InspectorX: Um Jogo para o Aprendizado em Inspeção de Software

Henrique Pötter[1], Marcelo Schots[1,2]

[1]Instituto de Matemática e Estatística – Universidade do Estado do Rio de Janeiro
Caixa Postal 15.064, Rio de Janeiro, RJ, 91.501-970

[2]Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ
Caixa Postal 68.511, Rio de Janeiro, RJ, 21945-970

henriquepotter.hp@gmail.com, schots@cos.ufrj.br

Abstract. *The benefits of inspection activities in software development are well known. However, for its effectiveness, it is necessary that the inspector is properly trained for the rapid identification of defects, especially the most common ones. In this context, this article describes InspectorX, a game aimed at the training and learning in software inspection, in order to allow the assimilation of concepts in a ludic manner.*

Resumo. *Os benefícios oriundos das atividades de inspeção no desenvolvimento de software são bem conhecidos. Porém, para a sua efetividade, é necessário que o inspetor esteja devidamente treinado para a rápida identificação de defeitos, especialmente os mais comuns. Neste contexto, este artigo descreve o InspectorX, um jogo voltado para o treinamento e aprendizado em inspeção de software, com o objetivo de permitir a assimilação dos conceitos de forma lúdica.*

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 em [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.

Na medida em que uma equipe de desenvolvimento de software se conscientiza dos defeitos de alta ocorrência, tais defeitos tendem a ser evitados no momento do desenvolvimento dos próximos artefatos, economizando tempo que seria gasto em manutenções corretivas posteriores [Fagan 1976]. Adicionalmente, as organizações

devem fazer do aprendizado com seus erros um instrumento para aumentar sua competitividade e qualidade [Kalinowski 2011]. 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.

Em [Thiry *et al.* 2010], é afirmado que jogos educativos podem ser aplicados como um complemento à capacitação de profissionais, tornando o aprendizado mais atrativo. Neste sentido, este artigo apresenta o InspectorX, um jogo desenvolvido com o objetivo de incentivar e prover apoio ao aprendizado em inspeção, mais especificamente na identificação de defeitos em artefatos do processo de desenvolvimento, e na a classificação destes defeitos de acordo com o tipo.

O restante do artigo está estruturado da seguinte forma: na Seção 2, são apresentados alguns conceitos relacionados ao aprendizado em inspeção e defeitos de software. Na Seção 3, o jogo InspectorX é detalhado, incluindo suas características e funcionalidades. A Seção 4 descreve alguns trabalhos relacionados. Por fim, a Seção 5 contém as considerações finais, incluindo limitações e trabalhos futuros.

2. Aprendizado em Inspeção e Defeitos de Software

O processo de aprendizado e conscientização dos defeitos está relacionado com o processo cognitivo e a aquisição de novas estruturas de conhecimento (*schemata*) [Sweller 1994]. Tais estruturas são construídas por meio de experiências anteriores que permitem responder a eventos similares de forma mais eficiente, orientando o processamento da informação.

A teoria do esforço cognitivo mostra que, para um aprendizado de longa duração voltado para problemas complexos, o processo de interação com a informação necessária para a solução desses problemas não deve exceder a capacidade máxima cognitiva [Sweller 1994]. Sob esta perspectiva, acredita-se que o aprendizado dos defeitos deve ser gradual para que haja a maximização da assimilação dos conceitos relacionados ao domínio de inspeção de software.

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, exercitados em pequenos exemplos práticos. Esta constatação motiva o desenvolvimento do InspectorX, um jogo voltado para o treinamento e aperfeiçoamento em tarefas de inspeção de artefatos do desenvolvimento de software, adaptado ao nível de experiência do jogador (no papel de inspetor).

3. O Jogo InspectorX

O jogo InspectorX foi construído com o objetivo de desenvolver a percepção do jogador na identificação e classificação de defeitos em artefatos de software, por meio do condicionamento da observação dos padrões apresentados, enriquecendo os *schemata*. Segundo [Robinson *et al.* 2008], a classificação dos defeitos e sua posterior análise ajudam a melhorar as estratégias de detecção de defeitos.

