeiro, Brasil: SBC, 2012. p. 1–10. Citado nas páginas 22, 23, 37, 38, 54 e 69.

SOUZA, É. F. de; FALBO, R. de A.; VIJAYKUMAR, N. L. Knowledge management initiatives in software testing: A mapping study. **Information and Software Technology**, v. 57, n. 0, p. 378 – 391, 2015. ISSN 0950-5849. Citado nas páginas 29 e 30.

SREEDHAR, G. **Design Solutions for Improving Website Quality and Effectiveness**. Andhra Pradesh, India: IGI Global, 2016. Citado na página 145.

TANG, Z.; JOHNSON, T. R.; TINDALL, R. D.; ZHANG, J. Applying heuristic evaluation to improve the usability of a telemedicine system. **Telemedicine Journal & E-Health**, Mary Ann Liebert, v. 12, n. 1, p. 24–34, 2006. Citado na página 145.

TAROUCO, L. M. R.; ROLAND, L. C.; FABRE, M.; KONRATH, M. L. P. *et al.* Jogos educacionais. **CINTED, UFRGS**, 2004. Citado na página 52.

THIRY, M.; ZOUCAS, A.; SILVA, A. C. da. Empirical study upon software testing learning with support from educational game. In: **XXIII International Conference on Software Engineering and Knowledge Engineering (SEKE)**. Miami Beach, USA: IEEE, 2011. p. 481–484. Citado nas páginas 23 e 54.

TOPI, H.; VALACICH, J. S.; WRIGHT, R. T.; KAISER, K.; JR, J. F. N.; SIPIOR, J. C.; VREEDE, G.-J. de. Is 2010: Curriculum guidelines for undergraduate degree programs in information systems. **Communications of the Association for Information Systems**, 2010. Citado nas páginas 22, 40 e 42.

TULLIS, T.; ALBERT, W. **Measuring the user experience: collecting, analyzing, and presenting usability metrics**. Waltham, USA: Elsevier, 2013. Citado na página 144.

VALLE, P. H. D.; BARBOSA, E. F.; MALDONADO, J. C. Cs curricula of the most relevant universities in brazil and abroad: Perspective of software testing education. In: **XVII International Symposium on Computers in Education (SIIE)**. Setúbal, Portugal: IEEE, 2015. p. 62–68. Citado nas páginas 15, 22, 25, 37, 40, 41, 42, 43, 61, 63, 64, 65 e 69.

_____. Um mapeamento sistemático sobre ensino de teste de software. In: **XXVI Simpósio Brasileiro de Informática na Educação**. Maceió, Brasil: SBC, 2015. p. 71 – 80. Citado nas páginas 13, 23, 24, 37, 38, 39, 40, 54, 59, 61, 62, 64, 65, 69, 92 e 111.

VALLE, P. H. D.; VILELA, R. F.; JÚNIOR, P. A. P.; INOCÊNCIO, A. C. G. Hedeg-heurísticas para avaliação de jogos educacionais digitais. In: **XVIII Conferência Internacional sobre Informática na Educação (TISE)**. Porto Alegre, Brasil: Nuevas Ideas En Informática Educativa, 2013. Citado nas páginas 45 e 47.

VEENENDAAL, E. van; CANNEGIETER, J. J. Test maturity model integration (tmmi)version 3.1. **TMMi Foundation**, Ireland, 2012. Citado na página 22.

- VINCENZI, A. M. R.; MALDONADO, J. C.; WONG, W. E.; DELAMARO, M. E. Coverage testing of java programs and components. **Science of Computer Programming**, Elsevier, Amsterdam, Netherlands, v. 56, n. 1-2, p. 211–230, abr. 2005. ISSN 0167-6423. Citado na página 37.
- VINCENZI, A. M. R.; VALLE, P. H. D.; BARBOSA, J. R.; LANA, C. A.; DELAMARO, M. E.; MALDONADO, J. C. Introdução ao teste de software com ferramentas para java. In: **Relatório Técnico Interno – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo**. São Carlos, Brasil: ICMC/USP, 2016. p. v.2. Citado nas páginas 70 e 98.
- WANG, M.; JIA, H.; SUGUMARAN, V.; RAN, W.; LIAO, J. A web-based learning system for software test professionals. **IEEE Transactions on Education**, 2011. Citado na página 40.
- WAZLAWICK, R. S. **Engenharia de Software: conceitos e práticas**. Rio de Janeiro: Elsevier, 2013. Citado nas páginas 28, 30 e 31.
- WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012. Citado na página 98.
- WONG, W.; BERTOLINO, A.; DEBROY, V.; MATHUR, A.; OFFUTT, J.; VOUK, M. Teaching software testing: Experiences, lessons learned and the path forward. In: **XXIV Conference on Software Engineering Education and Training (CSEE&T)**. Honolulu, USA: IEEE, 2011. p. 530–534. Citado nas páginas 37 e 38.
- WU, H. An effective equivalence partitioning method to design the test case of the web application. In: **International Conference on Systems and Informatics (ICSAI)**. Yantai, China: IEEE, 2012. p. 2524–2527. Citado na página 31.
- YANG, Q.; JIAO, C. The design innovation of chinese electronic game based on emotional experience. In: **International Conference on Information Technology, Computer Engineering and Management Sciences**. Nanjing, China: IEEE, 2011. p. 359–362. Citado na página 73.
- ZAMBIASI, S. P.; PINHEIRO, P. L. B. Os jogos de computador ea experiência interativa: Do espaço virtual ao real. In: **Computer on the Beach**. Florianópolis, Brasil: Universidade do Vale do Itajaí, 2010. p. 1–5. Citado na página 43.
- ZARRAONANDIA, T.; DIAZ, P.; RUIZ, M.; AEDO, I. Designing educational games by combining other game designs. In: **XII International Conference on Advanced Learning Technologies (ICALT)**. Rome, Italy: IEEE, 2012. p. 218–222. Citado na página 45.

PACOTE DO EXPERIMENTO DE AVALIAÇÃO DA USABILIDADE E QUALIDADE DO JOGO TESTING GAME

A.1 Considerações Iniciais

Neste Apêndice é apresentado os materiais utilizados no estudo de viabilidade realizado para avaliar a qualidade e usabilidade do jogo Testing Game. Na seção A.2, apresentam-se os materiais que são utilizados pelas duas etapas (Termo de Consentimento e Formulário de *Feedback*). Na seção A.3, apresentam-se os materiais utilizados apenas na primeira etapa do estudo de viabilidade (Formulário de Caracterização de Perfil e Modelo de Avaliação da Qualidade de Jogos Educacionais). Por fim, na Seção A.4, apresentam-se os materiais utilizados apenas na segunda etapa do estudo de viabilidade (Caracterização de Perfil e Modelo, Conjunto de Heurística e Relatório de Defeitos).

A.2 Materiais utilizados nas duas etapas

A.2.1 *Termo de Consentimento*

Você está sendo convidado para participar de um estudo de viabilidade do jogo Testing Game. Sua participação não é obrigatória e a qualquer momento você poderá desistir de participar e retirar seu consentimento.

O objetivo desse estudo é avaliar a qualidade do jogo Testing Game, que é um jogo para auxiliar o ensino de teste de software, desenvolvido no projeto de mestrado do aluno de mestrado Pedro Henrique Dias Valle.

A.2.1.0.1 Confidencialidade

Toda informação coletada neste estudo é confidencial, e seu nome não será identificado em momento algum. Da mesma forma, você se compromete a não comunicar os seus resultados enquanto não terminar o estudo, bem como manter sigilo dos documentos apresentados e que fazem parte do estudo.

A.2.1.0.2 Benefícios e Liberdade de Desistência

Eu _____ entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e ensinado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada à minha pessoa não seja incluída no estudo. Eu participo desse estudo de livre e espontânea vontade com o único intuito de contribuir para o avanço da pesquisa na área de Engenharia de Software.

Declaro que entendi os objetivos, riscos e benefícios da minha participação na pesquisa e concordo em participar.

Nome do Participante

Data: ____ / ____ / ____

A.2.2 Formulário de Feedback

Este questionário tem por objetivo obter o *feedback* dos participantes do estudo de viabilidade realizado. Neste questionário os participantes do estudo podem permitir a utilização dos dados obtidos no estudo de viabilidade. Além disso, o questionário visa obter sugestões e críticas do experimento executado.

- Nome:

- Você autoriza utilizar os dados e informações fornecidas por você no estudo realizado ?
Os dados serão utilizados de forma anônima.
 - () Autorizo
 - () Não Autorizo

- O treinamento do experimento foi de fácil entendimento.
 - () Discordo totalmente

- ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente
- O treinamento do experimento foi objetivo.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente
- O experimento foi cansativo.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente
- O experimento está bem estruturado.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente
- Na sua opinião, quais são as críticas ao experimento executado?
- Você tem alguma sugestão para o experimento executado?
- Você tem alguma sugestões para o jogo? Se sim, quais?

A.3 Materiais utilizados na primeira etapa do estudo de viabilidade (Avaliação da Qualidade)

A.3.1 *Formulário de Caracterização de Perfil*

Este formulário tem como objetivo verificar o nível de conhecimento dos participantes do estudo de viabilidade. Neste formulário há perguntas sobre o nível de conhecimento de cada sujeito em: teste de software, linguagem de programação Java, jogos educacionais e avaliação de jogos educacionais.

