UM ESTUDO SOBRE OS REQUISITOS DE JOGOS DE SIMULAÇÃO USADOS NO ENSINO DE ENGENHARIA DE SOFTWARE

RODRIGO MONTENEGRO POSSA

UM ESTUDO SOBRE OS REQUISITOS DE JOGOS DE SIMULAÇÃO USADOS NO ENSINO DE ENGENHARIA DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: RODOLFO SÉRGIO FERREIRA DE RESENDE

Belo Horizonte

Junho de 2011

Possa, Rodrigo Montenegro
P856e Um estudo sobre os requisitos de jogos de simulação usados no ensino de Engenharia de Software / Rodrigo Montenegro Possa. — Belo Horizonte, 2011
xx, 78 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento de Ciência da Computação.
Orientador: Rodolfo Sérgio Ferreira de Resende

1. Computação - Teses. 2. Engenharia de software - Teses. I.Orientador. II. Título.

CDU 519.6*32 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Um estudo sobre os requisitos de jogos de simulação usados no ensino de engenharia de software

RODRIGO MONTENEGRO POSSA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. RODOLFO SÉRGIO FERREIRA DE RESENDE - Orientador
Departamento de Ciência da Computação - UFMG

PROF. JOSÉ LUIS BRAGA
Departamento de Informática - UFV

PROF. CLARINDO ISAÍAS P. DA SILVA E PÁDUA
Departamento de Ciência da Computação - UFMG

PROF. EDUARDO MAGNO LAGES FIGUEIREDO
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 10 de junho de 2011.

Agradecimentos

Agradeço a Deus pela oportunidade de realizar mais este sonho, tendo me dado todas as condições possíveis para que esta graça fosse alcançada.

À Miriam, minha esposa, por ter me incentivado desde antes da seleção para o mestrado, pela cumplicidade durante minhas ausências e por ter sido solidária nas horas de pouca inspiração. Amor, obrigado por me ajudar a romper mais esta barreira!

A meus pais, pela forma digna com que me educaram, ensinando-me a ter perseverança para realizar novos sonhos. Obrigado também pela torcida e pelo incentivo. Valeu Dona Edna e Sr. Samuca!

A minha irmã, Sarah, por me incentivar, pela torcida, e pelo companheirismo.

A minha sogra, Clarice, pelo carinho, tratando-me como um filho, e pela torcida.

Aos amigos, em especial os familiares, os do DCC, os do SERPRO e os do TRT, obrigado pela solidariedade, pelas palavras positivas e pela torcida.

Ao meu orientador, Rodolfo, e à Daniela, praticamente minha co-orientadora, obrigado pelos ensinamentos, pelas críticas e elogios... sempre na medida e na hora apropriada.

"Há um menino
Há um moleque
Morando sempre no meu coração
Toda vez que o adulto balança
Ele vem pra me dar a mão"
(Milton Nascimento)

Resumo

Jogos de simulação podem ajudar no ensino e no aprendizado em diversas áreas da Engenharia de *Software*. Uma razão para sua adoção é que existem alguns experimentos que dão evidências que o jogo de simulação pode ser uma ferramenta efetiva para a melhoria do aprendizado, o entendimento de assuntos complexos e a melhoria da motivação dos estudantes. Uma questão de pesquisa importante é identificar quais requisitos tornam bem-sucedida a adoção de jogos de simulação nos cursos de Engenharia de *Software*. Neste sentido, apresentamos um estudo sobre os requisitos que podem ser usados para avaliar jogos de simulação em termos das teorias de aprendizado aplicadas. Nossa motivação é apoiar os desenvolvedores de jogos e os professores na criação e escolha de jogos de simulação para fins educacionais. Os requisitos foram levantados a partir de revisão da literatura da área. Os jogos de simulação existentes foram identificados e avaliados através de uma revisão sistemática da literatura. Como resultado, verificou-se que algumas teorias de aprendizado, apesar de terem potencial para serem aplicadas, estão sendo pouco contempladas pelos jogos de simulação.

Palavras-chave: Engenharia de *Software*, Jogos de Simulação, Teorias de Aprendizado.

Abstract

Simulation Games can help the teaching and learning in several areas of Software Engineering. One reason for their adoption is that exist some experiments providing evidence that simulation games can be an effective tool for enhancing learning and understanding of complex subjects, and also improving students' motivation. One important research issue is to identify which requirements make the results of the simulation game adoption successful in Software Engineering courses. In this way, we present a study of the requirements that can be used to evaluate simulation games in terms of the applied learning theories. Our motivation is to support games developers and instructors during the creation and selection of simulation games for educational purposes. The requirements have been extracted from the literature of the area. The existing simulation games have been identified and evaluated through a systematic literature review. The results show that some learning theories, even having potential to be applied, are rarely addressed by the simulation games.

Keywords: Software Engineering, Simulation Games, Learning Theories.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Um dos objetivos da Engenharia de *Software* é construir sistemas com qualidade, que além de corretos do ponto de vista funcional, atendam a requisitos não-funcionais cada vez mais estritos. Além disso, a busca por baixo custo e alta produtividade no desenvolvimento do *software* é constante. Infelizmente, as aplicações atuais são produtos complexos difíceis de desenvolver. A falta de boas práticas no desenvolvimento do *software* pode causar problemas típicos, como superação dos custos e prazos estimados, bem como um desempenho insatisfatório da aplicação durante os testes realizados pelo cliente e após a entrega.

Este contexto mudou o foco das organizações desenvolvedoras de *software* e pesquisadores em direção à maturidade das práticas de desenvolvimento de *software*. A criação de produtos de *software* complexos que satisfazem restrições de qualidade, custo e prazo requer a colaboração de vários profissionais cuja preparação reside em cursos de Engenharia de *Software* [Ludi & Collofello, 2001].

Tipicamente, o ensino tradicional de Engenharia de *Software* é baseado em dois componentes: as aulas teóricas, nas quais teorias de Engenharia de *Software* e conceitos relacionados são apresentados; e projetos, nos quais os estudantes devem trabalhar em grupos para desenvolver parte (geralmente pequena) de um *software* [Navarro, 2006]. Entretanto, para a preparação de um bom profissional, os cursos devem trazer consigo não apenas os princípios da Engenharia de *Software*, mas também o conhecimento adquirido em outras áreas, como o gerenciamento de projeto [Claypool & Claypool, 2005], devido a natureza peculiar das aplicações de *software*.

Outro ponto chave para um aprendizado bem-sucedido é a motivação. Se o conteúdo lecionado nas aulas teóricas for de aplicação imediata pelos alunos, eles serão mais engajados. Do contrário, que é o caso de ensino de Engenharia de *Software*, eles prestarão atenção com maior dificuldade, pois não podem imaginar que existe um

problema real por traz do que está sendo ensinado [Drappa & Ludewig, 2000].

A comunidade acadêmica reconheceu a insatisfação da indústria com relação a preparação dos estudantes [Callahan & Pedigo, 2002], e como resposta, criou algumas práticas que tentam ser mais próximas ao que acontece nas organizações do mundo real [McMillan & Rajaprabhakaran, 1999]. Dentre estas práticas, estão a análise de estudos de caso de grandes projetos, treinamentos que integram a sala de aula com a indústria, a execução de programas análogos aos programas de residência médica [Sampaio et al., 2005] e a utilização de jogos de simulação [Ludi & Collofello, 2001].

Segundo Navarro [Navarro, 2006], os jogos de simulação, nos quais os processos de *software* são praticados através de um ambiente baseado em jogo, fazem com que o aprendizado torne-se mais interessante, pois o estudante aplica de forma mais parecida e próxima da realidade os conhecimentos adquiridos. De acordo com Dantas e outros [Dantas et al., 2004], podem reduzir o tempo de treinamento, o orçamento e o risco, se comparados com a imersão dos alunos em projetos reais durante os treinamentos.

O presente trabalho apresenta uma análise de jogos de simulação para o ensino de Engenharia de *Software*. A principal contribuição desta análise é a identificação de quais requisitos tornam os resultados da adoção dos jogos de simulação bem-sucedida em cursos de Engenharia de *Software*. Desta forma, apresentamos um estudo sobre os requisitos que podem ser usados para avaliar os jogos de simulação em termos das teorias de aprendizado aplicadas. Nossa motivação é apoiar desenvolvedores de jogos e professores durante a criação ou escolha de jogos de simulação para propósitos educacionais.

Os requisitos identificados são apresentados na forma de um catálogo. Essa estrutura é uma adaptação da estrutura utilizada no trabalho de Hoffmann e outros [Hoffmann et al., 2004], que realizaram um estudo sobre os requisitos para ferramentas de gerenciamento de requisitos. Entretanto, os requisitos apresentados no trabalho de Hoffmann e outros são mais detalhados do que os deste trabalho. Aqui, eles possuem um nível maior de abstração, sendo que os detalhes são apresentados como possíveis formas de implementação dos requisitos.

1.1 Contextualização e Motivação

A literatura descreve o desenvolvimento de sistemas de *software* como um "problema cruel" (do inglês, *wicked problem*), uma vez que ele é caracterizado pela falta de definição precisa do problema, pela dificuldade de verificar as soluções propostas, pela inadequação de métodos e metodologias para garantir a qualidade dos resultados [Jef-

fries et al., 1981; Sutcliffe & Maiden, 1992], e também devido às interdependências complexas [DeGrace & Stahl, 1990] entre seus componentes.

Armarego [Armarego, 2002] identificou um dilema educacional no ensino de Engenharia de *Software*, no qual:

- Complexidade é adicionada em vez de ser reduzida à medida que se entende melhor o problema;
- Estratégias meta-cognitivas são fundamentais para o processo;
- Um rico conhecimento prévio e boa intuição são necessários para uma efetiva resolução do problema;
- Uma larga experiência é necessária de forma que similaridades e diferenças com relação a estratégias passadas são utilizadas para lidar com novas situações.

Outras questões que dificultam o ensino sobre processos de *software* foram identificadas por Navarro e Hoek [Navarro & Hoek, 2001]:

- O desenvolvimento de *software* é não-linear: atividades, tarefas e fases são repetidas e múltiplos eventos ocorrem ao mesmo tempo. Gerenciar dois projetos similares da mesma forma pode não produzir o mesmo resultado, devido a presença de vários fatores (possivelmente inesperados);
- O desenvolvimento de *software* envolve vários passos intermediários e escolhas contínuas entre múltiplas alternativas viáveis: mesmo com um planejamento cuidadoso, nem todos os eventos que podem ocorrer podem ser antecipados no início do projeto. Decisões difíceis devem ser tomadas; prós e contras, considerados; e conflitos, tratados;
- O desenvolvimento de *software* pode expor efeitos dramáticos com causas não-óbvias: enquanto o desenvolvimento de *software* apresenta vários relacionamentos de causa e efeito, existem outras situações que podem surgir em que a causa não é tão aparente. Por exemplo, a lei de Brooks [Brooks, 1978] enuncia que ao adicionar pessoas a um projeto que está atrasado, tipicamente faz com que o mesmo se torne ainda mais atrasado;
- A Engenharia de *Software* envolve múltiplas partes interessadas: clientes e outras pessoas da organização tomam decisões que impactam o desenvolvimento;

- A Engenharia de *Software* frequentemente envolve múltiplas metas conflitantes: o desenvolvimento de *software* inclui escolhas entre atributos como qualidade e custo, ou confiabilidade e desempenho.

Alguns dos problemas-chave no ensino de uma disciplina tão abstrata como a Engenharia de *Software* são listados por Schön [Schön, 1983]:

- É possível aprendê-la, mas não é didaticamente ou discursivamente ensinável: pode ser aprendida apenas em e através de operações práticas;
- É uma habilidade holística, sendo que as partes não podem ser aprendidas isoladamente;
- Depende da habilidade em reconhecer qualidades desejáveis e indesejáveis do mundo descoberto. Todavia, este reconhecimento não é algo que pode ser lecionado pelos professores. Em vez disso, deve ser aprendido através de prática;
- É um processo criativo em que o projetista chega para ver e realizar atividades de modo inovador. Portanto, nenhuma descrição prévia disso pode tomar o lugar de um aprendizado prático.

De acordo com Connolly e outros [Connolly et al., 2008], as características citadas anteriormente tornam o ensino do processo de desenvolvimento de *software* problemático ao se utilizar apenas abordagens didáticas tradicionais e a experiência prática fica muito aquém do que um estudante pode esperar no mundo real. Eles advogam um paradigma alternativo e complementar de ensino para a Engenharia de *Software* baseado nos princípios da Psicologia Construtivista. Além disso, Spiro e outros [Spiro et al., 1991] argumenta que o aprendizado baseado nos princípios do Construtivismo é provavelmente mais adequado em domínios que envolvem problemas mal-estruturados.

1.2 Objetivos

O objetivo inicial do projeto de mestrado era a investigação do domínio de jogos de simulação indo desde os aspectos conceituais até o desenvolvimento de um arcabouço (do inglês, *framework*) e de um jogo de simulação voltado para o ensino de Computação, em particular, para o ensino de Engenharia de *Software*. Esse arcabouço e esse jogo observariam diversos aspectos, como por exemplo, as teorias de aprendizado relacionadas à Psicologia Construtivista.

Entretanto, conforme discussão realizada no início do Capítulo 6, este escopo inicial foi reduzido em função de vários fatores. Em particular, retiramos do escopo

dos trabalhos a serem descritos nesta dissertação, o desenvolvimento de um arcabouço e de um jogo que funcionasse como prova de conceito. Em função disso, este trabalho descreve as atividades e resultados relacionados à identificação de requisitos que os jogos de simulação para o ensino Engenharia de *Software* devem satisfazer de modo a aplicar os princípios do Construtivismo.

Este trabalho possui dois objetivos específicos. O primeiro é identificar os jogos de simulação existentes para o ensino de Engenharia de *Software*. O segundo é avaliar estes jogos quanto à implementação dos requisitos levantados.

1.3 Limites do Trabalho

Existem vários tipos de jogos que se propõem a ensinar tópicos de Engenharia de *Software*. Há, por exemplo, jogos não-computacionais de tabuleiro (p. ex. [Taran, 2007]) e cartas (p. ex. [Baker et al., 2003]). Jogos computacionais, ou seja, projetados para serem jogados através de um *software*, podem ou não ser jogos de simulação. Dentre os tipos de jogos computacionais que não são de simulação existem, por exemplo, o *Lecture Quizz* [Wang et al., 2008], em que o jogador deve responder às perguntas de forma correta para prosseguir no jogo, e o *Open Software Solutions* [Sharp & Hall, 2000], desenvolvido na forma de um tutorial, em que o jogador caminha pelos ambientes de uma organização desenvolvedora de *software*, assistindo a apresentações, lendo documentos ou participando como ouvinte de uma reunião em cada um dos ambientes disponíveis.

O jogo de simulação é uma abordagem na qual o jogador assume um papel mais ativo, contribuindo de forma decisiva para o desenrolar da história por trás do jogo. Jogos não-computacionais também podem ser de simulação. O jogo *Problems and Programmers* [Baker et al., 2003], baseado em cartas, também é uma simulação na qual os jogadores participam de um projeto de software, cada um no papel de um empregado, e realizam ações que interferem no andamento do jogo. Ao final da seção 2.2, é apresentada uma definição mais precisa do termo "jogos de simulação".

Apesar de existirem vários tipos de jogos para o ensino de Engenharia de *Software*, este trabalho foca exclusivamente na identificação de requisitos para jogos de simulação computacionais. Os jogos considerados aqui podem ou não conter a metáfora de um ambiente profissional. Os que não contêm esta metáfora geralmente destinam-se ao ensino de técnicas isoladas de Engenharia de *Software*, como por exemplo o desenvolvimento de testes de unidade e a estimativa por pontos de função. Os jogos que transportam o aluno para um ambiente profissional permitem que ele se depare

com diversas situações, questões e problemas existentes no cotidiano das organizações desenvolvedoras de *software*, como problemas de comunicação entre os empregados, o surgimento de novos requisitos por parte do cliente, entre outros.

O trabalho foca principalmente nos princípios da Psicologia Construtivista. Conforme já visto na seção 1.1, abordagens baseadas no Construtivismo têm sido propostas por vários autores para apoiarem o ensino de Engenharia de *Software*.

1.4 Trabalhos Relacionados

Na literatura podem ser encontrados alguns trabalhos que identificam e avaliam jogos educacionais para o ensino de Engenharia de *Software*. Estes trabalhos diferem basicamente com relação aos tipos de jogos (computacionais ou não; de simulação ou não) considerados e às perguntas que se deseja responder na avaliação de cada jogo.

Todos os trabalhos identificados foram desenvolvidos por meio de uma revisão sistemática da literatura. Ao contrário de uma revisão da literatura convencional, a revisão sistemática é uma revisão metodologicamente rigorosa dos resultados de outros trabalhos. É um meio de identificar, avaliar e interpretar os trabalhos relevantes a uma questão particular de um trabalho de pesquisa [Kitchenham & Charters, 2007].

O trabalho de Von Wangenheim e Shull [Von Wangenheim & Shull, 2009] avaliou todos os tipos de jogos utilizados no ensino de Engenharia de *Software*. Foram identificados trabalhos que descrevem a aplicação de jogos não-computacionais (p. ex. tabuleiro e cartas), jogos computacionais, de simulação ou não (p. ex. tutoriais e conjunto de questões (do inglês, *quiz*)), e simuladores de laboratório (do inglês, *laboratory simulations* [Alessi & Trollip, 2001, pp. 237]), que não possuem características de jogo (p. ex. diversão). As seguintes perguntas foram respondidas nesse trabalho:

- "Quais os tipos de jogos usados no ensino de Engenharia de *Software*?"
- "Quão efetivos são os jogos para a educação quando comparados com outros métodos de ensino?"
- "Como os jogos podem ser melhor projetados para serem motivadores e úteis, e para melhorar o aprendizado?"

Para responder estas perguntas, Von Wangenheim e Shull analisaram os resultados dos experimentos de avaliação de cada um dos trabalhos identificados. Esses experimentos consistem, na maioria das vezes, de questionários, onde os alunos registram suas impressões sobre o jogo, ou de registros de desempenho de cada aluno em

avaliações realizadas antes e depois (pré-teste e pós-teste, respectivamente) do aluno praticar o jogo.

O trabalho de Von Wangenheim e Shull é uma versão atualizada e ampliada do trabalho de Connolly e outros [Connolly et al., 2007]. A diferença entre os dois trabalhos é que o trabalho de Connolly e outros desconsiderou os jogos não-computacionais e os simuladores de laboratório utilizados para fins educacionais.

Em trabalho relacionado a esta dissertação [Peixoto et al., 2011], publicamos os resultados de uma avaliação de jogos educacionais, levando em conta apenas os jogos de simulação baseados em computador. Nesse trabalho foram apresentadas as principais características dos jogos de simulação e as diferenças entre eles levando-se em consideração seus projetos de interação com a interface gráfica. Esse trabalho comparou os jogos de simulação com relação às seguintes características:

- Metas e regras;
- Retroalimentação (do inglês, *feedback*);
- Mundo virtual; e
- Características de jogo.

No trabalho de Peixoto e outros [Peixoto et al., 2011], realizamos uma revisão sistemática da literatura para responder à seguinte pergunta: "Como as características de projeto são promovidas nos jogos de simulação computacionais para a Engenharia de *Software*?". Para respondê-la, os jogos identificados foram jogados, a fim de que as características citadas acima pudessem ser avaliadas. No referido trabalho, o autor desta dissertação foi o responsável pelo processo de busca que identificou os jogos de simulação existentes para o ensino de Engenharia de *Software*.

1.5 Estrutura do Trabalho

O trabalho está estruturado da seguinte forma:

No Capítulo 2, é apresentada a revisão bibliográfica relacionada ao tema. Conceitos básicos, pré-requisitos para o entendimento e a avaliação de jogos de simulação, são apresentados.

O Capítulo 3 descreve o processo de busca para a identificação dos jogos de simulação que foram avaliados. Em seguida, cada jogo identificado é descrito.

O Capítulo 4 apresenta os requisitos identificados no estudo e organizados na forma de um catálogo.

O Capítulo 5 analisa como cada um dos requisitos identificados foi implementado pelos jogos avaliados.

O Capitulo 6 finaliza a apresentação com uma discussão sobre as contribuições e limitações do trabalho, além de sugestões de trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

2.1 Ensino de Engenharia de *Software*

2.1.1 Paradigmas de Ensino

De acordo com Alessi e Trollip [Alessi & Trollip, 2001, pp. 16], é necessário entender como as pessoas aprendem antes de desenvolver materiais efetivos para facilitar o aprendizado. À medida que você projeta um material educacional, deve sempre pensar sobre os princípios do aprendizado, avaliando se seu material os aplica e é compatível com eles.

Na história contemporânea da Pedagogia podem ser encontradas várias teorias psicológicas que descrevem os processos de desenvolvimento e aprendizado. As principais são descritas a seguir.

