

**Universidade do Estado do Rio de Janeiro
Instituto de Matemática e Estatística
Departamento de Informática e Ciência da
Computação**

**InspectorX: Um Jogo para o Aprendizado em
Inspeção de Software**

Autor: Henrique Alberto Brittes Pötter

Orientador: Marcelo Schots de Oliveira

Orientadora: Vera Maria Benjamim Werneck

RIO DE JANEIRO

Dezembro/2012

INSPECTORX: UM JOGO PARA O APRENDIZADO EM INSPEÇÃO DE SOFTWARE

Henrique Alberto Brittes Pötter

Monografia submetida ao corpo docente do Instituto de Matemática e Estatística da Universidade do Estado do Rio de Janeiro - UERJ, como parte dos requisitos necessários à obtenção do grau de Bacharel em Informática e Tecnologia da Informação.

Banca Examinadora:

Prof^o : _____
Marcelo Schots de Oliveira - Orientador
M.Sc., COPPE/UFRJ

Prof^o : _____
Vera Maria Benjamim Werneck
D.Sc, IME/UERJ

Prof^a: _____
Ana Letícia de Cerqueira Leite Duboc
D.Sc. IME/UERJ

Rio de Janeiro, 20 de dezembro de 2012.

A toda a sociedade, em cujos ombros eu me apoio.

AGRADECIMENTOS

Agradeço aos meus pais, que sempre estiveram ao meu lado e me deram condições para atingir os meus objetivos. Sou grato ao professor e orientador Marcelo Schots, que, desde que participei como discente na disciplina de Engenharia de Software por ele ministrada, me incentivou e orientou na elaboração do presente trabalho e na publicação de dois artigos, tendo sido experiências extremamente ricas, pelas quais serei eternamente grato. À professora Vera Werneck, que fez comentários que me auxiliaram na organização do trabalho, me ajudou no experimento do InspectorX e permitiu o desenvolvimento e finalização da minha monografia. Agradeço também à professora Letícia Duboc, por ter tornado possível o experimento realizado com o InspectorX em sala de aula e por ter prontamente aceito o convite para participar da banca examinadora. Por fim, agradeço a toda a UERJ e seus docentes do Instituto de Matemática e Estatística, os quais me deram a base de meus conhecimentos na área da ciência da computação e que, indiretamente, contribuíram para esta monografia, como professores, orientadores e exemplos a serem seguidos.

RESUMO

A inspeção de software consiste em verificar artefatos do desenvolvimento de software, à procura de defeitos ou anomalias, seguindo procedimentos bem estruturados (Fagan 1976). Como a etapa de detecção dos defeitos pelos inspetores é essencial para o sucesso do processo, torna-se imperativo que os inspetores estejam aptos a executar sua função, ou seja, identifiquem os defeitos de maneira rápida e eficiente.

Com o intuito de simular a etapa de identificação de defeitos por inspetores, permitindo que ganhem vivência, de maneira gradual, na atividade de inspeção, foi desenvolvido o jogo InspectorX (Pötter e Schots 2011) (Pötter *et al.* 2012). O jogo se enquadra na categoria de *serious games*, e propõe-se como uma maneira de treinamento em inspeção de software, possuindo níveis de dificuldade variados e modos de jogo distintos, para prover a flexibilidade na interação com a informação, conforme apresentado pela teoria da carga cognitiva.

Para facilitar o desenvolvimento de questões e o uso de taxonomias na classificação dos defeitos presentes nos artefatos, foi desenvolvida adicionalmente uma aplicação para a criação e o gerenciamento dos novos artefatos e taxonomias.

O jogo InspectorX contou com um estudo piloto, que demonstrou indícios positivos em sua utilização e apontou algumas necessidades de melhoria. A partir do estudo executado, foi possível observar que deve haver uma melhor sistematização na definição do contexto das questões e do método utilizado para se destacar os defeitos nos artefatos, pois podem afetar a qualidade do aprendizado; além disso, com base nas respostas dos participantes, percebeu-se que, para aumentar a imersão no papel de inspetor, trechos de artefatos com defeitos podem não ser suficientes, sendo necessário fornecer artefatos mais completos.

SUMÁRIO

1. Introdução	9
2. Revisão da literatura	11
2.1 Inspeção de software	11
2.2 Educação em Engenharia de Software	13
2.3 Aprendizado em Inspeção de Software	14
3. A Abordagem	16
3.1 O Jogo InspectorX	16
3.2 Trabalhos Relacionados	18
3.3 Questões da Pesquisa	20
3.4 As Questões do Jogo	21
3.5 Modos de Jogo	22
3.3.1 O Modo <i>Full inspection process</i>	23
3.3.2 O Modo <i>Defect crawler</i>	27
3.6 O Administrador de Questões	29
4. Implementação da Abordagem	34
4.1 Arquitetura	34
4.2 Interface do jogo InspectorX	38
4.3 Interface do Administrador de Questões	38
5. Avaliação da Abordagem	40
5.1 Planejamento	40
5.2 Execução	41
5.3 Análise de Dados	42
5.3.1 Dados Quantitativos	42
5.3.2 Dados Qualitativos	44
5.4 Ameaças à Validade	46
6. Conclusões	48
APÊNDICE A - Instrumentos utilizados na avaliação do jogo no contexto educacional de inspeção.	52
A.1 Formulário de Consentimento	52
A.2 Questionário de Caracterização	54
A.3 Questionário <i>Follow-up</i>	56
A.4 Questões utilizadas na execução do estudo	57

Lista de Figuras

1. Inspeções de Software nos Diferentes Artefatos (adaptada de [Kalinowski et al. 2004])	11
2. Etapas e papéis do processo de inspeção (adaptado de [Kalinowski et al. 2004])	12
3. Cobertura do jogo InspectorX, em relação às etapas do processo de [Sauer et al. 2000].	17
4. Relação pedagógica entre games, simulação e aprendizado (retirado de [Ulicsak e Wright 2010])	18
5. Ranking de pontuação do InspectorX	22
6. Tela de seleção dos modos de jogo	23
7. Fluxo de Atividades (<i>FullInspectionProcess</i>).	24
8. Tela de administração de partidas.	26
9. Tela de criação de uma partida.	26
10. Inspeção dos artefatos por um jogador no papel de inspetor, no modo <i>Full Inspection Process</i> .	27
11. Fluxo de Atividades (<i>DefectCrawler</i>).	28
12. Tela de administração de taxonomias.	29
13. Janela de criação de uma nova taxonomia.	30
14. Janela de criação de um novo tipo de defeito.	30
15. Tela de gerenciamento de questões.	32
16. Tela da interface da criação de uma nova questão.	33
17. Tela da interface da seleção do tipo de defeito presente no trecho selecionado.	33
18. Diagrama arquitetural do jogo InspectorX	34
19. Diagrama de implantação do jogo InspectorX	35
20. Modelo Relacional do Banco de Dados	37
21. Tela de acesso ao jogo InspectorX	38
22. Exemplo de demarcação de defeito	39
23. Gráfico da eficiência dos grupos no experimento	43
24. Gráfico da eficácia dos grupos no experimento	43
25. Gráfico da relação de participantes, que qualificaram o jogo com potencial de imersão, no papel de inspetor.	44
26. Tela do InspectorX do jogo no modo Defect crawler no nível fácil.	45

Lista de Tabelas

1. Níveis de dificuldade do jogo InspectorX.	21
2. Defeitos de requisitos, adaptados de (Shull 1998).	31
3. Defeitos de código, adaptados de (Jones 2009).	31
4. Divisão dos grupos versus níveis de dificuldades.	40
5. Alocação dos grupos.	41
6. Definição das métricas utilizadas.	42

Lista de Siglas e Símbolos

SoC Separation of Concerns.
DAL *Data Access Layer.*
DAO *Data Acess Obejct.*
RIA *Rich Interface Applications.*

CAPÍTULO 1

INTRODUÇÃO

A inspeção de software consiste na verificação visual de um produto de software para detectar e identificar anomalias, erros e desvios dos padrões e especificações (IEEE 2008). Em outras palavras, o processo de inspeção consiste em verificar o trabalho já desenvolvido para encontrar defeitos ou problemas por meio de testes estáticos (Fagan 1976). Os defeitos no desenvolvimento do software podem ocorrer em artefatos de diferentes etapas do processo de desenvolvimento, podendo estar presentes, por exemplo, em um documento de requisitos ou no código fonte (IEEE 1990).

Há diversos relatos na literatura sobre os benefícios obtidos em organizações de software por meio da inspeção, tais como os descritos por Jones (1985), Fagan (1986), e Salger *et al.*(2009). No entanto, para que os benefícios sejam efetivamente alcançados, as pessoas envolvidas no ato de inspeção devem estar aptas a identificar os defeitos mais comuns, a fim de se obter resultados mais satisfatórios.

Segundo Fagan (1976), na medida em que uma equipe de desenvolvimento de software se conscientiza dos defeitos de alta ocorrência, tais defeitos tendem a serem evitados no momento do desenvolvimento dos próximos artefatos, economizando tempo que seria gasto em manutenções corretivas posteriores. Adicionalmente, de acordo com Kalinowski (2011), as organizações devem fazer do aprendizado com seus erros um instrumento para aumentar sua competitividade e qualidade.

Assim, para garantir a qualidade do processo de inspeção, é de suma importância que os conceitos e técnicas envolvidos nas atividades relacionadas à inspeção estejam bem assimilados e sejam aplicados devidamente. Para este fim, pode-se fazer uso de jogos educativos, pois estes podem ser aplicados como um complemento à capacitação de profissionais, tornando o aprendizado mais atrativo (Thiry *et al.* 2010).

Sendo assim, este trabalho apresenta o InspectorX, um jogo com o objetivo de incentivar e prover apoio ao aprendizado em inspeção de forma

lúdica, por meio do condicionamento da observação dos padrões apresentados (Pötter e Schots 2011).

O jogo dispõe dois modos: o modo *Full Inspection Process*, que simula algumas etapas essenciais do processo de inspeção, e o modo *Defect Crawler*, que atua apenas na inspeção de artefatos, mais especificamente na análise e identificação de defeitos em artefatos do processo de desenvolvimento, permitindo também a classificação destes defeitos de acordo com o tipo.

A fim de verificar a sua eficiência no aprendizado das etapas da inspeção, foi realizada uma avaliação do jogo (Pötter *et al.* 2012).

O restante da monografia está organizado da seguinte forma: o Capítulo 2 apresenta uma revisão da literatura relacionada à inspeção de software e aprendizado com o uso de jogos; o Capítulo 3 trata da abordagem do jogo; no Capítulo 4, são descritos os detalhes da implementação do jogo InspectorX; e, por fim, o Capítulo 5 relata o planejamento, execução e resultados de um estudo realizado para avaliar o jogo.

CAPÍTULO 2

Revisão da Literatura

2.1 Inspeção de Software

A inspeção de software é uma técnica bem estruturada, que começou sendo usada em hardware e passou em seguida para projeto (*design*) e código (Fagan 1976). Esta técnica consiste na verificação visual de um produto de software para detectar e identificar anomalias, erros e desvios dos padrões e especificações (IEEE 2008). Em outras palavras, o processo de inspeção consiste em verificar o trabalho já desenvolvido, seguindo práticas bem definidas e com resultados que não sofram interferência de viés, para encontrar defeitos ou problemas por meio de testes estáticos (Fagan 1976).

O processo de inspeção se diferencia de outros processos, como a revisão ou *walkthrough*, pois tem o objetivo de identificar defeitos seguindo processos bem definidos e imparciais, enquanto os outros dois procuram fazer avaliações dos artefatos com comentários de possíveis problemas (IEEE Std. 1028-1997).

Os defeitos nos artefatos de software podem ser introduzidos em diferentes etapas do processo de desenvolvimento (IEEE 610.12 1990). Sendo assim, existe a possibilidade de se realizar inspeções nos diferentes artefatos de software ao longo de tal processo, como mostra a Figura 1.

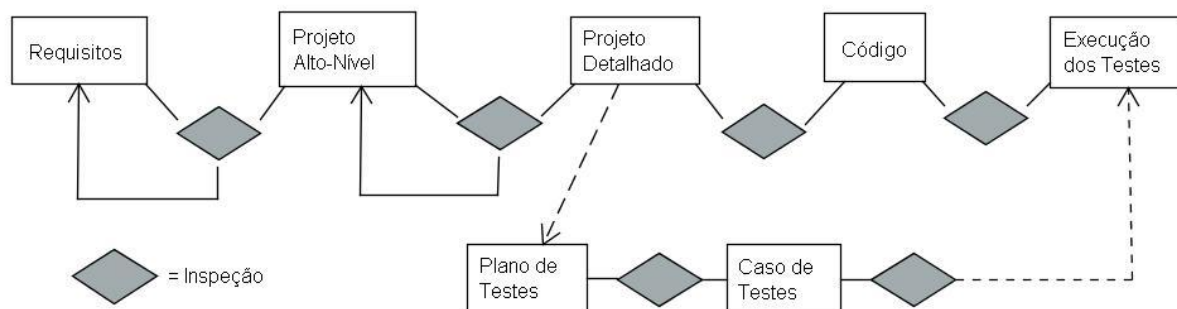


Figura 1. Inspeções de Software nos Diferentes Artefatos(adaptada de [Kalinowski *et al.* 2004])

Uma versão mais moderna do processo de inspeção, apresentada na Figura 2, possibilita o seu uso em equipes distribuídas, de acordo com (Sauer *et al.* 2000). Nesta versão, o processo é subdividido nas seguintes etapas:

- **Planejamento:** O moderador da inspeção define o escopo da inspeção, os artefatos a serem inspecionados, os inspetores envolvidos e técnicas a serem utilizadas.
- **Deteccção de defeitos:** Os inspetores selecionados inspecionam os artefatos definidos na etapa anterior, procurando defeitos, desvio de padrões ou outra inconsistência.
- **Coleção de defeitos:** O moderador revisa o trabalho realizado pelos inspetores e elimina redundâncias nas discrepâncias encontradas.
- **Discriminação de defeitos:** O moderador, o autor do artefato em questão e os inspetores se reúnem e discutem e classificam as discrepâncias encontradas, excluindo os falsos positivos.
- **Retrabalho:** Nesta etapa, o autor dos artefatos corrige as discrepâncias reais identificadas na etapa “Discriminação de defeitos”.
- **Continuação:** Após a etapa “Retrabalho”, o material corrigido é repassado para o moderador, que reavalia a correção e a qualidade do artefato que foi inspecionado.