- Nome:

- Sexo:
 - () Masculino
 - () Feminino

- Qual é o seu nível de escolaridade?
 - () Graduação
 - () Mestrado Incompleto
 - () Mestrado Completo
 - () Doutorado Incompleto
 - () Doutorado Completo

- Com que frequência você joga?
 - () Nunca
 - () Raramente
 - () Às vezes
 - () Muito
 - () Sempre

- Com relação ao seu conhecimento sobre jogos educacionais como você se considera?
 - () Não tenho conhecimento
 - () Básico
 - () Médio

- ☐ Avançado
 - ☐ Expert
- Você já participou outras vezes de avaliações jogos educacionais? Se sim, quantas vezes?
 - ☐ Nunca participei
 - ☐ 1 vez
 - ☐ 2 vezes
 - ☐ 3 vezes
 - ☐ Mais que 3 vezes
- Você já utilizou o modelo de avaliação de jogos educacionais proposto por Savi?
 - ☐ Sim
 - ☐ Não
- Você já jogou algum jogo de apoio ao ensino de teste de software?
 - ☐ Sim
 - ☐ Não
- Com relação ao seu conhecimento sobre a linguagem de programação Java como você se considera?
 - ☐ Não tenho conhecimento
 - ☐ Básico
 - ☐ Médio
 - ☐ Avançado
 - ☐ Expert
- Você já cursou alguma disciplina na graduação sobre teste de software?
 - ☐ Sim
 - ☐ Não
- Você já cursou alguma disciplina na pós-graduação sobre teste de software?
 - ☐ Sim
 - ☐ Não

- Com relação ao seu conhecimento sobre teste de software como você se considera?
 - () Não tenho conhecimento
 - () Básico
 - () Médio
 - () Avançado
 - () Expert

- Quanto tempo de experiência em teste de software na indústria você tem?
 - () Não tenho
 - () 1 ano
 - () 2 anos
 - () 3 anos
 - () Mais que 3 anos

A.3.2 Modelo de Avaliação da Qualidade de Jogos Educacionais

Este formulário foi obtido por meio do modelo de avaliação da qualidade de jogos educacionais proposto por Savi, Wangenheim e Borgatto (2011). Este modelo contém 37 perguntas que avaliam três dimensões, sendo elas: a motivação, experiência dos usuários e a aprendizagem.

- Horário de Início:

- Nome:

- O design do jogo é atraente.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- Houve algo interessante no início do jogo que capturou minha atenção.
 - () Discordo totalmente
 - () Discordo parcialmente

- ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- A variação (forma, conteúdo ou de atividades) ajudou a me manter atento ao jogo.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- O conteúdo do jogo é relevante para os meus interesses.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- O funcionamento deste jogo está adequado ao meu jeito de aprender.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- O conteúdo do jogo está conectado com outros conhecimentos que eu já possuía.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- Foi fácil entender o jogo e começar a utilizá-lo como material de estudo.

- () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- Ao passar pelas etapas do jogo senti confiança de que estava aprendendo.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- Estou satisfeito porque sei que terei oportunidades de utilizar na prática coisas que aprendi com o jogo.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- É por causa do meu esforço pessoal que consigo avançar no jogo.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- Temporariamente esqueci as minhas preocupações do dia-a-dia, fiquei totalmente concentrado no jogo.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente

- ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- Eu não percebi o tempo passar enquanto jogava, quando vi o jogo acabou.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- Me senti mais no ambiente do jogo do que no mundo real, esquecendo do que estava ao meu redor.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- Pude interagir com outras pessoas durante o jogo.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- Este jogo é adequadamente desafiador para mim, as tarefas não são muito fáceis nem muito difíceis.
 - ☐ Discordo totalmente
 - ☐ Discordo parcialmente
 - ☐ Indiferente
 - ☐ Concordo parcialmente
 - ☐ Concordo totalmente

- O jogo evolui num ritmo adequado e não fica monótono – oferece novos obstáculos, situações ou variações de atividades.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- Me diverti com o jogo.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- Quando interrompido, fiquei desapontado que o jogo tinha acabado.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- Eu recomendaria este jogo para meus colegas.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente

- Consegui atingir os objetivos do jogo por meio das minhas habilidades.
 - () Discordo totalmente
 - () Discordo parcialmente

- () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente
- Gostaria de utilizar este jogo novamente.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente
- Tive sentimentos positivos de eficiência no desenrolar do jogo.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente
- O jogo contribuiu para sua aprendizagem.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente
- O jogo foi para sua aprendizagem, comparando-o com outras atividades.
 - () Discordo totalmente
 - () Discordo parcialmente
 - () Indiferente
 - () Concordo parcialmente
 - () Concordo totalmente
- A experiência com o jogo vai contribuir para seu desempenho na vida profissional.

- ☐ Discordo totalmente
- ☐ Discordo parcialmente
- ☐ Indiferente
- ☐ Concordo parcialmente
- ☐ Concordo totalmente

- Horário de Término:

A.4 Materiais utilizados na segunda etapa do estudo de viabilidade (Avaliação da Usabilidade)

A.4.1 Formulário de Caracterização de Perfil

Este formulário tem como objetivo verificar o nível de conhecimento dos participantes do estudo de viabilidade. Neste formulário há perguntas sobre o nível de conhecimento de cada sujeito em: teste de software, linguagem de programação Java, jogos educacionais, avaliação de jogos educacionais e avaliação heurística.

- Nome:

- Sexo:
 - ☐ Masculino
 - ☐ Feminino

- Qual é o seu nível de escolaridade?
 - ☐ Graduação
 - ☐ Mestrado Incompleto
 - ☐ Mestrado Completo
 - ☐ Doutorado Incompleto
 - ☐ Doutorado Completo

- Com que frequência você joga?
 - ☐ Nunca
 - ☐ Raramente
 - ☐ Às vezes

- ☐ Muito
 - ☐ Sempre
- Com relação ao seu conhecimento sobre jogos educacionais como você se considera?
 - ☐ Não tenho conhecimento
 - ☐ Básico
 - ☐ Médio
 - ☐ Avançado
 - ☐ Expert
- Você já participou outras vezes de avaliações jogos educacionais? Se sim, quantas vezes?
 - ☐ Nunca participei
 - ☐ 1 vez
 - ☐ 2 vezes
 - ☐ 3 vezes
 - ☐ Mais que 3 vezes
- Você já utilizou o conjunto de heurísticas proposto por Nielsen?
 - ☐ Sim
 - ☐ Não
- Você já jogou algum jogo de apoio ao ensino de teste de software?
 - ☐ Sim
 - ☐ Não
- Com relação ao seu conhecimento sobre a linguagem de programação Java como você se considera?
 - ☐ Não tenho conhecimento
 - ☐ Básico
 - ☐ Médio
 - ☐ Avançado
 - ☐ Expert

- Você já cursou alguma disciplina na graduação sobre teste de software?
 - () Sim
 - () Não
- Você já cursou alguma disciplina na pós-graduação sobre teste de software?
 - () Sim
 - () Não
- Com relação ao seu conhecimento sobre teste de software como você se considera?
 - () Não tenho conhecimento
 - () Básico
 - () Médio
 - () Avançado
 - () Expert
- Quanto tempo de experiência em teste de software na indústria você tem?
 - () Não tenho
 - () 1 ano
 - () 2 anos
 - () 3 anos
 - () Mais que 3 anos

A.4.2 Avaliação Heurística

Este formulário foi obtido por meio do conjunto de heurísticas proposto por Nielsen (1994). Este conjunto de heurísticas contém 10 heurísticas que podem receber diferentes conceitos, que variam de 0 à 4, para os problemas identificados.

- Nome:
- Visibilidade do status do sistema: o usuário deve ser informado pelo sistema em tempo razoável sobre o que está acontecendo.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.

- () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- **Compatibilidade do sistema com o mundo real:** o modelo lógico do sistema deve ser compatível com o modelo lógico do usuário.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- **Controle do usuário e liberdade:** o sistema deve tornar disponíveis funções que possibilitem saídas de funções indesejadas.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- **Consistência e padrões:** o sistema deve ser consistente quanto à utilização de sua simbologia e à sua plataforma de hardware e software.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.

- () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- Prevenção de erros: o sistema deve ter um design que se preocupe com as possibilidades de erro.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- Reconhecimento ao invés de lembrança: as instruções para o bom funcionamento do sistema devem estar visíveis no contexto em que o usuário se encontra.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- Flexibilidade e eficiência de uso: o sistema deve prever o nível de proficiência do usuário em relação ao próprio sistema.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.

- () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- Estética e design minimalista: os diálogos do sistema devem conter somente informações relevantes ao funcionamento.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- Ajuda aos usuários no reconhecimento, diagnóstico e correção de erros: as mensagens devem ser expressas em linguagem clara, indicando as possíveis soluções.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.
- Ajuda e documentação: a informação desejada deve ser facilmente encontrada, de preferência deve ser contextualizada e não muito extensa.
 - () 0 - O problema que foi encontrado não é necessariamente um problema de usabilidade.
 - () 1 - Problema estético: Não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível.
 - () 2 - Baixa prioridade: O problema encontrado é de menor prioridade.
 - () 3 - Alta prioridade: o problema encontrado é de maior prioridade.
 - () 4 - Catástrofe: o problema encontrado deve ser corrigido imediatamente.
 - () Não é aplicável.

MODELO DE AVALIAÇÃO DA QUALIDADE DE JOGOS EDUCACIONAIS E AVALIAÇÃO HEURÍSTICA

B.1 Considerações Iniciais

Neste Apêndice, apresentam-se os principais conceitos dos instrumentos utilizados para avaliar o jogo Testing Game. Na Seção B.2, apresenta-se uma visão geral do modelo de avaliação de jogos educacionais na engenharia de software proposto por Savi, Wangenheim e Borgatto (2011). Na Seção B.3, apresentam-se os principais conceitos e particularidades da avaliação heurística. Além disso, é apresentado o conjunto de heurísticas proposto por Nielsen (1994). Por fim, na Seção B.4 são apresentadas as considerações finais deste Apêndice.