Cada questão do jogo é composta de um ou mais trechos de artefatos de software (e.g., documentos de requisitos ou código fonte). Cada trecho contém um ou mais defeitos que devem ser corretamente identificados e classificados pelo jogador, de forma a acumular pontos. Visando permitir um aumento gradual na complexidade das tarefas, obedecendo à teoria do esforço cognitivo, o jogo varia gradualmente o nível de dificuldade, conforme apresentado na Tabela 1.

Tabela 1. Níveis de dificuldade do jogo InspectorX

Nível	Nº de defeitos	Características
Básico	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	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	Cada questão possui mais de um defeito; o jogador terá que selecionar todos os trechos que contenham defeitos e classificá-los individualmente.

Para a classificação dos defeitos por categoria, podem ser utilizadas taxonomias existentes na literatura [Walia & Carver 2009]. A implementação atual utiliza como exemplo a taxonomia apresentada em [Shull 1998] para requisitos e a de [Jones 2009] para código fonte, descritas nas Tabelas 2 e 3, respectivamente.

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 (*loops*) 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).

O jogo também possui módulos de manutenção de usuário e ranking. O primeiro engloba as funcionalidades de cadastro, alteração e recuperação de senha. Para que o jogador possa acumular pontos, é necessário estar cadastrado e acessar o jogo (Figura 1, à esquerda). Já o ranking exibe a pontuação dos jogadores e classifica-os com base no número de pontos obtidos nas questões. Graças a esta funcionalidade, é possível

interromper o jogo a qualquer momento, e as questões respondidas naquela sessão são computadas no ranking para o usuário.

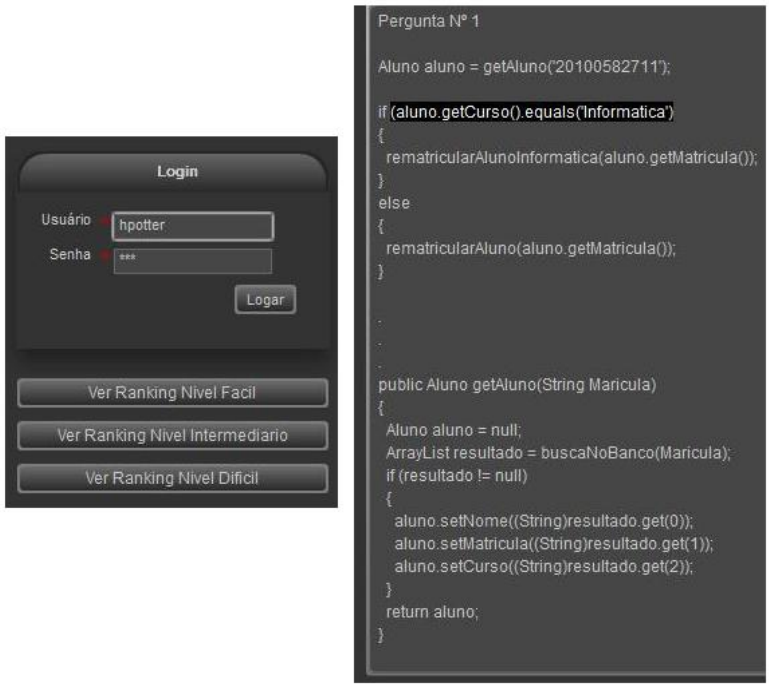

Figura 1. Tela de *login* (à esquerda) e tela de seleção do defeito (à direita)

A tarefa do jogador é selecionar o trecho que contenha o defeito (Figura 1, à direita) e classificá-lo a partir das taxonomias cadastradas no jogo. Para isto, após a seleção do trecho com potencial defeito, é exibida uma lista com os possíveis tipos de defeito cadastrados no jogo. Um exemplo deste cenário é exibido na Figura 2. Após responder todas as perguntas, o ranking é exibido para o jogador (Figura 3).

