JOGOS DE SIMULAÇÃO NO TREINAMENTO DE GERENTES DE PROJETOS DE SOFTWARE

Alexandre Ribeiro Dantas

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

_____
Prof. Cláudia Maria Lima Werner, D.Sc.

_____
Dr. Márcio de Oliveira Barros, D.Sc.

_____
Prof. Jano Moreira de Souza, Ph.D.

_____
Prof. Jorge Luis Nicolas Audy, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2003

DANTAS, ALEXANDRE RIBEIRO

Jogos de simulação no treinamento de gerentes de projetos de software [Rio de Janeiro] 2003

VIII, 98p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2003)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Treinamento baseado em jogos

2. Gerenciamento de projetos de software

I. COPPE/UFRJ II. Título (série)

Agradecimentos

Gostaria de expressar meus sinceros agradecimentos a todos que contribuíram e participaram da minha vida acadêmica e para o desenvolvimento deste trabalho.

A Deus a sabedoria, serenidade, coragem e oportunidades.

A todos que apoiaram, acreditaram, estimularam e contribuíram, de alguma maneira, família e amigos, para minha formação e evolução tanto como profissional como pessoa.

Aos meus orientadores Cláudia Maria Lima Werner e Márcio de Oliveira Barros, o ensino, a dedicação, a orientação e as oportunidades ao longo da minha formação na universidade.

Aos professores que aceitaram fazer parte da banca examinadora deste trabalho, Jano Moreira de Souza e Jorge Luis Nícolas Audy.

Aos professores do Programa de Engenharia de Sistemas e Computação e do curso de Bacharelado em Informática da Universidade Federal do Rio de Janeiro, desempenhando papel importante para minha formação profissional.

Aos amigos de muitas horas do Laboratório de Engenharia de Software e do curso de Bacharelado em Informática que estiveram comigo durante minha vida acadêmica, tornando-se, alguns, amigos para o restante da vida.

Aos coordenadores, professores, secretários e alunos da Pontifícia Universidade Católica do Rio Grande do Sul que me receberam de portas abertas para o período do mestrado sanduíche, dando todo o apoio necessário.

Aos alunos participantes dos estudos apresentados neste trabalho, em Porto Alegre e no Rio de Janeiro.

Aos funcionários do Programa de Engenharia de Sistemas e Computação.

À CAPES o auxílio financeiro.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

JOGOS DE SIMULAÇÃO NO TREINAMENTO DE GERENTES DE PROJETOS DE SOFTWARE

Alexandre Ribeiro Dantas

Dezembro/2003

Orientadores: Cláudia Maria Lima Werner
Márcio de Oliveira Barros

Programa: Engenharia de Sistemas e Computação

A aplicação de gerenciamento em projetos de software ainda é muitas vezes deficiente e inadequada, o que pode ser um fator de influência nos projetos com problemas de cronograma, custos, qualidade, ou que são cancelados. Uma das possíveis causas destas falhas em gerenciamento é a falta de compreensão das suas práticas e técnicas essenciais.

O treinamento de gerentes de projetos pode apresentar benefícios se um mecanismo prático for adicionado ao ensino tradicional. Simulações podem permitir ao estudante vivenciar situações e desafios gerenciais como em projetos reais, explorando alternativas, aprendendo lições sobre tomada de decisões e seus impactos no projeto. Além do conteúdo, a motivação e o engajamento do estudante também são fatores destacados para a eficiência do treinamento. Neste sentido, os jogos de simulação são apresentados como um mecanismo que alia as vantagens das simulações com os benefícios motivacionais dos jogos.

Neste trabalho é proposto e construído um jogo de simulação baseado em modelos dinâmicos para aplicação em um contexto de treinamento de gerentes de projetos de software. Desta forma, procura-se unir dois mercados crescentes na indústria: o treinamento da força de trabalho com o mercado de entretenimento e jogos digitais.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SIMULATION GAMES FOR TRAINNING SOFTWARE PROJECT MANAGERS

Alexandre Ribeiro Dantas

December/2003

Advisors: Cláudia Maria Lima Werner
Márcio de Oliveira Barros

Department: Systems and Computer Engineering

The adoption of project management in software projects is still deficient and inadequate, which can be an influent issue for projects with schedule, budget quality problems, or that are cancelled. One of the possible causes of these management threats is the lack of understanding of its essential practices and techniques.