B.2 Modelo de Avaliação de Jogos Educacionais na Engenharia de Software

O modelo de avaliação de jogos educacionais proposto por Savi, Wangenheim e Borgatto (2011) tem sido utilizado para avaliar a qualidade de jogos educacionais na área de engenharia de software, analisando a capacidade de motivação dos jogos, a experiências dos jogadores e a melhoria na aprendizagem. Na Figura 50 é ilustrado o modelo de avaliação de jogos educacionais, que contém 3 subcomponentes e 14 dimensões. Atualmente, as pesquisas que envolvem esse modelo de avaliação de jogos visam a avaliar jogos educacionais em diferentes áreas, tais como: Inspeção de Software, Estrutura de Dados, Teste de Software e Gerência de Projetos (BATTISTELLA; WANGENHEIM; WANGENHEIM, 2012; LOPES; MARQUES; CONTE, 2013; MESQUITA *et al.*, 2013; BEZERRA *et al.*, 2014).

Figura 50 – Modelo de Avaliação de Jogos Educacionais. (Adaptado de Savi, Wangenheim e Borgatto (2011))

O subcomponente **Motivação** é baseado no modelo de ARCS (KELLER, 1987). Esse subcomponente apresenta quatro categorias para representar a motivação em projetos educacionais, a saber: atenção, relevância, confiança e satisfação (SAVI; WANGENHEIM; BORGATTO, 2011). O subcomponente **Experiência do Usuário (UX)** verifica a interação do estudante com o jogo, considerando os pensamentos, sentimentos, prazeres, e outras percepções que envolvam interação (TULLIS; ALBERT, 2013).

As dimensões que correspondem ao subcomponente experiência do usuário são: imersão, desafio, competência, divertimento e interação social. Por fim, o subcomponente **Aprendizagem** é medido de acordo com os três primeiros níveis da taxonomia de Bloom (conhecimento, compreensão e aplicação) (ANDERSON; KRATHWOHL; BLOOM, 2001), além de outras duas dimensões denominadas: aprendizagem de curto prazo e aprendizagem de longo prazo. Essas duas últimas dimensões foram baseadas no modelo de avaliação de Moody e Sindre (2003).

B.3 Avaliação Heurística

A usabilidade está relacionada com a facilidade com que os usuários utilizam produtos de software para alcançarem objetivos específicos com eficácia, eficiência e satisfação. Geralmente, interfaces que têm uma boa usabilidade aumentam a produtividade dos usuários e diminuem a ocorrência de erros (ISO/IEC, 2010). A usabilidade é considerada um atributo de qualidade de software que avalia o quão fácil é os usuários utilizarem as interfaces do sistema, ou seja, a facilidade com que os usuários podem utilizar o software para realizar uma determinada tarefa (SREEDHAR, 2016).

Para avaliar a usabilidade de interfaces de *websites*, Nielsen (1994) propôs um conjunto de 10 heurísticas que foram fundamentadas em erros de usabilidade que eram encontrados com frequência em suas avaliações. Segundo Nielsen (1994), para a aplicação desse método é recomendado que se utilizem entre 3 a 5 avaliadores. Se essa quantidade for menor que 3, os avaliadores podem não encontrar todos os problemas de usabilidade e se essa quantidade for excedida, isso poderá tornar avaliação um método caro para sua aplicação (MOHAMED; JAAFAR, 2012).

Desde então, a avaliação heurística tem chamado a atenção de muitos pesquisadores e tem sido utilizada para avaliar a usabilidade de sistemas de diferentes domínios, tais como: medicina (TANG *et al.*, 2006), educação (LLING MICHAEL E MCKAY, 2016), jogos (PINELLE; WONG; STACH, 2008; MOHAMED; JAAFAR, 2012), entre outros.

Para que a avaliação heurística seja aplicada de forma eficiente, alguns passos devem ser seguidos (MOHAMED; JAAFAR, 2012), sendo eles:

- **Formação:** é realizado um treinamento para que os avaliadores se familiarizem com o conjunto de heurística que será utilizado e com o sistema que será avaliado;
- **Avaliação:** o sistema é avaliado por meio do conjunto de heurísticas para identificar problemas de usabilidade;
- **Taxa de classificação:** os problemas identificados no passo anterior são classificados de acordo com os conceitos de gravidade apresentados na Tabela 11; e
- **Processo de revisão:** é realizada uma análise para verificar quais foram os problemas de usabilidade identificados e suas respectivas taxas de classificação, considerando apenas a menor taxa de classificação.

Na Tabela 12, apresenta-se o conjunto de heurísticas proposto por Nielsen (1994) para avaliação da usabilidade de interfaces, foi utilizado para avaliar a usabilidade do jogo Testing Game.

Tabela 11 – Severidades para os problemas de usabilidade identificados na avaliação heurística

Conceitos	Descrição
0	O problema que foi encontrado não é necessariamente um problema de usabilidade
1	Problema estético: não é necessário corrigir o erro encontrado, a não ser que haja tempo disponível
2	Baixa prioridade: o problema encontrado é de menor prioridade
3	Alta prioridade: o problema encontrado é de maior prioridade
4	Catástrofe: o problema encontrado deve ser corrigido imediatamente

Tabela 12 – Conjunto de Heurísticas Proposto por Nielsen (1994) Para Avaliação da Usabilidade

Heurística	Descrição
H01	Visibilidade do status do sistema: o usuário deve ser informado pelo sistema em tempo razoável sobre o que está acontecendo.
H02	Compatibilidade do sistema com o mundo real: o modelo lógico do sistema deve ser compatível com o modelo lógico do usuário.
H03	Controle do usuário e liberdade: o sistema deve tornar disponíveis funções que possibilitem saídas de funções indesejadas.
H04	Consistência e padrões: o sistema deve ser consistente quanto à utilização de sua simbologia e à sua plataforma de hardware e software.
H05	Prevenção de erros: o sistema deve ter um design que se preocupe com as possibilidades de erro.
H06	Reconhecimento ao invés de lembrança: as instruções para o bom funcionamento do sistema devem estar visíveis no contexto em que o usuário se encontra.
H07	Flexibilidade e eficiência de uso: o sistema deve prever o nível de proficiência do usuário em relação ao próprio sistema.
H08	Estética e design minimalista: os diálogos do sistema devem conter somente informações relevantes ao funcionamento.
H09	Ajuda aos usuários no reconhecimento, diagnóstico e correção de erros: as mensagens devem ser expressas em linguagem clara, indicando as possíveis soluções.
H10	Ajuda e documentação: a informação desejada deve ser facilmente encontrada, de preferência deve ser contextualizada e não muito extensa.

A avaliação heurística pode ser realizada em qualquer fase do desenvolvimento de produtos de software, possibilitando que os problemas existentes sejam identificados antecipadamente (MOHAMED; JAAFAR, 2012). Desta forma, percebe-se que a avaliação heurística é uma boa opção para avaliar a usabilidade de produtos de software, pois é considerada uma avaliação rápida, barata e eficiente (NIELSEN, 1994). Neste trabalho, a avaliação heurística foi utilizada para avaliar a usabilidade do jogo Testing Game.

B.4 Considerações Finais

Neste Apêndice foram apresentados os principais conceitos dos modelos utilizados para avaliar o jogo Testing Game. O modelo de avaliação da qualidade de jogos educacionais na engenharia de software foi utilizado para avaliar a qualidade do jogo Testing Game na primeira etapa do estudo de viabilidade. O conjunto de heurísticas proposto por Nielsen (1994) foi utilizado para avaliar a usabilidade do jogo Testing Game na segunda etapa do estudo de viabilidade.

ANEXO

A

ARTIGOS PUBLICADOS

Um Mapeamento Sistemático Sobre Ensino de Teste de Software

Pedro Henrique D. Valle¹, Ellen F. Barbosa¹, José C. Maldonado¹

¹Instituto de Ciências Matemáticas e de Computação (ICMC/USP)
São Carlos/SP, Brasil, 13560-970

{pedrohenriquevalle@usp.br, francine@icmc.usp.br, jcmaldon@icmc.usp.br}

Abstract. Context: Software testing is an important activity to ensure quality for software products. However, there is a lack of qualified professionals and a lack of motivation to work with software testing. **Objective:** To identify the state of art about teaching software testing. **Method:** We performed a systematic mapping based on digital libraries and manual search. **Results:** We identified the main approaches of teaching software testing, as well as how to develop and evaluate them. Furthermore, we identified the languages addressed to teaching and the testing phases considered in these approaches. **Conclusion:** We characterized the state of art about teaching software testing approaches, observing that the most used ones are educational games and teaching software testing combined programming.

Resumo. Contexto: Teste é uma atividade importante na garantia da qualidade de produtos de software. No entanto, ainda há uma grande carência de profissionais qualificados e uma desmotivação para trabalhar com teste de software. **Objetivo:** Identificar o estado da arte referente ao ensino de teste de software. **Método:** Para isso, realizou-se a condução de um mapeamento sistemático utilizando máquinas de busca e a busca manual. **Resultados:** Identificaram-se as principais abordagens para o ensino de teste de software, bem como a forma de desenvolvê-las e avaliá-las. Além disso, identificaram-se as linguagens alvo e as fases de teste de software consideradas nessas abordagens. **Conclusões:** Caracterizou-se o estado da arte referente ao ensino de teste, observando que as abordagens mais utilizadas são jogos educacionais e ensino de teste com programação.

1. Introdução

O teste de software tem por objetivo executar programas ou produtos intermediários com entradas específicas analisando se esses produtos comportam-se de acordo com o esperado. O teste envolve basicamente quatro fases: o planejamento de teste, o projeto de casos de teste, a execução de testes e a avaliação dos resultados dos testes [Delamaro et al. 2007].