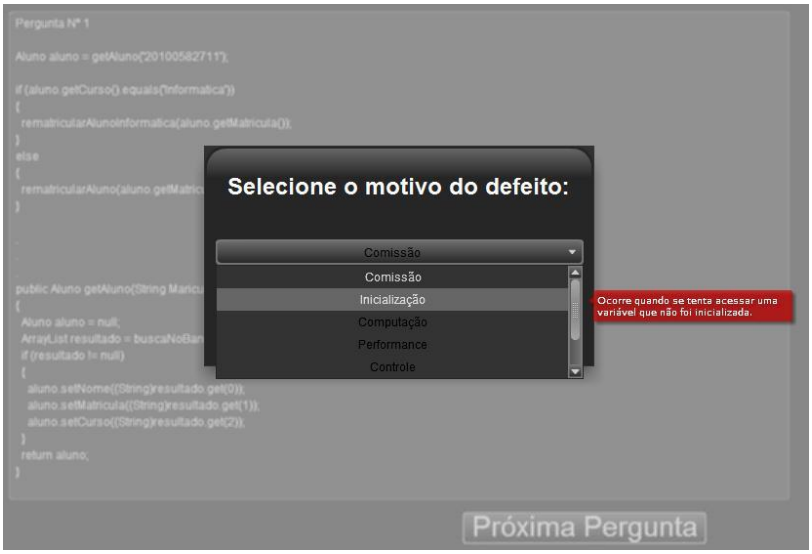

Figura 2. Tela de classificação de um defeito encontrado

Figura 3. Tela de *ranking* dos jogadores

As questões são descritas em um formato baseado em XML, cujos metadados delineiam no artefato em questão quais trechos possuem defeitos. Isto permite que novas questões sejam adicionadas, podendo inclusive ser direcionadas a domínios específicos e/ou cenários recorrentes em organizações. Uma interface visual foi criada visando facilitar a adição de novas questões. É importante notar que a qualidade da questão formulada impacta na qualidade do aprendizado, isto é, uma questão mal elaborada pode trazer distorções na compreensão dos conceitos.

3.1. Implementação da Abordagem

A arquitetura do jogo foi projetada visando flexibilidade, por meio de interfaces bem definidas. Sendo assim, a implementação foi dividida em quatro camadas, conforme exibido na Figura 4. Devido a esta flexibilidade, a substituição de qualquer camada não possui grande impacto nas demais camadas (e.g., uma versão do jogo para dispositivos móveis requereria apenas a substituição da camada de visão).

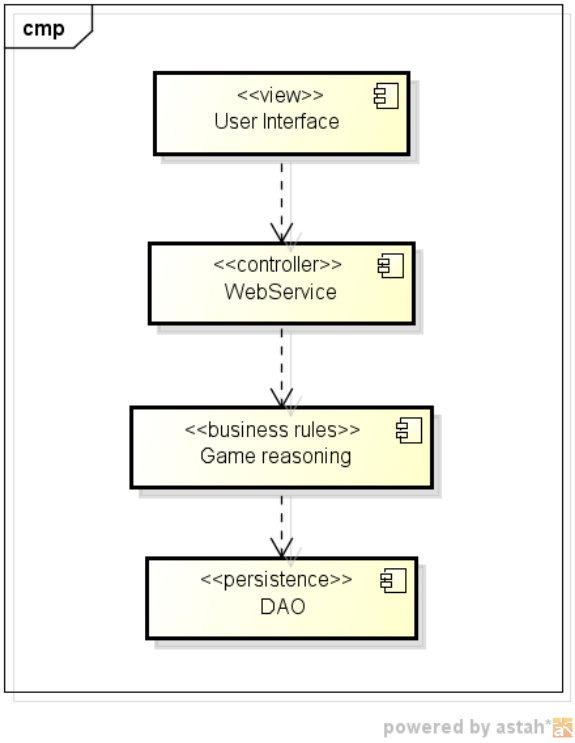

Figura 4. Arquitetura do jogo InspectorX

As camadas, suas responsabilidades e alguns detalhes de sua implementação são descritos a seguir:

- A camada de visão é responsável pela interação com o jogador, capturando e exibindo os dados. A fim de se obter a flexibilidade de capturar com facilidade os eventos de seleção da parte do texto que continha o defeito, fez-se uso da API gráfica do Adobe FLEX, que se baseia no conceito de RIA (*Rich Internet Applications*) [Allaire 2002]. Esta solução foi escolhida pela facilidade de uso e interatividade em tempo real com o jogador.
- A camada de controle foi criada como um *web service* para permitir que múltiplos usuários tenham acesso ao jogo e para fazer a comunicação da camada visual com a camada de negócio. A camada, construída na linguagem C#, possui uma interface que disponibiliza métodos para receber chamadas da camada visual e repassar à camada de negócio para o processamento dos dados.
- A camada de negócio é responsável por processar os dados e fazer chamadas à camada DAO, pois contém a lógica do jogo e regras de validação das respostas.
- Por fim, a camada DAO 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 independentemente de serem relacionais ou não.