2.1.1.1 Psicologia Comportamental

A Psicologia Comportamental começou no início do século XX, primeiramente com o trabalho de Thorndike e Pavlov. Para os adeptos deste paradigma, respostas instintivas básicas a estímulos naturais (p. ex. cachorro saliva quando sente o cheiro de uma comida) podem ser vinculadas a estímulos artificiais. O fisiólogo russo Ivan Petrovic Pavlov (1849-1936) deu a este comportamento o nome de condicionamento clássico [Alessi & Trollip, 2001, pp. 16-17].

Um exemplo de condicionamento clássico é a seguinte situação experimental verificada por Pavlov em laboratório [Coutinho & Moreira, 2001, pp. 58]: se alguém toca um sino enquanto dá comida a um cachorro, caso esta pessoa toque o sino uma outra vez, o cachorro irá salivar sem que se tenha comida por perto. O estímulo incondicional da comida faz surgir o estímulo incondicional do salivar. O sino tocando, por sua vez, é

um estímulo neutro que não faz surgir o estímulo do salivar, mas depois de um período de treino, torna-se condicional gerando uma resposta condicional, que é a salivação. O princípio do condicionamento clássico é que emparelhando-se repetidas vezes um estímulo neutro com um natural faz com que o estímulo neutro se torne condicional, e estimule uma resposta condicional.

Thorndike conduziu pesquisas que resultaram no que é agora conhecido por condicionamento operante [Alessi & Trollip, 2001, pp. 18]: o uso de recompensa e punição para modificar o comportamento. O condicionamento operante deu origem a escola comportamental de psicologia e aprendizado, o paradigma dominante de psicologia do aprendizado durante a maior parte do século XX.

A Psicologia Comportamental deu origem ao Projeto de Sistema Instrucional (do inglês, *Instructional System Design*) [Alessi & Trollip, 2001, pp. 18]. Trata-se de uma abordagem para o desenvolvimento da instrução usada primeiramente na indústria e no exército, onde tentou-se atender a necessidade de desenvolver uma grande quantidade de instruções efetivas que deviam promover o aprendizado. Sua ênfase é em especificar objetivos comportamentais (coisas que o aluno será capaz de fazer ao final da instrução), analisar tarefas de aprendizado e ensinar a especificar níveis para o desempenho do aluno. O Projeto de Sistema Instrucional recebe muitas críticas:

- São complexos e fornecem direção ao nível de currículo mais global, mas não no nível de lição;
- Ignoram importantes aspectos inobserváveis do aprendizado (p. ex. pensamento, reflexão, memória, motivação, individualidade dos alunos, entre outros);
- Embora eles sejam bons para o ensino de resultados previstos, eles frequentemente ignoram saídas não-pretendidas, mas valiosas.

2.1.1.2 Psicologia Cognitiva

Nos anos 70, a Psicologia Cognitiva ultrapassou a Comportamental como paradigma dominante de psicologia do aprendizado.

A vertente mais dominante da Psicologia Cognitiva é a abordagem de processamento da informação. Ela tenta descrever como a informação do mundo entra através dos nossos sentidos, é armazenada na memória, retida ou esquecida, e usada. Os adeptos deste paradigma argumentam que a informação é inicialmente armazenada em memória de curto prazo e deve ser usada e organizada para ser armazenada mais permanentemente em memória de longo prazo. A memória e o pensamento têm capacidade limitada, que leva em conta falhas na atenção e na memória. Há também a

noção do controle de execução, que coordena a percepção, a memória, o processamento e a aplicação da informação pelo aluno [Alessi & Trollip, 2001, pp. 19].

Subjacente a esta abordagem existe a suposição de que os sentidos e o cérebro seguem leis complexas muito sistemáticas e que nós podemos facilitar o aprendizado na medida em que podemos determinar essas leis. Alessi e Trollip identificaram algumas questões centrais da Psicologia Cognitiva que ajudam no projeto de ferramentas multimídia voltadas para a educação. Alguns deles são [Alessi & Trollip, 2001, pp.19-31]:

- **Percepção e atenção:** Para que a percepção dos elementos de uma lição ocorra, a atenção do aluno deve ser atraída e mantida. Isso ocorre através de características dos próprios alunos, incluindo o nível de envolvimento na lição, interesse pessoal no tópico, conhecimento prévio sobre o conteúdo, a dificuldade da lição para eles, e novidade ou familiaridade da informação;

- **Codificação:** Os estímulos devem ser codificados, ou seja, devem ser transformados em um formato que pode ser armazenado no cérebro. A codificação depende de uma série de fatores, incluindo o formato da informação no ambiente (p. ex. idioma utilizado), o meio de informação (p. ex. visual ou auditiva) e o inter-relacionamento de diferentes elementos de informação. Sugere-se ainda que o aprendizado é melhorado quando códigos de informação complementares são recebidos simultaneamente (p. ex. material visual e narração);

- **Memória:** as informações são mais facilmente lembradas quando organizadas na memória e não armazenadas de forma aleatória (p. ex. lembrar de 20 palavras em inglês que representam comidas é mais fácil do que lembrar de 20 palavras aleatórias lidas do dicionário). A repetição (p. ex. utilização e prática dos conceitos aprendidos) também é eficiente para a memorização;

- **Compreensão:** A informação que percebemos deve ser interpretada e integrada em nosso conhecimento atual do mundo. Devemos ser capazes de classificar a informação e não apenas de armazená-la e recuperá-la;

- **Aprendizado ativo:** A abordagem cognitiva considera que as pessoas não aprendem somente observando, mas também fazendo;

- **Motivação:** Deve-se desenhar os materiais para aumentar a motivação ou trabalhar a motivação que o aluno traz consigo;

- **Meta-cognição:** Trata-se da consciência da própria cognição por parte dos alunos. O aluno deve avaliar como está seu processo de aprendizado, revendo suas

atitudes, se for o caso. Algumas técnicas para a inclusão da meta-cognição em programas multimídia são: lembretes para parar e refletir, assistência com auto-avaliações e trabalho em dupla de forma que uma pessoa ajude a outra.

2.1.1.3 Psicologia Construtivista

O Construtivismo é uma visão filosófica que considera que a realidade que interessa é nossa interpretação individual do que percebemos. Contrapõe-se à filosofia objetivista (utilizada nas Psicologias Comportamental e Cognitiva) que diz que percebemos um mundo objetivo através de nossos sentidos e que o aprendizado é o processo de interpretar corretamente nossos sentidos e responder corretamente aos objetos e eventos do mundo real [Alessi & Trollip, 2001, pp. 31].

Existem diferentes escolas de pensamento construtivista [Piaget, 1968; Forman & McPhail, 1993]. Todas elas sustentam que o conhecimento não é recebido de fora, mas que nós o construímos em nossa mente. Em outras palavras, o aprendizado é o processo pelo qual as pessoas ativamente constroem conhecimento.

A abordagem construtivista espalhou-se rapidamente do início ao meio dos anos 90 nos projetos instrucionais, principalmente baseados em multimídia (p. ex. tutoriais, jogos e simuladores) [Alessi & Trollip, 2001, pp. 32]. Na opinião dos adeptos do Construtivismo, a educação estava tratando os alunos como "recipientes vazios onde o conhecimento é derramado", enquanto a educação deveria ser vista como alunos construindo ativamente seus conhecimentos com os professores atuando como treinadores, facilitadores ou mesmo parceiros no processo de aprendizado.

De acordo com Alessi e Trollip [Alessi & Trollip, 2001, pp. 32], os seguintes princípios ou sugestões são tipicamente promovidos como meios para apoiar esta construção do conhecimento:

- Mantenha o foco no aprendizado em vez de no ensino;
- Mantenha o foco nas ações e no pensamento dos alunos no lugar dos professores;
- Mantenha o foco no aprendizado ativo;
- Utilize a descoberta ou abordagens orientadas a descobertas;
- Incentive a construção de informação e projetos pelo aluno;
- Tenha uma fundamentação em cognição situada e sua noção associada de instrução ancorada;
- Utilize atividades de aprendizado cooperativo ou colaborativo;

- Use atividades de aprendizado autênticas ou intencionais;
- Enfatize a escolha e a negociação de metas, estratégias e avaliação de métodos pelo aluno;
- Incentive a autonomia pessoal por parte dos alunos;
- Apoie a reflexão pelo aluno;
- Apoie o senso de propriedade do aluno com relação ao aprendizado e as atividades;
- Incentive os alunos a aceitarem e refletirem sobre a complexidade do mundo real;
- Utilize tarefas autênticas e atividades que são pessoalmente relevantes para os alunos.

2.1.2 Teorias de Aprendizado

Teorias de aprendizado são teorias que descrevem como as pessoas aprendem. Uma boa escolha e utilização de teorias de aprendizado podem tornar o aprendizado mais eficiente e interessante.

A seguir apresentamos algumas teorias de aprendizado que seguem os princípios da Psicologia Construtivista, que é o foco deste trabalho, pois como foi visto anteriormente (vide seção 1.1), é possível criar a partir dela um paradigma de ensino alternativo adequado aos cursos de Engenharia de *Software*.

A teoria "Aprenda Fazendo" (do inglês, *Learning by Doing*) [Angehrn, 2004] é baseada na premissa de que uma pessoa aprende melhor não somente observando, mas também fazendo. Por exemplo, se alguém deseja aprender sobre processo de *software*, precisa participar de um projeto (real ou simulado) para praticar suas atividades.

A teoria do "Aprendizado através da Descoberta" (do inglês, *Discovery Learning Theory*) [De Jong & Van Joolingen, 1998] incentiva os alunos a aprenderem através da exploração, da experimentação, fazendo pesquisa, elaborando questões e procurando respostas. Esta teoria assume que o aluno aprende sobre algo de forma mais eficiente se ele o descobrir por sua própria conta, em vez de tê-lo dito explicitamente a ele.

As teoria do "Aprendizado Situado" (do inglês, *Situated Leaning*) e da "Instrução Ancorada" (do inglês, *Anchored Instruction*) [Alessi & Trollip, 2001, pp. 33] destacam o papel do contexto para a melhoria do aprendizado. A primeira sugere que o aprendizado deve sempre ocorrer em algum contexto, enquanto que a segunda sugere que o contexto deve ser semelhante ao mundo real. Como resultado, o conhecimento ou as habilidades aprendidas em um contexto particular são facilmente repetidos pelos alunos naquele

contexto. Portanto, a "Instrução Ancorada" é melhor do que o "Aprendizado Situado", no sentido de que os alunos podem ver as metas e os problemas tão reais quanto os que eles encontrarão em suas vidas.

A teoria do "Construcionismo" (do inglês, *Constructionism*) [Harel & Papert, 1991] coloca o estudante em um contexto onde ele deve construir ou produzir algo. Durante o processo de construção, o estudante deve negociar uma tarefa, elaborar planejamentos, realizar pesquisas, criar materiais, avaliá-los e revisá-los. No contexto da Engenharia de *Software*, um processo de construção pode ser o estudante no papel de um gerente com as tarefas de planejar e executar um projeto de *software*.

A teoria do "Aprendizado através da Reflexão" (do inglês, *Learning through Reflection*) afirma que um ambiente educacional deve incentivar o aprendizado de como aprender [Lieberman & Linn, 1991], oferecendo aos alunos a oportunidade de refletir sobre e discutir o que eles estão fazendo, seus acertos ou falhas, e o que eles farão a seguir. De acordo com Schön [Schön, 1987], esta teoria enfatiza a necessidade dos alunos de refletirem sobre suas experiências de aprendizado para tornar o objeto de aprendizado mais explícito, concreto e memorável.

O "Aprendizado Cooperativo e Colaborativo" (do inglês, *Cooperative and Collaborative Learning*) [Slavin, 1990] é uma teoria do aprendizado que promove as habilidades sociais, uma vez que os alunos devem ajudar uns aos outros a cumprirem uma meta compartilhada ou a aprenderem sobre algo. Neste caso, a interatividade inclui diálogos entre os alunos, além de outras atividades, e não somente a leitura ou a escrita. Além disso, cada participante exerce o papel tanto de alunos quanto de professores.

Na teoria da "Elaboração" (do inglês, *Elaboration Theory*) [Reigeluth & Rodgers, 1980], a ordem de complexidade da informação a ser apresentada deve aumentar gradualmente. Tarefas simples devem ser apresentadas primeiro, seguidas de versões mais complexas.

A essência da teoria da "Interação de Tratamento da Atitude" (do inglês, *Aptitude Treatment Interaction*) [Cronbach & Snow, 1977] é avaliar o comportamento dos alunos de forma que os professores possam adaptar o ambiente de aprendizado para torná-lo mais eficiente, considerando as individualidades de cada aluno.

Alguns trabalhos [Navarro, 2009; Connolly et al., 2008] afirmam que dentre as abordagens existentes para o ensino de Engenharia de *Software*, o jogo de simulação é a que possui maior potencial para aplicar diretamente as teorias de aprendizado baseadas nos princípios do Construtivismo.

2.1.3 Vantagens dos jogos sobre as outras abordagens

Um dos desafios para o ensino de Engenharia de *Software* é fornecer ao estudante experiência prática suficiente para lidar com as diversas situações inerentes ao desenvolvimento de *software*. É necessário que os estudantes identifiquem quais práticas e técnicas são úteis em várias situações.

Métodos tradicionais para o ensino centrados nos professores, impedem que os alunos apliquem na prática os conceitos vistos em sala de aula. Vários métodos alternativos foram propostos para contornar esta situação, como estudos de caso e a adoção de projetos, através dos quais os alunos interagem para desenvolver um *software* durante o período do curso. Alguns métodos de ensino consistem em dar mais realismo aos projetos, tornando-os mais parecidos com os projetos executados nas empresas desenvolvedoras de *software* [Kornecki et al., 2003; Gehrke et al., 2002]. Outros incluem o ensino de um ou mais assuntos (p. ex. testes de usabilidade, interação com as partes interessadas e aspectos de negócio) que são julgados pelos professores como bastante relevantes para o aprendizado dos alunos [Groth & Robertson, 2001; Crnkovic et al., 2003].

Dentre os métodos alternativos, a utilização de jogos educacionais computacionais é a mais atraente para a maioria dos alunos desta geração, uma vez que eles cresceram usando computadores, *video games*, telefones celulares e outras ferramentas da era digital. Segundo Prensky [Prensky, 2003], os alunos de hoje, por terem este histórico, são chamados de "Nativos Digitais" (do inglês, *Digital Natives*) e "são todos 'falantes' nativos da linguagem digital dos computadores, dos *video games* e da internet".

Os "Nativos Digitais" que jogaram muitos jogos eletrônicos na infância possuem habilidades como lidar com uma grande quantidade de informação de forma rápida, usando caminhos alternativos para obter informações. Esta nova geração prefere realizar atividades simultaneamente usando vários caminhos para uma mesma meta, em vez de realizar uma atividade de cada vez seguindo passos sequenciais. Eles preferem ser ativos, aprender através de tentativa e erro, e descobrir coisas por conta própria em vez de ler e ouvir. Eles querem ser tratados como "criadores e fazedores", em vez de "receptáculos a serem preenchidos com conteúdo" [Prensky, 2003].

Vale ressaltar que os jogos não vieram para substituir os métodos tradicionais de ensino. Vários experimentos (o trabalho de Von Wangenheim e Shull [Von Wangenheim & Shull, 2009] menciona muitos deles) mostraram que os jogos são mais efetivos quando utilizados para complementar os métodos tradicionais.

2.2 Jogos, Simuladores e Jogos de Simulação

Na literatura podem ser encontradas várias definições para os termos "Jogos", "Simuladores" e "Jogos de Simulação".

Heinich e outros [Heinich et al., 2002] definem o jogo como "uma atividade, em que os participantes seguem regras prescritas que diferem daquelas da vida real, visando atender uma meta desafiadora". Segundo Akilli [Akilli, 2007] esta definição carece de dois elementos vitais: diversão e criatividade. Akilli considera o jogo como sendo "uma atividade competitiva, criativa e agradável em sua essência, que é limitada por certas regras e requer certas habilidades".

Para Heinich e outros [Heinich et al., 2002], o termo simulação refere-se a "uma abstração interativa ou simplificação de algum fenômeno da vida real". Alessi e Trollip [Alessi & Trollip, 2001] descrevem este termo como sendo "qualquer tentativa de imitar um ambiente ou sistema real ou imaginário".

Embora sejam termos que se referem a conceitos diferentes, jogos e simulações têm muitas características em comum. Ambos possuem por trás um modelo que representa algum tipo de sistema e, em ambos, os alunos podem experimentar os efeitos de suas ações [Akilli, 2007]. Gredler [Gredler, 2004] identifica três importantes diferenças entre jogos e simulações:

- Em vez de tentar atender aos objetivos dos jogos, os participantes em uma simulação possuem sérias responsabilidades com privilégios que resultam em consequências associadas;

- A sequência de eventos de um jogo é tipicamente linear enquanto que uma sequência de simulação é não-linear. Em uma simulação, os participantes são confrontados com diferentes problemas, questões, ou eventos causados principalmente por suas decisões anteriores;

- Os jogos consistem de regras que descrevem movimentos possíveis, restrições, privilégios e penalidades para ações ilegais. As regras são totalmente imaginárias, sem relação com o mundo real. Uma simulação, por sua vez, é baseada em um conjunto dinâmico de relacionamentos entre diversas variáveis que são modificadas ao longo do tempo, refletindo processos causais autênticos.

Ainda segundo Gredler, as simulações e os jogos também diferem no sentido de que "simulações não são jogos por si só". Para também serem jogos, elas precisam de elementos estruturais adicionais como diversão, jogabilidade, regras, metas, entre outros. Um exemplo de simulação que não é um jogo é a simulação de laboratório

[Alessi & Trollip, 2001, pp. 213-214], onde os estudantes informam os parâmetros de entrada, executam a simulação e coletam os resultados, sem qualquer outro tipo de interação com o simulador.

Quando uma simulação reúne todos esses elementos inerentes a jogos, utiliza-se o termo "jogos de simulação". O conceito de jogos de simulação considerado neste trabalho, dentre os vários existentes na literatura, é enunciado a seguir: Trata-se de um exercício experimental que transporta os alunos para outro mundo, que simula um contexto ou uma situação do mundo real. Lá, eles aplicam seus conhecimentos, habilidades e estratégias para exercerem determinado papel [Gredler, 2004]. Neste caso, o jogador possui um papel ativo, cujas ações influenciam diretamente a execução dos eventos do jogo, bem como as variáveis da simulação.

Conforme mencionado na seção 1.3, este trabalho identificou e avaliou, no domínio da Engenharia de *Software*, apenas jogos de simulação, em vez de jogos e simulações separadamente.

2.3 Arquitetura dos Jogos de Simulação

Os jogos de simulação para o treinamento ou ensino de Engenharia de *Software* são construídos, em sua maior parte, baseados em três componentes [Dantas et al., 2004; Drappa & Ludewig, 2000]: o modelo de simulação, o simulador e a máquina de jogo. A seguir cada um desses componentes será detalhado.

2.3.1 Modelo de Simulação

O modelo de simulação é utilizado para representar a estrutura e o comportamento do processo de desenvolvimento a ser simulado [Dantas et al., 2004]. Ele pode ser configurado e analisado, representando diversas situações distintas que podem acontecer em grandes projetos, com equipes e prazos grandes, ao contrário de projetos-piloto.

Existem diversos métodos para a criação de modelos de simulação. Os mais importantes são a Dinâmica de Sistemas, os Eventos Discretos e o método Híbrido que serão explicados a seguir.

2.3.1.1 Dinâmica de Sistemas

A utilização de modelos de simulação na indústria é bem antiga. Em 1961, Forrester [Forrester, 1961] criou a método Dinâmica de Sistemas (do inglês, *System Dynamics*), que foi desenvolvido para modelar sistemas complexos para melhoria de políticas gerenciais e estruturas organizacionais. Neste método, os modelos são formulados usando-se

quantidades contínuas interconectadas em laços de retroalimentação (do inglês, *feedback loops*) de informação e causalidade circular. As quantidades são expressas em níveis, taxas e ligações representando os laços de retroalimentação.

De acordo com Martin e Raffo [Martin & Raffo, 2000], no contexto de projetos de *software*, a Dinâmica de Sistemas descreve, através de equações diferenciais, o processo de desenvolvimento em termos de fluxos que acumulam em vários repositórios. À medida que a simulação avança em pequenos intervalos de tempo, as mudanças nos fluxos e nos repositórios são computadas. Por exemplo, a taxa de geração de erros pode ser tratada como um fluxo e o número atual de erros pode ser tratado como um repositório. Um modelo de Dinâmica de Sistemas usado em uma organização desenvolvedora de *software* pode ser bastante útil na descoberta dos níveis dos repositórios onde um modelo torna-se instável, ou na previsão de efeitos de uma mudança nas variáveis dos sistemas.