No entanto, a indústria de teste de software defronta-se com a carência de mão-de-obra especializada nessa área [de Souza et al. 2012]. Isso deve-se, sobretudo, à dificuldade e ineficiência em ensinar teste de software por meio de aulas teóricas, pois ainda se fazem necessários ambientes práticos que motivem os estudantes a realizarem atividades relacionadas com o teste de software. Outro fator que pode contribuir para a

carência de profissionais qualificados é a desvalorização dos profissionais de teste que, em geral, são pouco reconhecidos pelo trabalho realizado [Smith et al. 2012].

Diante desse cenário, percebe-se que é imprescindível a utilização de diferentes abordagens para auxiliar o processo de ensino e aprendizagem de teste de software. Nesse contexto, entende-se como abordagem as metodologias e os instrumentos utilizados para auxiliar o ensino de algum conteúdo, em um dado domínio de conhecimento.

O objetivo deste trabalho é identificar o estado da arte referente ao ensino de teste de software. Para isso, serão analisadas por meio de um mapeamento sistemático quais são as abordagens utilizadas para auxiliar o ensino de teste, as fases de teste de software contempladas no ensino dessa atividade, as tecnologias utilizadas para o desenvolvimento das abordagens, bem como as linguagens alvo e, por fim, verificar como foram realizadas as avaliações das abordagens propostas para auxiliar o ensino de teste de software.

De acordo com os resultados obtidos neste trabalho, o ensino conjunto de teste de software com programação e jogos educacionais são as duas abordagens mais utilizadas para auxiliar o ensino de teste de software. Essa caracterização do estado da arte referente ao ensino de teste poderá auxiliar na identificação e no direcionamento de passos futuros para a realização de novas pesquisas relacionadas com esse tema, pois algumas novas direções podem ser seguidas a partir dos resultados obtidos neste mapeamento.

Este trabalho está organizado da seguinte forma: Na Seção 2 apresenta-se o planejamento deste mapeamento sistemático. Na Seção 2.1 apresentam-se as informações sobre a coleta dos dados, demonstrando a quantidade de trabalhos selecionados para análise e o ano de publicação dos mesmos. Na Seção 2.2 apresentam-se as respostas para as questões de pesquisa. Na Seção 3 discutem-se as ameaças à validade deste estudo. Na Seção 4 apresentam-se os principais conceitos das duas abordagens mais utilizadas para auxiliar o ensino de teste de software. Por último, na Seção 5, apresentam-se as considerações finais e os trabalhos futuros.

2. Planejamento do Mapeamento Sistemático

O mapeamento sistemático é um tipo de pesquisa muito utilizada quando há um cenário abrangente e que tem por objetivo reunir o máximo de informações disponíveis sobre uma determinada área do conhecimento [Kitchenham e Charters 2007]. Para a condução deste trabalho realizou-se um planejamento do mapeamento sistemático que contém as questões de pesquisa, *string* de busca, estratégias de busca, critérios de inclusão e exclusão. Devido à limitação de espaço, o protocolo deste mapeamento sistemático e os trabalhos selecionados para análise estão disponíveis em: (goo.gl/dTmSRR).

As questões de pesquisa para a realização do Mapeamento Sistemático são as seguintes:

- **Q1** - Quais são os tipos de abordagens que têm sido utilizadas para auxiliar o ensino teste de software?
- **Q2** - Quais são as fases de teste de software que têm sido contempladas no ensino de teste de software?
- **Q3** - Quais são tecnologias que têm sido utilizadas no desenvolvimento das abordagens identificadas na Q1 e quais são as linguagens alvo utilizadas para auxiliar o ensino de teste de software?

- **Q4** - Quais são as avaliações que têm sido realizadas para a validação das abordagens utilizadas para apoiar o ensino de teste de software?

Na Tabela 1 apresenta-se a *string* de busca utilizada neste mapeamento sistemático.

Tabela 1. *String* de busca genérica gerada para o Mapeamento Sistemático

Idioma	<i>String</i> de Busca
Inglês	("teaching" OR "learning" OR "training" OR "education") AND ("software testing" OR "software test" OR "software evaluation")

Para a inclusão dos trabalhos na análise, utilizaram-se os seguintes critérios de inclusão: (i) Os trabalhos devem estar escritos em inglês ou português; (ii) Os trabalhos devem conter as palavras chaves utilizadas na *string* de busca, no resumo e/ou título e/ou nas palavras-chave do artigo selecionado; e (iii) Os trabalhos devem apresentar abordagens que auxiliem o ensino de teste de software. A negação dos critérios apresentados anteriormente foi utilizada para exclusão dos trabalhos encontrados. O conjunto completo de critérios de exclusão está disponível em: (goo.gl/dTmSRR).

2.1. Coleta dos Dados

Na Tabela 2 apresenta-se a quantidade de trabalhos selecionados em cada passo de acordo com o planejamento estabelecido. É importante observar que grande parte dos trabalhos selecionados para análise foram identificados a partir das bases de dados IEEE Xplore, ACM e também pela busca manual. A busca manual foi realizada já que trabalhos importantes ou recentes poderiam não ter sido considerados na análise.

Base de Dados	Passo 1	Passo 2	Passo 3
Scopus	105	33	4
ACM	47	12	5
Compendex	228	36	3
IEEE	360	22	5
Busca Manual	—	—	8
TOTAL	740	103	25

Tabela 2. Quantidade de Trabalhos Selecionados

Figura 1. Estudos por Ano

Na Figura 1 apresenta-se a distribuição temporal dos trabalhos analisados neste mapeamento sistemático. Mesmo não havendo padrão ou tendência clara sobre a distribuição dos trabalhos ao longo dos anos, observa-se que houve um aumento no interesse dos pesquisadores com relação ao processo de ensino de aprendizagem de conteúdos relacionados com o teste de software a partir do ano de 2010.

A maior parte dos trabalhos selecionados foram publicados nas seguintes conferências: Software Engineering Education and Training (CSEE&T), Frontiers in Education Conference (FIE), Technical Symposium on Computer Science Education (SIGCSE), International Conference on Software Engineering (ICSE), Fórum de Educação em Engenharia de Software (FEES) e Simpósio Brasileiro de Informática na Educação (SBIE).

2.2. Análise dos Resultados

Por meio das respostas obtidas realizou-se a caracterização do estado da arte referente ao ensino de teste de software. É importante ressaltar que essa caracterização poderá auxiliar no desenvolvimento de novas pesquisas, pois há algumas decisões que podem ser tomadas, a partir dos resultados obtidos neste mapeamento sistemático, para o desenvolvimento de novos trabalhos que tenham como objetivo criar novas abordagens e/ou ambientes que auxiliem o ensino de teste de software.

2.2.1. Respostas para a Questão 1

A primeira questão de pesquisa diz respeito sobre quais são os tipos de abordagens utilizadas para auxiliar o ensino teste de software. Na Figura 2 apresenta-se um gráfico de bolha que ilustra o cenário do ensino de teste de software, apresentando as principais abordagens utilizadas, o método de validação dessas abordagens e as técnicas de teste de software contempladas em cada uma das abordagens.

Figura 2. Abordagens para Auxiliar o Ensino de Teste de Software

A seguir apresentam-se as descrições das abordagens identificadas para auxiliar o ensino de teste de software.

- **Módulos Educacionais:** São unidades concisas de estudo, compostas por conteúdos teóricos combinados com atividades práticas e avaliações, apoiadas por recursos tecnológicos e computacionais [Barbosa e Maldonado 2011];
- **Jogos Educacionais:** São jogos que proporcionam práticas educacionais atrativas e inovadoras, nos quais os usuários podem aprender de forma mais ativa, dinâmica e motivadora [Farias et al. 2012];
- **Ensino Conjunto de Teste com Programação:** É o ensino conjunto de conceitos básicos de programação e de teste de software. Muitas pesquisas demonstram que o ensino conjunto dessas disciplinas possui benefícios [de Souza et al. 2012];
- **Quiz:** É uma forma interativa de avaliar o conhecimento de usuários, por meio de questionários [Mustakerov e Borissova 2005];
- **Revisão por Pares:** Utilizar essa técnica para auxiliar o ensino de teste de software, permite um aprendizado lúdico e competitivo, no qual os estudantes aprendem uns com os outros [Smith et al. 2012];
- **Desenvolvimento Dirigido por Testes (TDD):** Nessa técnica o desenvolvedor escreve um caso de teste e em seguida produz um código que possa ser validado pelo teste [Edwards 2003];
- **Tutorial:** É uma ferramenta de ensino e aprendizagem que pode ser um programa de computador ou um texto que contém ou não imagens que auxiliam no processo de aprendizagem, demonstrando o passo a passo para a realização de alguma atividade [Liu et al. 2010];
- **Rede Social:** Essa abordagem permite que usuários comuniquem-se uns com outros trocando experiências no decorrer do processo de aprendizagem [Clarke et al. 2011];
- **Modelo de Residência de Software:** Essa abordagem inclui o ensino tradicional de conceitos relevantes de um determinado conteúdo e em seguida a realização de atividades práticas de com profundidade e/ou com especialização em algum assunto específico [Sampaio et al. 2005];
- **Aprendizagem Baseada em Problemas:** Essa abordagem permite que os estudantes trabalhem em equipe com o intuito de resolverem problemas, incentivando o desenvolvimento de habilidades como atitudes, auto iniciativa e cooperação [Figuerêdo et al. 2011];
- **Aprendizagem Baseada em Desempenho:** Essa abordagem utiliza a medição do desempenho dos usuários para esclarecer os principais objetivos e demonstrar as necessidades da aprendizagem individual do usuário [Wang et al. 2011];

Foram identificadas 11 abordagens diferentes para auxiliar o ensino de teste de software. As duas abordagens mais utilizadas são os **Jogos Educacionais** e o **Ensino de Teste com Programação**, as quais são discutidas na Seção 4. Juntas essas abordagens foram utilizadas em mais de 50% dos trabalhos analisados. É importante ressaltar que alguns trabalhos utilizaram mais de uma abordagem para apoiar o ensino de teste de software. A partir da Figura 2 percebe-se que o Teste Funcional é a técnica mais abordada.