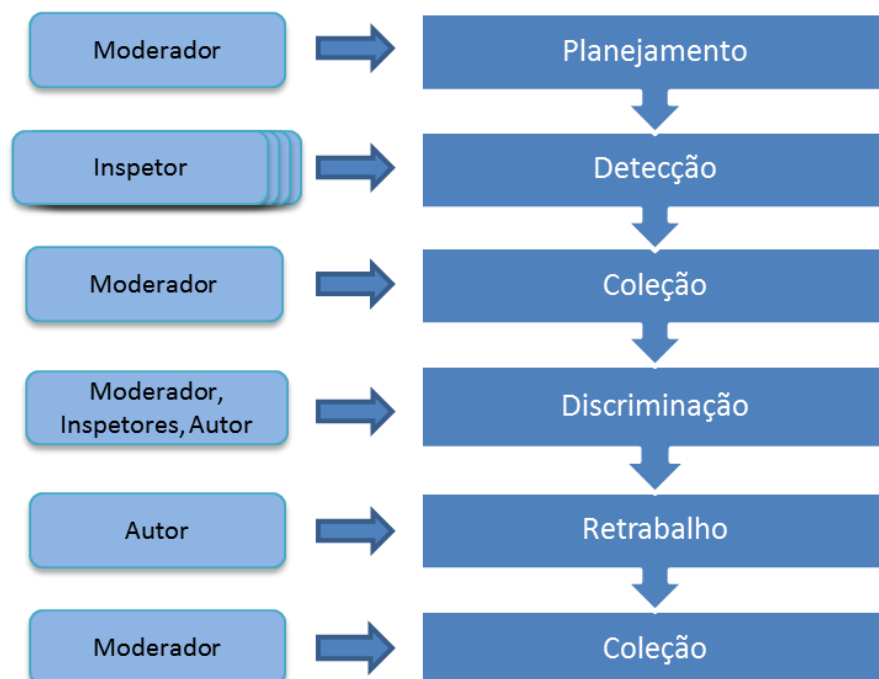


Figura 2. Etapas e papéis do processo de inspeção (adaptado de [Kalinowski *et al.* 2004])

Existe uma complexidade inerente na inspeção de software, devido aos seus procedimentos bem definidos e resultados imparciais, as quais demandam experiência e conhecimento por parte dos executores do processo. Com isto, para que o processo seja eficiente, é importante que todos os envolvidos adquiram conhecimento e sejam treinados em como atuar em seus devidos papéis ao longo do processo de inspeção.

2.2 Educação em Engenharia de Software

Na história da educação em engenharia de software, verifica-se a adoção de diversas novas abordagens de apoio ao ensino, como a integração entre disciplinas, jogos, projetos reais de software, entre outras (Silva *et al.* 2008). Este trabalho de Silva *et al.* (2008) também apresenta a importância de novas ferramentas digitais, ambientes e sistemas educacionais, tais como o WebCT (Goldberg *et al.* 1996), CoTeia (Arruda Jr. *et al.* 2002), Moodle (Moodle 2006) e Tidia/Ae (Projeto TIDIA (FAPESP), 2007), como plataformas de auxílio ao ensino.

Considerando essa nova tendência e a geração “Nativa Digital”, termo cunhado por Marc Prensky em 2007 (*apud* [Fernandes e Werner 2009]), vê-se a necessidade do desenvolvimento e da adaptação das linguagens e paradigmas de ensino. Neste quadro, os jogos educativos e digitais se enquadram como uma ferramenta de ensino, uma vez que se alinham com a linguagem da geração em questão (Fernandes e Werner 2009).

Segundo Thiry *et al.* (2010), embora haja diversos relatos sobre os benefícios obtidos com o uso de jogos no aprendizado, tal estratégia não tem sido muito explorada na prática de educação em engenharia de software, visto que as aulas muitas vezes são focadas no estudo de conceitos teóricos, de maneira expositiva, e exercitados em pequenos exemplos práticos.

Em Gee (2003), mostra-se a importância de criar ambientes que simulem cenários reais, expondo aos jogadores a informações contextualizadas no momento do seu uso prático, algo que ocorre com pouca frequência nas instituições de ensino. É enfatizado que as pessoas têm dificuldade em recordar informações que

lhes são passadas fora de contexto ou muito antes de serem efetivamente utilizadas. Também é discutida a importância de manter o desafio em um horizonte alcançável do usuário.

Porém, o aprendizado e a execução eficiente de tarefas pelo homem estão ligados à aquisição de novas estruturas do conhecimento (*schemata*) (Sweller 1994). Estas estruturas são formadas pela vivência de uma pessoa no ato de qualquer atividade, o que permite a ela responder de maneira mais eficiente ao reviver a mesma experiência. Em (Sweller 1994), é apresentada a teoria da carga cognitiva, a qual indica que o aprendizado deve ser gradual, devido à existência de um limite no momento da aquisição destas novas estruturas do conhecimento (*schemata*).

2.3 Aprendizado em Inspeção de Software

O processo de inspeção de software é composto de etapas bem definidas, o que o torna mais fácil de ser compreendido. A sua prática, no entanto, pode não ser tão simples e direta, pois envolve a necessidade de relação interpessoal e depende da experiência dos inspetores na fase de inspeção.

Neste âmbito, existem diversos tópicos que emergem como questionamentos ou temas de pesquisa no assunto, tais como:

- A relação com a qualidade de software, pois a detecção prematura de defeitos reduz o tempo em manutenção, os custos e os riscos (Kalinowski *et al.* 2004).
- O viés do autor do artefato, pois a fonte do erro humano no desenvolvimento dos artefatos (i.e., a fonte dos defeitos) pode estar relacionada com o efeito *Dunning-Kruger*, ou seja, um lapso de atenção ou a inexperiência (Kruger e Dunning 2009). Nesta discussão, pode-se inclusive justificar a necessidade de o inspetor não ser o próprio autor do artefato, em termos da imparcialidade necessária em relação aos defeitos.

- Técnicas de detecção de defeitos e métodos de localizar defeitos de forma mais eficiente (e.g., por meio de técnicas de leitura, como as apresentadas em (Basili *et al.* 1996) e (Conradi *et al.* 2003)).
- Administração, organização, catalogação e correção de defeitos encontrados ao longo do processo de inspeção.

Considerando essa natureza interdisciplinar, inerente à própria engenharia de software, cria-se uma dependência (i) do domínio de conceitos de desenvolvimento de software *a priori*, como os seus processos e ciclos de vida, e (ii) do entendimento de que a inspeção de software não é uma teoria, e sim uma prática necessária à qualidade do processo e do produto de software. Por isso, aulas de didática expositiva podem ser insuficientes para o esclarecimento destes pontos, dificultando o seu uso na prática.

Sendo assim, sob a perspectiva da teoria do esforço cognitivo de Sweller (Sauer *et al.* 2000), apresentada na seção 2.2, acredita-se que o aprendizado deve ser gradual, para que haja a maximização da assimilação dos conceitos relacionados ao domínio de inspeção de software.

Desta forma, considerando que é preciso experiência para boas inspeções e a forma mais eficiente de treinamento é a atuação prática na área de inspeção, optou-se pelo desenvolvimento de um jogo que permitisse, conforme o nível de experiência do jogador, o treinamento e aperfeiçoamento em tarefas de inspeção de artefatos do processo, sem os riscos de um cenário real.

CAPÍTULO 3

A Abordagem do InspectorX

Neste capítulo, são apresentados os fundamentos, jogabilidade e funcionalidade do jogo InspectorX.

3.1 O Jogo InspectorX

O jogo InspectorX (Pötter e Schots 2011) propõe auxiliar o ensino em um dos processos de qualidade de software, a inspeção de software (IEEE 2008) (Pressman 2001), considerando que a classificação dos defeitos e sua posterior análise ajudam a melhorar as estratégias de detecção de defeitos (Robinson *et al.* 2008). O jogo se encaixa na abordagem de ensino discutido no Capítulo 2, propondo tornar o aprendizado mais prazeroso e com todas as vantagens do meio digital e da internet.

O jogo tem como meta prover uma maneira prática, eficiente e lúdica de treinar a percepção dos defeitos mais comuns, por parte inspetores e desenvolvedores, e apresentar o processo de inspeção definido por Sauer (Sauer *et al.* 2000). A partir desta percepção, acredita-se que seja possível melhorar as práticas de desenvolvimento de software, evitando a recorrência dos defeitos no futuro.

O InspectorX foi construído com o objetivo de simular parcialmente um ambiente de inspeção de software e, assim, desenvolver a percepção do jogador na identificação e classificação de defeitos em artefatos de software, por meio do condicionamento da observação dos padrões apresentados no contexto da inspeção, enriquecendo as estruturas do conhecimento (*schemata*) (Sweller 1994).

Conforme apresentado no Capítulo 2, a versão moderna do processo de inspeção de Sauer (Sauer *et al.* 2000) não só possibilita o uso em equipes distribuídas, como também possui nomenclaturas mais descritivas dos processos de inspeção. Sendo assim, esta versão foi escolhida como base para o jogo, e a implementação cobre as etapas deste processo detalhadas na **Erro! Fonte de referência não encontrada..**

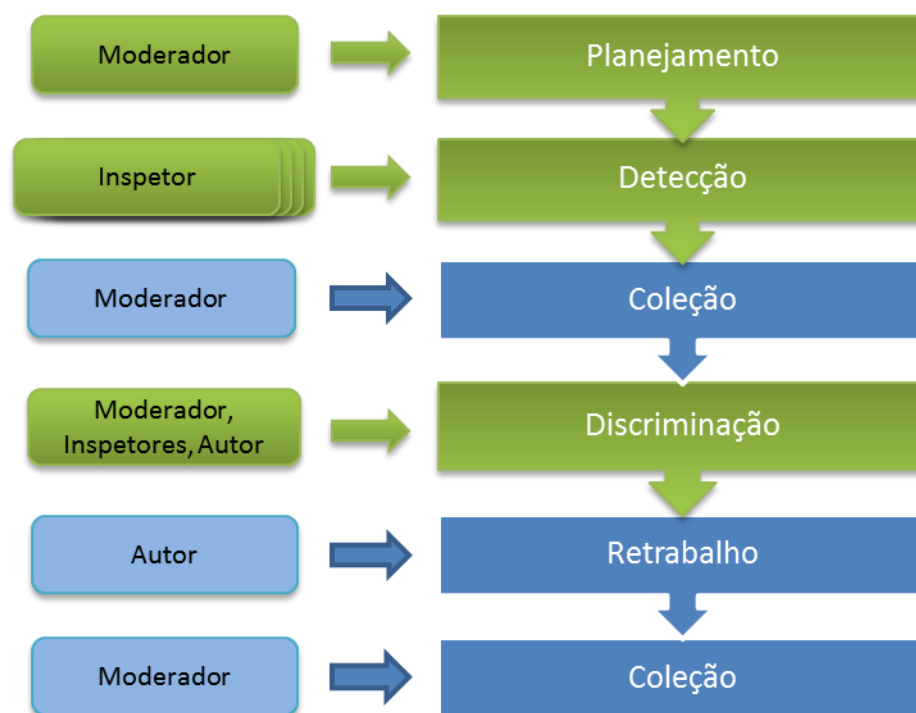


Figura 3. Cobertura do jogo InspectorX, em relação às etapas do processo de [Sauer *et al.* 2000].

Cabe ressaltar que a etapa de discriminação dos defeitos é parcialmente simulada na implementação do jogo, pois apenas o moderador decide quais discrepâncias são falsos positivos, não envolvendo os inspetores. No entanto, é possível utilizar a interface do jogo numa reunião entre o moderador, os inspetores e o autor do defeito, de forma que todos os papéis envolvidos nesta etapa possam executar esta etapa.

O InspectorX se encaixa na categoria de *serious game*, pois é um jogo que possui um objetivo educacional, e não apenas de entretenimento (Abt 1987). Jogos nesta categoria usam o potencial imersivo e de engajamento dos jogos em geral, com o intuito de desenvolver novas habilidades para fins específicos, envolvendo as áreas aprendizado, simulação e jogos (como pode ser visto na Figura 4). Além disso, eles possibilitam que os jogadores vivenciem situações que seriam difíceis ou impossíveis por questões de custo, tempo, logística e/ou segurança (Corti 2006).

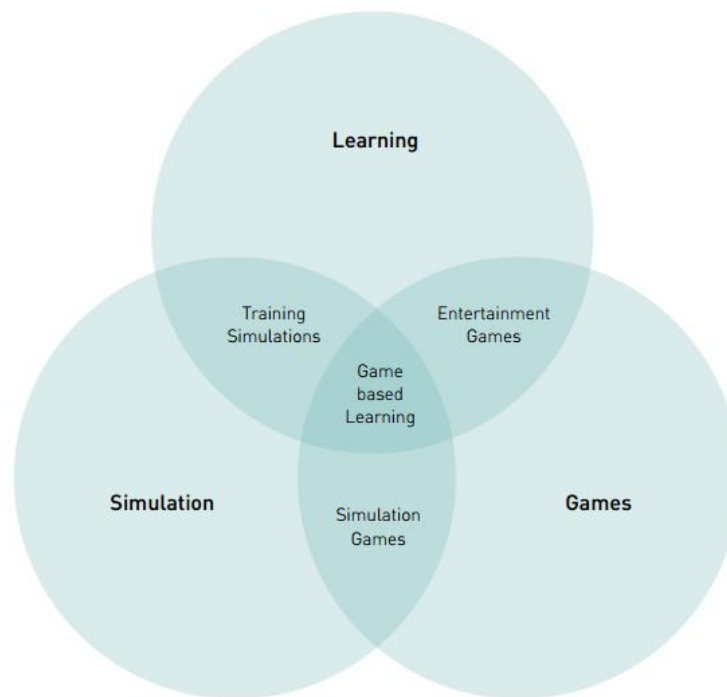


Figura 4.Relação pedagógica entre games, simulação e aprendizado (retirado de [Ulicsak e Wright 2010]).

3.2 Trabalhos Relacionados

Esta seção apresenta alguns trabalhos que também se utilizam de jogos para a educação em engenharia de software, e que serviram de inspiração para o presente trabalho. No início da elaboração deste trabalho, não foram encontrados trabalhos que utilizassem jogos para apoiar o processo de aprendizado em inspeção de software; mais recentemente, foi identificado um trabalho com objetivos semelhantes (Lopes *et al.* 2012), o qual se encontra ao final desta seção.

Em (Prikladnicki e von Wangenheim 2008), são apresentadas a motivação para o desenvolvimento de jogos educativos em engenharia de software, e mostram três exemplos de jogos voltados para gerência de projetos.

O jogo de cartas Problems and Programmers (Baker *et al.* 2003) visa auxiliar o ensino de engenharia de software por meio da simulação de processos de software. O jogo permite que o jogador perceba que inspeções proveem aumento de qualidade, mas impactam no tempo de entrega. Três tipos de impacto de defeito são considerados (simples, normais e graves), mas não há identificação ou classificação por parte do jogador. A forma de corrigir os defeitos varia conforme o impacto,

podendo ser desde a substituição por outra carta, no caso de defeitos simples, ou a seleção de novas cartas, no caso de defeitos graves.

No trabalho (McMeekin *et al.* 2009), são analisados os níveis de cognição associados a três diferentes técnicas de inspeção de código. Para a inspeção, foram utilizadas especificações em linguagem natural, diagramas de classes, o código do sistema e uma folha de papel, para o registro de defeitos. O jogo InspectorX (Pötter e Schots 2011) utiliza uma abordagem diferente: quando o nível de experiência do jogador em inspeção é baixo, os defeitos já são apresentados em tela, sendo necessária apenas a seleção do tipo de defeito, incentivando o aprendizado. Além disso, os erros e acertos são computados e corrigidos automaticamente.

Em (Thiry *et al.* 2010), são apresentados três jogos educativos que focam em áreas da melhoria de processo de software (Engenharia de Requisitos, Medição de Software, e Verificação e Validação de Software). O trabalho traz o resultado de experimentos para a avaliação da efetividade de aprendizagem, envolvendo alunos de graduação. Os experimentos comprovam a eficácia destes jogos para o ensino, mostrando que os jogos ajudam no entendimento dos conceitos envolvidos em cada área.

O jogo InspSoft (Lopes *et al.* 2012) tem uma proposta semelhante ao do InspectorX (apoiar o aprendizado em inspeção de software por meio de um jogo); no entanto, algumas características denotam algumas diferenças entre eles, tais como:

- O InspSoft possui uma etapa inicial de ensino do processo de inspeção por meio expositivo, tendo a atuação posterior do jogador, no papel de inspetor e durante a reunião de inspeção. Já no InspectorX, o foco é na atuação dos diferentes papéis no processo de inspeção, expondo as etapas ao longo do jogo de forma mais prática, tornando o aprendizado lúdico.
- O InspSoft possui como público alvo alunos de graduação, enquanto o InspectorX se propõe a ser utilizado também em empresas que procuram realizar treinamentos em inspeção de software.
- O jogo InspectorX não possui uma encenação fixa de uma situação de inspeção, ficando a cargo do administrador do jogo e dos outros jogadores atuar nos papéis da simulação de inspeção. O InspSoft procura criar um cenário fixo de inspeção com personagens que reagem com o jogador.