Abdel-Hamid [Abdel-Hamid, 1984] desenvolveu, através de Dinâmica de Sistemas, um modelo integrado que suporta a análise de interações entre atividades relacionadas como codificação, gestão de projeto, testes, contratações e demissões de empregados, entre outras ações gerenciais. Mais tarde, o trabalho realizado por Madachy [Madachy, 1994] resultou na modelagem de um processo de desenvolvimento mais detalhado. Ele utilizou a Dinâmica de Sistemas para modelar o efeito da realização de inspeções formais. Em particular, os impactos de custo, prazo e qualidade foram comparados para taxas diferentes de inspeção através do ciclo de vida dos projetos. Seu trabalho modelou a correlação entre fluxos de tarefas, erros e recursos humanos através de diferentes fases de desenvolvimento.

A figura 2.1, extraída do trabalho de Madachy [Madachy, 2008] apresenta um exemplo de um tipo de diagrama utilizado na Dinâmica de Sistemas, o Diagrama de Repositório e Fluxo (do inglês, *Stock and Flow Diagram*). Neste diagrama, os repositórios são representados na forma de retângulos ("pessoal" e "tarefas desenvolvidas"). O fluxo é representado por uma válvula ("taxa de desenvolvimento de *software*"), podendo conectar dois repositórios ou conectar um repositório a uma fonte ou sumidouro (do inglês, *source* e *sink* [Madachy, 2008, pp. 15], respectivamente). Fontes e sumidouros indicam, respectivamente, fornecimentos e recepções infinitas de informação. No caso da figura, o fluxo conecta o repositório "tarefas desenvolvidas" a uma fonte. Este fluxo é variável, dependendo dos valores da variável auxiliar "produtividade" e do repositório "pessoal". Neste caso, a quantidade de tarefas desenvolvidas aumenta a uma velocidade que depende da produtividade média da equipe multiplicada pela quantidade de empregados trabalhando no projeto.

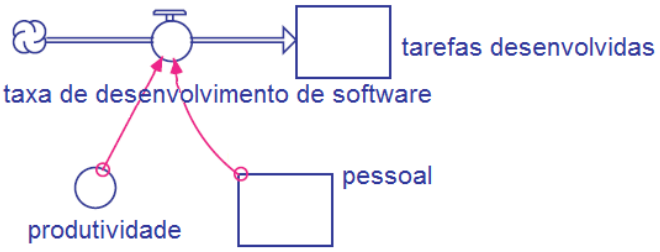

Figura 2.1. Diagrama de Repositório e Fluxo

2.3.1.2 Eventos Discretos

A simulação por Eventos Discretos (do inglês, *Discrete Events*) preocupa-se com a modelagem discreta do sistema que pode ser representado por uma série de eventos. O estado das variáveis de um sistema discreto modifica apenas em intervalos discretos no tempo [Banks et al., 2004].

Um sistema discreto típico é um sistema de filas descrito por seus chamadores, pela natureza das chegadas, pelo mecanismo do serviço, pela capacidade do sistema e pela disciplina do enfileiramento. O estado do sistema é o número de unidades no sistema e o estado do servidor (ocupado ou ocioso). Um evento é um conjunto de circunstâncias que acarreta mudanças instantâneas no estado do sistema.

No contexto do desenvolvimento de *software*, os desenvolvedores são representados como servidores em um modelo de simulação por Eventos Discretos. Eles realizam atividades de desenvolvimento, como codificação, inspeção, testes e retrabalho, em um determinado período. As entidades comuns são os artefatos dos processos, como documentos, pontos de função, código-fonte, entre outros. Alguns modelos de simulação de processos de desenvolvimento de *software* por Eventos Discretos podem ser encontrados nos trabalhos de Raffo [Raffo, 1996] e Padberg [Padberg, 2002].

2.3.1.3 Modelos Híbridos

Zhang [Zhang, 2008] identificou as seguintes diferenças entre os métodos Dinâmica de Sistemas e Eventos Discretos:

- Em Eventos Discretos, o tempo avança em grandes saltos até o próximo evento. Na Dinâmica de Sistemas, o tempo é dividido igualmente em pequenos passos, fazendo com que o estado do sistema simulado seja avaliado continuamente;
- Eventos Discretos possui um foco maior nos elementos individuais do que a Dinâmica de Sistemas. Esta, por sua vez, tende a retratar o sistema como um

todo;

- Através da Dinâmica de Sistemas, é possível representar as estruturas de laços fechados (do inglês, *closed-loop structures*). Nestas estruturas, as saídas da simulação podem influenciar diretamente as variáveis de entrada que lhes deram origem;

- Na modelagem por Eventos Discretos, cada entidade do sistema é monitorada durante a simulação. Na Dinâmica de Sistemas, aproximações são utilizadas no lugar de números exatos de entidades. Os valores médios das variáveis (repositórios e fluxos) são suficientes;

- O método Eventos Discretos é tipicamente empregado na construção de modelos não-determinísticos, o que facilita a representação de eventos e variáveis aleatórios.

No fim do século XX, vários autores identificaram a necessidade de desenvolverem modelos que unissem os benefícios das simulações por Eventos Discretos e por Dinâmica de Sistemas. Kellner e outros [Kellner et al., 1999] alertou para a necessidade de se explorar e desenvolver técnicas para modelagem de simulação híbrida e o entendimento de quando aplicá-las.

Martin e Raffo [Martin & Raffo, 2000] desenvolveram um modelo híbrido para representar os processos de desenvolvimento de *software* como uma série de passos discretos em um ambiente de projeto continuamente variável. O trabalho destes dois autores combina os Eventos Discretos modelados a partir do processo ISPW-6 [Kellner et al., 1991] com o modelo de Dinâmica de Sistemas desenvolvido por Abdel-Hamid [Abdel-Hamid, 1984]. Esta combinação de paradigmas cria oportunidade para examinar novos problemas e questões que são de extrema relevância para as partes interessadas de um projeto.

Através de um modelo de simulação híbrido de um projeto de *software* é possível representar, de acordo com Donzelli e Iazeolla [Donzelli & Iazeolla, 2001]:

- Em um nível mais alto de abstração, a estrutura do processo de *software*, suas atividades, interações e trocas de artefatos. Esses aspectos são melhor modelados através do método Eventos Discretos;

- No nível mais baixo de abstração, o comportamento das atividades e os fatores que influenciam o ambiente do projeto em execução. Esses aspectos são melhor modelados através do método Dinâmica de Sistemas;

A maioria dos jogos de simulação identificados neste trabalho (ver seção 3.1 para acessar a lista completa dos jogos) baseia-se em modelos de simulação híbridos, que utilizam aspectos da Dinâmica de Sistemas e de Eventos Discretos para contemplarem as seguintes características do projeto:

- Eventos Discretos: adequado para representar as regras que ocorrem em intervalos discretos no tempo, como as que devem ser executadas no início e no término de cada atividade. Além disso, é mais adequada para representar os elementos individuais (p. ex. empregados, cliente, artefatos, ferramentas e relacionamentos entre eles) do projeto que está sendo simulado;

- Dinâmica de Sistemas: adequado para representar as regras de causa e efeito que são executadas continuamente no projeto, fazendo com que atributos globais sejam modificados.

O modelo de simulação dos jogos também contempla o estado inicial dos elementos individuais da simulação, com os valores iniciais de cada um de seus atributos.

2.3.1.4 Exemplo de Modelagem Híbrida em um Jogo de Simulação

O modelo de simulação do jogo *The Incredible Manager* [Dantas et al., 2004], que é um dos jogos avaliados neste trabalho, utiliza o conceito de Meta-modelos de Dinâmica de Sistemas para representar a estrutura do processo de *software* (figura 2.2, que descreve os empregados, as atividades e as relações entre estes elementos), o comportamento do projeto (figura 2.3, que descreve o cenário de perda de produtividade devido à necessidade de comunicação entre os empregados) e o estado inicial do jogo (figura 2.4, que descreve os atributos iniciais para os empregados, bem como para as atividades e relacionamentos do projeto). As figuras foram extraídas do trabalho de Dantas e outros [Dantas et al., 2004].

A utilização de meta-modelos consiste em criar o modelo de simulação em alto nível de abstração, através de conceitos do fenômeno a ser estudado (p. ex. empregados, atividades, projeto, atributos), em vez de usar repositórios, taxas, variáveis auxiliares e equações diferenciais [Barros et al., 2002]. Um compilador realiza a tarefa de converter estes conceitos em equações da Dinâmica de Sistemas. Esta técnica torna mais rápida a elaboração dos modelos e permite a utilização da Dinâmica de Sistemas para representar aspectos inerentes a Eventos Discretos, além do cenário inicial dos jogos.

```
MODEL ProjectModel
{
   CLASS Developer
   {
      PROPERTY   Expertise      0;   # range [0, 1]
      PROPERTY   HourlyCost     0;   # in $
      PROC       Productivity   1;   # range [0, 1]
   };
   CLASS Activity
   {
      PROPERTY   ExpectedDuration 0; # in days
      STOCK      Cost             0;
      RATE (Cost) RTCost IF (Executing, Team.Cost, 0);
   };
   MULTIRELATION Precedences Activity, Activity (Successors);
   RELATION Team Activity, Developer;
};
```

Figura 2.2. Exemplo de modelo de estrutura usado no jogo *The Incredible Manager*

```
SCENARIO ProductivityLossDueCommunication ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      PROC  COHFactor Count ([Developer]);
      PROC  COHModifier LOOKUP (COHTable, COHFactor, 0, 30);
      TABLE COHTable 0, 0.015, 0.06, 0.135, 0.24, 0.375, 0.54;
      AFFECT Productivity Productivity * (1 - COHModifier);
   };
};
```

Figura 2.3. Exemplo de modelo de comportamento do projeto usado no jogo *The Incredible Manager*

```
DEFINE Exemplo ProjectModel
{
   Jimmy = NEW Developer
      SET Expertise = 0.4;
      SET HourlyCost = 20;
   Bobby = NEW Developer
      SET Expertise = 0.8;
      SET HourlyCost = 30;
   Design = NEW Activity
      SET ExpectedDuration 10;
      LINK Precedences Analysis;
      LINK Team Bobby;
   Coding = NEW Activity
      SET ExpectedDuration 25;
      LINK Precedences Design;
      LINK Team Bobby;
   ACTIVATE ProductivityLossDueCommunication
      CONNECT TheDeveloper Jimmy;
};
```

Figura 2.4. Exemplo de modelo de cenário inicial usado no jogo *The Incredible Manager*

2.3.2 Simulador

O segundo componente de um jogo de simulação é o simulador, que recebe o modelo como entrada e o interpreta, percorrendo iterativamente suas regras de causa e efeito para calcular o comportamento de cada elemento do fenômeno do mundo real que está sendo estudado.

Os primeiros simuladores utilizados na indústria interpretavam os modelos de simulação criados na forma de equações diferenciais. É o caso da ferramenta `Dynamo` [Pugh, 1976], que foi utilizada para criar e executar os modelos de simulação elaborados por Abdel-Hamid [Abdel-Hamid, 1984].

Os modelos de Madachy [Madachy, 1994] foram desenvolvidos e interpretados pela ferramenta de simulação `ITHINK` [Richmond, 1990], que utiliza diagramas visuais para construir modelos da Dinâmica de Sistemas. Estes diagramas utilizam conceitos como fluxos, repositórios, fontes, sumidouros, e variáveis auxiliares.

Normalmente, as duas ferramentas citadas acima realizam uma simulação não-interativa, na qual o usuário informa os valores iniciais para os parâmetros de entrada dos modelos, inicia a simulação e aguarda sua conclusão para visualizar os resultados. Simuladores interativos, através dos quais o usuário pode interromper a simulação, modificar os valores dos parâmetros de entrada e continuar a simulação a partir do ponto em que ela foi interrompida, são pré-requisitos para a criação de jogos de simulação educacionais [Dantas et al., 2004].

Um exemplo de simulador interativo, desenvolvido na ferramenta `ITHINK`, é o utilizado no trabalho de Collofello [Collofello, 2000]. Nesse simulador, uma interface amigável é apresentada ao usuário, onde ele define os valores iniciais da simulação. O simulador desse trabalho foi configurado para ser interrompido ao término de cada iteração do projeto de *software* simulado. Neste momento o usuário deve definir valores para a próxima iteração. Em seguida, a simulação prossegue considerando-se os novos valores. A figura 2.5 apresenta a interface gráfica onde o usuário informa valores de entrada para serem utilizados em cada iteração da simulação.

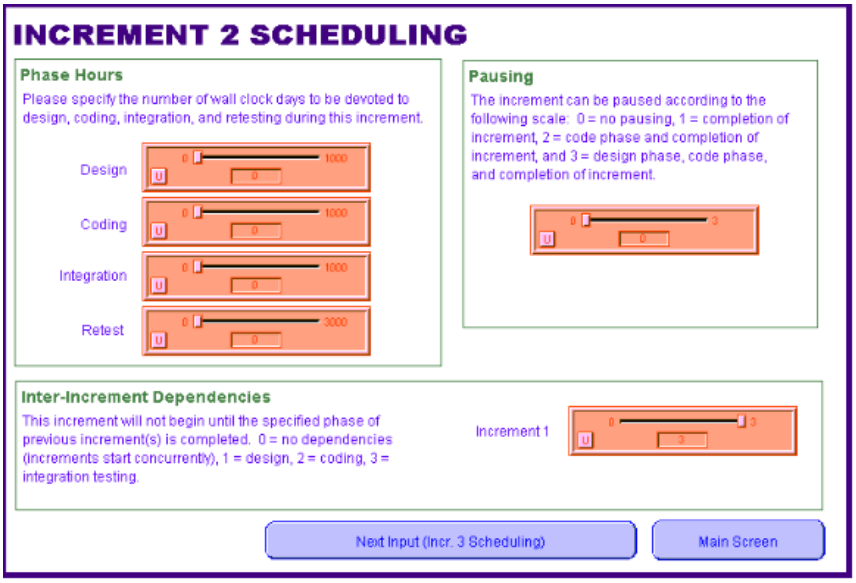

Figura 2.5. Tela de entrada para o simulador usado no trabalho de Collofello [Collofello, 2000]

Ao contrário do simulador desenvolvido no trabalho de Collofello, os simuladores de jogos de simulação educacionais não são responsáveis pela interação com o usuário.

Eles apenas interpretam o modelo de simulação e oferecem mecanismos para que as simulações possam ser interrompidas, reiniciadas e os valores dos modelos, modificados. Existe um componente que é o responsável por controlar a interação do usuário com o simulador por meio de uma interface visual típica de jogos. Trata-se da "Máquina de Jogo" que será descrita na próxima seção.

2.3.3 Máquina de Jogo

O terceiro componente de um jogo de simulação é a "Máquina de Jogo", com a qual o jogador interage, recebendo retroalimentação visual dos resultados da simulação e através da qual o jogador modifica os parâmetros do modelo.

Os primeiros jogos de simulação para o ensino de Engenharia de *Software* eram baseados em interfaces textuais. Nestes jogos, a interação ocorre através de comandos pré-definidos escritos pelo jogador ou de perguntas realizadas pelo jogo. No jogo SESAM [Drappa & Ludewig, 2000], que é um dos avaliados neste trabalho, o jogador interage com o simulador através de um subconjunto simples de sentenças em inglês. As mensagens que o SESAM envia ao jogador são traduzidas para linguagem natural usando-se um componente dicionário. Este mesmo componente é utilizado para analisar as mensagens enviadas pelo jogador.

O próximo passo na evolução da interação dos jogos com os usuários foi a utilização de interfaces gráficas. Este tipo de interface permite a representação dos objetos utilizados na simulação e do relacionamento entre eles de forma mais parecida com suas representações no mundo real. Desenvolvedores são representados como pessoas sentadas em suas mesas de trabalho. Pessoas da alta gerência podem aparecer no ambiente de desenvolvimento reclamando de um projeto que está atrasado.

A figura 2.6 mostra duas representações gráficas possíveis de um desenvolvedor no jogo *The Incredible Manager* [Dantas et al., 2004]. O desenvolvedor fica sonolento, quando nenhuma tarefa é atribuída a ele. Ele torna-se cansado quando está responsável por várias tarefas. O pânico é manifestado quando o projeto ou a tarefa especificamente atribuída ao desenvolvedor estão atrasados.

As interfaces gráficas dos jogos mais recentes são baseadas em mundos virtuais em três dimensões (3D). O mundo virtual é um ambiente de simulação computacional semelhante ao mundo real. Através dele, os usuários interagem por intermédio de *avatars*, que são representações dos usuários em um mundo virtual. O ambiente *Second Life* [Rymaszewski et al., 2007] é um dos mais populares utilizados na hospedagem de mundos virtuais 3D.

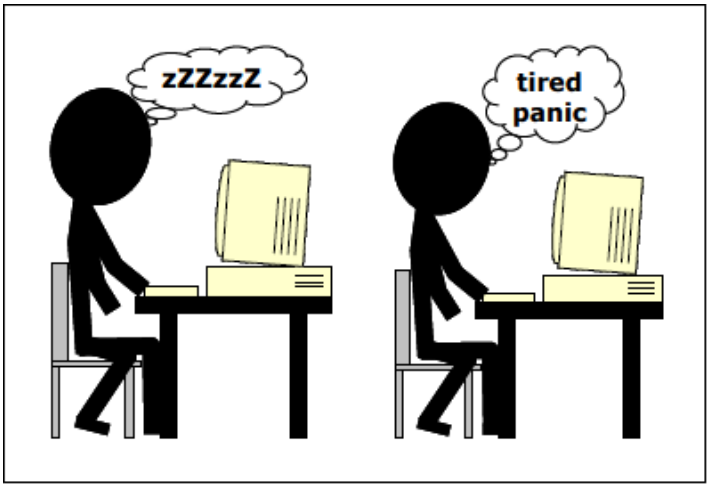

Figura 2.6. Possíveis situações do desenvolvedor no jogo *The Incredible Manager*

O *Second Life* fornece poderosas ferramentas para a criação de mundos virtuais 3D e uma linguagem para a criação dos roteiros (do inglês, *scripts*) que controlam os rumos da simulação. Além disso, possui suporte para a criação de jogos multi-usuários, que podem jogar em rede. Dentre as ferramentas de comunicação disponibilizadas pelo *Second Life* destacam-se o bate-papo (do inglês, *chat*), usado para conversas públicas entre os usuários, e a mensagem instantânea (do inglês, *instant messenger*), utilizada para conversas particulares entre dois ou mais usuários.

O jogo MO-SEProcess [Wang & Zhu, 2009] foi desenvolvido e hospedado no ambiente *Second Life*. Um mundo virtual representando uma organização desenvolvedora de *software* foi criado para o jogo. A figura 2.7 mostra parte deste mundo virtual. Nele, além de controlar os passos da simulação, cada jogador pode mover-se, escolher uma sala de trabalho, sentar-se em uma cadeira, levantar-se e comunicar-se com outros usuários através das ferramentas disponibilizadas pelo *Second Life*. Mais detalhes sobre este jogo são apresentados na seção 3.8.

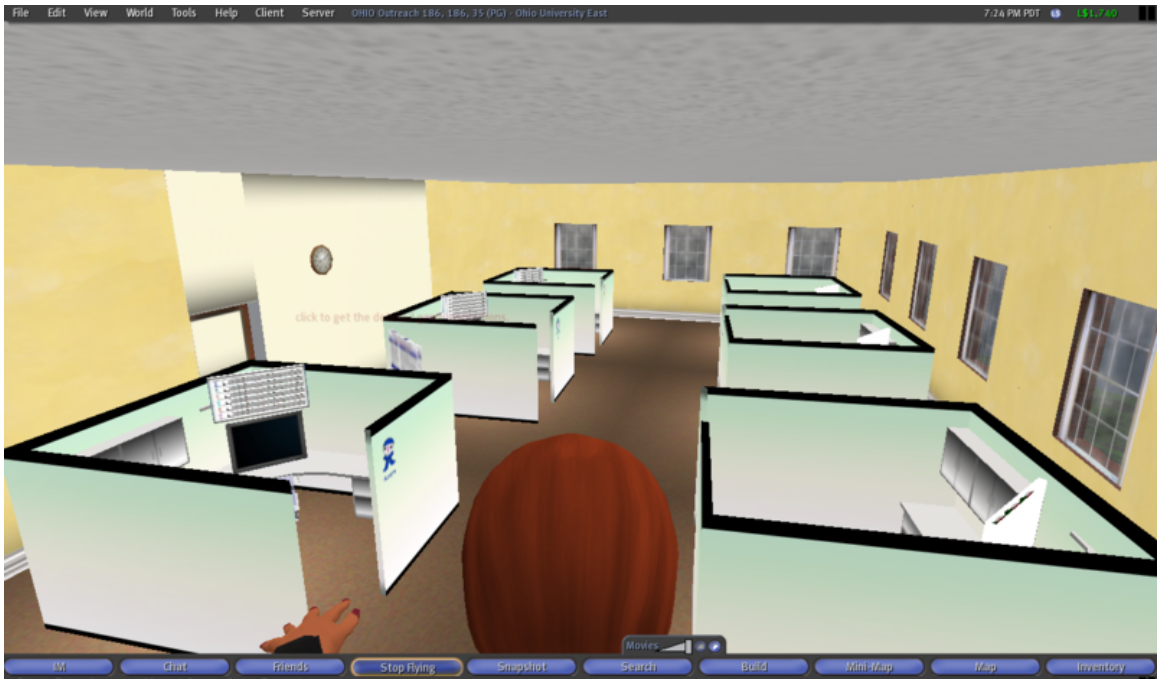

Figura 2.7. Tela do jogo MO-SEProcess

Capítulo 3

Jogos de Simulação Analisados

3.1 Identificação dos Jogos

Nesta dissertação foi realizada uma revisão sistemática da literatura considerando-se apenas jogos de simulação computacionais. Um dos passos para a realização de uma revisão sistemática da literatura é definir as questões de pesquisa a serem respondidas [Kitchenham & Charters, 2007]. Esta revisão visou responder às seguintes questões de pesquisa:

- Quais características dos jogos de simulação promovem a aplicação de teorias de aprendizado associadas aos princípios da Psicologia do Construtivismo?
- De que forma cada jogo avaliado implementa as características identificadas?