2.2.2. Respostas para a Questão 2

A segunda questão de pesquisa diz respeito sobre quais são as fases de teste de software contempladas no ensino de teste de software. Na Figura 3 ilustram-se as fases de teste de software identificadas e as porcentagens de trabalhos que as contemplaram.

Figura 3. Fases de teste de software contempladas nos trabalhos para auxiliar o ensino de teste

Como pode ser observado, aproximadamente 60% dos trabalhos analisados abordaram conteúdos relacionados com todas as fases de teste de software. Nos trabalhos analisados, a fase de projeto de caso de teste é a fase mais abordada no ensino dos conteúdos relacionados com o teste de software, sendo utilizado em aproximadamente 84% dos trabalhos selecionados. Além disso, há trabalhos que abordam apenas uma fase de teste de software como o trabalho proposto por [Farias et al. 2012], que propôs um jogo educacional para auxiliar o ensino de teste de software, especificamente para a fase de planejamento de teste.

2.2.3. Respostas para a Questão 3

A terceira questão de pesquisa diz respeito sobre quais são as tecnologias utilizadas no desenvolvimento das abordagens identificadas na Q1 e quais são as linguagens alvo utilizadas para auxiliar o ensino de teste de software, ou seja, as linguagens de programação utilizadas para escrever os códigos a serem testados.

Figura 4. a) Linguagens Alvo e b) Linguagens para o desenvolvimento das abordagens

Na Figura 4 a) observam-se as 9 tecnologias identificadas para auxiliar o desenvolvimento das abordagens identificadas na Q1, e as porcentagens de trabalhos que as utilizaram. A linguagem de programação Java foi a mais utilizada, sendo que aproximadamente 48% dos trabalhos a utilizaram. Isso pode estar relacionado com o fato

da linguagem Java ser bem conhecida entre os desenvolvedores de produtos de software e por prover a característica de portabilidade.

A segunda tecnologia mais utilizada foi a linguagem HTML, sendo utilizada em 15% dos trabalhos analisados. Por fim, as tecnologias PHP, OpenSim, Unity 3D, JavaScript, Flash e C++ foram utilizadas em aproximadamente 4% dos trabalhos analisados cada uma. É importante observar que alguns trabalhos utilizaram mais de uma linguagem de programação para o desenvolvimento das abordagens propostas. Por outro lado, outros não deixaram claro quais linguagens de programação foram utilizadas.

Na Figura 4 b) observam-se as linguagens alvo utilizadas pelas abordagens identificadas na Q1 para auxiliar o ensino de teste de software, a saber: C, Java, C# e C++, e a porcentagem de trabalhos que as abordaram. Aproximadamente 58% dos trabalhos analisados utilizaram a linguagem de programação Java como linguagem alvo para auxiliar o ensino de teste de software. Isso pode estar relacionado com o fato da linguagem Java ser bem disseminada no meio acadêmico e na indústria e também por haver muitas ferramentas de teste de software que auxiliam o teste de códigos escritos nessa linguagem. As outras linguagens alvo identificadas, tais como: C, C# e C++, foram utilizadas em 19%, 8%, 4% dos trabalhos analisados, respectivamente.

2.2.4. Respostas para a Questão 4

A quarta questão de pesquisa diz respeito sobre quais os tipos de avaliações que foram realizadas para validação das abordagens propostas para auxiliar o ensino de teste de software. Na Figura 5 ilustram-se os tipos de avaliações identificadas e a porcentagem de trabalhos que as utilizaram.

Figura 5. Avaliações realizadas para validação das abordagens propostas

As avaliações realizadas por meio de questionários foram utilizadas em 20% dos trabalhos analisados, como o trabalho proposto [de Oliveira 2013] que realizou a avaliação do Jogo Educacional proposto por meio de questionários com 127 avaliadores (jogadores). As avaliações por meio de estudo de caso correspondem a 28% dos trabalhos analisados. Porém, poucos detalhes sobre a realização dos estudos de caso foram disponibilizados. A avaliação empírica foi o tipo de avaliação mais utilizada para validação das abordagens propostas, sendo utilizada por 52% dos trabalhos. Nesse tipo de avaliação foram observadas as mudanças de atitude dos estudantes após a utilização das abordagens propostas, não havendo uma avaliação formal e sistemática. Esse tipo de

avaliação é interessante para avaliar o comportamento dos alunos ao utilizar as abordagens propostas.

3. Ameaças à Validade

Neste trabalho foram identificados três tipos de ameaças à validade que são:

1. **Validade Externa:** Refere-se à capacidade de generalizar os resultados do experimento. Apesar da pequena quantidade de trabalhos contemplados nesse mapeamento sistemático, os trabalhos contemplados são considerados relevantes na área.
2. **Validade Interna:** Refere-se à capacidade de tirar conclusões sobre os resultados obtidos. A busca automatizada dos trabalhos pode ser classificada neste tipo de ameaça, pois há a possibilidade de estudos importantes ou recentes não serem considerados. Para minimizar esse problema foram realizadas buscas manuais nos principais anais de congressos da área e buscas no Google Scholar.
3. **Validade de Construção:** Está relacionada, em geral, com a má definição da base teórica ou com a definição do processo de experimentação. A eficiência da *string* de busca pode ser classificada neste tipo de ameaça. A validação da *string* de busca utilizada neste trabalho foi realizada por meio de trabalhos controle. Além disso, formularam-se *strings* de busca alternativas com outros sinônimos das palavras utilizadas.

4. Ensino de Teste de Software com Programação e Jogos Educacionais

A partir dos resultados obtidos, observa-se que as duas abordagens mais utilizadas para auxiliar o ensino de teste de software são: i) jogos educacionais; e ii) ensino de teste de software com programação. O ensino de teste de software é pouco explorado no contexto de cursos de graduação, sendo pouco abordado na maioria dos cursos de computação e apenas ensinado no final da graduação. Entretanto, pesquisas indicam que o teste de software deve ser ensinado o mais cedo possível para que os estudantes desenvolvam suas habilidades de compreensão e análise [Edwards 2004, de Souza et al. 2012, de Souza et al. 2014].

Para resolver esses problemas, algumas iniciativas foram investigadas, dentre elas destaca-se o ensino de teste de software com programação. Essa abordagem sugere que o teste de software seja ensinado em paralelo com fundamentos de programação. O ensino conjunto dessas disciplinas pode ajudar o desenvolvimento de habilidades de compreensão e análise nos estudantes, pois para testar os estudantes precisam entender o comportamento de seus programas. Neste mapeamento sistemático identificaram-se algumas ferramentas para auxiliar o ensino de teste de software com programação, tais como: Web-CAT [Edwards 2003] e ProgTest [de Souza et al. 2012].

Jogos educacionais, por sua vez, têm sido muito utilizados para auxiliar o ensino de teste de software. Muitos pesquisadores afirmam que essas ferramentas podem facilitar a aprendizagem de conceitos e ideias, já que jogos motivam e proporcionam prazeres aos jogadores. Sendo assim, os usuários são incentivados a dedicarem parte de seu tempo e de seus esforços para vencerem os obstáculos propostos [Neto e da Fonseca 2013]. Portanto, percebe-se que os jogos educacionais possibilitam que seus usuários obtenham conhecimentos combinados com a diversão. Alguns jogos educacionais no domínio

de teste de software foram identificados neste mapeamento sistemático, tais como: *iTest Learning* [Farias et al. 2012], *Jogo das 7 Falhas* [Diniz e Dazzi 2011], *TestEG* [de Oliveira 2013], entre outros.

Os trabalhos analisados que utilizaram os jogos educacionais para auxiliar o ensino de teste de software contemplaram apenas a técnica de Teste Funcional. Desta forma, seria interessante a criação de jogos educacionais que abordassem conteúdos relacionados com o Teste Estrutural e o Baseado em Defeitos. Portanto, seria pertinente criar uma ferramenta que combine o ensino de teste de software com programação e jogos educacionais para apoiar o ensino de teste de software incremental. Uma alternativa para isso seria a criação de um jogo educacional que abordasse conteúdos de teste de software em conjunto com fundamentos de programação.

5. Conclusões e Trabalhos Futuros

Por meio do mapeamento sistemático realizado neste trabalho, observou-se que há uma carência de trabalhos que abordam o ensino de teste de software. Para a realização dessa análise foram selecionados 25 trabalhos. A partir dos estudos selecionados identificaram-se 11 abordagens diferentes para auxiliar o ensino de teste de software, dentre elas: jogos educacionais e ensino de teste com programação, as mais utilizadas.

Aproximadamente 60% dos trabalhos analisados contemplaram as quatro fases de teste de software. Para uma melhor formação de profissionais de teste seria interessante que uma maior quantidade de estudos abordassem todas essas fases, pois assim os mesmos contemplaram o ensino de teste de forma geral. Para o desenvolvimento das abordagens identificadas foram utilizadas diferentes tecnologias, sendo que a linguagem Java foi utilizada em 42% dos trabalhos analisados. Além disso, a mesma foi utilizada em 58% dos trabalhos como linguagem alvo. Para validação das abordagens a avaliação empírica foi a mais utilizada, aproximadamente de 52% dos trabalhos a utilizaram. Porém, poucos detalhes foram apresentados sobre a condução dessas avaliações.

Neste cenário, percebe-se uma carência de ambientes virtuais que motivem o ensino de teste de software. Sendo assim, como trabalhos futuros pretende-se desenvolver um jogo educacional que aborde o ensino conjunto de teste de software com programação. Neste cenário o mapeamento sistemático desenvolvido contribuiu para identificar os principais elementos para o desenvolvimento desse jogo. Esse jogo educacional contemplaria as quatro fases de teste. A linguagem Java poderia ser utilizada para o desenvolvimento do jogo, bem como linguagem alvo. Por fim, para a validação seria realizada uma avaliação empírica por meio de experimentos controlados.