4. Trabalhos Relacionados

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.

Em [Prikladnicki & von Wangenheim 2008], é apresentada a motivação para o desenvolvimento de jogos educativos em engenharia de software, e mostram três exemplos de jogos voltados para gerência de projetos. Assim como no presente trabalho, o artigo mostra a relevância do uso de jogos como forma de treinamento. No entanto, o foco do trabalho destes autores está em gerência de projetos, enquanto o InspectorX contempla tarefas de inspeção de software.

No trabalho de [McMeekin *et al.* 2009], os níveis de cognição são analisados com relação a três diferentes técnicas de inspeção de código. 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 utiliza uma abordagem diferente: quando o nível de experiência do jogador em inspeção é baixo, os defeitos já são apresentados em tela, sendo necessário apenas classificá-los, estimulando o aprendizado. Além disso, erros e acertos são computados e corrigidos automaticamente.

Em [Thiry *et al.* 2010], 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) são apresentados. 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 trabalho dos autores não trata de atividades práticas relativas ao processo de inspeção de software.

5. Conclusões

Este artigo apresentou o InspectorX, um jogo voltado para o treinamento e aprendizado em inspeção de software. Espera-se que o jogo seja uma maneira prática e eficiente de treinar a percepção dos jogadores para os defeitos mais comuns, melhorando as práticas de desenvolvimento de software e evitando a recorrência de defeitos no futuro.

Uma das limitações deste trabalho é que há uma dificuldade em elaborar questões que reflitam contextos mais ricos, em que apenas trechos de artefatos podem não ser suficientes (e.g., quando a identificação e/ou o entendimento do defeito envolve outros artefatos). A interface atual do jogo é mais voltada a defeitos em um único artefato, não sendo apropriada para representar defeitos dispersos em vários artefatos.

Um dos próximos passos desta pesquisa é a execução de um estudo de observação. O objetivo, descrito segundo a abordagem GQM [Basili *et al.* 1994] é **analisar** o jogo InspectorX **com o propósito de** caracterizar **com relação à** eficiência, eficácia e melhora no processo cognitivo envolvido em inspeções de artefatos de software, **do ponto de vista de** profissionais da indústria e alunos, **no contexto de** projetos de software. Serão utilizados dados quantitativos (tais como o tempo gasto e o número de acertos) e qualitativos, obtidos por meio de um questionário *follow-up*.

Outros trabalhos futuros incluem (i) integrar o InspectorX a técnicas de leitura, comumente utilizadas em tarefas de inspeção, (ii) permitir a inspeção visual de artefatos baseados em imagem, como diagramas UML, por exemplo, e (iii) incorporar outras classificações de defeitos (e.g. [Walia & Carver 2009]).

Agradecimentos

Os autores agradecem aos alunos Vinícius Ramires, Caio Cesár, Julio Afonso, Leandro de Carvalho e Leonardo Avramesco pelo auxílio no desenvolvimento da aplicação, e agradecem também à CAPES pelo apoio financeiro.

Referências

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.

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.

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.

Kalinowski, M., 2011. “Uma Abordagem Probabilística para Análise Causal de Defeitos de Software”, Tese de D.Sc., COPPE/UFRJ, 357p, Rio de Janeiro.

McMeekin, D. A., von Konsky, 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.

Prikladnicki, R., von Wangenheim, C. G., 2008. “O Uso de Jogos Educacionais para o Ensino de Gerência de Projetos de Software”. In: *I Fórum de Educação de Engenharia de Software (FEES'08)*, XXII Simpósio Brasileiro de Engenharia de Software, Campinas, Brasil.

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.

Shull, F., 1998. “Developing Techniques for Using Software Documents: A Series of Empirical Studies”, Ph.D. Thesis, University of Maryland, College Park.

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.

Walia, G. S., and Carver, J. C., 2009. “A Systematic Literature Review To Identify and Classify Software Requirement Errors”. *Information & Software Technology*, 51, 7, pp. 1087-1109.