- As questões do jogo InspectorX são flexíveis e administráveis; no jogo InspSoft elas são fixas, compiladas junto ao executável do jogo.
- Em termos de implementação, a aplicação do jogo InspSoft é executada localmente com recursos fixos, enquanto o InspectorX é distribuído como cliente magro (*thinclient*), consumindo serviços web com um banco de dados centralizado para armazenamento das questões e dados de desempenho dos jogadores (conforme detalhado no capítulo 4). Embora haja uma dependência de conexão à Internet, isto traz a vantagem de que cada instância do jogo pode utilizar questões atualizadas e em maior variedade.

Apesar destas diferenças, ambos os jogos propõem uma forma de aprendizado nos processos da inspeção de software alinhada com a emergente geração nativa digital, apresentada no Capítulo **Erro! Fonte de referência não encontrada..**

3.3 Questões da Pesquisa

Para este trabalho, foram identificadas as seguintes questões de pesquisa:

- Questão principal
 - Como apoiar o aprendizado em inspeção de software?
- Questões secundárias:
 - O uso de jogos pode apoiar o aprendizado em inspeção?
 - A separação em níveis de dificuldade ajuda o aprendizado?
 - Os níveis de dificuldade definidos para o jogo estão de acordo com os níveis de dificuldade na escala de aprendizado, do ponto de vista dos inspetores/jogadores?
 - As características definidas para o InspectorX tornam o jogo mais atrativo ao jogador?
 - Como criar o estado de *flow*¹ durante o aprendizado (isto é, como tornar o aprendizado lúdico)?

¹O estado de *flow* é um estado ideal de motivação intrínseca, o qual a pessoa se encontra completamente imersa em sua atividade [Csíkszentmihályi&Csíkszentmihályi 1988].

- O fluxo de atividades definido para o jogo é capaz (e até que ponto) de simular etapas do processo de inspeção?

As questões de pesquisa foram utilizadas para direcionar as perguntas na obtenção de dados qualitativos e usadas na avaliação do jogo InspectorX (Capítulo 5).

3.4 As Questões do Jogo

Cada questão do jogo é composta de um artefato de software (e.g., um trecho de documento de requisitos ou de código fonte) que contém um ou mais defeitos, que devem ser corretamente identificados e classificados individualmente por um jogador (ou seja, o inspetor), de forma a acumular pontos.

Visando permitir um aumento gradual na complexidade da interação do jogador com o jogo, e tentando obedecer, em hipótese, à teoria do esforço cognitivo (Fagan 1976), o jogo varia gradualmente em complexidade de acordo com o nível de dificuldade, conforme apresentado na Tabela 1.

Tabela 1. Níveis de dificuldade do jogo InspectorX

Nível	Nº de defeitos	Pontuação (por defeito corretamente detectado)	Características
Básico	1	1	O defeito é exibido de forma destacada para o jogador; a função do jogador é apenas identificar qual o tipo de defeito, utilizando a lista que lhe é exibida.
Intermediário	1	2	A diferença entre este nível e o nível básico é que o jogador deve selecionar o trecho em que o defeito se encontra, sem indicações por parte do jogo; ao selecionar o defeito, o jogador deve classificá-lo, utilizando a lista exibida.
Avançado	> 1	2	Cada questão possui mais de um defeito, o jogador tem que selecionar todos os trechos que contenham defeitos e classificá-los individualmente.

Nos níveis intermediário e avançado, o jogo computa um ponto na identificação do trecho com defeito e mais um ponto na classificação correta.

O jogo dispõe também de um ranking separado por níveis de dificuldade, que mostra a posição dos jogadores ordenada pelos pontos acumulados, ao responder corretamente as questões, nos dois modos de jogo (Figura 5).

Resultado Final		
Usuario	Pontuacao	Email
henrique	2	henriquepotter.hp@gmail.com
teste1	1	teste@gmail.com
teste2	2	teste2@gmail.com
teste3	2	teste3@gmail.com
teste4	2	teste4@gmail.com
schots	1	marceloschots@yahoo.com.br

Voltar

Figura 5. Ranking de pontuação do InspectorX.

3.5 Modos de Jogo

Devido à complexidade inerente ao processo de inspeção, foram criados modos de jogo para flexibilizar a jogabilidade, o que permite uma aproximação gradual do jogador com os processos da inspeção, tendo partidas mais casuais ou mais imersivas.

O jogo apresenta dois modos: o *Full Inspection Process*, o qual simula o processo de inspeção de software como um todo (isto é, as etapas do processo cobertas pelo jogo, conforme apresentado na **Erro! Fonte de referência não encontrada.**), e o *DefectCrawler*, o qual possui foco apenas na atividade de inspeção de artefatos de software.

A tela de seleção do modo de jogo (Figura 6) é apresentada ao jogador após ele acessar o jogo, possibilitando a seleção do modo de jogo. Os modos de jogo são detalhados nas subseções a seguir.



Figura 6. Tela de seleção dos modos de jogo.

3.3.1 O modo *Full inspection Process*

Neste modo, os jogadores podem optar por um dos dois papéis essenciais do processo de inspeção (Sauer *et al.* 2000), a saber:

- **Moderador:** Neste papel, o jogador pode atuar duas atividades do processo de inspeção (Sauer *et al.* 2000), seguindo o fluxo das atividades apresentadas na Figura 7.

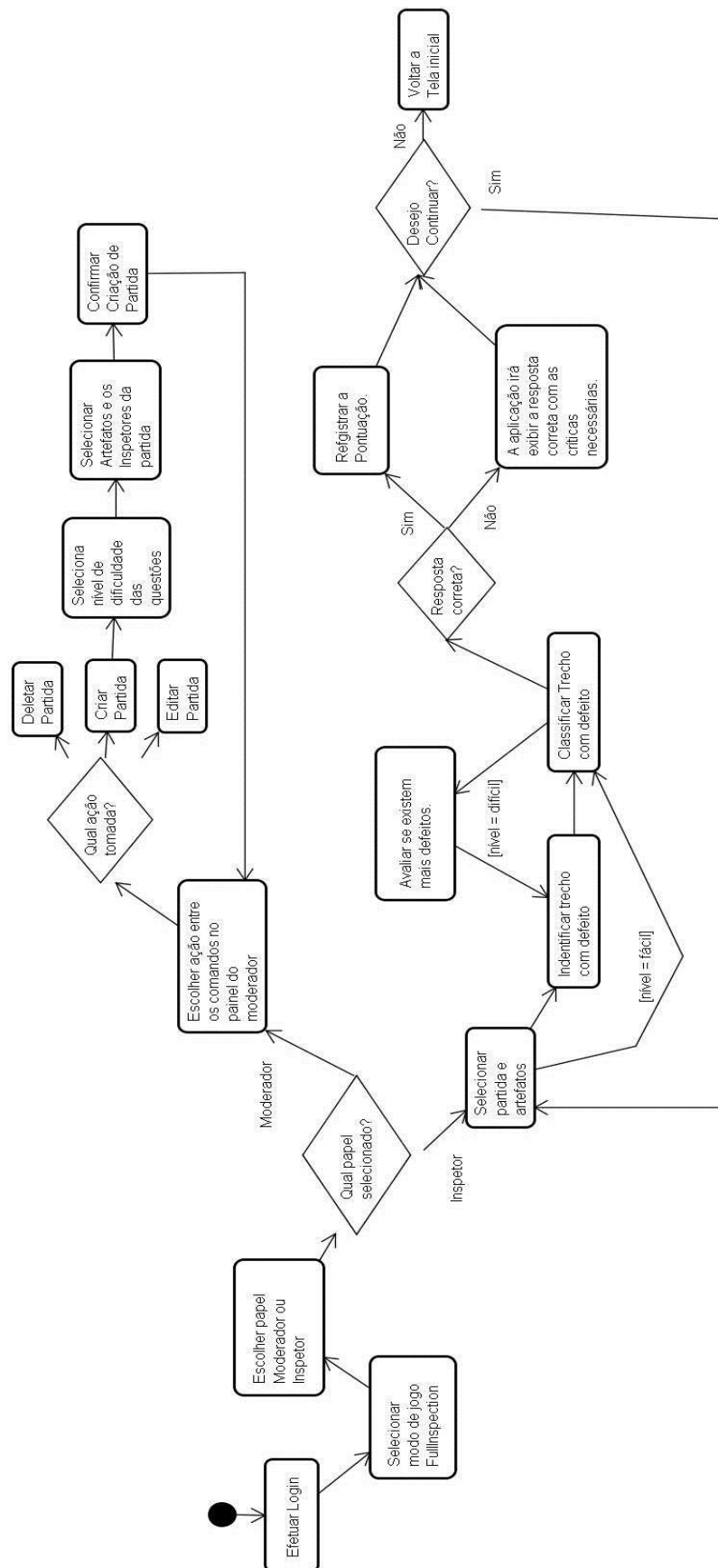


Figura 7. Fluxo de Atividades (*Full Inspection Process*)

- Planejamento da inspeção (Figuras 5 e 6), referente à etapa “Planejamento” (Sauer et al. 2000), na qual o jogador irá definir o escopo de artefatos e os inspetores, associando um ao outro (relacionamento muitos-para-muitos). Cada rodada de inspeção é denominada como uma partida.
- Avaliação da inspeção, referente à etapa “Discriminação de defeitos” (Sauer et al. 2000), na qual o jogador irá analisar os defeitos apontados pelos inspetores decidindo se foram corretamente identificados (nos casos em que a resposta diverge da pré-cadastrada pela abordagem).
- **Inspetor:** Neste papel, o jogador irá analisar e reportar defeitos em um determinado escopo de artefatos de software, referente à etapa “Detecção” (Sauer et al. 2000), definidos pelo moderador. O jogador terá uma lista de partidas, nas quais ele foi incluído por algum moderador, e deverá inspecionar todos os artefatos para terminar uma partida. Após avaliar os artefatos, estes poderão ser posteriormente validados pelo moderador da partida.

O moderador é avaliado quanto à classificação dos defeitos relatados pelos inspetores, apontando corretamente os falsos positivos. Já os inspetores são avaliados em relação à identificação correta dos defeitos nos artefatos de software.

A Figura 8 mostra a tela do jogador, atuando pelo papel de moderador, a qual permite o mesmo gerenciar todas as partidas já criadas, bem como criar novas partidas.

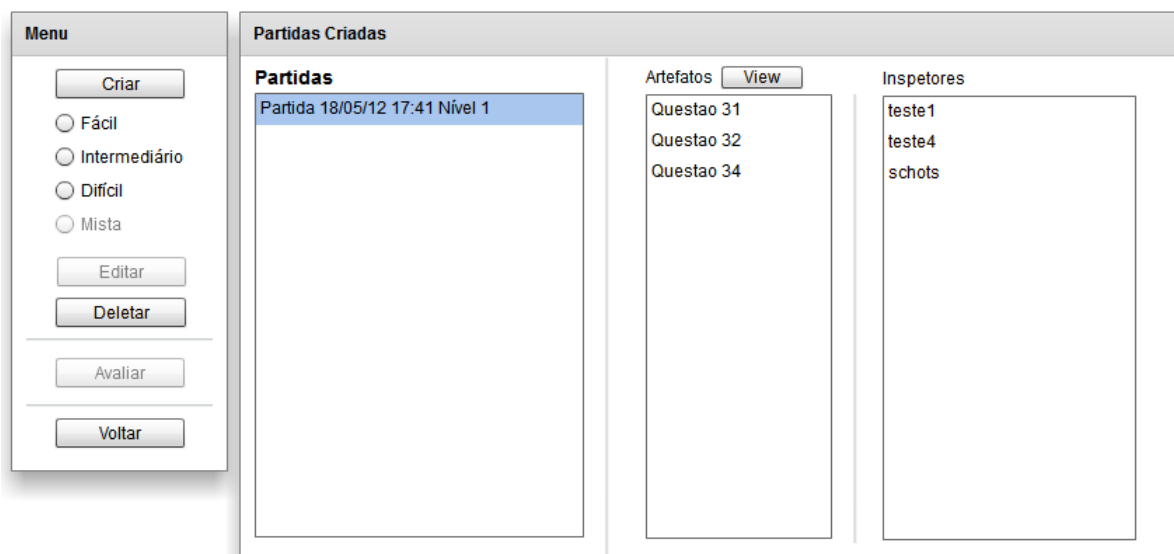


Figura 8. Tela de administração de partidas.

A Figura 9 ilustra a interface de criação de uma nova partida, onde permite o moderador seleccionar tanto os artefatos a serem inspecionados quanto os inspetores que atuarão sobre estes artefatos.

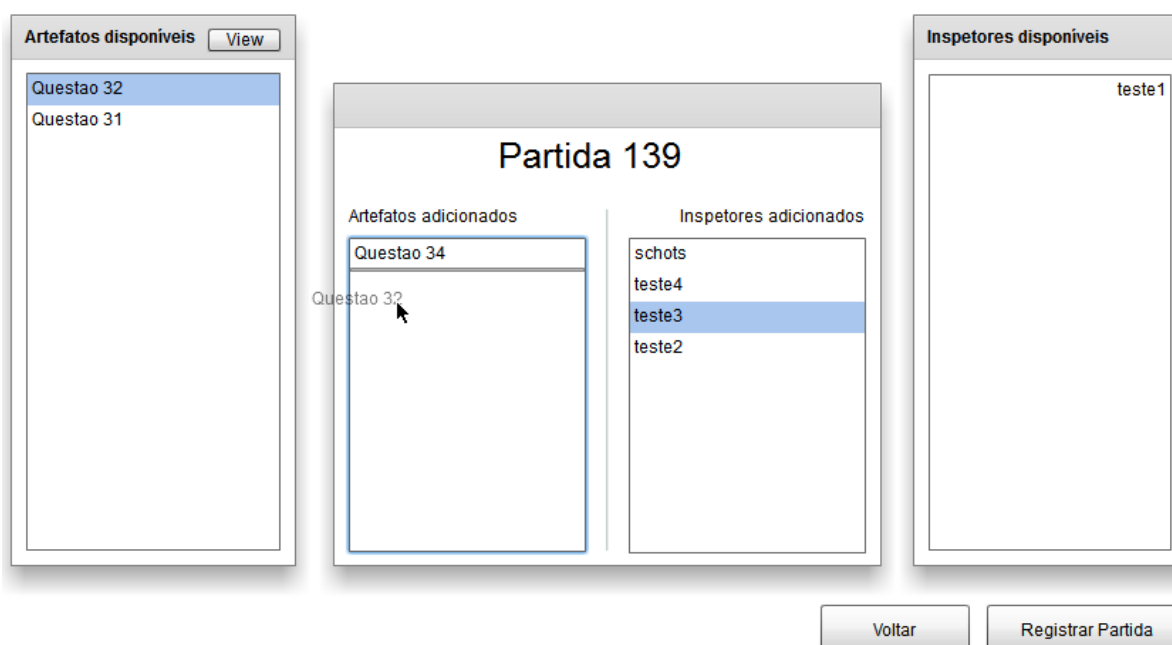


Figura 9. Tela de criação de partidas.