6. Agradecimentos

Os autores agradecem à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)(PROEX-9035919/M) pelo apoio concedido a este trabalho.

Referências

Barbosa, E. e Maldonado, J. (2011). *E-Infrastructures and Technologies for Lifelong Learning: Next Generation Environments*, chapter Collaborative development of educational modules: a need for lifelong learning.

- Clarke, P. J., Pava, J., Wu, Y., e King, T. M. (2011). Collaborative web-based learning of testing tools in se courses. In *Technical symposium on Computer science education*.
- de Oliveira, B. C. (2013). Testeg-um software educacional para o ensino de teste de software. In *Universidade Federal de Lavras*.
- de Souza, D. M., da Silva Batista, M. H., e Barbosa, E. F. (2014). Avaliação de qualidade de um ambiente de apoio ao ensino de programação. *Revista Novas Tecnologias na Educação*.
- de Souza, D. M., Maldonado, J. C., e Barbosa, E. F. (2012). Aspectos de desenvolvimento e evolução de um ambiente de apoio ao ensino de programação e teste de software. In *Simpósio Brasileiro de Informática na Educação*.
- Delamaro, M. E., Maldonado, J. C., e Jino, M. (2007). *Introdução ao Teste de Software*. Elsevier.
- Diniz, L. L. e Dazzi, R. L. (2011). Jogo para o apoio ao ensino do teste de caixa-preta. In *Simpósio Brasileiro de Informática na Educação*.
- Edwards, S. H. (2003). Teaching software testing: Automatic grading meets test-first coding. In *Conference on Object-oriented Programming, Systems, Languages, and Applications*. ACM.
- Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. In *Technical Symposium on Computer Science Education*. ACM.
- Farias, V., Moreira, C., Coutinho, E., e Santos, I. S. (2012). itest learning: Um jogo para o ensino do planejamento de testes de software. In *Fórum de Educação em Engenharia de Software*. Simpósio Brasileiro de Engenharia de Software.
- Figueiredo, C. d. O., dos Santos, S. C., Borba, P. H., e Alexandre, G. H. (2011). Using pbl to develop software test engineers. In *International Conference on Computers and Advanced Technology in Education*.
- Kitchenham, B. e Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University and Durham University Joint Report.
- Liu, H., Kuo, F.-C., e Chen, T. Y. (2010). Teaching an end-user testing methodology. In *Software Engineering Education and Training*.
- Mustakerov, I. e Borissova, D. (2005). A conceptual approach for development of educational web-based e-testing system. *Expert Systems with Applications*.
- Neto, J. F. B. e da Fonseca, F. d. S. (2013). Jogos educativos em dispositivos móveis como auxílio ao ensino da matemática. *Revista Novas Tecnologias na Educação*, 11.
- Sampaio, A., Albuquerque, C., Vasconcelos, J., Cruz, L., Figueiredo, L., e Cavalcante, S. (2005). Software test program: a software residency experience. In *International Conference on Software Engineering*.
- Smith, J., Tessler, J., Kramer, E., e Lin, C. (2012). Using peer review to teach software testing. In *International Conference on International Computing Education*. ACM.
- Wang, M., Jia, H., Sugumaran, V., Ran, W., e Liao, J. (2011). A web-based learning system for software test professionals. *IEEE Transactions on Education*.

CS Curricula of the Most Relevant Universities in Brazil and Abroad: Perspective of Software Testing Education

Pedro Henrique Dias Valle, Ellen Francine Barbosa, José Carlos Maldonado
Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP)
São Carlos/SP, Brazil, 13560-970
Email: pedrohenriquevalle@usp.br, francine@icmc.usp.br, jcmaldon@icmc.usp.br

Abstract—Software testing is a relevant activity to provide evidencies of the quality of software products. However, few courses in the computing area provide an adequate body of knowledge to the students and few of them pursue the software development practices and activities related to VV&T (Verification, Validation, and Testing), specifically testing, leading to a recognized lack of professionals that master the underlying VV&T concepts and principles. In this paper, we provide an overview concerning the software testing education in the most relevant brazilian universities and abroad. We analyze the reference curricula proposed by SBC (Brazilian Computer Society) and ACM (Association for Computing Machinery) for undergraduate computer courses as well as the curricula of the most relevant universities in Brazil and abroad. We identify the effort allocated, the main approaches and the main tendencies to facilitate the teaching of software testing. It is clear that there is a need to integrate software testing education with other disciplines along the CS undergraduate courses.

Keywords—Software Testing, Curricula Recommendations, Integrated Education.

I. INTRODUCTION

Software testing aims at executing programs or models with specific inputs analyzing if these products behave as expected, leading to a detailed analysis of the software with the goal to identify failures and after, through debugging, to eliminate the faults that originate the failures. The software testing is a dynamic activity since it is based on the execution of programs or other executable intermediate artifacts. [1]. Software testing involves four phases: test planning, test case design, test execution and evaluation of testing results. These phases must be planned and applied along the software development processes, according to previously established testing strategies [2].

Despite the software testing being recognized as a relevant activity in quality assurance of software products, the test industry has found a lack of specialized professionals

in this area [3]. This fact can be related with the difficulty and inefficiency in software testing education, since there is a lack of well established environments and methods that motivate the students to learn contents related to the software testing activity [4].

To know how the testing professionals are trained, we analyzed the reference curricula proposed by Brazilian Computer Society (SBC) and by Association for Computing Machinery (ACM) for undergraduate courses in Computer Science. The goal of the reference curricula is to provide support and guidelines for the definition, implementation and evaluation of undergraduate courses in Computer Science. These curricula recommend that the software testing contents be addressed as one of the topics of Software Engineering course, called Verification, Validation and Testing Software (VV&T) [5, 6, 7, 8, 9].

The SBC and ACM recommendations, however, do not explicitly motivate to provide the students an integrated view of software testing contents with other disciplines and practices related to the development of software products, such as: Algorithms, Object-Oriented Programming, Data Structures, Analysis and Requirements Specification, Software Reuse, among others. As a consequence, the students do not acquire a “test culture” adequate for software product development, since they do not embody the habit of rigorously testing their programs from the moment they learn to program. It is clear that it is indispensable to revisit the methodology of software testing education integrating the software testing contents with other disciplines along the CS undergraduate courses.

The aim of this paper is to characterize how software testing education has been approached in the CS undergraduate courses at universities in Brazil and abroad. We identify the effort allocated, the main approaches and the main tendencies to facilitate the software testing education.

This paper is organized as follows: In Section II it is presented the research methodology used for the selection of universities that had their CS reference curricula analyzed, as well as the reference curricula proposed by SBC and ACM. In Section III, we present an overview about software testing education, providing an analysis of the reference curricula proposed by SBC and ACM and of the CS curricula of the most relevant universities in the world. In Section IV, we present the main approaches identified to support software testing education. In Section V, we discuss the main tendencies in teaching of software education. Finally, in Section VI we present the conclusions and future work.

II. METHODOLOGY

For the selection of the brazilian universities that had their curricula analyzed we used the ranking made available by

the RUF (Ranking Universitário Folha)¹ of the year 2014. The RUF is an annual evaluation about the higher teaching of the Brazil accomplished by the newspaper Folha de São Paulo. In this evaluation are ranked the most relevant brazilian universities, including publics and privates, by analyzing five indicators: research, internationalization, innovation, education and marketing.

For the selection of the most relevant universities in the world, we used the ranking 2014-2015 made available by Times Higher Education², that is an english magazine about articles and news of the higher education. This magazine provides annually a ranking of the most relevant universities in the world.

Finally, we analyzed the reference curricula made available by SBC³ and ACM⁴ for CS undergraduate courses: Computer Science, Computer Engineering, Information Systems, Software Engineering, among others. We conducted a systematic mapping to identify the main tendencies for teaching software testing, aiming at contributing to the reformulation of current practices [10].

III. DIAGNOSIS OF THE SOFTWARE TESTING EDUCATION

Analysing the curricula of the selected national and international universities, we obtained an diagnosis of how software testing professionals are educated and trained along their undergrad activities. In addition, we perform an analysis of the reference curricula proposed by SBC and ACM. We summarize the teaching practices of software testing at the selected universities in Brazil and abroad.

A. Brazilian Scenario

The reference curricula proposed by SBC for CS undergraduate courses suggests that the contents related to the software testing could be approached in one of the topics of the Software Engineering discipline, called Verification, Validation and Testing (VV&T), in which few lectures are dedicated to testing [5, 11, 12]. This recommendation has induced a practice in which the contents are addressed isolated with few integration with other disciplines.

We analyzed the curricula of the courses in Computer Science, Computer Engineering, Information Systems and Software Engineering of the 25 most relevant universities in Brazil according to the *ranking* made available by RUF in 2014. In Table I we present the brazilian universities that had their curricula analyzed.