The alliance of a practical mechanism with traditional teaching methodologies can provide benefits for project management training. By using simulations, students can live management situations and challenges as they appear in real projects, exploring alternatives, learning lessons from decision making and their impacts upon the project. Besides the contents, student's motivation and engagement are considered to be important factors for training efficiency. In this sense, simulation games are introduced as a mechanism that joins the aspects of simulations with the motivational benefits of games.

In this work a simulation game based on dynamic models is presented and applied in a training context for software project managers, representing the union of two growing markets in industry: the task force training with the entertainment and digital games market.

Índice Analítico

1 INTRODUÇÃO 1

1.1. O FRACASSO EM PROJETOS DE SOFTWARE 1
1.2. TREINAMENTO E ENTRETENIMENTO 3
1.3. MOTIVAÇÃO E OBJETIVO 4
1.4. ORGANIZAÇÃO 5

2 TREINAMENTO EM GERENCIAMENTO DE PROJETOS DE SOFTWARE 6

2.1. GERENCIAMENTO DE PROJETOS E TOMADA DE DECISÕES 6
2.2. TREINAMENTO E PRÁTICA 9
2.3. TREINAMENTO E JOGOS 12
2.4. TREINAMENTO COM SIMULAÇÃO EM GERENCIAMENTO DE PROJETOS 14
2.5. EFETIVIDADE DE SIMULAÇÕES NO TREINAMENTO 18
2.6. CONCLUSÕES 21

3 MODELOS PARA PROJETOS DE DESENVOLVIMENTO DE SOFTWARE 22

3.1. MODELOS PARA SISTEMAS DE DINÂMICA COMPLEXA 22
3.2. DINÂMICA DE SISTEMAS 25
3.3. LIMITAÇÕES E EXTENSÕES DA DINÂMICA DE SISTEMAS 29
3.3.1. Limitações da Dinâmica de Sistemas para os jogos 33
3.4. CONCLUSÕES 33

4 APLICAÇÃO DE JOGOS DE SIMULAÇÃO NO TREINAMENTO DE GERENTES DE PROJETOS DE SOFTWARE 35

4.1. ESTUDOS EXPERIMENTAIS INICIAIS COM TÉCNICAS DE SIMULAÇÃO 35
4.2. PROPOSTA DE UM JOGO DE SIMULAÇÃO 39
4.2.1. Os personagens 40
4.3. ESTRUTURA PARA UM JOGO DE SIMULAÇÃO 40
4.3.1. O modelo 41
4.3.2. O simulador 42
4.3.3. A máquina de jogo 44
4.3.3.1. Início da fase 44
4.3.3.2. Planejamento do projeto 45
4.3.3.3. Aceitação do planejamento 46
4.3.3.4. Execução do projeto 46
4.3.3.5. Fim da fase 48
4.4. LIÇÕES PARA O JOGADOR 48
4.5. CONCLUSÕES 49

5 *THE INCREDIBLE MANAGER* 51

5.1. O MODELO 51
5.2. O SIMULADOR 52
5.3. A MÁQUINA DE JOGO 54
5.3.1. Início do jogo 57
5.3.2. Planejamento e aceitação do projeto 58
5.3.3. Execução do projeto 60
5.3.4. Fim da fase 63
5.4. CONCLUSÕES 64

6 UM ESTUDO DA APLICAÇÃO DE JOGOS DE SIMULAÇÃO EM GERENCIAMENTO DE PROJETOS DE SOFTWARE 65

6.1. PLANEJAMENTO 65
6.2. PARTICIPANTES 66
6.3. INSTRUMENTOS 67
6.4. PROCEDIMENTO 67
6.4.1. Sessão de treinamento 68
6.4.2. Primeira sessão de simulação 68
6.4.3. Primeira sessão de discussão 68
6.4.4. Segunda sessão de simulação 68
6.4.5. Segunda sessão de discussão 69
6.5. RESULTADOS 69
6.6. CONCLUSÕES 71

7 CONSIDERAÇÕES FINAIS 73

7.1. CONTRIBUIÇÕES 73
7.2. LIMITAÇÕES E TRABALHOS FUTUROS 75

REFERÊNCIAS BIBLIOGRÁFICAS 77

APÊNDICE A - MODELO PARA O PROJETO CTRLPESC 82

APÊNDICE B - QUESTIONÁRIOS UTILIZADOS NO ESTUDO DA APLICAÇÃO DO JOGO DE SIMULAÇÃO 96

B.1. QUESTIONÁRIO DE CARACTERIZAÇÃO DO PARTICIPANTE 96
B.2. QUESTIONÁRIO DE AVALIAÇÃO QUALITATIVA 97
B.3. DADOS QUANTITATIVOS DE DESEMPENHO 98

"Ménon - E como hás de encontrar uma coisa de que não sabes absolutamente nada? Na tua ignorância, que princípio tomarás para te guiar nesta investigação? E se, por acaso, encontrasses a virtude, como a reconhecerias, se nunca a conheceste?