A interface dos inspetores, apresentada na Figura 10, possibilita ao jogador inspecionar todos os artefatos que foram incluídos por algum moderador e que ainda não foram inspecionados.

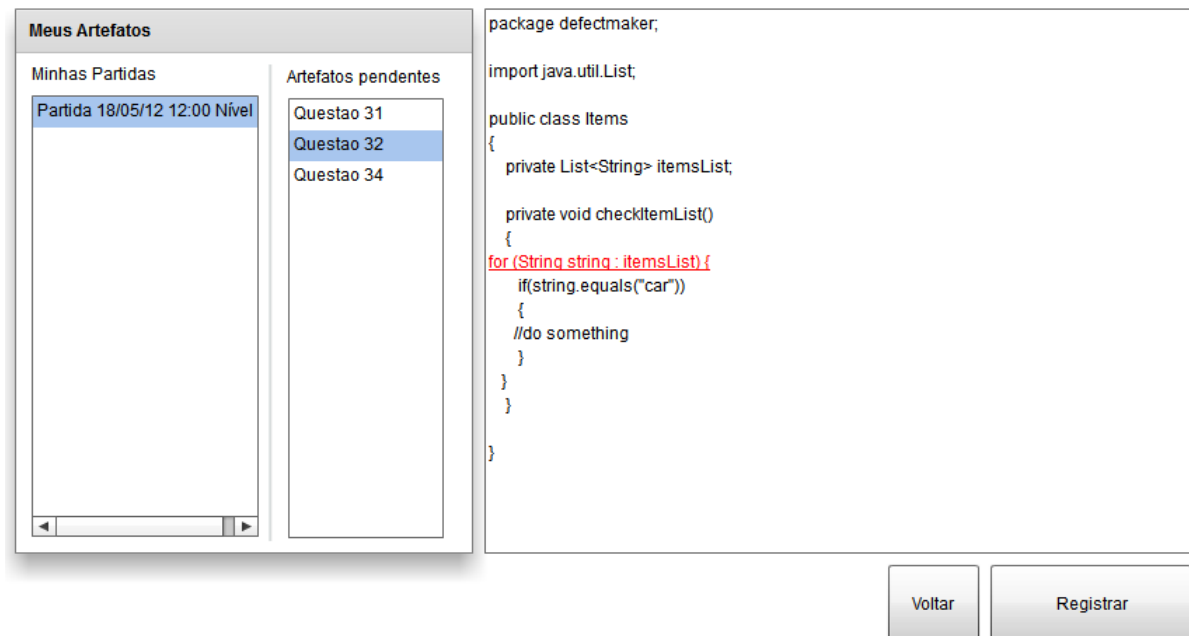


Figura 10. Inspeção dos artefatos por um jogador no papel de inspetor, no modo *FullInspectionProcess*.

3.3.2 Modo *Defect Crawler*

Neste modo de jogo, os jogadores apenas inspecionam os artefatos de software, com o objetivo de identificar defeitos e classificá-los, seguindo o fluxo de atividades apresentados na Figura 11. Cada defeito corretamente encontrado e classificado pontua de acordo com a Tabela 1, variando com o nível de dificuldade seguindo as seguintes regras:

- O jogador nunca inspeciona novamente o mesmo artefato, caso já tenha apontado os defeitos no mesmo (independente de ter errado ou acertado).
- Os artefatos são embaralhados usando o algoritmo de Fisher-Yates (Durstensfeld 1964). Este algoritmo permite embaralhar a lista de artefatos de maneira imparcial, e assim não favorece posições na lista.

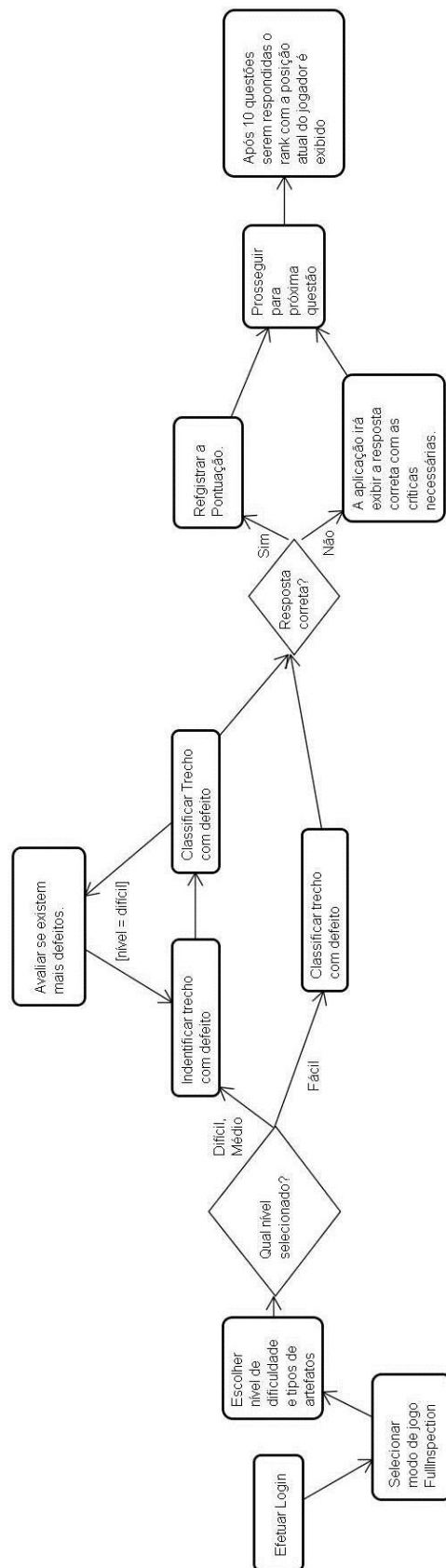


Figura 11. Fluxo de Atividades (*Defect Crawler*)

3.6 Administrador de Questões

Visando facilitar o gerenciamento e criação de novos artefatos de software, foi desenvolvida uma aplicação desktop usando a linguagem Java.

Os artefatos e seus respectivos defeitos, bem como novas taxonomias, podem ser cadastrados por qualquer pessoa, sendo esta, de preferência, experiente nos processos de inspeção. Assim, o administrador do jogo efetua o mapeamento entre o defeito e seu tipo, com base na taxonomia escolhida.

O módulo administrador de questões possibilita a customização de taxonomias (Figura 12) para classificação dos defeitos nas questões, assim como a criação e manutenção das questões. A criação de novos artefatos pode ser separada em 3 etapas, que são detalhadas a seguir.

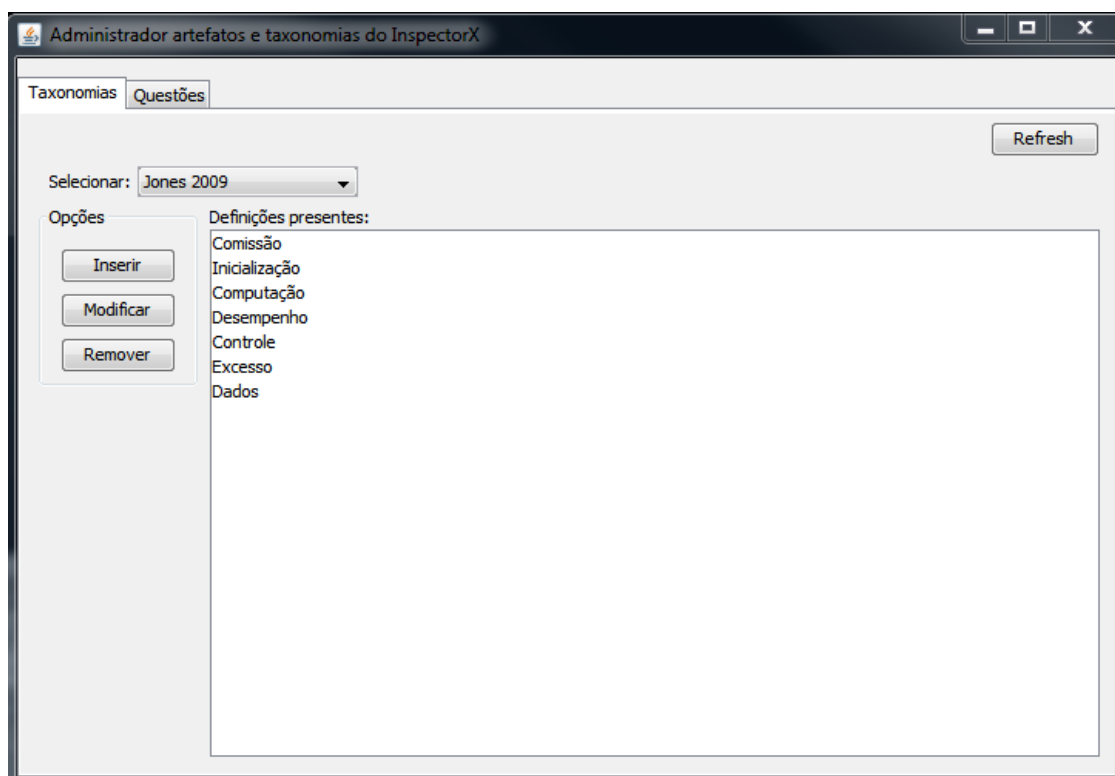
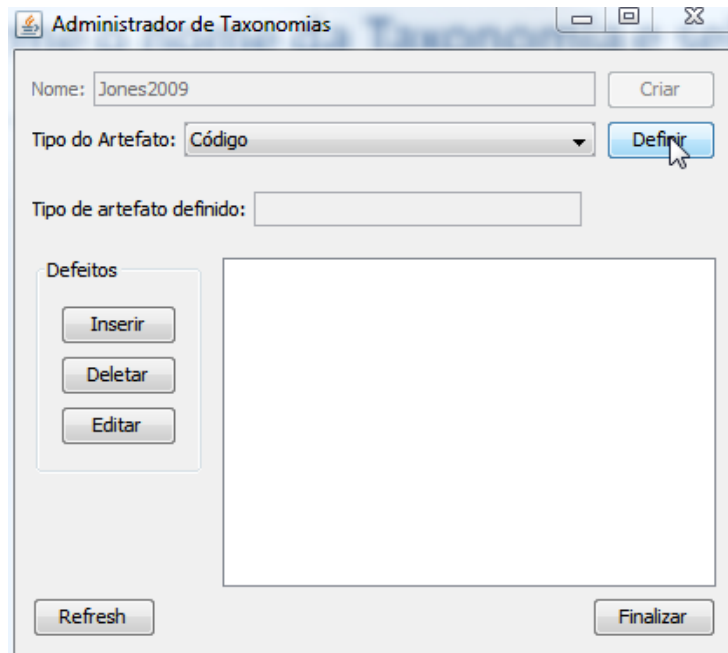


Figura 12. Tela de administração de taxonomias.

Etapas 1: Criação da Taxonomia

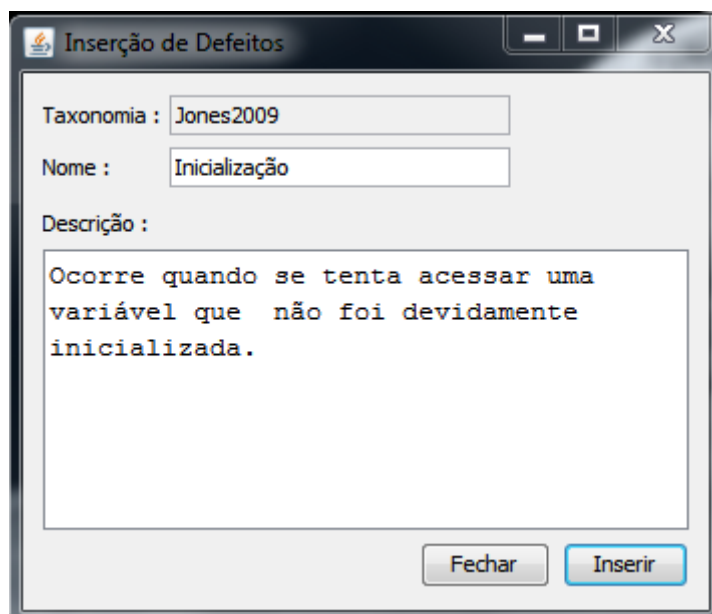
Nesta etapa devem ser definidos o nome da taxonomia, o tipo de artefato ao qual ela se refere (por exemplo, código fonte ou requisito) e os respectivos tipos de defeitos

por ela definidos (Figuras 13 e 14), pois eles serão usados na classificação dos defeitos presentes nos artefatos de software.



A janela intitulada "Administrador de Taxonomias" possui um formulário para criar uma nova taxonomia. No topo, há um campo "Nome:" com o valor "Jones2009" e um botão "Criar". Abaixo, há um campo "Tipo do Artefato:" com o valor "Código" e um botão "Definir". Segue-se um campo "Tipo de artefato definido:". Na parte inferior esquerda, há um grupo de botões "Defeitos" contendo "Inserir", "Deletar" e "Editar". Na parte inferior, há botões "Refresh" e "Finalizar".

Figura 13. Janela de criação de uma nova taxonomia de defeitos.



A janela intitulada "Inserção de Defeitos" mostra o detalhamento de um novo tipo de defeito. Ela contém campos para "Taxonomia:" (valor "Jones2009") e "Nome:" (valor "Inicialização"). Abaixo, há um campo "Descrição:" com o texto "Ocorre quando se tenta acessar uma variável que não foi devidamente inicializada.". Na parte inferior, há botões "Fechar" e "Inserir".

Figura 14. Janela de criação de um novo tipo de defeito (detalhamento da taxonomia).

Para a classificação dos defeitos por categoria, podem ser utilizadas quaisquer taxonomias existentes na literatura. A implementação atual permite que seja adicionada qualquer taxonomia para classificação dos defeitos desde que siga

o formato dos exemplos abaixo, assim como as taxonomias do trabalho de (Shull 1998) (Tabela 2) para requisitos e a de (Jones 2009) (Tabela 3) para código fonte.

Tabela 2. Defeitos de requisitos, adaptados de (Shull 1998)

Tipos de Defeito	Descrição
Omissão	Deve-se à omissão ou negligência de alguma informação necessária ao desenvolvimento do software.
Ambiguidade	Ocorre quando uma determinada informação não é bem definida, permitindo assim uma interpretação subjetiva, que pode levar a múltiplas interpretações.
Fato incorreto	Informações dos artefatos do sistema que são contraditórias com o conhecimento que se tem do domínio da aplicação.
Inconsistência	Ocorre quando duas ou mais informações são contraditórias entre si.
Informação estranha	Informação desnecessária incluída nos requisitos do software que esta sendo desenvolvido.

Tabela 3. Defeitos de código, adaptados de (Jones 2009)

Tipos de Defeito	Descrição
Comissão	Ocorre quando existe algum segmento de código que foi implementado incorretamente, i.e., cuja implementação é diferente do que foi especificado.
Inicialização	Ocorre quando se tenta acessar uma variável que não foi inicializada.
Computação	Similar ao defeito de comissão; ocorre quando um valor é definido erroneamente para uma variável.
Desempenho	Algumas rotinas executam comandos ou laços (<i>loops</i>) desnecessários.
Controle	Ocorre quando um comando de desvio condicional é usado de forma incorreta.
Excesso	Existem trechos de código irrelevantes e desnecessários.
Dados	Ocorre quando uma estrutura de dados é manipulada de forma incorreta (por exemplo, quando se tenta acessar um índice inexistente de um vetor/matriz).

Etapa 2: Criação dos Artefatos de Software

Após o cadastro de uma taxonomia, é possível cadastrar as questões do jogo, e antes da criação de uma questão, deve ser selecionada alguma das taxonomias previamente cadastradas (Figura 15).