In general, the brazilian universities address software testing education in the scope of the Software Engineering discipline, in the topic Verification, Validation and Testing (VV&T) without integration with other disciplines. In this case, usually, less than ten hours are assigned to the teaching of software testing in these courses, i.e., there is no time enough to adequately address a software testing practice in the CS undergrad courses

TABLE I. THE MOST RELEVANT BRAZILIAN UNIVERSITIES

Brazilian Universities	Abbreviation
University of Sao Paulo	USP
Federal University of Minas Gerais	UFMG
Federal University of Rio de Janeiro	UFRJ
Federal University of Rio Grande do Sul	UFRGS
State University of Campinas	UNICAMP
São Paulo State University	UNESP
Federal University of Santa Catarina	UFSC
University of Brasilia	UNB
Federal University of Paraná	UFPR
Federal University of São Carlos	UFSCAR
Federal University of Pernambuco	UFPE
Federal University of São Paulo	UNIFESP
Federal University of Ceara	UFC
Federal University of Bahia	UFBA
Federal University of Santa Maria	UFSM
Federal University of Fluminense	UFF
State University of Rio de Janeiro	UER
J	
Pontifical Catholic University of Rio Grande do Sul	PUC
RS	
Federal University of Viçosa	UFV
Pontifical Catholic University of Rio de Janeiro	PUC-RIO
Federal University of Rio Grande do Norte	UFR
N	
Federal University of Goiás	UFG
State University of Maringá	UEM
State University of Londrina	UEL
Federal University of Paraíba	UFPB

There are also few brazilian universities that have a specific and compulsory discipline for software testing,

¹ Available at: <http://ruf.folha.uol.com.br>

² Available at: <http://www.timeshighereducation.co.uk/>

³ Available at: <http://www.sbc.org.br/>

⁴ Available at: <http://www.acm.org/education/curricula-recommendations>

such as: University of Brasilia (UNB), Federal University of Ceará (UFC), Pontifical Catholic University of Rio Grande do Sul (PUCRS), Federal University of Rio Grande do Norte (UFRN) and Federal University of Goiás (UFG). Few other universities have an elective discipline for software testing, such as: Federal University of Rio de Janeiro (UFRJ) and Institute of Mathematics and Computer Science of University of São Paulo (ICMC/USP).

However, it should be observed that the contents of software testing are methodologically addressed without providing the students an integrated view of other relevant disciplines for software product development, such as: Algorithms, Object-Oriented Programming, Data Structures, Analysis and Requirements Specification, Software Reuse, Data Base, among others.

B. International Scenario

ACM has also proposed reference curricula for CS undergraduate courses: Computer Science, Software Engineering, Computer Engineering and Information Systems. ACM suggests that contents related to the software testing could be approached in topics of disciplines such as Software Engineering, Verification and Validation of Software, Computer Systems Engineering and Programming Fundamentals, inducing more integrated practices of testing with other disciplines. However, like in the brazilian universities, less than ten hours are assigned to the teaching of software testing in these courses, i.e., there is no time enough to adequately address a software testing practice in the CS undergrad courses [13, 7, 9].

We also performed a search in the CS curricula of the most relevant universities in the world with the goal to identify how the contents related to the software testing have been addressed. In Table II we present the international universities that had their curricula analyzed.

Table II. THE MOST RELEVANT UNIVERSITIES IN THE WORLD OF COMPUTER SCIENCE AREA

International Universities	Country
Massachusetts Institute of Technology	United States
Stanford University	United States
California Institute of Technology	United States
Princeton University	United States
University of Cambridge	United Kingdom
Imperial College London	United Kingdom
University of Oxford	United Kingdom
Swiss Federal Institute of Technology Zürich	Switzerland
University of California - Los Angeles	United States
University of California - Berkeley	United States
Georgia Institute of Technology	United States
École Polytechnique Fédérale de Lausanne	Switzerland
National University of Singapore	Singapore
Carnegie Mellon University	United States
Cornell University	United States
Northwestern University	United States
University of Illinois at Urbana Champaign	United States
University of Technology	Netherlands
University of Science and Technology	Hong Kong
Harvard University	Japan
University of California - Santa Barbara	United States

In the curricula of these universities, the contents related to the software testing are considered in the topics of the disciplines of Software Engineering and/or Programming, as proposed by ACM. The software testing contents are addressed isolatedly, without a relationship with other disciplines during the CS undergraduate courses.

It can be observed, with few exceptions, that in the CS courses, either in Brazil or abroad, software testing is addressed only in the disciplines of Software Engineering and/or Programming Fundamentals. Few lectures are assigned to the teaching of software testing, providing to students only an overview of the contents of this area. In addition, software testing education is approached only at the end of the undergraduate courses. Therefore, the software testing professionals, in general, do not receive a good education at the universities to start working inside companies.

It would be interesting to integrate software testing education with other disciplines along the CS undergraduate courses, with projects and tasks that would lead the students to explore in an integrated perspective the skills and competences developed in the course, including software testing. In this direction, it would be mandatory to revisit and modify the reference curricula proposed by SBC and ACM, since they provide crucial directions and elements for the definition, implementation and evaluation of CS undergraduate courses. There is the need to induce and motivate software testing contents be addressed in an integrated perspective with other disciplines along the CS undergraduate course, making evident to students the importance of testing their programs (products) before being released. Consequently, the developers would create the habit of testing their programs since they start learning how to program. This would lead to the creation of an adequate

“test culture” adequate among students and professionals.

IV. APPROACHES TO SUPPORT SOFTWARE TESTING EDUCATION

In this perspective, the need to improve software testing education is more than evident. It is indispensable to develop environments and tools to facilitate the teaching and training of software testing, providing to the students and professionals skills and motivation to work with software testing [14]. Recently, different approaches, methodologies and tools have been used to facilitate software testing education in an integrated approach along CS undergraduate courses.

To characterize the main initiatives used in software testing education, we carried out a systematic mapping [10]. In this study we identified 11 different initiatives to support software testing education: Educational Modules, Peer Review, Educational Games, Quiz, Tutorial, Test Driven Development (TDD), Problem-Based Learning, Integrated Teaching of Software Testing and Programming, Social Network, Software Residence and Based-Learning Performance. Following, we present a description of the four most used of these initiatives.

- **Educational Modules:** They are concise units of study, composed of theoretical contents combined with practical activities and evaluations, supported by technological and computing resources [15];
- **Educational Games:** They are games that facilitate the learning of concepts and ideas, providing attractive educational practices, in which the users can learn more actively, dynamic and motivated, allowing to acquire knowledge combined with the fun [16];
- **Test Driven Development (TDD):** It is a software development technique in which the developers write automated test cases for each functionality. Then, code is produced that can be validated by the previously written tests [17] and;
- **Integrated Teaching of Software Testing and Programming:** It is the teaching of basic concepts of programming together with the software testing concepts. Several studies have shown that the integrated teaching of these topics provides benefits, developing the skills of analysis and understanding in the students [18, 19, 20].

Educational games and the integrated teaching of software testing and programming are the two most used initiatives identified to support the software testing education. We perceived that these initiatives can be considered as the main tendencies in software testing education. These initiatives are briefly discussed next.

V. TENDENCIES OF SOFTWARE TESTING EDUCATION EVERY SOFTWARE PRODUCT MUST BE PROPERLY TESTED BEFORE being released to the user. Despite the software testing being recognized as an important activity in the quality assurance of software products, many students feel unmotivated to learn the contents related to the software testing. In addition, many software testing professionals are devalued because, in general, their salaries are lower compared with other positions in the software development companies [20, 4].

However, this scenario has been changed and some companies have changed their allocation strategies and accountability of these professionals in the development teams, once that they perceived that these activities are entirely related. Although there is no scientific evidence, researchers say that testers with good programming skills perform tests with better efficiency [3]. Thus, many companies are now understanding that software testing is an important activity for the development of high quality software products.

As mentioned before, it is fundamental to develop environments and tools that facilitate the teaching and training of software testing, providing to the students and professionals training and motivation to work with software testing [14]. To address these problems, software testing tools and environments have been used to facilitate the software test execution process, allowing to detect the greater number of defects so that it can provide evidences of the quality of software products. Currently, there are several software testing tools, such as: JaBUTi, muJava, SPACES, among others [21, 22, 23]. In the website of the Núcleo de Apoio à Pesquisa in Software Livre (NAPSoL)⁵ some open source software testing tools are available. Such tools can be used to support the software testing education.

However, the isolated adoption of these tools is not enough to motivate and facilitate software testing education. It is also necessary to use different methodologies and to approach the testing contents and concepts in an integrated perspective along the CS undergraduate courses. Following, we present the two most relevant initiatives in this direction: the integrated perspective of software testing education and educational games.

C. Integrated Perspective of Software Testing Education Contents

The integrated teaching of software testing and programming can help the development of skills of analysis and comprehension in the process of training students, because for testing is necessary that the students know the behavior and semantics of their programs [17, 3]. We present some of the initiatives in this direction.

⁵ Available at: <http://napsol.icmc.usp.br/>

The paper proposed by Corte and Maldonado [20] had the goal to provide mechanisms for integrating the programming fundamentals teaching with software testing education. For this, they developed an integrated educational module of object-oriented programming and software testing. This module divided the contents of programming fundamentals, identifying which the software testing contents could be applied to the programming contents. Thus, since the beginning of the CS undergraduate course, students are exposed to an integrated perspective exploring the contents and fundamentals of programming and software testing.

Code Hunt [24] is a game that also explores the software testing education with programming. In this game, the players must find a code fragment observing the expected result for the code fragment uncovered. The Code Hunt offers to the user a faulty code and some tips on what should be the result of the program when it is correct. For the code to be considered correct, the player must use several test cases [25].

As the code fragment is written the test cases change, enabling that the players obtain new experiences. The game Code Hunt teaches programming using software testing techniques. This technique is known like Test Driven Development (TDD), in which developers guided test cases produce code that can be validated by the tests [24].

Others initiatives have been proposed for the integration of software testing contents with others disciplines. Among them, we find the tool called ProgTest, which is an environment for the submission and evaluation of practical exercises based on the software testing activities. The ProgTest environment [3] evaluates practical exercises written in programming languages Java and C. For the evaluation of the exercises, this tool uses the functional, structural and fault-based test criteria. By using this tool, the teachers can automate the process of evaluating the codes and the test case sets that, in general, is carried out manually by the teacher. [3].