Sócrates - Compreendo, Ménon, o que queres dizer. Que magnífico argumento para uma discussão! Não é possível o homem procurar o que já sabe, nem o que não sabe, porque não necessita de procurar aquilo que sabe, e, quanto ao que não sabe, não podia procurá-lo, visto não saber sequer o que havia de procurar."

Platão, "Ménon", 387-361 aC.

Capítulo

1

Introdução

1.1. O fracasso em projetos de software

Com base em relatos e pesquisas sobre o estado da prática de projetos de desenvolvimento de software (GIBBS, 1994, BROWN, 1996, REEL, 1999, STANDISH GROUP, 1994, 1999, 2003), podemos observar que muitos projetos falham na indústria. Muitas vezes, os custos do desenvolvimento aumentam, enquanto a produtividade declina e o número de defeitos cresce exponencialmente em função do tamanho do software a ser entregue (BROWN, 1996).

O STANDISH GROUP (1994, 1999, 2003) apresenta os resultados de pesquisas na tentativa de identificar o escopo das falhas dos projetos de software e os principais fatores que levam os projetos a falhar. Segundo os relatórios, na indústria norte-americana, foram gastos aproximadamente 275 bilhões de dólares por ano em mais de 200.000 projetos de desenvolvimento de aplicações de tecnologia da informação. Estimativas feitas para o ano de 1995 indicavam que seriam gastos 81 bilhões de dólares em projetos cancelados e 59 bilhões adicionais em projetos concluídos fora das estimativas originais. No ano de 1994, apenas 16% dos projetos foram concluídos com sucesso, ou seja, dentro do prazo, custos e funcionalidades previstas. Por outro lado, 31% dos projetos foram cancelados e 53% dos projetos foram concluídos com extrapolação de cronograma, orçamento e deficiência nas funcionalidades que são entregues operacionais (em média apenas 61% das funcionalidades especificadas inicialmente). Para o ano de 1998, estes valores apresentam uma aparente melhora, porém os projetos problemáticos – cancelados e completados com dificuldades – ainda são a norma na indústria. Os projetos problemáticos experimentaram, em média, aumentos em 189% da estimativa de custo e 222% sobre a estimativa de cronograma originais.

As causas para os relatos de insucesso dos projetos de desenvolvimento de software têm sido buscadas em diferentes origens. Para BARROS (2001), há duas

linhas de explicação para os insucessos dos projetos de software. Uma associa os problemas do desenvolvimento a fatores tecnológicos e a crescente complexidade dos produtos. Outra linha associa as falhas dos projetos a problemas de gerenciamento, comunicação e tratamento de incertezas. BARROS (2001) também destaca que, apesar da grande evolução das técnicas e ferramentas de desenvolvimento ao longo dos anos, as técnicas utilizadas no gerenciamento de projetos de software são antigas e nem sempre são diretamente aplicáveis nos projetos recentes. A indústria, por sua vez, nem sempre utiliza uniformemente o que é conhecido como as melhores práticas (GIBBS, 1994).

Segundo BROWN (1996), uma das grandes causas de um gerenciamento deficiente ou inadequado é a falta de compreensão das suas práticas e técnicas essenciais. WHITE e FORTUNE (2002) apresentam os resultados de uma pesquisa para avaliar a prática corrente em gerenciamento de projetos de forma a estudar as possíveis causas da disparidade entre as promessas oferecidas pelas técnicas e métodos de gerenciamento e os resultados obtidos na prática. A pesquisa foi enviada para gerentes de projetos representando 620 organizações públicas e privadas. Os resultados indicam que uma grande quantidade dos participantes aplica apenas um pequeno número de métodos, metodologias, ferramentas e técnicas. A maioria relata limitações e desvantagens na utilização do que foi aplicado. Dentre as limitações citadas, destacam-se a inadequação para projetos complexos, dificuldade de modelar o mundo real, excesso de documentação, consumo de tempo e dificuldades na predição de problemas. Os resultados também indicam que os critérios utilizados para definir um projeto de sucesso mais citados são os mais difundidos na literatura de gerenciamento de projetos (isto é, dentro do cronograma, dentro do orçamento, e de acordo com a especificação), somados à necessidade de atender os objetivos organizacionais. Os fatores que são críticos para o sucesso do projeto mais destacados pelos participantes foram a existência de objetivos claros, suporte da gerência sênior, fundos e recursos adequados e cronogramas realistas. O STANDISH GROUP (1999) também identifica estes fatores como críticos, além do envolvimento do usuário, uso de marcos de projeto pequenos e a presença de gerentes de projeto experientes.