O processo de busca utilizado na revisão sistemática da literatura para identificar os jogos de simulação existentes para o ensino de Engenharia de *Software* consistiu em pesquisar as bases de dados de um conjunto de seis periódicos e conferências reconhecidos internacionalmente. São eles: *ACM Digital Library*, *Sage Simulation&Gaming*, *IEEEXplore*, *ScienceDirect*, *SpringerLink* e *Wiley Interscience database*.

Estas bases de dados foram escolhidas por já terem sido utilizadas em revisões sistemáticas de outras pesquisas na Engenharia de *Software* [Von Wangenheim & Shull, 2009] e por possuírem artigos revisados de periódicos e conferências sobre Engenharia de *Software*, ou especificamente, sobre jogos de simulação.

Foi submetida a seguinte expressão de pesquisa nestas bases de dados: *"software engineering" and ("simulation" or "game")*. Além disso, esta busca foi refinada para retornar trabalhos publicados em um período de doze anos (entre 1o de janeiro de 1999 e 31 de dezembro de 2010). Os resultados da busca totalizaram 2583 trabalhos

candidatos. Através de um critério de inclusão e exclusão, restringimos o resultado a artigos relevantes para nosso trabalho. Este critério consistiu em aplicar os seguintes passos:

- Leitura do título para eliminar trabalhos irrelevantes;
- Leitura do resumo e das palavras-chaves para eliminar trabalhos adicionais irrelevantes;
- Leitura do restante dos trabalhos, considerando somente aqueles que tratam das questões de pesquisa apresentadas no início desta seção.

Como nosso interesse é apenas em jogos de simulação computacionais, desconsideramos jogos de simulação não-computacionais, como os jogados em tabuleiro [Taran, 2007], e tutoriais multimídia [Sharp & Hall, 2000] sobre Engenharia de *Software*.

Através desta abordagem o resultado inicial foi reduzido para 17 trabalhos relevantes que descrevem 7 jogos de simulação para o ensino de Engenharia de *Software*. A tabela 3.1 apresenta a lista dos jogos de simulação identificados, bem como as referências para seus respectivos trabalhos. O jogo de simulação SimVBSE [Jain & Boehm, 2006] também foi identificado nessa busca, entretanto foi desconsiderado, pois ele não está mais disponível para ser jogado. Além do mais, o trabalho que o descreve não apresenta detalhes suficientes sobre a arquitetura de suas partes (máquina de jogo, simulador e modelo de simulação). Esta falta de informação impediu que as questões de pesquisa fossem devidamente respondidas para este jogo.

Tabela 3.1. Jogos de simulação identificados através da revisão sistemática da literatura.

Jogo	*Trabalhos Identificados*
qGame	[Knauss et al., 2008]
SESAM	[Drappa & Ludewig, 1999] e [Drappa & Ludewig, 2000]
SimjavaSP	[Shaw & Dermoudy, 2005]
SimSE	[Navarro & Hoek, 2002], [Navarro & Hoek, 2004], [Navarro & Hoek, 2005a], [Navarro & Hoek, 2005b], [Navarro & Hoek, 2007] e [Navarro, 2009]
The Incredible Manager	[Barros et al., 2002] e [Barros et al., 2006]
TREG	[Vega et al., 2009a] e [Vega et al., 2009b]
MO-SEProcess	[Ye et al., 2007], [Zhu et al., 2007] e [Wang & Zhu, 2009]

Vale ressaltar que este processo de busca é o mesmo utilizado no trabalho de trabalho Peixoto e outros [Peixoto et al., 2011], resultando nos mesmos jogos de simulação identificados por esse trabalho. A principal diferença desta dissertação para o trabalho de Peixoto e outros está nas questões de pesquisa.

A seguir são apresentados os jogos de simulação identificados através do processo de busca descrito acima.

3.2 qGame

O qGame é um jogo de simulação mono-usuário *online* para o ensino do tópico de Engenharia de Requisitos. O principal desafio do jogo é evitar os erros durante desenvolvimento do produto. Revisões de artefatos ajudam a superar este desafio, mas consomem tempo, que pode ser necessário para a execução de outras atividades. O mesmo vale para a prática de documentar os artefatos, como a especificação de requisitos e o projeto do sistema. Logo, o jogador precisa balancear velocidade e qualidade, devendo corrigir os erros o quanto antes no projeto. Isso ajuda a experimentar as vantagens da Engenharia de Requisitos realizada de forma sistemática.

O jogo é baseado no conceito de "*quantum* de *software*" onde cada item de informação do projeto (p. ex. um requisito) é um *quantum*. Na interface gráfica do jogo, os *quanta* são representados por círculos coloridos sendo trocados pelas partes interessadas (desenvolvedores nos papéis de analista de requisitos, projetistas e implementadores, além do cliente). A troca dessas informações leva a falhas de comunicação. A consequência é que nem todos os requisitos são representados no produto final ou mesmo requisitos incorretos são implementados e entregues ao cliente.

A figura 3.1 apresenta a tela do jogo. Um *quantum* na cor verde indica os requisitos da forma como foram enunciados pelo cliente. O surgimento de um *quantum* azul no decorrer do processo indica que o requisito foi mal-entendido e, por consequência, foi projetado ou implementado incorretamente. O desaparecimento de um *quantum* indica que o requisito do cliente não foi projetado ou implementado.

3.3 SESAM

O SESAM é um jogo de simulação mono-usuário utilizado para o ensino de gerenciamento de projeto de *software* em que os alunos exercem o papel de um gerente de projeto. O jogador pode contratar empregados, atribuir tarefas a eles, controlar o progresso do projeto, entre outras. O simulador registra internamente os valores de

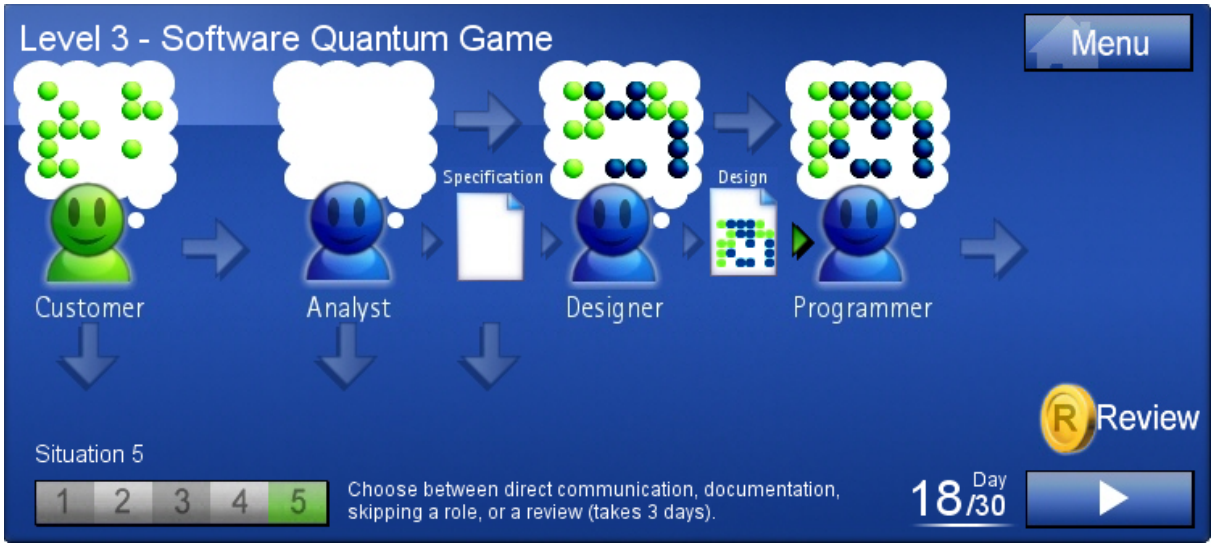

Figura 3.1. Tela do jogo qGame

variáveis e fornece informações sobre o estado atual do projeto para o usuário. Quando o jogo termina, o jogador recebe o resultado na forma de um placar e analisa seu desempenho.

Existem duas versões do jogo: uma textual e outra com interface gráfica. Nesta avaliação foi levada em conta apenas a versão textual, visto que os dois trabalhos identificados no processo de busca que descrevem o SESAM referem-se a esta versão.

Na versão textual, o jogador interage com o jogo através de uma linguagem natural que pode ser interpretada pelo simulador. Ao iniciar o jogo, uma mensagem informa o jogador sobre o cenário. A meta do jogador consiste em tornar o projeto um sucesso, não esgotando o prazo nem o orçamento definidos no cenário inicial, além de entregar um produto com poucos erros ao cliente. A figura 3.2 apresenta a tela do jogo, que consiste em diálogos do jogador, no papel de gerente de projeto, com seus empregados.

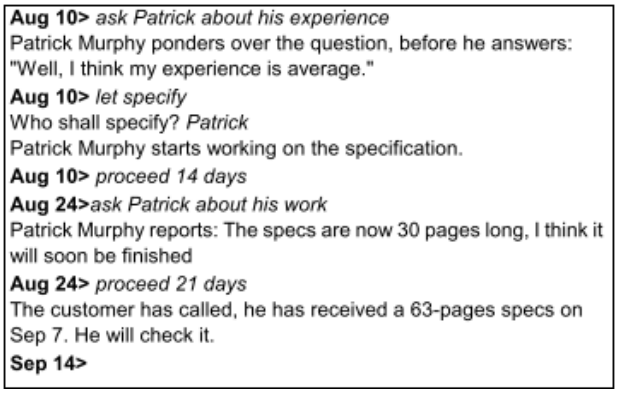

Figura 3.2. Tela principal do jogo SESAM

Ao término da simulação, o jogador recebe o resultado na forma de um depoimento do cliente, que relata suas impressões sobre o produto de *software* desenvolvido, bem como sobre o projeto de *software* realizado.

3.4 SimjavaSP

O SimjavaSP é um jogo de simulação mono-usuário para o ensino de gerenciamento de projeto. Os alunos exercem o papel de um gerente de projeto com a tarefa de desenvolver um produto de *software* hipotético dentro do prazo e do orçamento disponíveis, e com qualidade aceitável. Os jogadores podem realizar ações como demitir empregados, modificar o tempo alocado para tarefas de validação e verificação, atribuir uma tarefa, entre outras. As interações com o jogo ocorrem através de painéis de controle gráficos que mostram o estado atual dos atributos do processo e do produto, permitindo-se a execução de ações gerenciais. O jogo termina quando o projeto está 100% completo ou quando o jogador esgota os recursos de orçamento ou de prazo disponíveis para o projeto.

Assim como em projetos reais de *software*, o jogador, no papel de gerente, é inteiramente responsável por planejar, gerenciar pessoas, dirigir e controlar o projeto. Deste modo, os alunos percebem que os resultados do projeto estão diretamente relacionados a suas próprias decisões.

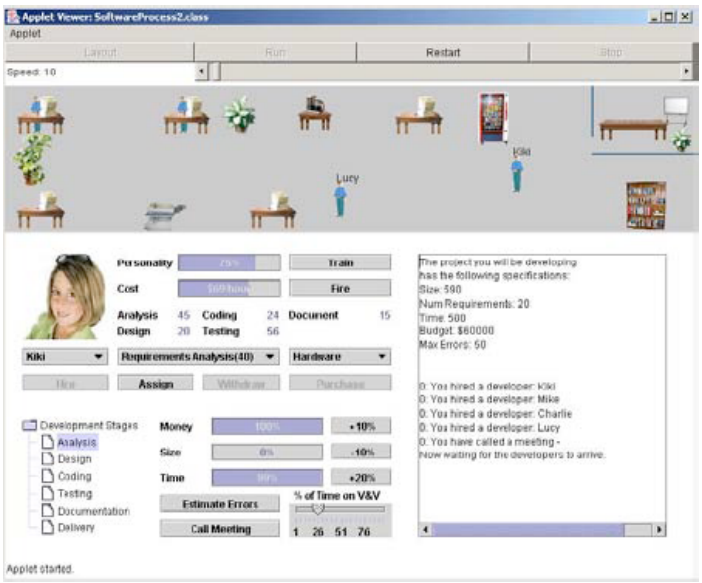

Figura 3.3. Tela do jogo SimjavaSP

A figura 3.3 apresenta a tela do jogo. No painel superior é possível ver uma animação com os empregados trabalhando no projeto. No painel central, o jogador pode

obter informações sobre cada um dos empregados, além de efetuar ações sobre eles, como treiná-los ou demiti-los. No painel inferior, são exibidas informações sobre o projeto no geral, bem como sobre cada uma de suas etapas (análise, desenho, codificação, testes, documentação e entrega). Neste painel, o jogador pode executar ações sobre o projeto como um todo (p. ex. convocar uma reunião).

3.5 SimSE

O SimSE é um ambiente de simulação mono-usuário personalizado e baseado em jogo. É destinado a educar alunos com relação aos tópicos de gerenciamento de projeto e processos de *software*. Atualmente, personalizações para seis modelos estão disponíveis, cada um correspondendo a um tipo de ciclo de vida de projeto que se deseja ensinar: Cascata, Incremental, Programação Extrema (do inglês, *Extreme Programming*), Prototipação Rápida, RUP (abreviação do inglês, *Rational Unified Process*) e Inspeções.

No jogo os alunos exercem o papel de um gerente de projeto e precisam gerenciar uma equipe de desenvolvedores para completarem com sucesso um projeto de *software*. O jogador guia o processo através de ações como atribuir tarefas, dar descanso aos empregados, comprar ferramentas, entre outras. Os valores dos atributos dos objetos (empregados, ferramentas, artefatos e o projeto como um todo) do jogo podem ser monitorados, como a energia e o humor dos empregados, os gastos realizados e o orçamento disponível para o projeto.

A figura 3.4 apresenta a tela principal do jogo. A interface gráfica também pode ser personalizável, mas na prática todos os modelos disponibilizados pelo jogo trabalham com a ideia de um escritório onde os empregados estão dispostos em suas mesas. Sempre que um empregado inicia uma determinada ação, um balão em cima de sua figura é apresentado com um texto descrevendo a respectiva ação. No painel superior, é possível escolher cada um dos objetos do jogo. Os atributos do objeto selecionado podem ser monitorados no painel inferior. No painel direito, o jogador pode escolher um empregado e realizar uma ação sobre ele. Por fim, no painel inferior direito, é possível controlar o progresso do jogo, fazendo com que o tempo progrida até o próximo evento (p. ex. conclusão de uma determinada tarefa ou interrupção do trabalho por motivo de doença) ou avance uma determinada quantidade.

A meta do jogo é desenvolver um projeto de *software* dentro de certas restrições, como prazo, orçamento e qualidade do produto. O jogo termina quando o orçamento ou o prazo disponíveis para o projeto são esgotados ou quando o jogador decide entregar

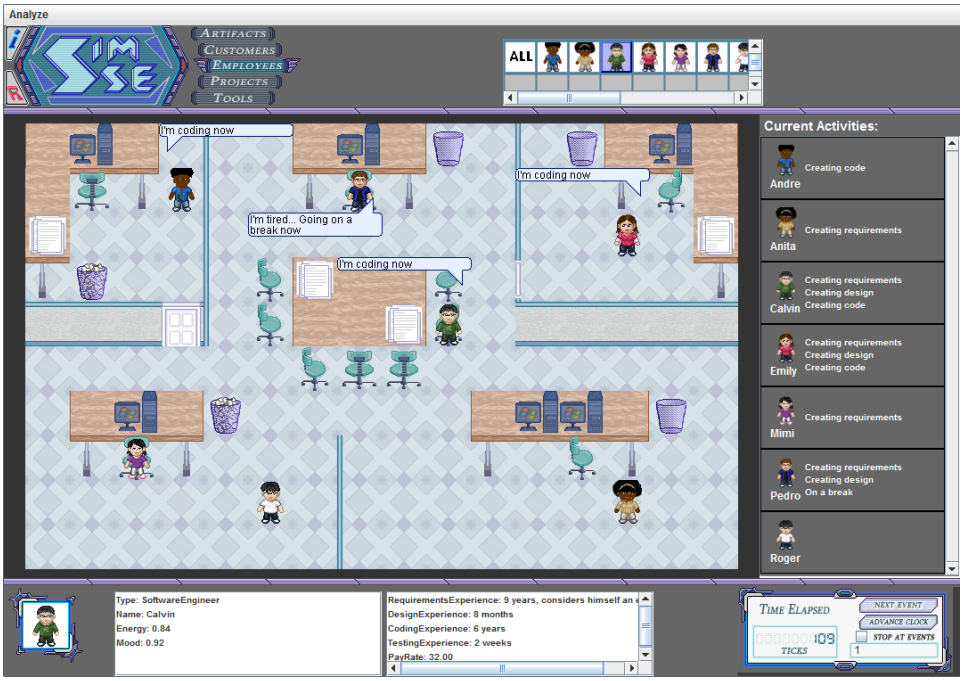

Figura 3.4. Tela principal do jogo SimSE

o produto ao cliente. O placar do jogo, baseado em parâmetros como qualidade do produto, custo do projeto e prazo para conclusão, é apresentado ao jogador. O SimSE oferece ainda a funcionalidade de ferramenta explicativa que apresenta um histórico dos atributos de cada objeto, bem como a descrição das regras que governaram a história do jogo. A figura 3.5 mostra a ferramenta explicativa apresentando um gráfico com o histórico do atributo "energia" do empregado "Mimi".

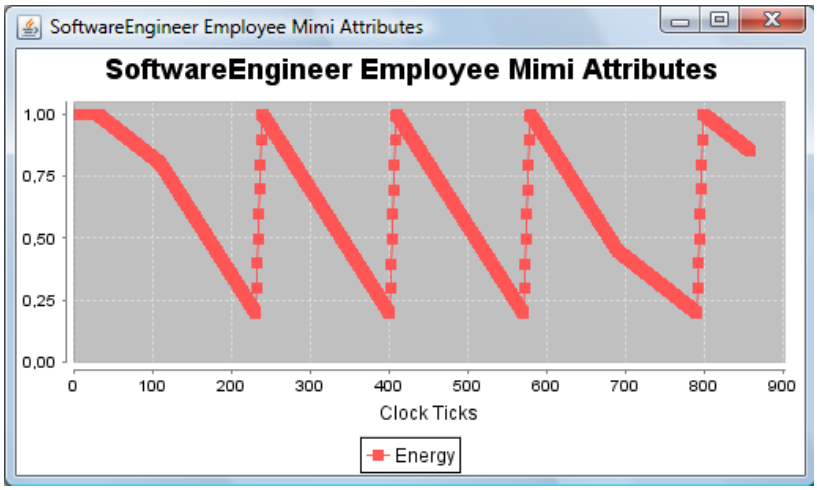

Figura 3.5. Ferramenta explicativa do SimSE

3.6 *The Incredible Manager*

O *The Incredible Manager* é um jogo de simulação mono-usuário em que o aluno atua como gerente de projeto responsável por planejar, executar e controlar um projeto de *software*. A meta do jogo é completar um projeto dentro do custo e do prazo estabelecidos durante a fase de planejamento e aprovados pelas partes interessadas. A execução do projeto ocorre em turnos contínuos, que consomem os recursos planejados. O aluno precisa monitorar a execução do projeto e realizar ações corretivas quando necessário. Efeitos visuais e relatórios oferecem retroalimentações, mostrando desenvolvedores exaustos, tarefas atrasadas, entre outras informações.