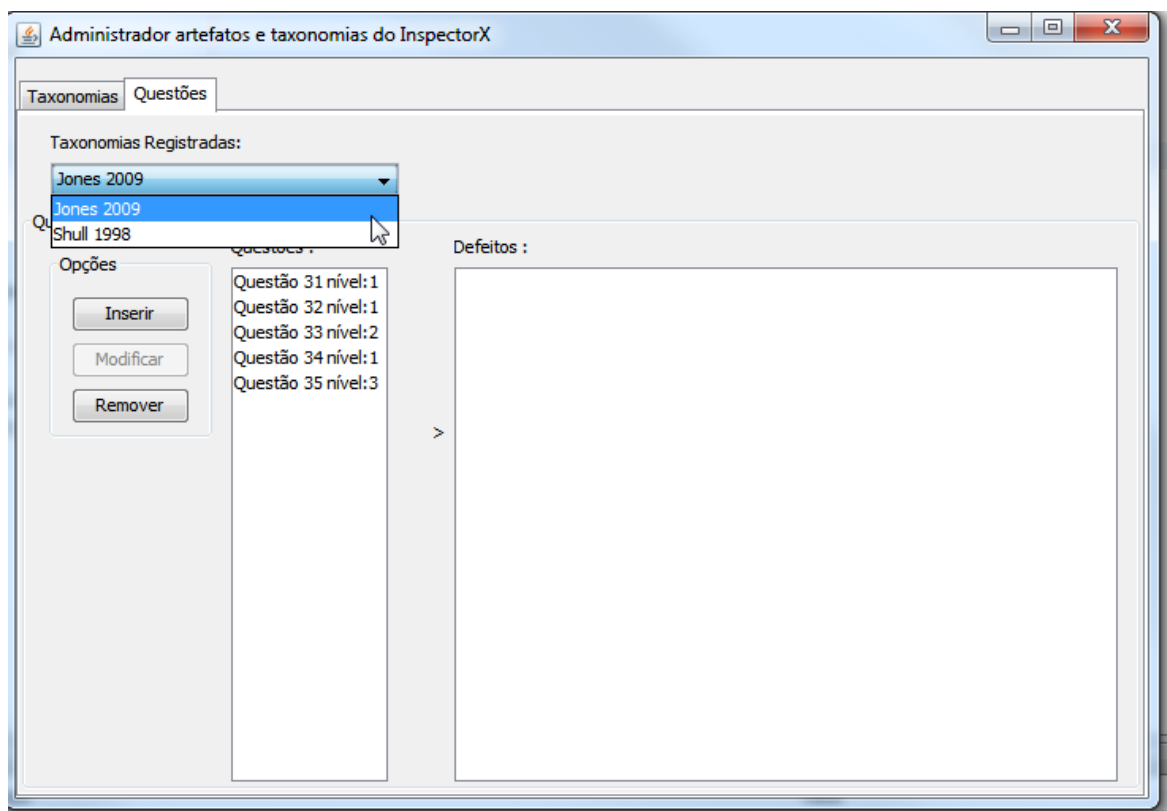


Figura 16. Tela de gerenciamento de questões

Após a seleção do tipo de taxonomia a ser usado para classificação dos defeitos da questão, basta selecionar a opção de inserir.

Etapas 3: Identificação dos defeitos no artefato

Após inserir o artefato de software, devem-se marcar quais trechos correspondem ao defeito (Figura 16), selecionando o seu tipo, sendo este pré-cadastrado na taxonomia vigente, e adicionar uma crítica descritiva do defeito (Figura 17). Esta crítica é apresentada ao jogador quando ele errar na identificação do tipo de defeito ou na seleção do trecho correto. Para que seja possível definir mais de um defeito no artefato por questão, basta selecionar o nível difícil.

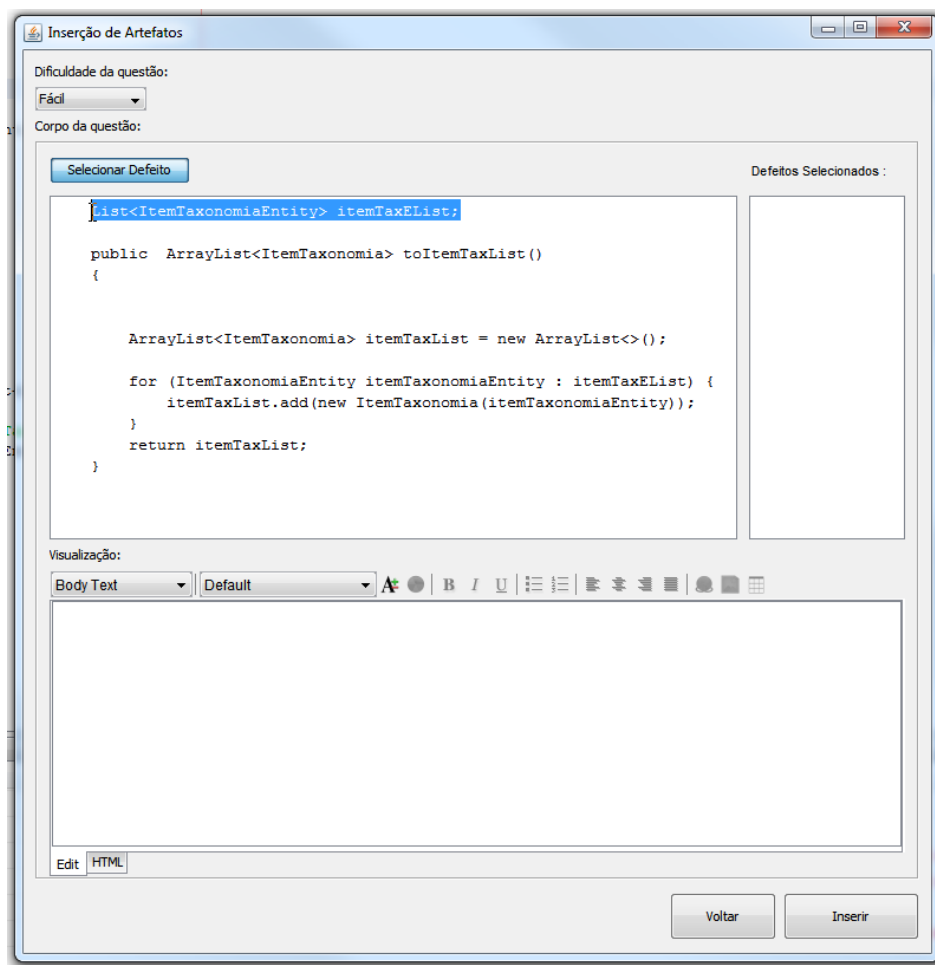


Figura 16. Tela da interface da criação de uma nova questão

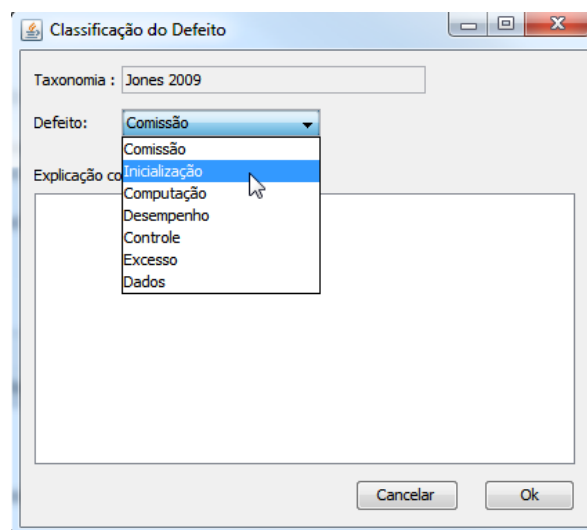


Figura 17. Tela da interface da seleção do tipo de defeito presente no trecho selecionado.

CAPÍTULO 4

Implementação da Abordagem InspectorX

Neste capítulo, são explicados os detalhes da implementação do jogo.

4.1 Arquitetura

Visando facilitar o desenvolvimento sem limitações de plataformas para o lado cliente, o jogo foi dividido em interfaces bem definidas que permitiram a aplicação do conceito de separação de interesses (SoC – *Separation of Concerns*), dividindo a implementação em três camadas e disponibilizando as funções do jogo como serviços, conforme exibido na Figura 18 e na Figura 19. Esta arquitetura facilitou, por exemplo, o desenvolvimento da aplicação de administração de questões.

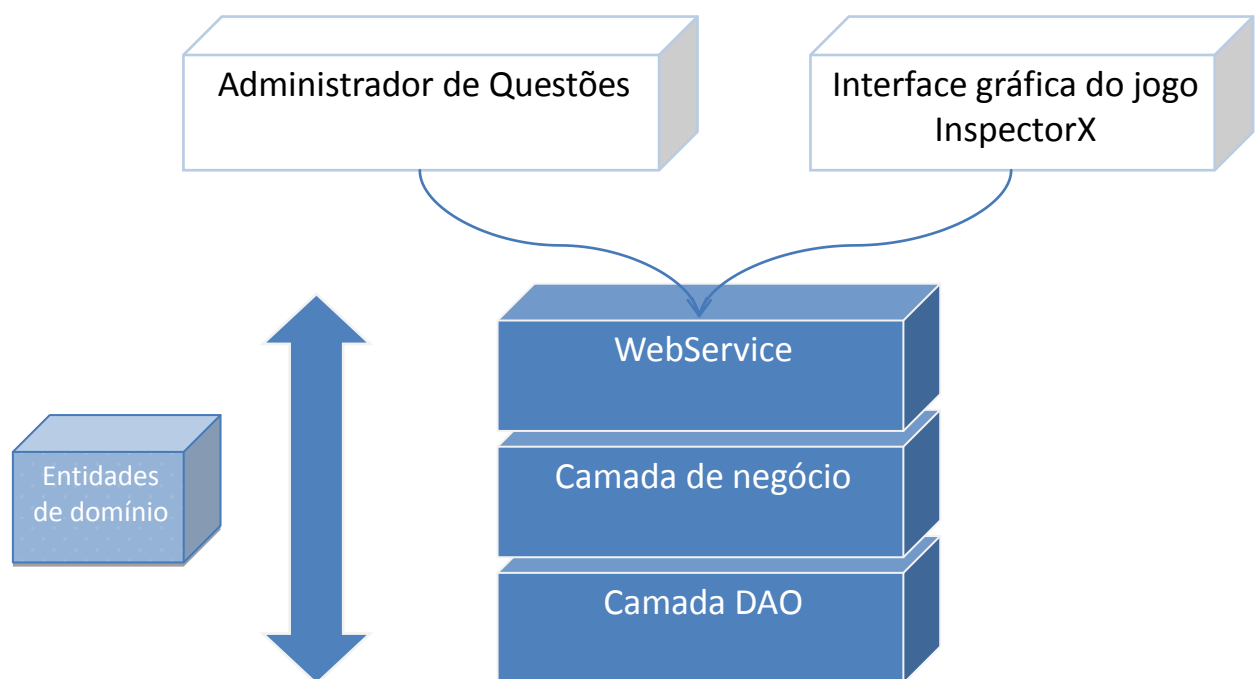


Figura 17. Diagrama arquitetural do jogo InspectorX

Cada uma destas camadas foi codificada na linguagem C#, sendo descritas a seguir:

- A camada do WebService é responsável por disponibilizar as funções essenciais do jogo, por meio da plataforma de serviços web. A aplicação de serviço web foi desenvolvida com a plataforma ASP.NET Web Services.

- A camada de negócio é responsável por processar os dados e aplicar as regras de negócio do jogo, como por exemplo, pesquisar questões ainda não resolvidas por um jogador ou embaralhar as questões usando o algoritmo de Fisher-Yates (Durstensfeld 1964). Esta camada foi desenvolvida como uma biblioteca consumida no webservice.
- Por fim, a camada DAO (*Data Access Object*) contém os métodos que fazem acesso ao banco de dados. Estes métodos são responsáveis por executar *queries* requisitadas pela camada de negócio. Esta camada foi construída utilizando o framework LINQ (*Language Integrated Query*) (Marguerie et al. 2008), um conjunto de extensões da plataforma .NET que permite realizar pesquisas em conjuntos de dados, sejam estes relacionais ou não. Esta camada foi desenvolvida como uma biblioteca e é consumida pela camada de negócio.

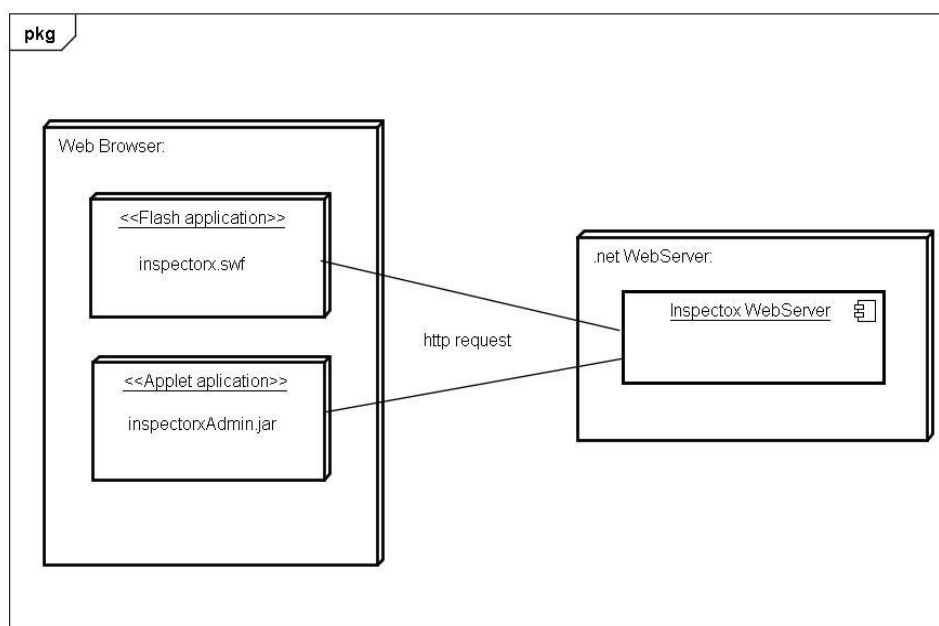


Figura 19. Diagrama de implantação do jogo InspectorX

A Figura 20 detalha o modelo do banco de dados do jogo InspectorX. Para permitir a persistência dos dados de cada partida em cada modo de jogo, de forma que o usuário atuasse em dois papéis (moderador e inspetor), foi usada a tabela “Usuario_Partida” para guardar os registros de cada partida para cada tipo de usuário. A relação de muitos-para-muitos entre as tabelas “Questão” e

“TrechoDefeito” permite que sejam persistidas as relações dos defeitos em cada artefato e, ao mesmo tempo, que mantenha todas as definições do tipo do defeito e da taxonomia vigente. A tabela “Trecho_Resposta” persiste os trechos selecionados pelos inspetores em cada partida, tornando possível a fase de reunião de inspeção onde o moderador identifica os falsos positivos.

4.2 Interface do jogo InspectorX

A aplicação de interface do jogo InspectorX (Figura 19) foi desenvolvida na plataforma *Adobe Flex*, baseada no conceito de RIA (*Rich Interface Applications*) (Allaire 2002) e codificada em *ActionScript*. Esta plataforma foi escolhida, com o intuito de aproveitar a flexibilidade da interação em tempo real com o jogador (Pötter e Schots 2011) e pela possibilidade de se desenvolver animações com facilidade.

A aplicação do jogo apenas consome os métodos do serviço web, o qual já contém as regras de negocio necessárias ao jogo.