Web-CAT also is a tool that was developed to support the software testing education with programming [3]. This tool have the goal to support the submission and evaluation of practical exercises of programming assisting the test driven development [26]. Web-CAT evaluates the exercises that was submitted analyzing the following criteria: i) readability/project, analyzed manually by an assistant teacher; ii) style/codification, analyzed automatically by static analysis tools; and iii) test/correction, analyzed automatically by software testing tools, among the programming languages included, there are Java, C, C++ and Pascal [26].

Marmoset is an environment for the submission of

programming practical exercises based on software testing activities. The main goal of this environment was to improve the students learning in relation to programming contents and to know how students learn to program. Test cases used to test the programs submitted to Marmoset can be of four types:

i) student test, provided by the students; ii) public testing, provided to students before they begin their exercises; iii) release test, made available by teachers in specific conditions or after the end of the deadline; and iv) secret tests, they are written by the teacher and made available only at the end of the deadline of the exercises [27].

D. Educational Games to Support the Software Testing Education

Educational games raised the interest of educational technology researchers because there was a large increase of the video game industry in recent years. In addition, many researchers argue that these tools can facilitate the learning of concepts and ideas, because these games motivate and provide pleasures to the players while they play and learn. Therefore, we perceived that the educational games allow their users obtaining knowledge combined with the fun [16]. In this context, some educational games have been proposed to support software testing education, for instance: iTest Learning [16], Jogo das 7 Falhas [28], JETS [29], U-TEST [30], iLearnTest [31], among others.

However, the educational games previously referred to, dealt only with the contents of functional test and do not have an integrated teaching methodology with other courses. Therefore, the students do not have an integrated vision of software testing concepts with other disciplines, compromising their skills. This fact have contributed to the lack of an adequate “test culture” to the development of software products, since they did not have the habit of adequately testing their first programs.

This perspective has motivated us to develop an educational game that would address the teaching of software testing together with the teaching of programming fundamentals. The idea is to provide the students an integrated teaching of software testing contents with other disciplines of the Computer Science curricula. Thus, the students would have a better perspective of integrating testing in the scope of software product development. This would improve the students “test culture”. The game will address the functional test, structural test and fault-based test, providing a more motivating environment to the students.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we identified how software testing education has been approached in CS undergraduate courses in the most relevant universities in Brazil and abroad. We performed an analysis in the CS reference curricula proposed

by SBC and ACM as well as in the CS curricula of the most relevant universities in Brazil and in the world. We observed that, in general, software testing contents are not addressed in an integrated perspective with other disciplines. It can be observed, with few exceptions, that in the CS courses, either in Brazil or abroad, software testing is addressed only in the disciplines of Software Engineering and/or Programming Fundamentals. Few lectures are assigned to the teaching of software testing, providing to students only an overview of the contents of this area.

We identified 11 different approaches to support software testing education, among them: Educational Games, Integrated Teaching of Software Testing Teaching and Programming, Educational Modules and Test Driven Development. In this paper we presented the two main tendencies in software testing education: i) integration of software testing contents teaching with other disciplines along the CS undergraduate courses; and ii) use of educational games in software testing education. We identified tools and games for supporting these tendencies of software testing education.

It should be highlighted that it would be interesting to integrate software testing education with other disciplines along the CS undergraduate courses, with projects and tasks that would lead the students to explore in an integrated perspective the skills and competences developed in the course, including software testing. In this direction, it would be mandatory to revisit and modify the reference curricula proposed by SBC and ACM, since they are provide crucial directions and elements for the definition, implementation and evaluation of CS undergraduate courses.

Since software testing is considered a difficult discipline to be taught only by lectures and theoretical classes, the development of tools and environments to facilitate software testing education is mandatory. This scenario motivated the development of an educational game that will address the software testing education along with programming teaching, combining the two approaches more used to support the software testing education: Educational Games and Integrated Teaching of Software Testing and Programming.

REFERENCES

- [1] M. E. Delamaro, J. C. Maldonado, and M. Jino, *Introdução ao Teste de Software*. Elsevier, 2007.
- [2] R. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw Hill, Inc., 2010.
- [3] D. M. de Souza, J. C. Maldonado, and E. F. Barbosa, "Aspectos de desenvolvimento e evolução de um ambiente de apoio ao ensino de programação e teste de software," in *Anais do Simpósio Brasileiro de Informática na Educação*, 2012.
- [4] J. Smith, J. Tessler, E. Kramer, and C. Lin, "Using peer review to teach software testing," in *Proceedings of the ninth annual international conference on International computing education research*. ACM, 2012.
- [5] SBC, "Currículo de referência para cursos de graduação em bacharelado em ciência da computação e engenharia de computação," in *Sociedade Brasileira de Computação*, 2005.
- [6] L. Cassel, A. Clements, G. Davies, M. Guzdial, R. McCauley, A. McGettrick, B. Sloan, L. Snyder, P. Tymann, and B. W. Weide, "Computer science curriculum 2008: An interim revision of cs 2001," in *Association for Computing Machinery*, 2008.
- [7] R. J. LeBlanc, A. Sobel, J. L. Diaz-Herrera, T. B. Hilburn et al., *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. IEEE Computer Society, 2006.
- [8] C. Mao, "Towards a question-driven teaching method for software testing course," in *International Conference on Computer Science and Software Engineering*. IEEE, 2008.
- [9] H. Topi, J. S. Valacich, R. T. Wright, K. Kaiser, J. F. Nunamaker Jr, J. C. Sipior, and G.-J. de Vreede, "Is 2010: Curriculum guidelines for undergraduate degree programs in information systems," *Communications of the Association for Information Systems*, 2010.
- [10] P. H. D. Valle, E. F. Barbosa, and J. C. Maldonado, "Um mapeamento sistemático sobre ensino de teste de software," in *Anais do XXVI Simpósio Brasileiro de Informática na Educação*, 2015.
- [11] SBC, "Currículo de referência para cursos de licenciatura em computação," in *Sociedade Brasileira de Computação*, 2002.
- [12] SBC., "Currículo de referência para cursos de graduação em bacharelado em sistemas de informação," in *Sociedade Brasileira de Computação*, 2003.
- [13] ACM, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, 2013.
- [14] W. Wong, A. Bertolino, V. Debroy, A. Mathur, J. Offutt, and M. Vouk, "Teaching software testing: Experiences, lessons learned and the path forward," in *Conference on Software Engineering Education and Training*, 2011.
- [15] E. Barbosa and J. Maldonado, *E-Infrastructures and Technologies for Lifelong Learning: Next Generation Environments*, 2011, ch. Collaborative development of educational modules: a need for lifelong learning.
- [16] V. Farias, C. Moreira, E. Coutinho, and I. S. Santos, "itest learning: Um jogo para o ensino do

- planejamento de testes de software.” in *V Fórum de Educação em Engenharia de Software*. Natal: Simpósio Brasileiro de Engenharia de Software, 2012.
- [17] S. H. Edwards, “Teaching software testing: Automatic grading meets test-first coding,” in *Conference on Object-oriented Programming, Systems, Languages, and Applications*. ACM, 2003.
- [18] D. M. de Souza, M. H. da Silva Batista, and E. F. Barbosa, “Avaliação de qualidade de um ambiente de apoio ao ensino de programação,” *Revista Novas Tecnologias na Educação*, vol. 12, 2014.
- [19] E. Barbosa, M. Silva, C. Corte, and J. Maldonado, “Integrated teaching of programming foundations and software testing,” in *Frontiers in Education Conference*, 2008.
- [20] C. K. D. Corte and J. C. Maldonado, “Ensino integrado de fundamentos de programação e teste de software,” Master’s thesis, Universidade de São Paulo. Instituto de Ciências Matemáticas e de Computação, 2006.
- [21] A. Vincenzi, W. Wong, M. Delamaro, and J. Maldonado, “Jabuti: A coverage analysis tool for java programs,” in *Simpósio Brasileiro de Engenharia de Software*, 2003.
- [22] Y.-S. Ma, J. Offutt, and Y. R. Kwon, “Mujava: An automated class mutation system: Research articles,” *Software Testing, Verification and Reliability*, 2005.
- [23] D. Barbosa, W. Andrade, P. Machado, and J. Figueiredo, “Spaces—uma ferramenta para teste funcional de componentes,” in *Simpósio Brasileiro de Engenharia de Software*, 2004.
- [24] N. Tillmann, J. Bishop, N. Horspool, D. Perelman, and T. Xie, “Code hunt - searching for secret code for fun,” *Proceedings of the 7th International Workshop on Search-Based Software Testing*, June 2014.
- [25] N. Tillmann, P. de Halleux, T. Xie, and J. Bishop, “Code hunt: Gamifying teaching and learning of computer science at scale,” *Conference on Learning at Scale*, 2014.
- [26] S. H. Edwards and M. A. Perez-Quinones, “Web-cat: automatically grading programming assignments,” in *Conference on Innovation and Technology in Computer Science Education*. ACM, 2008.
- [27] J. Spacco, W. Pugh, N. Ayewah, and D. Hovemeyer, “The marmoset project: an automated snapshot, submission, and testing system,” in *Symposium on Object-oriented programming systems, languages, and applications*. ACM, 2006.
- [28] L. L. Diniz and R. L. S. Dazzi, “Jogo das sete falhas: Um jogo educacional para apoio ao ensino do teste caixa preta,” in *Anais do Computer on the Beach*, 2011.
- [29] T. G. da Silva and F. M. Muller, “Jogos sérios em mundos virtuais: uma abordagem para o ensino-aprendizagem de teste de software,” Master’s thesis, Universidade Federal de Santa Maria, 2012.
- [30] M. Thiry, A. Zoucas, and A. C. da Silva, “Empirical study upon software testing learning with support from educational game,” in *International Conference on Software Engineering and Knowledge Engineering*, 2011.
- [31] T. P. B. Ribeiro and A. C. R. Paiva, “ilearntest: Jogo educativo para aprendizagem de testes de software,” Master’s thesis, Faculdade de Engenharia da Universidade do Porto, 2014.