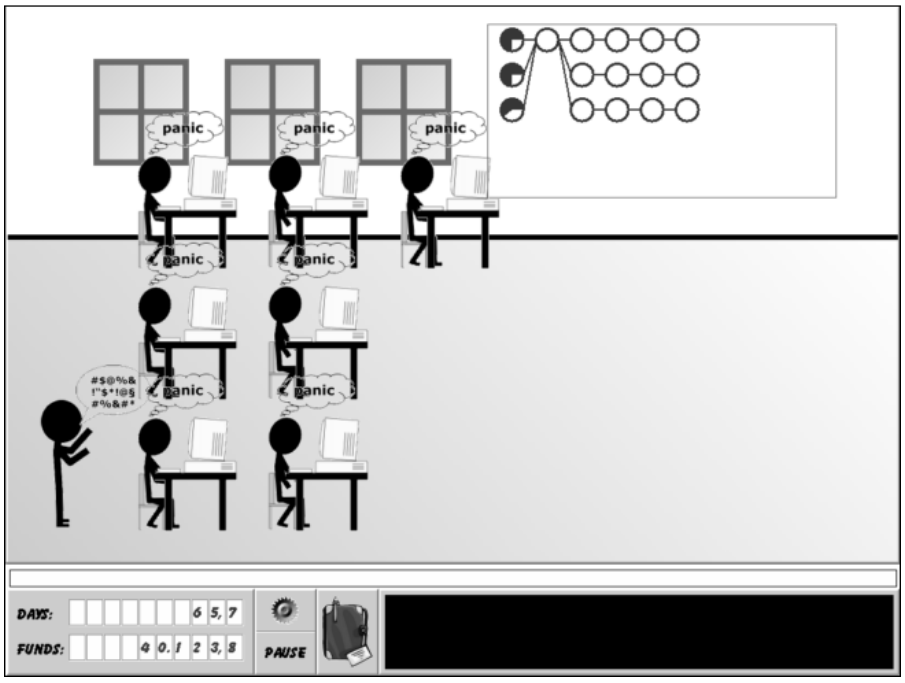

Figura 3.6. Tela principal do jogo *The Incredible Manager*

A figura 3.6 apresenta a interface gráfica do jogo, correspondente à fase de controle do projeto. Nesta fase, o jogador, no papel de gerente, deve monitorar as reações dos empregados, verificando se os mesmos estão em pânico, ociosos ou cansados. O sentimento de pânico, por exemplo, indica que o projeto está atrasado, sem fundos para gastos adicionais ou a tarefa sob responsabilidade do desenvolvedor está atrasada.

Na parte inferior da interface é possível verificar os recursos restantes do projeto (dias e fundos). O jogador deve efetuar um replanejamento do projeto para tomar ações corretivas, como realocar os empregados a uma nova tarefa, demiti-los ou contratar novos desenvolvedores.

3.7 TREG

O TREG é um jogo de simulação destinado ao ensino do tópico "oficina de requisitos". A oficina de requisitos de *software* é uma das formas de ajudar no levantamento de requisitos junto ao cliente e outras partes interessadas. Trata-se de uma sessão de trabalho conduzida por um facilitador que envolve partes interessadas importantes para o projeto e é realizada para se chegar a um consenso sobre os requisitos do projeto, revisar os requisitos existentes e documentar os novos.

Metáforas da vida real são usadas para associar uma tarefa comum da vida real com uma situação de engenharia de requisitos. Por exemplo, quando o aluno está sendo treinado na técnica de oficina de requisitos, uma metáfora de cozinha é proposta. O aluno atua no papel de um chef de cozinha que precisa encontrar ingredientes para a receita "realizando oficinas". O jogo sugere 14 ingredientes para realizar uma oficina com sucesso. A figura 3.7 mostra a parte do ambiente que utiliza esta metáfora.

Figura 3.7. Tela do jogo TREG

Quando o aluno escolhe um dos ingredientes para iniciar a sessão de treino, uma situação relacionada ao ingrediente escolhido é apresentada a ele, revelando múltiplos caminhos que não funcionam.

O *Second Life* [Rymaszewski et al., 2007] foi usado como plataforma para o desenvolvimento do jogo. Trata-se de um mundo virtual 3D onde usuários têm a possibilidade de criarem suas partes daquele mundo. A escolha do *Second Life* foi devido às características de imersão e colaboração inerentes a ele que são necessárias para o jogo TREG. Foram utilizadas as capacidades do *Second Life* para criar a construção principal, salas metafóricas, salas de simulação, jogadores virtuais, entre outros elementos.

A versão disponível do TREG é mono-usuário. Entretanto, ao entrar em contato com os autores do jogo, verificou-se que existe ainda uma versão multi-player não validada e indisponível do mesmo. Como os trabalhos que descrevem o TREG referem-se à versão mono-usuário, esta será considerada no restante deste trabalho.

3.8 MO-SEProcess

O MO-SEProcess é um jogo de simulação *online* multi-usuário, desenvolvido no ambiente *Second Life* (conforme explicado na seção 2.3.3), para o ensino de processos de *software*. Ele é baseado no modelo de simulação do SimSE correspondente ao ciclo de vida Cascata. Os alunos podem escolher um de seis papéis de desenvolvedores disponíveis, formando uma equipe de desenvolvimento junto com outros jogadores. O placar da equipe é apresentado ao final do jogo, se eles entregarem o produto antes do prazo previsto.

Figura 3.8. Tela do jogo MO-SEProcess

A figura 3.8 apresenta a interface gráfica vista por um dos jogadores participantes de uma partida do MO-SEProcess. No painel central, o jogador pode controlar as atividades de seu empregado. É possível solicitar que o empregado realize atividades inerentes às etapas de requisitos, projeto, codificação ou testes. Em cada uma das etapas, o jogador pode solicitar que o empregado elabore artefatos, revise-os, ou os

corrija. A quantidade de erros do artefato sob responsabilidade do jogador pode ser visualizada, bem como a energia restante do respectivo empregado.

No painel superior, é possível acompanhar as atividades dos outros empregados, verificando em que etapa cada um está trabalhando e quais atividades estão sendo realizadas. A partir destas informações, os jogadores podem utilizar recursos de comunicação como bate-papos ou mensagens instantâneas para entrarem em acordo sobre como irão atuar no decorrer do jogo.

A meta do jogo é entregar o produto ao cliente o mais completo possível e com poucos erros. Após a entrega, o jogo apresenta o placar da equipe, que pode variar de 0 a 100 pontos.

Capítulo 4

Requisitos Identificados

O estudo que identificou os requisitos deste trabalho foi realizado em duas etapas. A primeira ocorreu através de revisão da literatura na área de projetos de ambientes educacionais multimídia (p. ex. os trabalhos de Alessi e Trollip [Alessi & Trollip, 2001],Von Wangenheim e Shull [Von Wangenheim & Shull, 2009], e Moreno-Ger e outros [Moreno-Ger et al., 2008]) à procura de boas práticas que viabilizassem a aplicação das teorias de aprendizado apresentadas na seção 2.1.2. Através desta etapa foi possível fazer um levantamento inicial dos requisitos.

Na segunda etapa do estudo, foi feita uma revisão sistemática da literatura para responder as questões de pesquisa mencionadas na seção 3.1. A revisão sistemática, portanto, foi responsável por identificar os jogos apresentados no capítulo 3, por inspecioná-los a procura de novos requisitos e por verificar como eles implementam cada um dos requisitos identificados. A inspeção e a verificação ocorreram através da leitura dos trabalhos que descrevem cada jogo (vide tabela 3.1), bem como através de experimentação dos jogos.

A atividade de experimentação consistiu em jogar cada um dos jogos identificados. A experimentação dos jogos teve o objetivo de procurar características que viabilizassem a aplicação das teorias de aprendizado (novos requisitos) ou que atendesse os requisitos levantados na primeira etapa do estudo (forma de implementação do requisito). Cada jogo foi experimentado por 30 minutos a 3 horas, dependendo de sua complexidade.

Os requisitos identificados são apresentados na próxima seção na forma de um catálogo de requisitos. A estrutura do catálogo utilizado neste trabalho é uma adaptação da estrutura usada por Hoffmann e outros [Hoffmann et al., 2004]. Esses autores descreveram os requisitos para ferramentas de gerenciamento de requisitos e usaram a estrutura hierárquica de requisitos para produtos de *software* proposta no padrão

ISO/IEC para Avaliação de Produtos de *Software* [ISO/IEC, 1991]. Neste trabalho foi utilizada uma hierarquia com dois níveis. O primeiro nível corresponde a um título de subseção e nomeia um determinado assunto de interesse em projetos de ambientes educacionais multimídia. O segundo nível corresponde ao texto que discute vários aspectos relacionados ao assunto. A seguir, o enunciado do requisito tenta simplesmente resumir este texto. Além disso, o segundo nível descreve, após o enunciado do requisito, quais as possíveis formas para sua implementação.

O trabalho de Hoffmann e outros apresentou ainda a priorização de cada um dos requisitos identificados. Esta priorização foi realizada pelas partes interessadas nos requisitos. Neste trabalho, a atividade de priorização dos requisitos não foi realizada.

É importante ressaltar que os requisitos identificados aqui são de interesse dos seguintes papéis:

- O professor que deseja utilizar jogos de simulação como abordagem complementar ao ensino de Engenharia de *Software*. Os requisitos podem auxiliá-lo na escolha dos jogos com maior potencial para aplicação das teorias de aprendizado. Em jogos personalizáveis (p. ex. o SimSE [Navarro, 2006]), o professor pode ainda ter o papel de personalizador, que é quem irá disponibilizar novos modelos de simulação ou alterar os modelos existentes, visando contemplar determinada situação do mundo real ainda não abordada pelo jogo.

- O desenvolvedor de jogos de simulação que, além dos conhecimentos técnicos sobre projeto de jogos, deve conhecer os requisitos que podem tornar os jogos mais eficientes para o aprendizado, bem como suas possíveis formas de implementação;

- O aluno que será o usuário (jogador) dos jogos de simulação escolhidos pelos professores visando complementar seus conhecimentos sobre determinados tópicos de Engenharia de *Software*. Ele aprenderá através das teorias de aprendizado empregadas pelos jogos. Daí a importância dos requisitos identificados neste trabalho, pois suas implementações viabilizam a aplicação destas teorias.

4.1 Transferência do Aprendizado

Uma vez que o aprendizado em uma lição multimídia (p. ex. um jogo de simulação) é frequentemente um precursor da aplicação ou utilização do conhecimento no mundo real, é interessante que o conhecimento adquirido em tais ambientes educacionais seja facilmente transferido para uma situação real [Alessi & Trollip, 2001, pp. 29]. A transferência do conhecimento consiste em aplicar o que é aprendido em um ambiente

educacional nas atividades do mundo real, tais como ser capaz de voar em um avião depois de usar um programa de simulação de voo.

Requisito 1. Um jogo de simulação para o ensino de Engenharia de *Software* (JSES) deve dar suporte à transferência do aprendizado.

A seguir, apresentamos as características mais comuns nos jogos existentes que podem melhorar a transferência do aprendizado. A primeira é permitir que o estudante possua um papel ativo no jogo. Por exemplo, o estudante pode realizar um papel de gerente em um projeto de *software*, tomando decisões sobre processos, ciclos de vida, contratando ou alocando pessoas para então verificar como estas decisões afetam o sucesso do projeto como um todo [Von Wangenheim & Shull, 2009].

A próxima característica é alocar o aluno a um trabalho hipotético ou uma situação de problema, onde ele precisa realizar o trabalho ou encontrar e implementar soluções. A situação de problema mais comum nos jogos de simulação de Engenharia de *Software* é o gerenciamento de projeto, no qual os estudantes atuam como gerente com o intuito de realizar um projeto dentro do prazo e do orçamento, além de entregar produtos de boa qualidade a partir dele.

4.2 Interatividade

A interatividade dos jogos de simulação é o que os torna atraentes e efetivos [Akilli, 2007]. Esta característica diz respeito tanto às ações do jogador quanto às reações do jogo. A interatividade dá ao jogador sensação de controle sobre o jogo, motivando mais os alunos durante o processo de aprendizado. Eles preferem que seja desta forma, ao invés de um ambiente onde o aprendizado ocorre através de observação passiva [Alessi & Trollip, 2001, pp. 229].

Requisito 2. Um JSES deve prover interatividade.

De acordo com Alessi e Trollip [Alessi & Trollip, 2001, pp. 250], quatro tipos de ação podem ser executadas pelo jogador em um jogo de simulação. Essas ações são comuns à maioria dos jogos analisados por este trabalho. São elas:

- Fazer uma escolha;
- Manipular um objeto;

- Reagir a um evento;
- Coletar informação.

Para cada ação, a reação correspondente é importante para dar retroalimentação aos alunos sobre o impacto de tais ações no decorrer do jogo [Alessi & Trollip, 2001, pp. 254].

Entretanto, o aluno não deve ter controle total sobre o jogo. É interessante que os jogos controlem as opções que serão disponibilizadas para o jogador em cada momento do jogo. Jogos de simulação que possuem esta característica são chamados "prescritivos" [Navarro, 2006]. Processos de *software* como o RUP (do inglês, *Rational Unified Process*) são altamente densos com inúmeros passos prescritivos. Se tornarmos estes passos disponíveis em qualquer ponto do jogo, o jogador certamente ficaria confuso com tantas alternativas. Em vez disso, as opções disponíveis em cada passo são limitadas, dependendo da parte do processo em que o jogador se encontra.

4.3 Complexidade do Mundo

Uma crítica que os educadores construtivistas fazem aos ambientes de educação tradicionais é que o conhecimento e as habilidades ensinados são muito simplificados [Alessi & Trollip, 2001, pp. 35]. Ambientes educacionais melhores devem ser projetados com informação, problemas e múltiplas abordagens de soluções, de forma semelhante ao que é presenciado pelas pessoas em suas vidas e trabalhos no mundo real [Savery & Duffy, 1995].

A natureza rápida e flexível das simulações permite que as experiências sejam repetidas e que diferentes situações sejam introduzidas e praticadas. A meta para todos os participantes é que cada um exerça um papel específico, trate as questões, ameaças ou problemas que surgirem em uma situação, e experimente os efeitos de suas decisões. A situação pode tomar diferentes rumos, dependendo das ações e reações dos participantes. Isto é, uma simulação é um estudo de caso envolvente de uma realidade social ou física específica na qual os participantes assumem papéis com responsabilidades e restrições bem-definidas.

REQUISITO 3. Um JSES deve refletir a complexidade das situações e dos problemas encontrados no mundo.

De acordo com Gredler [Gredler, 2004], uma característica importante das simulações é a existência de um modelo adequado da situação complexa do mundo real com

que o estudante interage. Navarro [Navarro, 2006] define este modelo como *preditivo*. Um modelo preditivo especifica relações de causa e efeito que as ações do jogador irão disparar durante a simulação.

Nem todas as características de um fenômeno precisam ser fielmente replicadas em um jogo de simulação. Ao contrário de simulações de laboratório, onde a precisão é de suma importância, nos jogos de simulação aplicados para fins educacionais às vezes é interessante omitir, simplificar, modificar ou mesmo adicionar detalhes e características. Tudo isso visando a um aprendizado mais eficiente. O tempo é uma característica do mundo real que pode ser alterada (eliminada, modificada, etc.) em um jogo de simulação [Alessi & Trollip, 2001, pp. 241]. Trata-se do período de tempo sobre o qual um determinado fenômeno ocorre. Uma vez que projetos podem durar dias, meses ou até anos, o projeto de *software* que é simulado em um jogo precisa ter seu tempo reduzido de forma que as simulações ocorram em tempo hábil.

4.4 Meta-cognição

Etimologicamente, a palavra meta-cognição significa além da cognição, ou seja, a habilidade de conhecer o próprio ato de conhecer. Alguns autores concluíram que bons alunos são mais capazes tanto de usar estratégias para adquirir, organizar e usar seus conhecimentos, quanto de controlar seus processos cognitivos [Flavell & Wellman, 1977]. A meta-cognição pode influenciar a motivação, porque o fato dos alunos serem capazes de controlar e gerenciar seus próprios processos cognitivos dá a eles a noção da responsabilidade por seus desempenhos escolares e cria confiança nas suas próprias capacidades [Jones, 1988]. Logo, supõe-se que a prática da meta-cognição conduz a uma melhora da motivação e da atividade cognitiva, e portanto a uma melhora no processo de aprendizado. Tem sido sugerido que os projetistas e os professores precisam prestar atenção não somente à cognição dos alunos, mas também à meta-cognição dos mesmos [Mayer, 1998].

Segundo Dixon [Dixon, 1991], o foco da maioria dos métodos de ensino é proporcionar experiências, com pouca atenção ao que realmente está sendo aprendido a partir delas. Dixon também advertiu que "frequentemente confundimos experiência com aprendizado, ou proporcionar experiência com educação" e "a experiência por si só não faz nada pelos alunos - ou por qualquer um - a menos que eles aprendam algo cognitivo no processo".

De acordo com Upchurch e Sims-Knight [Upchurch & Sims-Knight, 1999], fornecer ambientes de aprendizado em que os estudantes podem realizar atividades de

reflexão tem quatro resultados positivos que aumentam as habilidades meta-cognitivas:

- Os alunos aprendem melhor o material do curso;
- Eles aprendem como monitorar e regular seus aprendizados;
- Eles aprendem como modificar a maneira como eles trabalham para tornarem-se melhores alunos;
- Eles desenvolvem as atitudes positivas para si e para o aprendizado persistente que lhes permitam exercer a enorme quantidade de práticas deliberadas necessárias para adquirir conhecimento em suas áreas e se engajar no aprendizado contínuo ao longo da vida.

REQUISITO 4. Um JSES deve facilitar a reflexão sobre o conteúdo aprendido.

De modo geral, várias atividades de reflexão foram propostas para o ensino de Engenharia de *Software*, tais como os ensaios [Upchurch & Sims-Knight, 1999], onde os estudantes precisam escrever um texto reflexivo sobre suas experiências de aprendizado, e perguntas direcionadas [Boyer et al., 2010] elaboradas pelos professores com o objetivo de melhorar a compreensão e a decomposição de um problema, incentivando o planejamento antes da implementação, promovendo auto-explicações e revelando lacunas no conhecimento.

Em um jogo de simulação aplicado ao ensino de Engenharia de Software, um elemento com grande potencial no apoio à reflexão é a sessão de esclarecimentos (do inglês, *Debriefing Session*) [Peters & Vissers, 2004], que consiste em, ao final do jogo, verificar o quão próximo o desempenho do aluno ficou do ideal planejado pelos projetistas do jogo e orientá-lo quanto ao que ele deve fazer para obter um resultado melhor. Na pesquisa realizada por Peixoto e outros [Peixoto et al., 2011], também identificou-se a sessão de esclarecimentos como um importante elemento que os jogos devem oferecer visando à prática da reflexão, consequentemente, à melhora da meta-cognição dos alunos.

Um outro elemento que pode ser usado para promover uma atividade de reflexão é a ferramenta explicativa. Ela é responsável por fornecer retroalimentações detalhadas sobre o resultado do jogo, ainda que, ao contrário das sessões de esclarecimentos, não faça uma comparação do desempenho do aluno com o resultado ideal planejado pelos projetistas. A importância desta ferramenta também foi enfatizada na revisão sistemática da literatura realizada por Von Wangenheim e Shull [Von Wangenheim & Shull, 2009].

4.5 Trabalho em Equipe

A Engenharia de *Software* diz respeito tanto aos conhecimentos técnicos quanto às pessoas e ao trabalho em equipe, uma vez que as questões pessoais são centrais em projetos de *software* [Pieterse et al., 2006; Hazzan & Tomayko, 2004]. Portanto, o trabalho em equipe deve ser um importante objetivo de aprendizado para os alunos de Engenharia de *Software*.

Infelizmente, as habilidades do trabalho em equipe, como comunicação e colaboração, têm sido frequentemente sub-valorizadas em cursos de computação que se concentram principalmente em contribuições individuais [Hilburn & Bagert, 1999]. Consequentemente, conforme observado por Denning [Denning, 1992], os empregadores consideram que as novas pessoas não sabem como se comunicar e que eles possuem experiência e preparação insuficientes para trabalhar como parte de uma equipe.

De acordo com Lingard e Berry [Lingard & Berry, 2002], na maioria dos cursos em que projetos em grupo são realizados, os estudantes recebem pouca orientação sobre como ser um membro efetivo na equipe. Seus trabalhos propõem atividades que visam promover o entendimento do processo em grupo bem como estabelecer relacionamentos confortáveis das pessoas com os outros membros da equipe. Dentre as atividades, a *role-playing* [Luca & Heal, 2006] pode frequentemente melhorar as habilidades de trabalho em grupo e a participação dos membros da equipe. Nesta atividade, uma equipe é selecionada e um papel hipotético específico é atribuído a cada um dos seus membros. Em seguida, o grupo deve ser organizado para resolver algum problema relacionado ao projeto. No trabalho realizado por Luca e Heal [Luca & Heal, 2006], foi mostrado que a atividade de *role-playing* foi uma abordagem de ensino eficiente e bem recebida pelos estudantes.