A interface de login do jogo InspectorX é apresentada em um formato de caixa de diálogo com o título "Login". No topo, há um campo de texto para "Usuário" e um campo de texto para "Senha", ambos com um ícone de olho desativado. Abaixo dos campos, há um botão "Efetuar Login". Na base da caixa de diálogo, há dois links: "Esqueci minha senha" e "Não sou cadastrado". Abaixo da caixa de diálogo, há três botões separados: "Ranking Nível Fácil", "Ranking Nível Intermediário" e "Ranking Nível Difícil".

Figura 19. Tela de acesso ao jogo InspectorX

4.3 Interface do Administrador de Questões

A interface do administrador de questões foi desenvolvida na linguagem Java e com o *widget toolkit* do framework *Swing* (Robinson e Vorobiev 2003). Assim como na aplicação da interface do jogo InspectorX, a aplicação do administrador de questões consome o serviço web que já contém regras de negocio e métodos de consulta ao banco de dados.

As questões criadas pelo administrador de questões são transformadas da codificação ASCII para HTML, e depois são inseridas no banco de dados. No nível fácil, a formatação dos textos para HTML possibilita a demarcação dos trechos com defeito, com o uso de *tags* de HTML, como no exemplo da Figura 22.

<pre>// RF02. Deve ser calculada uma projeção do saldo até o final do mês usando como base o saldo da primeira semana; neste sistema, deve-se assumir que 1 mês equivale a 4 semanas // Assuma como um trecho de uma classe. public float calcularProjecaoMensal(float saldoSemanal) { saldoMensalProj = saldoSemanal*4; saldoDiárioMédia = saldoMensalProj/28 return saldoMensalProj; }</pre>	<p>Trecho: saldoDiárioMédia = saldoMensalProj/28</p> <p>Tipo: Excesso</p> <p>Explicação: O trecho não tem utilidade no contexto da função.</p>
--	---

Figura 22. Exemplo de demarcação de defeito

Com este método, o texto demarcado aparece como um *link*, semelhante ao formato dos *browsers* de internet, na interface do jogo, e a chamada ao método de JavaScript “event:erro” é capturada pela aplicação *flash* do jogo quando o *link* é selecionado.

CAPÍTULO 5

Avaliação da Abordagem InspectorX

Neste capítulo, é apresentado um estudo de caráter experimental realizado com o jogo InspectorX.

5.1 Planejamento

Para o estudo em questão, foi usado apenas o modo *DefectCrawler*, com o objetivo de responder as questões de pesquisa apresentadas no capítulo 3 tópico 3.2. Foram elaboradas 20 questões, com trechos de código fonte em Java e Portugol, sendo 10 questões para os níveis fácil e intermediário, respectivamente. Deve ser destacado, que para este estudo, não foi avaliado o nível avançado e nem o modo *Full inspection process*.

Para coleta de dados quantitativos, seria usada a pontuação e o tempo de resposta de cada questão. Sendo assim, foi implementado um temporizador em cada questão, o que tornou possível saber quanto tempo cada jogador levou para realizar cada questão.

Foram reunidos 11 alunos de graduação do curso Qualidade de Software e Modelagem de Sistemas do curso de Ciência da Computação e 1 aluno de Mestrado da UERJ. Criaram-se 3 grupos, com base nos perfis coletados por meio de um questionário de caracterização, o grupo 1 jogaria apenas no nível Intermediário, o 2 jogaria no Fácil e no Intermediário e o 3 jogaria no Fácil de acordo com a Tabela 4.

Tabela 4. Divisão dos grupos *versus* níveis de dificuldades

	Nível fácil	Nível intermediário
Grupo 1		X
Grupo 2	X	X
Grupo 3	X	

Após a realização do jogo, foi enviado aos participantes um questionário *follow-up*, com perguntas que visavam responder as questões de pesquisa, coletando dados qualitativos do jogo.

5.2 Execução

O estudo contou com 12 participantes, seguindo os seguintes passos ao longo do experimento:

- i. Preenchimento do questionário de caracterização dias, antes do estudo.
- ii. Por meio da análise feita nas respostas dos questionários de caracterização, foi feito um levantamento de perfis, que incluíam experiência em desenvolvimento de software, inspeções de software e desenvolvimento Java.
- iii. Criação de grupos heterogêneos em relação à experiência nas práticas de software a Tabela 5.

Tabela 5. Alocação dos grupos

Grupo	Formação Acadêmica	Média da Experiência em Engenharia de software (0-4)	Média da Experiência em Qualidade de software (0-4)	Média da Experiência em Inspeção de software (0-4)	Média da Experiência em Desenvolvimento em Java (0-4)
1	Graduação em andamento	2,5	3	2	2,25
2	Mestrado em andamento	2,8	2,6	1,2	2,6
	Graduação em andamento				
3	Graduação em andamento	3	1,666	1,6667	2,6667

- iv. No dia do experimento, foi realizado um treinamento do jogo e uma explicação da taxonomia usada nas questões.
- v. Execução do jogo InspectorX.
- vi. Envio do questionário *follow-up*, após o experimento.

5.3 Análise dos Dados

Considerando que este foi apenas um estudo exploratório, e contou com poucos participantes, observa-se que os dados quantitativos podem não ser suficientes para mostrar possíveis comportamentos de forma generalizada.

Durante a análise, observou-se que, em uma das questões no nível intermediário, o trecho com o defeito não foi cadastrado corretamente, o que impediu a pontuação dos participantes que selecionaram o trecho correto. Devido a isto, esta questão foi anulada.

5.1 Dados quantitativos

Para clarificar, os termos usados a seguir, segue a Tabela 6.

Tabela 6. Definição das métricas utilizadas

Eficiência	Relação entre tempo gasto e a pontuação, por questão.
Eficácia	Porcentagem de acertos em relação ao total de questões.

Observando as Figuras 22 e 23, percebe-se que os grupos 2 e 3, que jogaram no nível 1, tiveram desempenho inferior ao do grupo 1 que jogou no nível 2². Apesar deste comportamento nos dados, contrariar as expectativas, de que o grupo do nível fácil deveria ter tido um desempenho melhor, pode ser devido à alocação dos grupos. Outra possibilidade para este comportamento se evidenciou, na análise qualitativa, onde os participantes apontaram que ter o trecho destacado, dificultava a identificação do defeito, pois tirava a atenção do contexto do trecho, marcado no código.

² Os dados do nível intermediário do grupo 2 não foram usados na comparação de desempenho, pois o grupo possuía viés de aprendizado da etapa anterior (jogo no nível fácil).

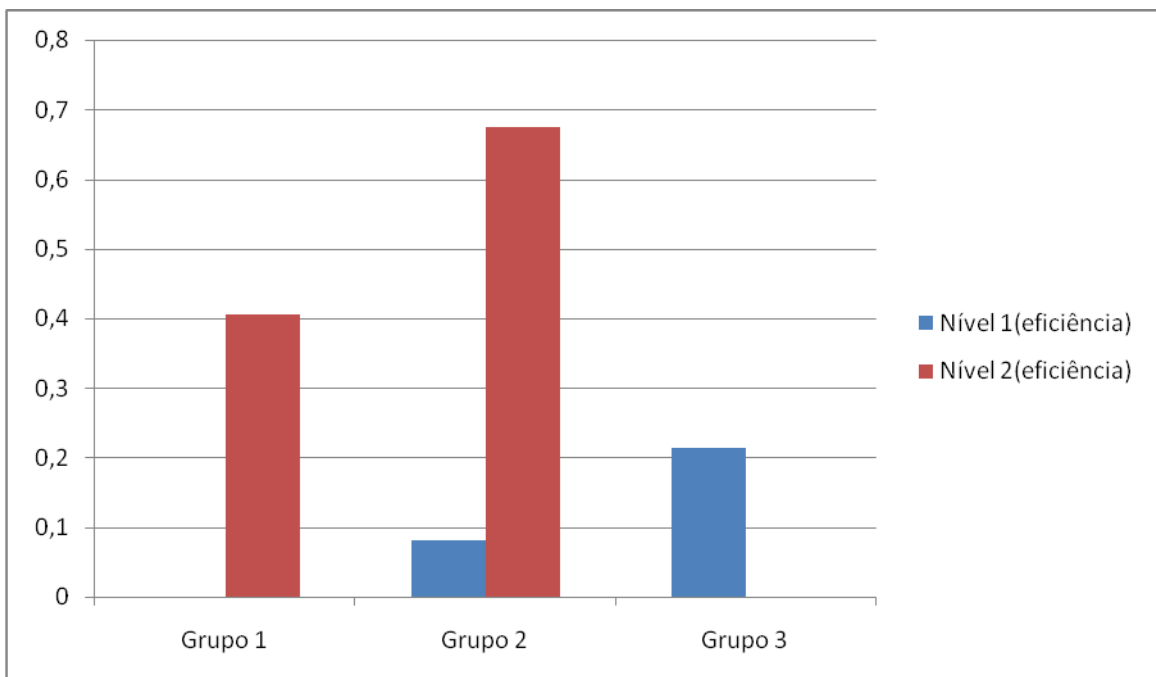


Figura 23. Gráfico da eficiência dos grupos no experimento

Ao analisar os dois gráficos também fica evidente, a melhora no desempenho no grupo 2, tanto em eficácia quanto em eficiência, porém não se pode afirmar que foi devido a um aprendizado em inspeção, pois pode ter tido influencia de outros fatores (como discutido na seção 6).

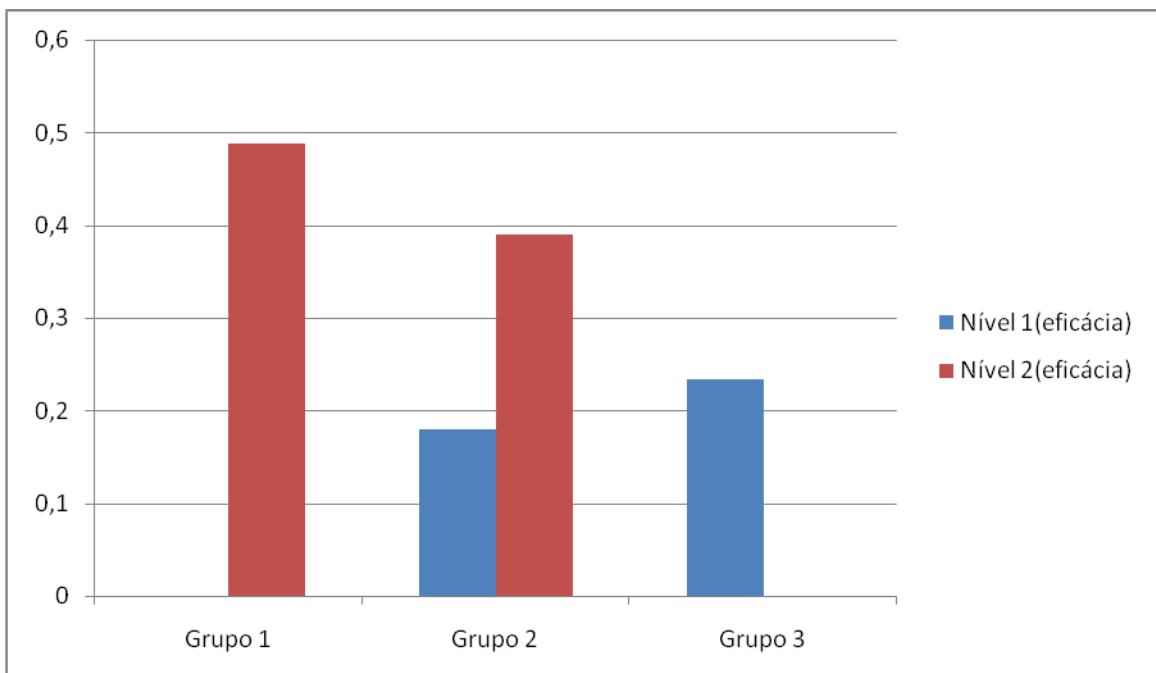


Figura 24. Gráfico da eficácia dos grupos no experimento

5.2 Dados qualitativos

A análise qualitativa mostrou que o jogo teve uma boa recepção por parte dos jogadores. Os participantes apontaram que a ferramenta tem potencial para ser utilizada no aprendizado e treinamento de inspeção. Uma das questões interessantes como o potencial de estado de *flow* (Imersão) (Csíkszentmihályi e Csíkszentmihályi 1988), se encontra destacado na Figura 24.

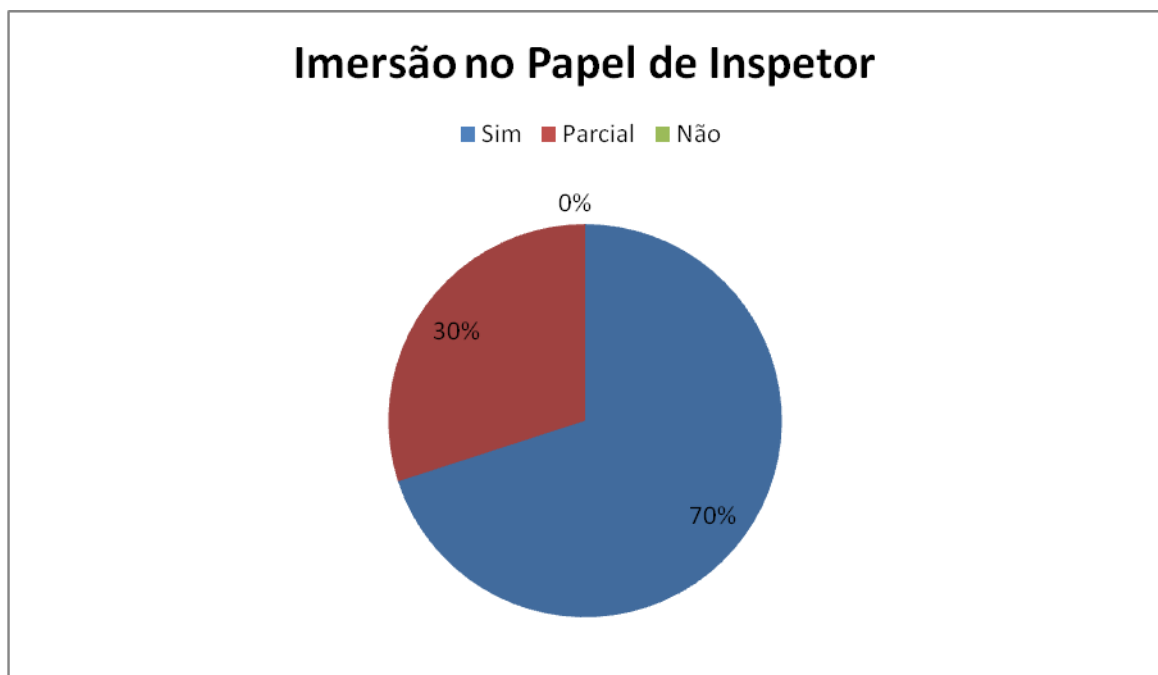


Figura 25. Gráfico da relação de participantes, que qualificaram o jogo com potencial de imersão, no papel de inspetor.

Outros pontos importantes, coletados na análise qualitativa foram:

- Alguns jogadores reportaram que, no nível fácil, o trecho do defeito, já estar destacado dificultava a sua análise, pois focava a sua atenção para o trecho em detrimento do contexto do artefato, o qual é essencial, para a classificação do defeito.
- Quase todos os jogadores apontaram a necessidade de melhoria na explicação da taxonomia de defeitos utilizada. Os tipos de defeito em que houve mais confusão foram o de comissão (segmento de código cuja implementação é

diferente do que foi especificado) e o de computação (ocorre quando um valor é definido erroneamente para uma variável).