Requisito 5. Um JSES deve dar suporte ao desenvolvimento de habilidades no trabalho em equipe.

Jogos multi-usuário, que disponibilizam aos jogadores papéis que colaboram entre si, facilitam o apoio à atividade de *role-playing*, permitindo que os alunos desenvolvam habilidades no trabalho em equipe.

Jogos mono-usuário, através dos quais os jogadores interagem com Caracteres Não-jogadores (do inglês *Non Player Character*), permitem o desenvolvimento parcial das habilidades do trabalho em equipe. Neste tipo de jogo, há limitações para se representar algumas situações que ocorrem no mundo real, como as falhas de comunicação e os conflitos entre os membros de uma equipe.

Ressalta-se que este requisito é essencial apenas para jogos de simulação que possuem a metáfora de um ambiente profissional, cujo contexto é uma organização ou unidade desenvolvedora de *software*. Jogos que não possuem esta metáfora não devem necessariamente atender a este requisito, pois seu foco geralmente é o aprendizado de uma técnica ou método particular de Engenharia de *Software* e não o trabalho em equipe. Todos os jogos identificados neste trabalho utilizam a metáfora de um ambiente profissional. Logo, o requisito 5 é essencial para todos eles.

4.6 Diferenças entre os Alunos

Os alunos possuem diferentes estilos de aprendizado e nem todas as pessoas aprendem da mesma forma ou com a mesma rapidez. Alguns aprendem melhor assistindo ou ouvindo algo. Outros, lendo. Por fim, outros aprendem melhor fazendo ou através de ambientes que proporcionam experiências práticas. Portanto, durante o desenvolvimento e a implementação de um curso, é importante considerar os estilos de aprendizado dos alunos [Zapalska & Brozik, 2006].

De modo semelhante, os jogos educacionais precisam adaptar seus comportamentos de acordo com o perfil do jogador, de modo a satisfazer as necessidades especiais de cada estudante [Moreno-Ger et al., 2008]. Todavia, existem ainda poucas iniciativas que tentam facilitar o projeto e a integração de comportamentos adaptativos nos jogos educacionais.

REQUISITO 6. Um JSES deve ser adaptável a diferentes estilos de aprendizado.

A interação entre usuário e o jogo pode ser utilizada para adaptar o comportamento dos jogos. Por exemplo, pode ser detectado de forma transparente se o jogador está tentando resolver um problema insistentemente, e então dar a ele uma dica ou diminuir a dificuldade da tarefa gradualmente [Hunicke, 2005].

Um comportamento adaptativo simples que pode ser implementado em jogos de simulação é aumentar gradualmente a complexidade no decorrer do jogo. De acordo com Hunicke [Hunicke, 2005], neste caso, o aumento da dificuldade do jogo é ainda relativamente estático, o que pode conduzir a inadequações entre a habilidade do jogador e os desafios do jogo.

4.7 Rastreamento entre Teorias de Aprendizado e Requisitos

Conforme mencionado no início do Capítulo 4, os requisitos identificados aqui são aqueles cuja implementação facilita a aplicação de uma ou mais teorias de aprendizado baseadas no Construtivismo. A tabela 4.1 apresenta o rastreamento entre as teorias de aprendizado e os requisitos identificados. A marcação com `X` indica que a implementação de um requisito facilita a aplicação de determinada teoria de aprendizado. Caso a aplicação de uma teoria seja condicionada à forma como o requisito é implementado, a marcação com *C* é utilizada.

Na referida tabela, os nomes dos requisitos foram substituídos por mnemônicos de forma que a tabela coubesse em apenas uma página. Os seguintes mnemônicos foram utilizados:

- `TRANSF`: para representar o requisito 1, "Um JSES deve dar suporte à transferência do aprendizado";
- `INTERAT`: para representar o requisito 2, "Um JSES deve ser interativo";
- `COMPLEX`: para representar o requisito 3, "Um JSES deve refletir a complexidade das situações e dos problemas encontrados no mundo";
- `REFLEX`: para representar o requisito 4, "Um JSES deve facilitar a reflexão sobre o conteúdo aprendido";
- `EQUIPE`: para representar o requisito 5, "Um JSES deve dar suporte ao desenvolvimento de habilidades no trabalho em equipe";
- `ADAPT`: para representar o requisito 6, "Um JSES deve ser adaptável a diferentes estilos de aprendizado".

O requisito 1 ("Um JSES deve dar suporte à transferência do aprendizado") é rastreado a partir da teoria "Aprenda Fazendo", pois para implementá-lo os jogos devem colocar o aluno em um papel ativo no jogo, onde ele pode desempenhar ações semelhantes às da vida real. As teorias "Aprendizado Situado" e "Instrução Ancorada" também são rastreadas para este requisito, uma vez que a transferência do aprendizado requer que o jogo ocorra em um contexto com o qual o aluno irá se deparar na vida real (os mesmos problemas, questões e situações). O "Construcionismo", por sua vez, rastreia o requisito 1 desde que o problema a ser resolvido no jogo seja um problema de construção, ou seja, que ele tenha que construir ou produzir algo. A realização de um

projeto de software, que ao final entrega um produto ao cliente, pode ser considerada um problema de construção.

O requisito 2 ("Um JSES deve ser interativo") é rastreado a partir da teoria "Aprenda Fazendo", uma vez que o aluno, através de suas ações, participa ativamente do jogo, em vez de ser um mero observador. Outra teoria a partir da qual o requisito é rastreado é o "Aprendizado através da Descoberta", pois através das ações e retro-alimentações disponíveis, os alunos podem explorar e experimentar melhor o jogo, a procura de melhores soluções para os problemas a serem resolvidos.

O requisito 3 ("Um JSES deve refletir a complexidade das situações e dos problemas encontrados no mundo") é rastreado a partir da teoria "Aprendizado através da Descoberta", pois à medida em que o jogador explora o jogo, as regras de causa e efeito são reveladas, permitindo que ele tenha um desempenho cada vez melhor a partir do conhecimento sobre as mesmas.

O requisito 4 ("Um JSES deve facilitar a reflexão sobre o conteúdo aprendido") é rastreado a partir da teoria "Aprendizado através da Reflexão", pois as ferramentas para reflexão sobre o conteúdo aprendido visam a melhorar a meta-cognição do aluno, incentivando o aprendizado de como aprender melhor.

O requisito 5 ("Um JSES deve dar suporte ao desenvolvimento de habilidades no trabalho em equipe") é rastreado a partir da teoria "Aprendizado Cooperativo e Colaborativo", uma vez que o trabalho em equipe visa a que seus membros cumpram determinadas metas em comum.

O requisito 6 ("Um JSES deve ser adaptável a diferentes estilos de aprendizado") é rastreado a partir da teoria "Interação de Tratamento de Atitude" no sentido de que determinados aspectos do jogo são modificados para se adequarem ao comportamento apresentado pelo aluno durante o processo de aprendizado. A teoria da "Elaboração" também rastreia o requisito 6, desde que a adaptação consista em aumentar gradualmente a dificuldade do jogo, à medida que o jogador apresente um desempenho cada vez melhor.

Tabela 4.1. Rastreamento entre Teorias de Aprendizado e Requisitos.

Teorias vs. Requisitos	TRANSF	INTERAT	COMPLEX	REFLEX	EQUIPE	ADAPT
Aprenda Fazendo	X	X				
Aprendizado através de Descoberta		X	X			
Aprendizado Situado	X					
Instrução Ancorada	X					
Construcionismo	*C*					
Aprendizado através de Reflexão				X		
Aprendizado Cooperativo e Colaborativo					X	
Elaboração						*C*
Interação do Tratamento de Atitude						X

Capítulo 5

Análise dos Jogos com Relação aos Requisitos Identificados

A seguir é analisado como os jogos implementam os requisitos apresentados no capítulo anterior. Alguns requisitos são implementados parcialmente ou não são implementados por determinados jogos, o que é justificado no decorrer do capítulo, caso a caso.

Ao final do capítulo, uma tabela comparativa entre os jogos é apresentada.

5.1 Requisito 1: Um JSES deve dar suporte à transferência do aprendizado

Os jogos SESAM, SimjavaSP, SimSE e MO-SEProcess implementam completamente o requisito 1. O SimSE, o SESAM, o SimjavaSP e o *The Incredible Manager* são semelhantes no que diz respeito ao papel ativo que o aluno exerce durante o jogo. Em todos eles, o estudante assume o papel de um gerente de projeto que possui a tarefa de realizar um projeto no prazo e dentro do orçamento, além de ter que entregar, exceto no jogo *The Incredible Manager*, um produto de boa qualidade.

Por sua vez, cada jogador do MO-SEProcess pode exercer um papel específico em um projeto de *software*. Seis papéis de engenheiros de *software* estão disponíveis, cada um com tarefas e metas específicas. A tarefa dos jogadores também é entregar um projeto dentro do prazo e do orçamento, construindo um produto com poucos erros.

Os jogos qGame e TREG implementam parcialmente o requisito 1. No jogo qGame, o jogador não participa do jogo exercendo um papel específico. Ele simplesmente controla o fluxo dos itens de informação (*quantum* de requisitos, projeto e código) ao longo do projeto, definindo se atividades de verificação e documentação serão reali-

zadas e a ordem das atividades. O problema a ser resolvido aqui é entregar o produto com poucos erros e dentro do prazo ao cliente.

Em algumas fases do jogo TREG, o jogador não exerce nenhum papel específico, devendo apenas realizar corretamente uma determinada tarefa, como indicar quais são as entradas e saídas de cada atividade de uma oficina de requisitos. Em outras fases, de fato o jogador atua no papel de coordenador da oficina, devendo, por exemplo, escolher as pessoas corretas para participarem das reuniões. O trabalho hipotético a ser realizado no jogo é a execução correta de uma oficina para levantamento de requisitos.

O jogo TREG é o único no qual o jogador não tem que resolver um problema de construção. O trabalho hipótético a ser realizado no jogo é a execução correta de uma oficina para levantamento de requisitos. Não se trata de um problema de construção, uma vez que nenhum produto (p. ex. requisitos levantados e prontos para serem especificados) é entregue no final. É importante ressaltar que a ausência de um problema de construção não torna o jogo menos aderente ao requisito 1, mas o impede de aplicar a teoria de aprendizado do "Construcionismo".

5.2 Requisito 2: Um JSES deve prover interatividade

O SimSE implementa todos os tipos de ações citados na apresentação do requisito 2 (vide seção 4.2). O jogador faz uma escolha quando seleciona um empregado sobre o qual uma ação será realizada. Para cada passo da simulação, o jogador pode coletar informações sobre empregados, artefatos, ferramentas e outros objetos. Cada empregado é um objeto que pode ser manipulado selecionando-se a ação a ser realizada sobre ele, tais como atribuir uma tarefa, aumentar seu salário, entre outras. Os eventos aos quais o jogador deve reagir podem ser ações aleatórias (p. ex. falha catastrófica do sistema) ou autônomas (p. ex. o empregado descansa quando seu nível de energia cai para 0.2 ou menos) disparadas pelo jogo. As principais retroalimentações do jogo são na forma de atualizações dos valores dos objetos de acordo com regras pré-definidas e vinculadas às ações dos usuários.

O SimSE é prescritivo, porque, em qualquer momento, apenas um subconjunto de ações estão disponíveis para o jogador. Por exemplo, no início do jogo, o jogador não pode atribuir uma tarefa de revisão para um empregado, visto que nenhum artefato começou a ser feito.

A interatividade do SESAM ocorre através de ações e retroalimentações textuais. Com o comando `Ask`, é possível coletar informações sobre empregados (p. ex. `Ask J. Smith for his experiences`). O comando `Let` serve para escolher um empregado e atribuir uma tarefa a ele (p. ex. `Let J. Smith write the specification`). De forma diferente do SimSE, o jogador precisa reagir a eventos originados somente de ações autônomas como o envio da especificação de requisitos para o cliente após sua conclusão. Ações aleatórias não foram implementadas neste jogo.

A maior parte das retroalimentações fornecidas pelo SESAM são mensagens relacionadas às ações de usuário (p. ex. *Patrick Murphy reports: The specs are now 30 pages long, I think it will soon be finished*) e às ações autônomas (p. ex. *The customer has called, he has received a 63-pages specs on Sep 7. He will check it*). A prescritividade do SESAM reside nas restrições das ações dos usuários, no sentido de que o jogador somente pode realizar uma ação se suas restrições forem satisfeitas.

Uma vez que o *The Incredible Manager* tem o foco em gestão de projeto em vez de processo de *software*, a maior parte das ações disponíveis para o usuário são relacionadas às atividades de planejamento e replanejamento. O jogador realiza uma escolha quando contrata um empregado. Ao atribuir uma tarefa para um empregado e definir o esforço para sua execução, o jogador está manipulando um objeto. O plano de projeto pode ser aprovado ou não pelo gerente sênior. A recusa deste artefato é um evento ao qual o gerente precisa reagir. A coleta de informação é feita através de um painel que contém dados como a quantidade de dias e o orçamento restantes para o projeto.

Entre as retroalimentações do *The Incredible Manager* estão as expressões gráficas dos empregados (p. ex. pânico, cansaço e ócio) e do gerente sênior (p. ex. andando de um lado para o outro se o projeto está atrasado). Um aspecto prescritivo do jogo é que ele não permite a execução do projeto se o plano foi recusado pelo gerente sênior.

No SimjavaSP, atributos relevantes do processo e do produto são apresentados para os jogadores através de um painel. Ele também consiste de botões para disparar ações sobre o projeto e uma barra deslizante que permite ao jogador alocar esforço para atividades de validação e verificação. Outro painel apresenta atributos do empregado selecionado. Através deste painel, é possível atribuir tarefas para um empregado e realizar uma ação (p. ex. treiná-lo ou demiti-lo) sobre ele. Eventos, conhecidos como "Truques sujos" (do inglês, *Dirty Tricks* [Dawson, 2000]), foram implementados e os jogadores precisam reagir a eles (p. ex. *hardware* e *software* com defeitos, antecipação do prazo pelo cliente e saída de desenvolvedores). Estes eventos são ações aleatórias.

Retroalimentações relacionadas tanto a relações de causa e efeito longas quanto curtas são ilustradas na simulação. Por exemplo, um estudante pode notar que o

projeto vai ficar fora do prazo e decide corrigir isto através da contratação de vários desenvolvedores. Esta estratégia terá o efeito imediato de diminuir o orçamento com rapidez, bem como de causar um aumento na comunicação entre os desenvolvedores, que tentarão coordenar seus esforços de desenvolvimento. A longo prazo, isto pode realmente trazer um impacto negativo para o projeto, como atrasá-lo ainda mais.

No MO-SEProcess, a ação de escolha ocorre quando o jogador escolhe um artefato sobre o qual o jogador realizará uma tarefa (p. ex. criar requisito e revisar projeto). Todo artefato é um objeto a ser manipulado pelo jogador no papel de empregado. O empregado pode também ser manipulado, por exemplo, quando o jogador permite que ele descanse ou o envia de volta ao trabalho. O jogador pode coletar informação sobre o progresso do projeto examinando o progresso e a quantidade de erros de cada artefato, ou examinando o progresso de integração do projeto como um todo. O jogador pode também obter a situação dos outros jogadores através de um painel de situações. Os eventos aos quais o jogador deve reagir são ações autônomas (p. ex. a conclusão de um artefato) e ações de outros jogadores (p. ex. o início de uma tarefa por outro empregado). As ações do jogador podem causar retroalimentações como a perda de produtividade de uma tarefa devido a alocação de vários empregados para sua execução. O jogo é prescritivo, no sentido de que cada jogador só pode controlar as ações de seu respectivo empregado.

O qGame implementa parcialmente o requisito 2. O único tipo de ação disponível no jogo é "fazer uma escolha". O jogador faz uma escolha quando decide qual é a próxima atividade a ser executada com relação ao momento atual do projeto. Ele pode tomar as seguintes decisões: permanecer na mesma atividade, documentar ou não um artefato, revisar ou não um artefato, e saltar etapas do processo. Todos os eventos do qGame, que são o início e o término das atividades, são controlados pelo jogador. Portanto, o jogo não oferece ações autônomas ou aleatórias. Os objetos (empregados, cliente e *quanta*) também não podem ser diretamente manipulados e nenhuma informação pode ser obtida a partir deles.

As retroalimentações oferecidas pelo jogo são na forma da quantidade e das cores dos círculos que representam os *quanta*, indicando a quantidade de requisitos que está sendo correta ou incorretamente transformada no produto a ser entregue ao cliente. O qGame é prescritivo no sentido de que o jogador não pode executar uma determinada atividade após outra se a primeira estava prevista para ser executada antes no ciclo de vida do projeto considerado pelo jogo (p. ex. realizar a etapa de requisitos após a codificação e revisar requisitos antes de documentá-lo).

No TREG a implementação do requisito 2 também é parcial. Neste jogo, em cada tarefa a ser realizada, o jogador pode interagir com o jogo somente através de

ações de escolha e de coleta de informações. O jogador realiza uma escolha quando seleciona uma das alternativas propostas em cada tarefa, como, por exemplo, escolher as pessoas mais apropriadas para participarem de uma reunião ou escolher as entradas e saídas corretas de cada uma das atividades da etapa de análise de requisitos. O jogo disponibiliza, em cada tarefa, uma janela onde o aluno pode visualizar os prós e contras de cada alternativa possível para ajudar a responder a tarefa. Esta análise consiste em uma ação de coleta da informação.

Retroalimentações informativas são dadas no TREG através de Caracteres Não-jogadores, usados para orientar os jogadores, e de monitores que informam o placar e o tempo do jogo.

A prescritividade no TREG consiste em não deixar o aluno prosseguir no jogo caso não tenha finalizado corretamente uma tarefa que exija uma resposta certa. Algumas tarefas não possuem respostas corretas, mas o jogo seguirá por certo rumo dependendo da escolha feita pelo aluno.

5.3 Requisito 3: Um JSES deve refletir a complexidade das situações e dos problemas encontrados no mundo real

Todos os jogos implementam completamente o requisito 3. O SimSE possui um modelo preditivo onde várias regras de causa e efeito encontradas nos processos de *software* foram modeladas. Por exemplo, o jogo implementa as seguintes regras quando o modelo de simulação utilizado é o do ciclo de vida Cascata:

- "Se um projeto está atrasado e você adicionar mais pessoas, o projeto tornar-se-á ainda mais atrasado";
- "Utilizar poucas pessoas bem qualificadas é mais produtivo do que utilizar muitas pessoas pouco qualificadas";
- "Faça um projeto antes de codificar".

Para modelar e simular estas regras, um construtor de modelos e um simulador foram criados. Esses dois componentes possuem características típicas de Dinâmica de Sistemas (p. ex. regras contínuas de causa e efeito) e de Eventos Discretos (p. ex. regras discretas, disparadas quando o simulador inicia ou termina uma ação, além de entidades individuais como empregados, artefatos e ferramentas, além de). Ações

aleatórias (p. ex. falhas catastróficas do sistema que resultam na perda de uma parte significativa do projeto) foram modeladas para representar as incertezas inerentes aos processos de *software* do mundo real.

Alguns dos efeitos do mundo real demonstrados pelo jogo são exagerados. De acordo com Navarro [Navarro, 2006], regras semelhantes à realidade fazem com que as lições sejam apresentadas em um nível imperceptível. Por exemplo, uma regra multiplicou por 10 o efeito da presença de erros em um documento de requisitos sobre o número de erros que são introduzidos no documento de projeto. Sem esta amplificação, o efeito desta regra seria muito difícil de detectar durante o jogo. Simplificações também foram feitas. Vários detalhes e mesmo alguns aspectos centrais do processo foram deixados de fora. Segundo Navarro, a inclusão de muitos detalhes pode oprimir o jogador e distraí-lo das lições que o modelo tenta ensinar.

O SESAM usa um modelo preditivo híbrido cuja modelagem é realizada através de uma linguagem desenvolvida pelos criadores do jogo. Na simulação, as regras deste modelo são convertidas para uma estrutura de Gramática de Grafos (do inglês, *Graph Grammars*) [Drappa & Melchisedech, 1995]. Através de regras de reescrita de grafos, a simulação realiza as mudanças de estado apropriadas no projeto de *software* simulado. A notação de Gramática de Grafos originalmente proposta por Göttler [Göttler, 1979], que já contemplava aspectos de Eventos Discretos (p. ex. tratamento de eventos autônomos disparados em intervalos discretos de tempo, além de entidades individuais como empregados e artefatos), foi adaptada para o jogo visando possibilitar o processamento de eventos de usuários bem como a representação de um aspecto da Dinâmica de Sistemas: o conceito de taxa, que é uma função matemática que descreve mudanças contínuas em um atributo (p. ex. tamanho do documento) à medida que o tempo da simulação avança.