- Um dos jogadores apontou a necessidade de melhoria na explicação de algumas questões; segundo ele, alguns enunciados pareciam não estar claros.
- Um jogador do grupo 1 e um do grupo 2 apontaram a necessidade de utilizar artefatos de código mais completos para maior imersão no papel de inspetor.

Um ponto que foi possível observar, tanto pela análise quantitativa quanto pela qualitativa, é que, conforme afirmado anteriormente (Pötter e Schots 2011), a qualidade da questão formulada possui impacto direto na qualidade do aprendizado, isto é, uma questão mal elaborada pode trazer distorções na compreensão dos conceitos.


Não foi encontrado um método para avaliar se a maneira com que o defeito foi destacado seguiu a devida complexidade, no contexto da teoria da carga cognitiva (Sweller 1994). Porém, o menor desempenho dos grupos que realizaram o jogo no nível fácil e o fato de que muitos jogadores sentiram mais dificuldade com o defeito já destacado (Figura 25), mostra que a aplicação da teoria da carga cognitiva, da forma como está presente no jogo para o nível fácil, pode não estar apropriada.

```
package defectmaker;

import java.util.List;

public class Items
{
    private List<String> itemList;

    private void checkItemList()
    {
        for (String string : itemList) {
            if(string.equals("car"))
            {
                //do something
            }
        }
    }
}
```



Próximo Artefato

Figura 26. Tela do InspectorX do jogo no modo Defect crawler no nível fácil.

5.3 Ameaças à Validade

Embora alguns cuidados tenham sido tomados no estudo em questão, foram detectadas as seguintes ameaças à validade, segundo a taxonomia de Wohlin et al. (2000):

- **Validade Interna:** (i) não é possível garantir que as informações obtidas no questionário de caracterização são condizentes com a realidade; (ii) foi observado que a descrição das questões afeta diretamente o desempenho dos participantes; no entanto, todas as questões tiveram uma porcentagem aceitável de acertos; (iii) como todos os participantes aprenderam a taxonomia da classificação dos defeitos de código fonte (Jones 2009) no mesmo dia do estudo, não é possível saber se eles a assimilaram corretamente e o quanto isto impactou no desempenho dos mesmos; (iv) apesar dos cuidados na codificação do jogo e os resultados serem consistentes com testes em pequena escala, não há como saber se possíveis bugs no código afetaram os resultados.
- **Validade Externa:** (i) com exceção de um participante, todos os demais são alunos de graduação, o que pode limitar o escopo de análise na didática do InspectorX em ensinar inspeção, porém vale notar que os grupos foram balanceados e muitos dos participantes trabalhavam na indústria; (ii) o estudo foi realizado utilizando somente questões com trechos/artefatos de código, o que limita o universo de aplicação do InspectorX; pretende-se efetuar outros estudos utilizando questões com artefatos de requisito; (iii) apenas o modo *Defect Crawler* foi estudado; assim, os resultados podem não condizer com a experiência de um participante no modo *Full Inspection Process*.
- **Validade de conclusão:** o tamanho da amostra não é significativo para evidenciar padrões consistentes que permitam resultados conclusivos.
- **Validade de construção:** (i) como o jogo foi acessado na web e o cálculo do tempo de duração de cada questão é feito no servidor, não há como prever se ocasionais congestionamentos na rede afetaram o tempo de

resposta do jogo no computador do participante; (ii) não foi analisado o nível de dificuldade que pode ser inerente ao próprio trecho de código (e.g., algumas questões podem ser mais difíceis do que outras em função da medida de sua complexidade).

CAPÍTULO 6

Conclusões

Esta monografia apresentou, com o jogo InspectorX, a possibilidade de usar *serious games*, no ensino e treinamento de alguns pontos importantes da inspeção de software. Com o levantamento de trabalhos relacionados no Capítulo 2, mostrou-se que já existe interesse em jogos para ensino em áreas da engenharia de software, e já existe esforço de pesquisa relevante na área.

Para o desenvolvimento da mecânica de jogo, foi usada a teoria da carga cognitiva (Sweller 1994). Com isso mostrou-se a necessidade de flexibilizar a interação com o jogo, e assim, a criação de níveis de dificuldade, de acordo com a experiência do jogador.

O estudo piloto mostrou que o jogo tem potencial para o uso no aprendizado de inspeção de software, no entanto também apontou para a importância de artefatos de software serem mais completos e não apenas trechos. O estudo também destacou a necessidade de atenuar o destaque visual do defeito no nível fácil, de maneira que não retire a atenção do jogador do restante do artefato.

De acordo com a experiência e conhecimento adquiridos com o estudo piloto e ao longo do desenvolvimento do projeto, destacam-se as seguintes melhorias identificadas: (i) atenuar o destaque dos defeitos em artefatos de software do jogo no nível fácil; (ii) desenvolver cenários mais completos e descritivos na adição de novas questões do jogo, evitando ambiguidade; (iii) melhorar a interface com o usuário, com a finalidade de desenvolver um ambiente mais similar ao de jogos; (iv) implementar os serviços web com as regras de negócio e modificar o sistema de gerenciamento de banco de dados do jogo com tecnologias mais portáteis de código aberto, tais como Java e MySQL, facilitando assim sua distribuição e implantação.

Referências bibliográficas

- Abt, C. Serious Games. Washington, D.C.: University Press of America, 1987.
- Allaire, J., 2002. "Macromedia Flash MX – A Next-Generation Rich Client", *Technical Report, Macromedia*, March.
- Baker, A., Oh Navarro, E., van der Hoek, A., 2003. "Problems and Programmers: an Educational Software Engineering Card Game. In: *25th International Conference on Software Engineering (ICSE'03)*, pp. 614–619, Portland, Oregon.
- Basili, V., Caldiera, G., Rombach, H., 1994. "Goal Question Metric Paradigm", *Encyclopedia of Software Engineering*, v. 1, n. edited by John J. Marciniak, John Wiley & Sons, pp. 528-532.
- Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M. V., 1996."The empirical investigation of Perspective-Based Reading" *Empirical Software Engineering*, Volume 1, Issue 2, pp 133-164
- Conradi, R., Mohagheghi, P., Arif, T., Hegde, L. C., Bunde, G. A., Pedersen, A. 2003."Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment" in: 17th European Conference, Darmstadt, Germany, July 21-25.
- Corti, K., 2006. Games-based Learning; a serious business application, White Paper, PIXELearning Limited, February.
- Csikszentmihályi, M., Csikszentmihályi, F. Optimal experience: psychological studies of flow in consciousness. Cambridge University Press, 1988, 419 p.
- Durstenfeld, R., 1964. "Algorithm 235: Random permutation". *Communications of the ACM (CACM)*, 7 (7), p. 420, July.
- Fagan, M. E., 1976. "Design and Code Inspection to Reduce Errors in Program Development", *IBM Systems Journal*, 15, 3, pp. 182-211.
- Fagan, M. E., 1986. Advances in Software Inspections, *IEEE Transactions on Software Engineering*, SE-12, 7, pp. 744-751, July.
- Fernandes, L., Werner, C., 2009. "Sobre o uso de Jogos Digitais para o Ensino de Engenharia de Software", In: *Fórum de Educação de Engenharia de Software (FEES'08)*, Simpósio Brasileiro de Engenharia de Software, Fortaleza, Brasil.
- Gee, J., 2003. "What Video Games Have to Teach Us About Learning and Literacy". Palgrave Macmillan 2003, 256 p.
- IEEE, 1990. *IEEE Std. 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology*.
- IEEE, 2008. *IEEE Std. 1028-2008: IEEE Standard for Software Reviews and Audits*.

- Jones, C. L., 1985. "A process-integrated approach to defect prevention", *IBM Systems Journal*, 24, 2, pp. 150-167.
- Jones, C., 2009. *Software Engineering Best Practices*, McGraw-Hill Inc., New York, USA.
- Lopes, A. C., Marques, A. B., Conte, T., 2011. "Avaliação do Jogo InspSoft: Um Jogo para Ensino de Inspeção de Software". In: *Fórum de Educação de Engenharia de Software (SBES'12)*, XXVI Simpósio Brasileiro de Engenharia de Software, Rio Grande do Norte, Brasil.
- Kalinowski, M., 2011. "Uma Abordagem Probabilística para Análise Causal de Defeitos de Software", Tese de D.Sc., COPPE/UFRJ, 357p, Rio de Janeiro.
- Kruger, J., Dunning, D., 2009, "Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments", *Journal of Personality and Social Psychology*, Vol 77(6), Dec 1999, 1121-1134.
- McMeekin, D. A., von Kinsky, B. R., Chang, E., and Cooper, D. J. A., 2009. "Evaluating Software Inspection Cognition Levels Using Bloom's Taxonomy". In: *22nd Conference on Software Engineering Education and Training (CSEET '09)*, Hyderabad, India.
- Marguerie, F., Eichert, S. and Wooley, J., 2008. *Linq in Action*. Manning Publications Co., Greenwich, USA.
- Pötter, H., Schots, M., 2011. "InspectorX: Um Jogo para o Aprendizado em Inspeção de Software". In: *Fórum de Educação de Engenharia de Software (FEES'11)*, XXV Simpósio Brasileiro de Engenharia de Software, São Paulo, Brasil.
- Pötter, H., Schots, M., 2012. "Evolução e um Estudo Exploratório do Jogo InspectorX". In: *Fórum de Educação de Engenharia de Software (FEES'12)*, XXVI Simpósio Brasileiro de Engenharia de Software, Rio Grande do Norte, Brasil.
- Pressman, R.S., 2001, *Software Engineering: A Practitioner's Approach*, Fifth Edition, McGraw Hill.
- Prikladnicki, R., von Wangenheim, C. G., 2008. "O Uso de Jogos Educacionais para o Ensino de Gerência de Projetos de Software". In: *Fórum de Educação de Engenharia de Software (FEES'08)*, XXII Simpósio Brasileiro de Engenharia de Software, Campinas, Brasil.
- Robinson, M., Vorobiev, P., "Swing, Second Edition", Manning, 2003. 912p.
- Robinson, B., Francis, P., Ekdahl, F., 2008. "A Defect-Driven Process for Software Quality Improvement", In: *2nd International Symposium on Empirical Software Engineering and Measurement (ESEM'08)*, pp. 333-335, Kaiserslautern, Germany.
- Salger, F., Engels, G., and Hofmann, A., 2009. "Inspection Effectiveness for Different Quality Attributes of Software Requirement Specifications: an

- Industrial Case Study". In: *7th ICSE Workshop on Software Quality (WOSQ'09)*, pp. 15-21, Vancouver, Canada.
- Sauer, C., Jeffery, D.R., Land, L., Yetton, P., 2000, "The Effectiveness of Software Development Technical Review: A Behaviorally Motivated Program of Research", *IEEE Transactions on Software Engineering*, 26 (1): 1-14, January.
- Shull, F., 1998. "Developing Techniques for Using Software Documents: A Series of Empirical Studies", Ph.D. Thesis, University of Maryland, College Park.
- Silva, M., Borges, V., Barbosa, E., Maldonado, J., 2008. "Novas tendências em educação de Engenharia de Software: um estudo de caso no domínio de Teste de Software", In: *Fórum de Educação de Engenharia de Software (FEES'08)*, - Fórum de Educação em Engenharia de Software, Campinas-SP, Brasil.
- Sweller, J., 1994. Cognitive Load Theory, Learning Difficulty and Instructional Design. *Learning and Instruction*, 4, pp. 295-312.
- Thiry, M., Zoucas, A., Gonçalves, R. Q., e Salviano, C., 2010. "Aplicação de Jogos Educativos para Aprendizagem em Melhoria de Processo e Engenharia de Software". In: *Workshop Anual do MPS (WAMPS'10)*, pp. 118-127, Campinas, Brasil.
- Ulicsak, M., Wright, M., 2010, "Futurelab: innovation in education. Games in Education: Serious Games". Disponível em:
http://archive.futurelab.org.uk/resources/documents/lit_reviews/Serious-Games_Interviews.pdf .Acesso em 07/07/2012.

APÊNDICE A

Instrumentos utilizados na avaliação do jogo no contexto educacional de inspeção.

Este apêndice apresenta os instrumentos utilizados na avaliação do jogo InspectorX.

4.1 Formulário de Consentimento

O formulário de consentimento foi entregue, a priori, aos participantes do estudo, com o intuito de obter o consentimento. Para conscientizar o participante, das condições do estudo, o formulário as apresenta conforme apresentado abaixo.

<p>Avaliação do Jogo InspectorX Formulário de Consentimento</p>

O estudo a ser conduzido visa avaliar a viabilidade do jogo InspectorX no apoio ao aprendizado da inspeção de artefatos de software e do processo de inspeção. O estudo consiste em tarefas referentes a inspeções de software.

Eu declaro ter mais de 18 anos de idade e concordar em participar do estudo conduzido por Henrique Alberto Brittes Pötter na Universidade do Estado do Rio de Janeiro. Eu entendo que os trabalhos que eu desenvolver serão estudados, visando entender atributos de qualidade dos procedimentos e técnicas propostos.

Eu estou ciente de que meu nome não será divulgado em hipótese alguma. Também estou ciente de que os dados obtidos por meio deste estudo serão mantidos sob confidencialidade, e os resultados serão posteriormente apresentados de forma agregada, de modo que um participante não seja associado a um dado específico.

Da mesma forma, comprometo-me a não comunicar os meus resultados enquanto o estudo não for concluído, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e apresentado. Também entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada à minha pessoa não seja incluída no estudo. Por fim, declaro participar de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

Pesquisador responsável

Henrique Alberto Brittes Pötter

Professores responsáveis

Marcelo Schots de Oliveira

Vera Maria Benjamin Werneck

Nome (em letra de forma): _____

Assinatura: _____ Data: _____

4.2 Questionário de Caracterização

O questionário de caracterização foi entregue, a priori, aos participantes do estudo, com o intuito de caracterizar o perfil de cada participante, possibilitando uma análise qualitativa e quantitativa, mais precisa, dos dados.

Avaliação do Jogo InspectorX Questionário de Caracterização do Participante	
Código do participante:	

Prezado(a) participante,

Este formulário contém algumas perguntas sobre sua experiência acadêmica e profissional.

1. Formação acadêmica

- ☐ Doutorado concluído
- ☐ Doutorado em andamento
- ☐ Mestrado concluído
- ☐ Mestrado em andamento
- ☐ Graduação concluída
- ☐ Graduação em andamento

Instituição acadêmica: _____
Ano de ingresso: _____ Ano de conclusão / Ano previsto de conclusão: _____

2. Formação geral

- a) Qual é sua experiência com desenvolvimento de software orientado a objetos (OO)? (marque todos os itens que se aplicam)

- ☐ Já li material sobre desenvolvimento de software OO.
- ☐ Já participei de um curso sobre desenvolvimento de software OO.
- ☐ Nunca desenvolvi software OO.
- ☐ Tenho desenvolvido software OO para uso próprio.
- ☐ Tenho desenvolvido software OO como parte de uma equipe, relacionado a um curso.

() Tenho desenvolvido software OO como parte de uma equipe, na indústria.

- b) Por favor, detalhe sua resposta. Inclua o número de semestres ou número de anos de experiência relevante em desenvolvimento de software. (por exemplo, “Eu trabalhei por 2 anos como programador de software na indústria”)

- c) Por favor, indique o grau de sua experiência nas áreas a seguir, com base na escala abaixo:

0 = Nenhum

1 = Estudei em aula ou em livro

2 = Pratiquei em projetos em sala de aula

3 = Utilizei em projetos pessoais

4 = Utilizei em projetos na indústria

Área de conhecimento	Grau de experiência				
Engenharia de software	0	1	2	3	4
Qualidade de software	0	1	2	3	4
Inspeção de software	0	1	2	3	4
Desenvolvimento de software em Java	0	1	2	3	4

Desde já, agradecemos a sua colaboração.

Henrique Alberto Brittes Pötter

Marcelo Schots de Oliveira

Vera Maria Benjamin Wernec

4.3 Questionário *Follow-up*

O questionário *follow-up* foi entregue a após a execução do estudo, via e-mail, para os participantes. O objetivo deste questionário era coletar dados para a análise qualitativa do jogo InspectorX. Para se economizar espaço foi retirado a caixa de texto, referente à resposta do participante, após cada questão.

Questionário Follow-up

- 1) O que você achou do jogo Inspector X?

- 2) O que você achou da divisão de níveis de dificuldade distintos? Justifique.
(Apenas para o grupo G2).

Bom () Razoável () Ruim()

- 3) Você sentiu a necessidade de mais (ou menos) níveis de dificuldade?
Justifique.

- 4) Você utilizaria o jogo Inspector X como forma de treinamento ou aprimoramento da aprendizagem?

- 5) Do que você gostou no jogo Inspector X?

- 6) Do que você não gostou (ou o que você mudaria) no jogo Inspector X?

- 7) Você se sentiu no papel de um verdadeiro inspetor de software? Justifique.

() Sim () Parcialmente () Não

4.4 Questões utilizadas na execução do estudo

Foram apenas utilizadas questões para os níveis fácil e intermediário. As diferenças entre os níveis foram apresentadas na Tabela 1. A ordem atual das questões não representa a ordem na qual foram apresentadas aos alunos, pois elas foram embaralhadas usando o algoritmo de Fisher-Yates (Durstensfeld 1964).

Questões utilizadas no nível fácil

Questão 1

<pre>import java.util.List; public class Dispositivo{ ArrayList<int> dispositivosEstoList; public static void main(String[] args) { } private void setDispositivosEstoque(int[] dispositivosId) { dispositivosEstoList = new ArrayList<int>(); int dispLenght = dispositivosId.length(); for(int i=0; i= dispLenght ; i++) { dispositivosList.add(dispositivosId[i]); } } }</pre>	<p>Trecho: for(int i=0;i=dispositivosId.length;i++)</p> <p>Tipo: Dados</p> <p>Explicação: O loop for irá acessar uma posição inexistente do array 'dispositivosId', quando 'i = dispositivosId.length()' . Lembrando que se as posições de um array vão de 0 a n e o tamanho do array será n+1.</p>
---	--

Questão 2

<pre>import java.util.List; public class Items { private List<String> itemsList; private void checkItem() { for (String string : itemsList) { if(string.equals("car")) { //do something } } } }</pre>	<p>Trecho: for (String string : itemsList) {</p> <p>Tipo: Inicialização</p> <p>Explicação: A variável 'itemsList' pode não ter sido inicializada, seria recomendado uma verificação antes de tentar acessá-la.</p>
---	---

<pre> } } } </pre>	
--------------------------------	--

Questão 3

<pre> import java.util.List public class ItemTaxonomia { String name; ItemTaxonomia subItem; public ItemTaxonomia(String name) { this.name=name; itemTaxList = new ArrayList<>(); } public ArrayList<ItemTaxonomia> getDefaultList() { itemTaxList.add(new ItemTaxonomia("Comercial")); itemTaxList.add(new ItemTaxonomia("Industrial")); itemTaxList.add(new ItemTaxonomia(subItem.name)); } return itemTaxList; } </pre>	<p>Trecho: itemTaxList.add(new ItemTaxonomia(subItem.name));</p> <p>Tipo: Inicialização</p> <p>Explicação: A variável "subItem" não foi devidamente inicializada.</p>
---	--

Questão 4

<p>// RF01: Para o cálculo de saldo mensal, deve ser retirada a alíquota de 5% de imposto sobre o valor de cada produto.</p> <p>// Assuma como um trecho de uma classe e a importação "import.java.util.List" foi feita.</p> <pre> double percentual = 15.0 / 100.0; private ArrayList<Produto>calcularSaldoMensal(ArrayList<Pr oduto> prodList) { for(Produto prod : prodList) { prod.valor=prod.valor-(prod.valor*percentual); } return prodList; } </pre>	<p>Trecho: double percentual = 15.0 / 100.0;</p> <p>Tipo: Comissão</p> <p>Explicação: O valor correto é 5 %, porém foi implementado como 15%.</p>
---	--

Questão 5

<pre>// RF06: Deve ser verificada a unicidade do RG do usuário no momento do cadastro, não possibilitando dois usuários com o mesmo RG no sistema. public void cadastrarUsuario(Usuario user) throws Exception { bool rgExistente = verificarSeRGJaExiste(user); if(rgExistente) { cadastrar(user); } }</pre>	<p>Trecho: if(rgExistente)</p> <p>Tipo: Controle</p> <p>Explicação: A implementação diverge da especificação porque pede pra não cadastrar usuário se o RG for igual, e o código faz exatamente o contrário. O booleano "rgExistente" deveria estar negado.</p>
---	--

Questão 6

<pre>// RF02. Deve ser calculada uma projeção do saldo até o final do mês usando como base o saldo da primeira semana; neste sistema, deve-se assumir que 1 mês equivale a 4 semanas // Assuma como um trecho de uma classe. public float calcularProjecaoMensal(float saldoSemanal) { saldoMensalProj = saldoSemanal*4; saldoDiárioMédia = saldoMensalProj/28 return saldoMensalProj; }</pre>	<p>Trecho: saldoDiárioMédia = saldoMensalProj/28</p> <p>Tipo: Excesso</p> <p>Explicação: O trecho não tem utilidade no contexto da função.</p>
---	---

Questão 7

<pre>//Considera apenas o algoritmo, independente de linguagem. lucro_jan = 300; lucro_fev=500; lucro_mar=100; lucro_trimestralMedio = (lucro_jan+lucro_fev+lucro_mar)/2;</pre>	<p>Trecho: (lucro_jan+lucro_fev+lucro_mar)/2</p> <p>Tipo: Computação</p> <p>Explicação: O cálculo da média ponderada nesse caso seria a soma do lucro dos três meses divididos pelo número de meses no caso 3.</p>
---	---

Questão 8

<pre>// assuma a existência dos métodos getListaPacientes(String tipo, String nomeMedico) e setListData(Paciente[]) e a importação “import.java.util.List” foi feita. private void getMeusPacientes() { ArrayList<Paciente> pacList = getListaPacientes(“SUS”, “Dr.Alberto”); Paciente[] pacArray = new Paciente[pacList.size()]; if(!pacList.isEmpty()){ lst_Pacientes.setListData(pacList.toArray()); else JOptionPane.showMessageDialog(this, “Nenhum paciente encontrado!”); } }</pre>	<p>Trecho: Paciente[] pacArray = new Paciente[pacList.size()];</p> <p>Tipo: Excesso</p> <p>Explicação: O trecho não tem nenhuma utilidade para a função do método .</p>
--	--

Questão 9

<pre>// Assuma como um trecho de uma classe e a importação “import.java.util.List” foi feita. Assuma também que a inicialização da classe GPSCalculos é custosa. private ArrayList<int> calcularTodasAsDistancias(ArrayList<G PS>posicoesList, GPS posicaoInicial) { ArrayList<int> distancias = new ArrayList<int>(); for(GPS novaPosicao : posicoesList) { GPSCalculos gpsc = new GPSCalculos(posicaoInicial.getLat(), posicaoInicial.get Long()); distancias.add(gpsc.getDistanceFrom(novaPosicao.getLat(), novaPos icao.getLong())); } return distancias; }</pre>	<p>Trecho: GPSCalculos gpsc = new GPSCalculos(latitudelnI, lo ngetudelnI);</p> <p>Tipo: Desempenho</p> <p>Explicação: O objeto da classe GPSCalculos poderia ter sido instanciado por fora do laço “for”, evitando que o objeto fosse criado toda vez que o laço fosse executado.</p>
--	--

Questão 10

<pre>// Assuma como um trecho de uma classe e a existência do método enviarMensagemParaMedico(Paciente pac). Paciente pac; private void checarStatusPaciente() { If(pac.emergenciaStatus==true) { enviarMensagemParaMedico(pac); } }</pre>	<p>Trecho: If(pac.emergenciaStatus ==true)</p> <p>Tipo: Inicialização</p> <p>Explicação: A variável pacAlerta não foi devidamente inicializada.</p>
--	--

Questões utilizadas no nível Intermediário

Questão 11

<pre>//Considera apenas o algoritmo, independente de linguagem. lucro_jan = 300; lucro_fev=500; lucro_mar=100; lucro_trimestralMedio = lucro_jan+lucro_fev+lucro_mar/3;</pre>	<p>Trecho: lucro_jan+lucro_fev+lucro_mar/3</p> <p>Tipo: Computação</p> <p>Explicação: Para o cálculo da média ponderada, era necessário colocar parênteses, de forma que a divisão fosse feita devidamente.</p>
---	--

Questão 12

<pre>// Assuma como um trecho de uma classe e a classe Paciente possui um método implementado getCPF() que retorna o cpf. // OBS: O CPF é usado como chave primária na tabela Paciente. private Paciente getPacienteEmergencia(Paciente[] pacientes) { int counter = pacientes.lenght-1; PacientepacEmergencia; while(counter>=0)</pre>	<p>Trecho: while(counter>=0)</p> <p>Tipo: Desempenho</p> <p>Explicação: Este laço irá desnecessariamente percorrer todo o array mesmo que o cpf já tenha sido encontrado.</p>
--	---

<pre> { if(pacientes[counter].getCPF()=='525.173.944-31') { pacEmergencia = pacientes[counter]; } Counter--; } return pacEmergencia; } </pre>	
---	--

Questão 13

<p>// RF07. No momento do cadastro do sistema, se o usuário já estiver cadastrado, este deve ser informado com uma mensagem de alerta; senão, cadastrá-lo no banco de dados. // Assuma como trechos de uma classe.</p> <pre> private void cadastrarUsuario(String usuario,Stringsenha) { if (verificarExistenciaCadastro(senha)) showMessage('Este nome de usuário já está cadastrado!'); else cadastrarNoBancoDeDados(usuario,senha); } private boolean verificarExistenciaCadastro(String usuario) { return verificaExistenciaNoBanco(usuario); } </pre>	<p>Trecho: if (verificarExistenciaCadastro(senha))</p> <p>Tipo: Comissão</p> <p>Explicação: No trecho, é passada a senha por parâmetro no lugar do nome do usuário. Sendo assim a implementação difere da especificação.</p>
---	---

Questão 14

<p>// Assuma a existência do método public Item[] pegarEstoque() neste contexto</p> <pre> public static void main(String[] args) { } private void verificarEstoque() { Itens[] estoque = pegarEstoque(); } </pre>	<p>Trecho: for (int i=0;i<=quantidadeEstoque;i++)</p> <p>Tipo: Dados</p> <p>Explicação: O loop for irá acessar uma posição inexistente do array</p>
--	---

<pre>int quantidadeEstoque = estoque.lenght; for (int i=0;i<=quantidadeEstoque;i++) { Item eltem = estoque[i]; } }</pre>	<p>‘estoque, quando ‘i= quantidadeEstoque’ . Lembrando que as posições de um array vão de 0 a n-1 e o tamanho do array é n.</p>
--	---

Questão 15

<p>// O Sistema deve bloquear mais de 10 acessos simultâneos. // Assuma a existência do método public Usuario[] getLoggedUsers() neste contexto</p> <pre>int usuarioLim = 10; int usuariosConectados = getLoggedUsers().lenght; if(usuariosConectados==usuarioLim) { // bloquear novos acessos }</pre>	<p>Trecho: if(usuarioLim==usuarios Conectados)</p> <p>Tipo: Controle</p> <p>Explicação: O controle do ‘if’ irá falhar caso o número de usuários conectados seja maior que o número de usuários limite; o correto deveria ser “if(usuariosConectados > usuarioLim)”.</p>
---	--

Questão 16

<p>// Lembrando que “Math.pow” em java calcula potência e “Math.PI” é uma constante do pi.</p> <pre>import java.util.*; public class Teste{ public double calcularAreaCirculo(double raio){ double areaCirculo; if(raio<=0) areaCirculo= Math.pow(raio, 2)*Math.PI*raio; return areaCirculo; } }</pre>	<p>Trecho:if(raio<=0)</p> <p>Tipo: Controle</p> <p>Explicação: A condicional ‘if(raio<=0)’ condiciona o cálculo da área do círculo de forma inadequada, pois raios de valor negativo serão considerados. Deveria ser ‘if(raio>=0)’.</p>
---	---

Questão 17

<pre>//Lenbrando que "Math.pow" em java calcula potência e "Math.PI" é uma constante do pi. import java.util.*; public class Teste{ public double calcularPerimetroCirculo(double raio){ double perimetroCirculo; perimetroCirculo = 2*Math.PI+raio; return perimetroCirculo; } }</pre>	<p>Trecho: <code>areaCirculo = Math.pow(raio, 1)*Math.PI+raio;</code></p> <p>Tipo: Computação</p> <p>Explicação: O raio está sendo somado ao resto da equação, quando deveria estar sendo multiplicado.</p>
--	--

Questão 18

<pre>// RF03: No preenchimento do formulário do cadastro, o nome de usuário deve possuir no mínimo 5 caracteres, e o e-mail digitado pelo usuário deve ser um email válido. // Assuma a existência do método public boolean validarEmail(String email), que verifica o e-mail de entrada como um email válido, retornando "true" caso o seja. private void validarFormulario(String usuairo,String email) throws Exception { if(usuario.isEmpty) throw new Exception("O usuário não foi preenchido."); if(usuario.lenght()<=0) throw new Exception("O número de caracteres para o usuário deve ser 5"); if(email.isEmpty) throw new Exception("O email não foi preenchido."); if(!validarEmail(email)) throw new Exception(""); }</pre>	<p>Trecho: <code>if(usuario.lenght()<=0)</code></p> <p>Tipo: Comissão</p> <p>Explicação: A implementação difere na especificação, foi explicitado que o número de caracteres mínimo para o nome de usuário deve ser 5.</p>
---	--

Questão 19

//Assuma que este é um trecho de uma classe, e que String.Empty é uma primitiva de linguagem que representa uma String vazia.

```
private void validarLogin(String usr,String email)
{
    If(usr!=null && usr!=String.Empty)
If(usr==null)
    // valida o usuário
}
```

Trecho: If(usr==null)

Tipo: Controle

Explicação: A condicional 'If(usr==null)' após a condicional 'If(usr!=null&&usr!=String.Empty)' impossibilitará a execução de qualquer trecho de código abaixo dela, pois caso a variável usr passe na primeira condicional, ela será bloqueada pela segunda (que é como uma negação da primeira).

Questão 20

// Assuma como um trecho de uma classe e a importação "import.java.util.List" foi feita. Assumir também a existência da classe EmptyListException

```
public void calcularMediaNotas(ArrayList
alunosList)
{
    If( (alunosList!=null) && (alunosList.size!=0)
){
        int i;
        float notaSoma;
        int listSize = alunosList.size();
        for(i=0;i<list.size-1;i++)
            notaSoma+= alunosList.get(i).ProvaNota;

        float media = notaSoma/i;
        } else {
            throw new EmptyListException();
        }
}
```

Trecho: float media = notaSoma/i;

Tipo: Computação

Explicação: A última posição da lista não equivalerá à quantidade de alunos, pois a lista começa do "0", sendo assim o cálculo da média estará sempre errado.