Assim como o SimSE, o SESAM também simplifica alguns detalhes e aspectos centrais dos processos de *software*. Todavia os efeitos das regras não são exagerados. A parametrização dos modelos é baseada em dados empíricos, que cobrem milhares de projetos de *software* e outras fontes da literatura. Além disso, o comportamento quantitativo dos modelos foi demonstrado comparando-se os resultados da simulação com os dados obtidos a partir do modelo de estimativa de custos COCOMO II [Boehm et al., 1995].

Conforme já foi dito na seção 2.3.1.4, o *The Incredible Manager* utiliza um modelo preditivo híbrido que é uma extensão da Dinâmica de Sistemas originalmente proposta por Forrester [Forrester, 1961]. O atual modelo do jogo simplifica algumas características do mundo real, visto que é incapaz de representar várias situações e eventos inesperados, como múltiplos desenvolvedores trabalhando juntos na mesma tarefa e

interações sociais. Este modelo é uma adaptação do modelo de Dinâmica de Sistemas criado por Abdel-Hamid [Abdel-Hamid, 1984], que é é um dos primeiros modelos de simulação de laboratório aplicados a projetos de *software*.

O SimjavaSP também usa um modelo preditivo, que é interpretado por um simulador baseado em Eventos Discretos, o Simjava [Howell & Mcnab, 1998]. O uso de Eventos Discretos, neste caso, torna mais fácil a representação de ocorrências aleatórias (p. ex. os "Truques Sujos" apresentados por Dawson [Dawson, 2000] e os eventos benéficos) e o comportamento individual dos objetos da simulação (p. ex. atividades e empregados). Todavia o simulador Simjava também permite que regras contínuas, que são características da Dinâmica de Sistemas, sejam implementadas.

O modelo de simulação do jogo é baseado no modelo de laboratório utilizado por Merrill e Collofello [Merrill & Collofello, 1997] e não apresenta exageros nas regras de causa e efeito. Todavia, simplifica alguns aspectos dos processos de *software*. Por exemplo, somente 6 "Truques Sujos" [Dawson, 2000] e 5 eventos benéficos foram implementados no jogo.

Uma vez que o modelo de simulação do MO-SEProcess é o mesmo modelo de ciclo de vida Cascata utilizado no SimSE, ele também é preditivo, simplifica detalhes e aspectos de processos de *software*, e apresenta exageros nas regras de causa e efeito. A diferença reside na modelagem e na simulação. Enquanto o SimSE possui suas próprias ferramentas de modelagem e simulação, o MO-SEProcess usou a linguagem-script `Linden`. Esta linguagem é fornecida pelo ambiente *Second Life* [Rymaszewski et al., 2007], que é o ambiente para criação de mundos virtuais online sobre o qual o jogo foi desenvolvido.

No qGame os requisitos do cliente são considerados legítimos e válidos no início do projeto simulado. À medida que o projeto de *software* se desenrola, estes requisitos vão sendo transformados em requisitos do sistema, depois em projeto e, por fim, em código. O modelo preditivo descreve regras que interferem na transformação destes itens de informação (*quantum*) ao longo do projeto, fazendo com que os requisitos, os itens do projeto e os itens de código sejam mal-interpretados, esquecidos ou elaborados com defeitos. O grau de falhas em cada atividade depende das ações realizadas pelo jogador. Uma ação de revisão, detecta e corrige erros. Uma ação de documentação, melhora a comunicação e diminui as mal-interpretações. A não execução de uma etapa (p. ex. implementa sem realizar atividades de projeto) economiza momentaneamente o tempo, mas aumenta a quantidade de erros nos artefatos e, por consequência, faz com que o tempo do projeto aumente a longo prazo.

O qGame simplifica algumas características do mundo real, como a quantidade de desenvolvedores desempenhando cada papel no projeto (apenas um) e o fato de as

atividades não poderem ser realizadas em paralelo. O qGame exagera nos efeitos das regras de forma que o aluno possa verificar, por exemplo, que em um projeto onde as revisões são frequentes, os erros são sempre menores do que em um projeto onde não são realizadas revisões.

O TREG utiliza um modelo preditivo baseado no gênero de "Estórias Ramificadas" (do inglês, *Branching Stories*), na técnica de Cenários para especificação do *software* e em diagramas de máquinas de estado para modelar o comportamento dos elementos do sistema [Vega et al., 2009a]. Os cenários são usados para descrever várias situações simuladas no jogo. Todos os cenários são conectados formando uma rede de relacionamentos usando um grafo de estórias ramificadas. Estórias ramificadas é um gênero de simulação tradicional que oferece ao estudante a opção de responder questões de múltiplas escolhas. O diagrama de máquinas de estado apresenta o comportamento dinâmico de uma entidade baseado em suas respostas aos eventos e são usados para exibirem diferentes tipos de comportamento dos objetos do jogo.

O TREG exagera na metáfora utilizada ao combinar o uso de ingredientes de uma receita com as melhores práticas da técnica de oficinas de requisitos. Os ingredientes utilizados na metáfora implementada pelo jogo foram retirados do trabalho de Gottesdiener [Gottesdiener, 2002].

Todos os jogos analisados trabalham com tempo reduzido de forma que os projetos ou as oficinas de requisitos simulados durem no máximo poucas horas, em vez de durarem dias, meses ou anos.

5.4 Requisito 4: Um JSES deve facilitar a reflexão sobre o conteúdo aprendido

Nenhum dos jogos de simulação avaliados oferece sessões de esclarecimentos, ou seja, não realizam uma comparação do resultado do aluno com um desempenho ideal definido durante o projeto do jogo. Portanto, nenhum deles implementa completamente o requisito 4.

Os jogos SimSE e TREG implementam parcialmente o requisito 4, pois disponibilizam uma ferramenta explicativa. O SimSE possui uma ferramenta explicativa que fornece aos jogadores uma representação gráfica de como o processo de desenvolvimento simulado comportou-se ao longo do tempo e explicações das regras subjacentes ao jogo de simulação. Esta ferramenta pode ser executada no fim de um jogo de forma que os estudantes visualizem o registro de eventos, regras e valores de atributos que foram gravados durante o jogo.

Um estudo observacional mostrou que os estudantes que utilizaram uma versão-protótipo do SimSE sem a ferramenta explicativa foram significativamente mais confusos e sem-confiança sobre a lógica por trás dos resultados do que aqueles que utilizaram uma versão do jogo com esta ferramenta [Navarro, 2006].

O TREG utiliza uma técnica chamada *Machinima* para oferecer retroalimentações ao usuário. Esta técnica consiste em usar os mecanismos do jogo 3D para gerar uma gravação de um vídeo com elementos do mundo virtual representando o desempenho do usuário no jogo. A *Machinima* é utilizada para filmar algumas situações problemáticas em uma oficina de requisitos. Nela, o aluno poderá ver um vídeo com as consequências de suas escolhas, aprendendo desta forma uma nova técnica de oficina baseada em padrões de colaboração.

Os outros jogos não incluem uma ferramenta explicativa. Em vez disso, eles somente apresentam os valores dos atributos do sistema entregue, os sentimentos do cliente sobre estes valores ou uma indicação de sucesso ou falha do projeto simulado. Nenhuma informação sobre o registro de eventos, regras e valores de atributos ao longo do tempo está disponível nestes jogos.

5.5 Requisito 5: Um JSES deve dar suporte ao desenvolvimento de habilidades no trabalho em equipe

O único jogo multi-usuário dentre os avaliados neste trabalho é o MO-SEProcess. Neste jogo, as pessoas trabalham juntas para realizarem um projeto de *software* que segue o ciclo de vida Cascata. Cada pessoa exerce um papel no processo de desenvolvimento. A meta do jogo é entregar o *software* ao cliente o mais completo possível e com a menor quantidade de erros. Isso requer boa colaboração entre os jogadores. Este jogo implementa completamente o requisito 5.

Como o MO-SEProcess utiliza o ambiente do *Second Life*, é possível interagir com os outros jogadores através das ferramentas de comunicação disponibilizadas por este ambiente como bate-papos e mensagens instantâneas.

Nos jogos qGame, SESAM, SimjavaSP, SimSE, *The Incredible Manager* e TREG são disponibilizados Caracteres Não Jogadores que interagem com o jogador através de mudanças de expressão, emissão de frases ou painéis com informações sobre os mesmos. Estes jogos implementam parcialmente o requisito 5.

Conforme já mencionado na seção 3.7, de acordo com os autores do TREG,

existe uma versão multi-usuário do jogo. Entretanto, esta versão foi desconsiderada neste trabalho por não estar disponível para ser jogada.

5.6 Requisito 6: Um JSES deve ser adaptável a diferentes estilos de aprendizado

Nenhum jogo é adaptável dinamicamente. Além disso, o SESAM, o *The Incredible Manager*, o SimjavaSP, o MO-SEProcess e o TREG não implementam qualquer tipo de comportamento adaptativo.

Os jogos SimSE e qGame permitem que os alunos escolham modelos de simulação ou níveis de dificuldade diferentes antes de iniciar uma partida. Uma vez que se trata de uma troca manual, estes jogos implementam parcialmente o requisito 6. No SimSE, o jogador pode escolher um de seis modelos de simulação, cada um representando um tipo de processo de desenvolvimento (Cascata, Inspeção, Incremental, Programação Extrema, Prototipação Rápida e RUP).

O qGame disponibiliza 3 níveis de dificuldade. No primeiro, o jogador não pode interferir nas ordens das atividades e apenas atividades de elaboração de artefatos estão disponíveis, como especificar, projetar e implementar. No segundo nível, é possível interferir na ordem das atividades. Por exemplo, pode-se optar por não realizar a especificação dos requisitos. No terceiro e último nível, que é o representado na figura 3.1 o jogador pode submeter os artefatos a atividades de revisão a fim de diminuir a quantidade de defeitos. Além disso, o jogador pode não documentar as atividades de análise de requisitos e de projeto.

5.7 Aplicação das Teorias de Aprendizado pelos Jogos Avaliados

A tabela 5.1 apresenta quais jogos implementam cada um dos requisitos identificados neste trabalho. A marcação com `X` indica que o jogo implementa um determinado requisito, enquanto que a marcação com *P* indica uma implementação parcial.

O jogo SimSE é o que implementa um maior número de requisitos e, consequentemente, aplica uma maior quantidade de teorias de aprendizado. Isto não quer dizer que esse jogo seja mais eficiente do que os outros. De acordo com Navarro [Navarro, 2009], embora uma abordagem educacional que aplique mais teorias de aprendizado do que outra não ser necessariamente a melhor, definitivamente vale mais a pena explorá-la.

Confrontando-se esta tabela com a tabela 4.1, é possível verificar que algumas teorias de aprendizado estão sendo bastante aplicadas pelos jogos. A teoria "Aprenda Fazendo", por exemplo, cuja aplicação é promovida pela implementação do requisitos 1 ("Um JSES deve dar suporte à transferência de aprendizado") e 2 ("Um JSES deve prover interatividade"), é contemplada por todos os jogos, pois os dois requisitos foram implementados pelo menos parcialmente por todos eles. O mesmo ocorre com as teorias da "Instrução Ancorada", do "Aprendizado Situado" e do "Aprendizado através da Descoberta".

A teoria do "Construcionismo", promovida condicionalmente pela implementação do requisito 1, desde que o problema a ser resolvido seja de construção, também foi aplicada pela grande maioria dos jogos. Apenas o jogo TREG não disponibiliza um problema de construção para ser resolvido pelo jogador.

A teoria do "Aprendizado Cooperativo e Colaborativo", de aplicação facilitada pela implementação do requisito 5 ("Um JSES deve melhorar as habilidades no trabalho em equipe") foi contemplada em todos os jogos. Entretanto, a maioria deles implementou parcialmente o requisito.

As outras teorias de aprendizado foram pouco aplicadas pelos jogos. A teoria "Aprendizado através de reflexão", por exemplo, que possui aplicação facilitada através da implementação do requisito 4 ("Um JSES deve facilitar a reflexão sobre o conteúdo aprendido"), foi aplicada, e mesmo assim parcialmente, apenas em dois jogos: SimSE e TREG. O mesmo ocorreu com as teorias da "Elaboração" e da "Interação do Tratamento de Atitude", devido a não implementação do requisito 6 ("UM JSES deve ser adaptável a diferentes estilos de aprendizado") pela maioria dos jogos.

Estes resultados sugerem que os desenvolvedores dos jogos de simulação não têm se preocupado, em tempo de projeto, com as teorias de aprendizado que serão utilizadas pelos jogadores (alunos) enquanto estiverem jogando para aprenderem sobre determinado tópico de Engenharia de Software. Os requisitos implementados pela grande maioria dos jogos são justamente aqueles inerentes a própria natureza do jogo de simulação.

O que faz um jogo de simulação ser diferente em relação a jogos de outra natureza (p. ex. tutoriais e *quizzes*) é o fato de o jogador ter um papel ativo no contexto do jogo (requisito 1), de poder controlar, através de ações, os rumos da simulação (requisito 2) e de representar as complexidades do mundo real através de um modelo preditivo (requisito 3). Logo, o fato de se desenvolver um jogo de simulação geralmente faz com que ele, por sua própria natureza, atenda aos três primeiros requisitos identificados neste estudo.

Os elementos de projeto e implementação de um jogo de simulação que fariam

com que os mesmos fossem aderentes aos outros requisitos identificados neste trabalho, como por exemplo, disponibilizar sessões de esclarecimento (requisito 4), tornar o jogo multi-usuário (requisito 5) e criar mecanismos para adaptação dinâmica do jogo ao estilo de aprendizado dos alunos (requisito 6) não são inerentes à natureza de um jogo de simulação. Talvez por isso, a eles não têm sido dada a devida atenção por parte dos desenvolvedores dos jogos.

Tabela 5.1. Implementação dos requisitos identificados no estudo pelos jogos avaliados.

Requisitos vs. Jogos	MO-SEProcess	qGame	SESAM	SimjavaSP	SimSE	*The Incredible Manager*	TREG
1: Um JSES deve dar suporte à transferência do aprendizado	X	*P*	X	X	X	X	*P*
2: Um JSES deve prover interatividade	X	*P*	X	X	X	X	*P*
3: Um JSES deve refletir as situações do mundo real	X	X	X	X	X	X	X
4: Um JSES deve facilitar a reflexão sobre o conteúdo aprendido					*P*		*P*
5: Um JSES deve melhorar as habilidades no trabalho em equipe	X	*P*	*P*	*P*	*P*	*P*	*P*
6: Um JSES deve ser adaptável a diferentes estilos de aprendizado		*P*			*P*		

Capítulo 6

Conclusão

Este trabalho apresentou um estudo dos requisitos que podem auxiliar a escolha e o desenvolvimento de jogos de simulação para o ensino de Engenharia de *Software*. O levantamento dos requisitos levou em consideração as boas práticas para o desenvolvimento de jogos, as características de jogos de simulação existentes e a aplicação de teorias de aprendizado, especialmente as relacionadas a Psicologia Construtivista. Os jogos de simulação existentes para o ensino de Engenharia de *Software* foram identificados e verificou-se como é a implementação dos requisitos em cada um deles.

Conforme mencionado na seção 1.2, fazia parte do objetivo do projeto de mestrado, a investigação do domínio de jogos de simulação envolvendo também o estudo e a construção de um arcabouço de desenvolvimento e de um jogo de simulação. Em função de diversos fatores, indo desde o problema da dedicação parcial ao mestrado até os aspectos da complexidade de uma implementação mesmo que parcial do jogo, decidiu-se deixar o trabalho de desenvolvimento do arcabouço e do jogo fora do escopo dos trabalhos descritos nesta dissertação.

A maioria dos jogos identificados neste trabalho destina-se ao ensino do tópico de Gerenciamento de Projetos. Percebe-se que há uma carência de jogos de simulação voltados para o ensino de outros tópicos de Engenharia de *Software*, como Engenharia de Usabilidade, Testes, entre outros.

Apesar do jogo de simulação ser considerado uma abordagem que tem grande potencial para aplicar teorias de aprendizado baseadas no Construtivismo, observou-se através dos resultados deste trabalho que algumas teorias foram pouco contempladas pelos jogos avaliados. Uma teoria pouco contemplada é o "Aprendizado através da reflexão" que, dentre os requisitos identificados, é facilitada pela implementação do requisito 4 do catálogo ("Um JSES deve facilitar a reflexão sobre o conteúdo aprendido"). Este requisito foi implementado apenas pelos jogos SimSE e TREG. Mesmo assim são

implementações parciais, pois nenhum dos dois oferece sessões de esclarecimentos ao final do jogo, através das quais os alunos poderiam refletir sobre o que eles deveriam fazer para obterem um melhor desempenho. Verificou-se que os requisitos mais atendidos são justamente aqueles inerentes a própria natureza dos jogos de simulação e que os diferenciam dos demais jogos, como por exemplo, os tutoriais.

A utilização dos requisitos apresentados neste trabalho pelos projetistas de novos jogos de simulação, pode fazer com que no futuro as teorias de aprendizado sejam sistematicamente contempladas.

6.1 Contribuições do Trabalho

Através de uma revisão sistemática da literatura, os jogos de simulação existentes para o ensino de Engenharia de *Software* foram identificados. As seguintes questões de pesquisa foram respondidas para cada um dos jogos:

- Quais características dos jogos de simulação promovem a aplicação de teorias de aprendizado associadas aos princípios da Psicologia do Construtivismo?

- De que forma cada jogo avaliado implementa as características identificadas?

Ao contrário do trabalho de Peixoto e outros [Peixoto et al., 2011], este trabalho não focou prioritariamente o projeto das interfaces gráficas dos jogos, mas também o projeto de outras partes de um jogo de simulação, como o simulador e o modelo de simulação. Logo, algumas características identificadas podem ser inerentes também a um desses dois componentes. A seção 2.3 descreveu detalhadamente cada uma das partes que compõem um jogo de simulação.

O diferencial deste trabalho para o de Von Wangenheim e Shull [Von Wangenheim & Shull, 2009] é que as características descobertas não são baseadas somente nos resultados dos experimentos de validação dos jogos com os alunos, mas também na análise de cada uma das partes dos jogos (máquina de jogo, simulador e modelo de simulação) e na análise dos jogos como um todo, simplesmente jogando-os no lugar de um aluno. Estas novas formas de análise foram necessárias, visto que os alunos não entendem suficientemente sobre teorias de aprendizado para poderem identificar as características desejadas neste trabalho. Entretanto, os resultados dos experimentos não foram desconsiderados, visto que algumas características destacadas pelos alunos tratam, de fato, da aplicação de uma ou mais teorias de aprendizado.

6.2 Limitações do Trabalho

A metodologia utilizada neste trabalho não nos permite dizer que foram identificados todos os requisitos cujas implementações facilitam a aplicação das teorias de aprendizado listadas aqui. Uma abordagem metodológica que talvez rendesse melhores resultados consiste em realizar uma revisão sistemática da literatura não apenas para identificar os jogos e verificar como eles implementam os requisitos levantados previamente, mas também para a identificação inicial dos requisitos, que neste trabalho foi realizada através de uma revisão convencional da literatura. A revisão sistemática permitiria que uma quantidade maior de trabalhos sobre projetos de ambientes educacionais multimídia fosse identificada. Uma possível questão de pesquisa para esta revisão sistemática inicial seria: "Quais são as boas práticas preconizadas pelos trabalhos na área de projetos de ambientes educacionais multimídia que podem facilitar a aplicação das teorias de aprendizado baseadas no Construtivismo pelos jogos de simulação?".

Outra limitação deste trabalho diz respeito à validação dos requisitos com algumas partes interessadas, como alunos, professores e desenvolvedores de jogos de simulação. Os requisitos foram identificados e validados de forma restrita. A atividade de experimentação dos jogos, que consistiu em jogar cada um dos jogos de simulação para o ensino de Engenharia de Software a procura de requisitos ou formas de implementação de requisitos, é um exemplo de atividade que poderia ter a participação de mais pessoas. Jogos multi-usuário, como o MO-SEProcess, no qual pode-se atuar em 6 papéis distintos, poderiam ser jogados por mais de uma pessoa ao mesmo tempo.

Conforme mencionado na seção 3.1, apenas algumas bases de periódicos e conferências foram consideradas no processo de busca que identificou os jogos existentes para o ensino de Engenharia de *Software*. Foram escolhidas as bases mais renomadas, portanto alguns jogos de simulação não foram identificados a partir desta busca. No Brasil, por exemplo, existem outros jogos de simulação, como o X-MED [Prikladnicki & Von Wangenheim, 2008], que visa ao ensino do tópico medição de software.

É importante ressaltar que os requisitos identificados neste trabalho possuem um foco maior na melhoria do aprendizado dos alunos. Requisitos cujos principais interessados são os professores não foram identificados, uma vez que as teorias de aprendizado consideradas seguem os princípios da Psicologia do Construtivismo, cujos adeptos defendem um foco maior no aluno e no aprendizado.

Apesar de os requisitos identificados parecerem ser aplicáveis a jogos de simulação para o ensino de qualquer domínio, não podemos afirmar isso com certeza, uma vez que a metodologia utilizada para a identificação dos requisitos considerou exclusivamente trabalhos e jogos relacionados ao domínio de Engenharia de *Software*.

6.3 Trabalhos Futuros

O estudo realizado pode ser ampliado de forma a considerar outras teorias de aprendizado, baseadas nas Psicologias Comportamental e Cognitiva, pois os jogos de simulação também aplicam princípios destes paradigmas de ensino. O estudo pode ainda considerar requisitos relacionados a diversão, uma vez que o jogo deve atingir um equilíbrio entre diversão e valor educacional para ser efetivo [Prensky, 2003].

O estudo pode ainda ser estendido para contemplar outras abordagens como jogos que não são de simulação, jogos não-computacionais e outras abordagens práticas (p. ex. projetos de curso). Uma extensão mais ousada seria considerar também abordagens mais próximas dos princípios das Psicologias Comportamental e Cognitiva (p. ex. aulas e palestras).

Pode-se ainda avaliar experimentalmente junto aos alunos de Engenharia de *Software* os jogos identificados neste trabalho. Estes experimentos serviriam para verificar a relação entre a aplicação das teorias de aprendizado e a eficiência dos jogos para o ensino. Com esse trabalho, pode-se descobrir qual aspecto de uma teoria é prioritário em relação a outro.

Atualmente, estamos desenvolvendo um jogo de simulação que visa a atender os requisitos identificados neste trabalho. Sua implementação está sendo realizada de forma componentizada, com clara separação entre os componentes modelo de simulação, simulador e máquina de jogo. Desta forma, será possível o desenvolvimento de outros jogos de simulação através da reutilização de um ou mais componentes deste jogo. Acreditamos que será possível a transformação deste jogo em um arcabouço de *software* que irá permitir a implementação eficiente e flexível de novos jogos.

Referências Bibliográficas

Abdel-Hamid, T. K. (1984). *The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective*. PhD thesis, Massachussets Institute of Technology, Cambridge (MA), EUA.

Akilli, G. (2007). Games and simulations: A new approach in education? In Gibson, D.; Aldrich, C. & Prensky, M., editores, *Games and Simulations in Online Learning: Research and Development Frameworks*, pp. 1--20. Information Science Pub., Pennsylvania State University, USA.

Alessi, S. M. & Trollip, S. R. (2001). *Multimedia for Learning. Methods and Development*. Allay and Bacon, Needham Heights, Massachussetts, USA.

Angehrn, A. A. (2004). Advanced social simulations: Innovating the way we learn how to manage change in organizations. *International Journal of Information Technology Education*, 2004.

Armarego, J. (2002). Advanced software design: a case in problem-based learning. In *CSEET '02: Proceedings of the 15th Conference on Software Engineering Education and Training*, p. 44, Washington, DC, USA. IEEE Computer Society.

Baker, A.; Navarro, E. & van der Hoek, A. (2003). Problems and programmers: an educational software engineering card game. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pp. 614 – 619.

Banks, J.; Carson, J.; Nelson, B. L. & Nicol, D. (2004). *Discrete-Event System Simulation (4th Edition)*. Prentice Hall, 4 edição.

Barros, M. d. O.; Dantas, A. R.; Veronese, G. O. & Werner, C. M. L. (2006). Model-driven game development: experience and model enhancements in software project management education. *Software Process: Improvement and Practice*, 11(4):411--421.

Barros, M. d. O.; Werner, C. M. L. & Travassos, G. H. (2002). A system dynamics metamodel for software process modeling. *Software Process: Improvement and Practice*, 7(3-4):161--172.

Boehm, B.; Clark, B.; Horowitz, E.; Westland, C.; Madachy, R. & Selby, R. (1995). Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering*, 1(1):57–94.

Boyer, K. E.; Lahti, W.; Phillips, R.; Wallis, M. D.; Vouk, M. A. & Lester, J. C. (2010). Principles of asking effective questions during student problem solving. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on Computer science education*, pp. 460--464, New York, NY, USA. ACM.

Brooks, Jr., F. P. (1978). *The Mythical Man-Month: Essays on Softw.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edição.

Callahan, D. & Pedigo, B. (2002). Educating experienced it professionals by addressing industry's needs. *IEEE Software*, 19:57–62.

Claypool, K. & Claypool, M. (2005). Teaching software engineering through game design. *SIGCSE Bull.*, 37(3):123--127.

Collofello, J. (2000). University/industry collaboration in developing a simulation based software project management training course. In *Software Engineering Education Training, 2000. Proceedings. 13th Conference on*, pp. 161 – 168.

Connolly, T. M.; Stansfield, M. & Hainey, T. (2007). An application of games-based learning within software engineering. *British Journal of Educational Technology*, 38(3):416--428.

Connolly, T. M.; Stansfield, M. & Hainey, T. (2008). Using games-based learning to teach software engineering. *Web Information Systems and Technologies*, 8(5):304–313.

Coutinho, M. T. d. C. & Moreira, M. (2001). *Psicologia da Educação: Um Estudo dos Processos Psicológicos de Desenvolvimento e Aprendizagem Humanos, Voltado para a Educação.* Editora Lê.

Crnkovic, I.; Land, R. & Sjögren, A. (2003). Is software engineering training enough for software engineers? In *Proceedings of the Sixteenth Conference on Software Engineering Education and Training*, Madri, Espanha. IEEE.

Cronbach, L. & Snow, R. (1977). *Aptitudes and Instructional Methods: A Handbook for Research on Interactions*. Irvington, New York, NY, USA.

Dantas, A. R.; Barros, M. O. & Werner, C. M. L. (2004). Treinamento experimental com jogos de simulação para gerentes de projeto de software. *18º Simpósio Brasileiro de Engenharia de Software - SBES 2004. Brasília: SBC/UNB.*

Dawson, R. (2000). Twenty dirty tricks to train software engineers. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pp. 209--218, New York, NY, USA. ACM.

De Jong, T. & Van Joolingen, W. R. (1998). Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, 68(2):179–201.

DeGrace, P. & Stahl, L. H. (1990). *Wicked problems, righteous solutions*. Yourdon Press, Upper Saddle River, NJ, USA.

Denning, P. J. (1992). Educating a new engineer. *Commun. ACM*, 35(12):82--97.

Dixon, J. R. (1991). New goals for engineering education. *Mechanical Engineering*, 113:56–62.

Donzelli, P. & Iazeolla, G. (2001). Hybrid simulation modelling of the software process. *Journal of Systems and Software*, 59(3):227 – 235.

Drappa, A. & Ludewig, J. (1999). Quantitative modeling for the interactive simulation of software projects. *Journal of Systems and Software*, 46(2-3):113 – 122.

Drappa, A. & Ludewig, J. (2000). Simulation in software engineering training. In *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, pp. 199--208, New York, NY, USA. ACM.

Drappa, A. & Melchisedech, R. (1995). The use of graph grammar in a software engineering education tool. *Electronic Notes in Theoretical Computer Science*, 2:73--80.

Flavell, J. H. & Wellman, H. M. (1977). Metamemory. In Kail, R. V. & Hagen, J. W., editores, *Perspetives on the development of memory and cognition*, pp. 3--33. Erlbaum, Hillsdale, NJ, USA.

Forman, E. & McPhail, J. (1993). Vygotskian perspectives on children's collaborative problem-solving activities. In Forman, E. A.; Minick, N. & Stone, C. A., editores, *Contexts for learning. Sociocultural dynamics in children's development.* Oxford University Press, Oxford, UK.

Forrester, J. W. (1961). *Industrial Dynamics.* The M.I.T Press, 1st edição.

Gehrke, M.; Giese, H.; Nickel, U. A.; Niere, J.; Tichy, M.; Wadsack, J. P. & Zündorf, A. (2002). Reporting about industrial strength software engineering courses for undergraduates. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pp. 395--405, New York, NY, USA. ACM.

Gottesdiener, E. (2002). *Requirements by collaboration: workshops for defining needs.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Göttler, M. (1979). Semantical description by two-level graph grammars for quasihierarchical graphs. In Nagl, M. & Schneider, H. J., editores, *Proceedings WG78 Graphs, Data Structures, Algorithms, Series Applied Computer Science Nr. 13.* Hanser Verlag, Munich, Germany.

Gredler, M. E. (2004). Games and simulations and their relationships to learning. In Jonassen, D. H., editor, *Handbook of Research on Educational Communications and Technology*, chapter 21, pp. 571–581. Lawrence Erlbaum, Hillsdale, NJ, USA.

Groth, D. & Robertson, E. (2001). It's all about process: project-oriented teaching of software engineering. In *Proceedings of the 14th Conference on Software Engineering Education and Training*, pp. 7–17.

Harel, I. & Papert, S. (1991). *Multimedia for Learning. Methods and Development.* Ablex, Norwood, NJ, USA.

Hazzan, O. & Tomayko, J. E. (2004). Reflection processes in the teaching and learning of human aspects of software engineering. *Conference on Software Engineering Education and Training*, 0:32–38.

Heinich, R.; Molenda, M.; Russell, J. & Smaldino, S. (2002). *Instructional media and technologies for learning.* Merrill Prentice Hall, Upper Saddle River, NJ, 7a edição.

Hilburn, T. & Bagert, D. (1999). A software engineering curriculum model. In *Proceedings of the 1999 ASEE Annual Conference.* ASEE.

Hoffmann, M.; Kuhn, N.; Weber, M. & Bittner, M. (2004). Requirements for requirements management tools. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pp. 301--308, Washington, DC, USA. IEEE Computer Society.

Howell, F. & Mcnab, R. (1998). simjava: A Discrete Event Simulation Library For Java. In *In International Conference on Web-Based Modeling and Simulation*, pp. 51--56.

Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pp. 429--433, New York, NY, USA. ACM.

ISO/IEC (1991). ISO/IEC 9126: Information technology - software product evaluation - quality characteristics and guidelines for their use. In *International Organization for Standardization*.

Jain, A. & Boehm, B. (2006). Simvbse: Developing a game for value-based software engineering. In *Software Engineering Education and Training, 2006. Proceedings. 19th Conference on*, pp. 103 –114.

Jeffries, R.; Turner, A. A.; Polson, P. & Atwood, M. E. (1981). The processes involved in designing software. In Anderson, J. R., editor, *Cognitive Skills and their Acquisition*, pp. 225–283. E. Erlbaum, Hillsdale (NJ).

Jones, B. F. (1988). Text learning strategy instruction: Guidelines from theory and practice. In Weinstein, C. E.; Goetz, E. T. & Alexander, P. A., editores, *Learning and study strategies: Issues in assessment, instruction, and evaluation*, pp. 23--260. Academic Pres, New York, NY, USA.

Kellner, M.; Feiler, P.; Finkelstein, A.; Katayama, T.; Osterweil, L.; Penedo, M. & Rombach, H. (1991). Ispw-6 software process example. In *Proceedings of the 6th International Software Process Workshop: Support for the Software Process*. IEEE Computer Society Press.

Kellner, M. I.; Madachy, R. J. & Raffo, D. M. (1999). Software process simulation modeling: Why? what? how? *Journal of Systems and Software*, 46(2-3):91 – 105.

Kitchenham, B. & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Knauss, E.; Schneider, K. & Stapel, K. (2008). A game for taking requirements engineering more seriously. In *Multimedia and Enjoyable Requirements Engineering - Beyond Mere Descriptions and with More Fun and Games, 2008. MERE '08. Third International Workshop on*, pp. 22 –26.

Kornecki, A.; Khajenoori, S.; Gluch, D. & Kameli, N. (2003). On a partnership between software industry and academia. In *Proceedings of the 16th Conference on Software Engineering Education and Training, 2003*, pp. 60–69.

Lieberman, D. A. & Linn, M. C. (1991). Learning to learn revisited: Computers and the development of self-directed learning skills. *Journal of Research on Computing in Education*, 23(3):373–395.

Lingard, R. & Berry, E. (2002). Teaching teamwork skills in software engineering based on an understanding of factors affecting group performance. *Frontiers in Education, Annual*, 2:S3G1–6.

Luca, J. & Heal, D. (2006). *Is role-play an effective teaching approach to assist tertiary students to improve teamwork skills?*, pp. 473–477.

Ludi, S. & Collofello, J. (2001). An analysis of the gap between the knowledge and skills learned in academic software engineering course projects and those required in real: projects. *31st ASEE/IEEE Frontiers in Education Conference*, 1:8–11.

Madachy, R. J. (1994). *A Software Project Dynamics Model For Process Cost, Schedule And Risk Assessment.* PhD thesis, University of Southern California.

Madachy, R. J. (2008). *Software process dynamics.* Wiley, Hoboken, NJ.

Martin, R. & Raffo, D. (2000). A model of the software development process using both continuous and discrete models. *International Journal of Software Process Improvement and Practice*, 5:2–3.

Mayer, R. E. (1998). Cognitive, metacognitive, and motivational aspects of problem solving. *Instructional Science*, 26(1–2):49--63.

McMillan, W. W. & Rajaprabhakaran, S. (1999). What leading practitioners say should be emphasized in students' software engineering projects. In *CSEET '99: Proceedings of the 12th Conference on Software Engineering Education and Training*, p. 177, Washington, DC, USA. IEEE Computer Society.

Merrill, D. & Collofello, J. S. (1997). Improving software project management skills using a software project simulator. In *Software Project Simulator, Frontiers in Education Conference*, pp. 1361--1366.

Moreno-Ger, P.; Burgos, D.; Martínez-Ortiz, I.; Sierra, J. L. & Fernández-Manjón, B. (2008). Educational game design for online education. *Computers in Human Behavior*, 24(6):2530 – 2540.

Navarro, E. (2006). *SimSE: a software engineering simulation environment for software process education*. PhD thesis, University of California, Irvine, California, USA.

Navarro, E. O. (2009). On the role of learning theories in furthering software engineering education. In Ellis, H. J. C.; Demurjian, S. A. & Naveda, J. F., editores, *Software Engineering: Effective Teaching and Learning Approaches and Practices*, chapter 1, pp. 38–59. IGI Global.

Navarro, E. O. & Hoek, A. V. (2001). Adapting game technology to support individual and organizational learning. In *Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering*, Buenos Aires, Argentina.

Navarro, E. O. & Hoek, A. v. d. (2002). Towards game-based simulation as a method of teaching software engineering. In *Frontiers in Education, 2002. FIE 2002. 32nd Annual*, volume 3, pp. S2G--13 vol.3.

Navarro, E. O. & Hoek, A. v. d. (2004). Simse: an educational simulation game for teaching the software engineering process. *SIGCSE Bull.*, 36:233--233.

Navarro, E. O. & Hoek, A. v. d. (2005a). Design and evaluation of an educational software process simulation environment and associated model. In *Software Engineering Education & Training, 18th Conference on*, pp. 25 –32.

Navarro, E. O. & Hoek, A. v. d. (2005b). Software process modeling for an educational software engineering simulation game. *Software Process: Improvement and Practice*, 10(3):311--325.

Navarro, E. O. & Hoek, A. v. d. (2007). Comprehensive evaluation of an educational software engineering simulation environment. In *Software Engineering Education Training, 2007. CSEET '07. 20th Conference on*, pp. 195 –202.

Padberg, F. (2002). A discrete simulation model for assessing software project scheduling policies. *Software Process: Improvement and Practice*, 7(3-4):127--139.

Peixoto, D. C. C.; Possa, R. M.; Resende, R. F. & Pádua, C. I. P. S. (2011). An Overview of the Main Design Characteristics of Simulation Games in Software Engineering Education. In *Software Engineering Education and Training, 2011. CSEET '11. IEEE 24st Conference on (to appear)*, Honolulu, USA.

Peters, V. A. M. & Vissers, G. A. N. (2004). A Simple Classification Model for Debriefing Simulation Games. *Simulation & Gaming*, 35(1):70--84.

Piaget, J. (1968). *Six Psychological Studies*. Vintage Books, New York, NY, USA.

Pieterse, V.; Kourie, D. G. & Sonnekus, I. P. (2006). Software engineering team diversity and performance. In *SAICSIT '06: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pp. 180--186, Republic of South Africa. South African Institute for Computer Scientists and Information Technologists.

Prensky, M. (2003). Digital game-based learning. *Comput. Entertain.*, 1(1):21--21.

Prikladnicki, R. & Von Wangenheim, C. G. (2008). O uso de jogos educacionais para o ensino de gerência de projetos de software. In *FEES - Fórum de Educação em Engenharia de Software*, volume 1, pp. 37--45, Campinas, Brasil.

Pugh, A. L. (1976). *Dynamo User's Manual*. The M.I.T Press, 5th edição.

Raffo, D. M. (1996). *Modeling software processes quantitatively and assessing the impact of potential process changes on process performance*. PhD thesis, Pittsburgh, PA, USA. AAI9622438.

Reigeluth, C. M. & Rodgers, C. A. (1980). The elaboration theory of instruction: Prescriptions for task analysis and design. *NSPI Journal*, 19(1):16–26.

Richmond, B. (1990). *ITHINK User's Guide and Technical Documentation*. High Performance Systems Inc., Hanover, NH.

Rymaszewski, M.; Wallace, M.; Winters, C.; Ondrejka, C.; Batstone-Cunningham, B. & Rosedale, P. (2007). *Second Life: the office guide*. Wiley Press.

Sampaio, A.; Albuquerque, C.; Vasconcelos, J.; Cruz, L.; Figueiredo, L. & Cavalcante, S. (2005). Software test program: a software residency experience. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pp. 611--612, New York, NY, USA. ACM.

Savery, J. R. & Duffy, T. M. (1995). Problem based learning: An instructional model and its constructivist framework. *Educational Technology*, 35(5):31--38.

Schön, D. (1983). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, NY, USA.

Schön, D. A. (1987). *Educating the Reflective Practitioner*. Jossey-Bass, San Francisco, CA, USA.

Sharp, H. & Hall, P. (2000). An interactive multimedia software house simulation for postgraduate software engineers. In *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, pp. 688--691, New York, NY, USA. ACM.

Shaw, K. & Dermoudy, J. (2005). Engendering an empathy for software engineering. In *ACE '05: Proceedings of the 7th Australasian conference on Computing education*, pp. 135--144, Newcastle, New South Wales, Australia. Australian Computer Society, Inc.

Slavin, R. E. (1990). *Cooperative Learning*. Prentice Hall, Englewool Cliffs, NJ, USA.

Spiro, R. J.; Feltovich, P. J.; Jacobson, M. & Coulson, R. (1991). Cognitive flexibility, constructivism and hypertext: random access instruction for advanced knowledge acquisition in ill-structured domains. In Duffy, T. M. & Jonassen, D. H., editores, *Constructivism and the Technology of Instruction. A conversation*. Ed. Erlbaum, Hillsdale, NJ, USA.

Sutcliffe, A. G. & Maiden, N. A. M. (1992). Analysing the novice analyst: cognitive models in software engineering. *International Journal of Man-Machine Studies*, 36(5):719 – 740.

Taran, G. (2007). Using games in software engineering education to teach risk management. In *Software Engineering Education Training, 2007. CSEET '07. 20th Conference on*, pp. 211 –220.

Upchurch, R. L. & Sims-Knight, J. E. (1999). Reflective essays in software engineering. In *in Proceedings of the 29 th ASEE/IEEE Frontiers in Education Conference*, volume 3, p. 13A6/20. IEEE Computer Society.

Vega, K.; Fuks, H. & Carvalho, G. (2009a). Training in requirements by collaboration: Branching stories in second life. In *Sistemas Colaborativos (SBSC), 2009 Simposio Brasileiro de*, pp. 116 –122.

Vega, K.; Pereira, A.; Carvalho, G.; Raposo, A. & Fuks, H. (2009b). Prototyping games for training and education in second life: Time2play and treg. In *Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on*, pp. 1 –8.

Von Wangenheim, C. G. & Shull, F. (2009). To game or not to game? *IEEE Software*, 26(2):92–94.

Wang, A.; Ofsdahl, T. & Morch-Storstein, O. (2008). An evaluation of a mobile game concept for lectures. In *Software Engineering Education and Training, 2008. CSEET '08. IEEE 21st Conference on*, pp. 197 –204.

Wang, T. & Zhu, Q. (2009). A software engineering education game in a 3-d online virtual environment. In *Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on*, volume 2, pp. 708 –710.

Ye, E.; Liu, C. & Polack-Wahl, J. (2007). Enhancing software engineering education using teaching aids in 3-d online virtual worlds. In *Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE '07. 37th Annual*, pp. T1E–8 –T1E–13.

Zapalska, A. & Brozik, D. (2006). Learning styles and online education. *Campus-Wide Information Systems*, 23(5):325–335.

Zhang, H. (2008). *Qualitative & Semi-Quantitative Modelling and Simulation of the Software Engineering Processes*. PhD thesis, The University of New South Wales, Sydney, Australia.

Zhu, Q.; Wang, T. & Tan, S. (2007). Adapting game technology to support software engineering process teaching: From simse to mo-seprocess. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 5, pp. 777 –780.