MANDL-STRIEGNITZ e LICHTER (1998) também apresentam resultados de dois estudos sobre o gerenciamento de projetos de software na indústria. O primeiro estudo foi conduzido de forma a verificar e compreender as práticas correntes, os problemas relacionados ao gerenciamento no desenvolvimento de projetos de software. O segundo estudo foi conduzido para verificar se as informações subjetivas e qualitativas coletadas pelo primeiro estudo poderiam ser relacionadas com dados

quantitativos e qualitativos de medições detalhadas realizadas sobre projetos reais de desenvolvimento de software. Os resultados dos estudos indicam que o gerenciamento é um fator influente no sucesso ou fracasso do projeto.

Segundo MANDL-STRIEGNITZ e LICHTER (1998), muitos gerentes de projeto são promovidos para cargos gerenciais a partir de êxitos apresentados em atividades técnicas, sem ter o treinamento adequado, nem adquirir as experiências e habilidades apropriadas para o gerenciamento de projetos. As organizações e os gerentes de projeto não sabem a importância e a utilidade das técnicas básicas de gerenciamento, resultando em deficiências no acompanhamento e controle dos projetos. Desta forma, o tempo geralmente investido nas atividades de gerenciamento é reduzido e inadequado. Dados quantitativos importantes não são coletados, técnicas essenciais de riscos e estimativas não são conhecidas nem aplicadas e as restrições de cronograma sempre têm prioridade sobre as atividades de garantia de qualidade.

1.2. Treinamento e entretenimento

O treinamento da força de trabalho é uma necessidade crescente para atender às demandas de uma indústria cada vez mais competitiva, gerando um enorme orçamento direcionado para os mais diversos tipos de aprendizado para adultos, tais como dirigir negócios, melhorar processos e gerenciar pessoas, riscos e organizações. Somente nos Estados Unidos, este orçamento foi estimado em mais de 100 bilhões de dólares no ano 2000, o que representa apenas uma parte do mercado mundial de treinamento, estimado em mais de 2 trilhões de dólares (PRENSKY, 2001).

Apesar da grandeza deste mercado, da crescente disponibilidade de cursos on-line e dos financiamentos feitos pelas empresas aos seus funcionários, os participantes de treinamentos muitas vezes os consideram monótonos e não se motivam a fazê-los, levando a taxas de evasão maiores de 50% (PRENSKY, 2001).

Um outro mercado crescente na indústria, também estimado em trilhões de dólares, é o de entretenimento, especialmente jogos digitais, com vendas estimadas em 7 bilhões de dólares em 2002, apresentando um crescimento de 115% em relação ao ano de 1995 (IDSA, 2003).

Nos Estados Unidos, aproximadamente 50% dos habitantes são usuários de jogos digitais e 65% dos mais freqüentes usuários são maiores do que 18 anos (IDSA, 2003). A média de idade de um jogador norte-americano é de 29 anos enquanto a idade média da força de trabalho norte-americana é de 39 anos (IDSA, 2003, PRENSKY, 2001). Adultos representam uma parte considerável do mercado de jogos de entretenimento digitais e a medida que funcionários mais antigos são substituídos

por mais novos, as novas gerações de empregados terão sempre vivido em um mundo interativo, sob efeito das tecnologias digitais de entretenimento. Por outro lado, treinamento é necessário, e as formas tradicionais de treinamento nem sempre funcionam bem pela incapacidade em transmitir para os estudantes uma grande quantidade de material técnico e de forma pouco atrativa.

1.3. Motivação e objetivo

O treinamento de gerentes de projetos de desenvolvimento de software pode ser um fator importante na formação de melhores gerentes e na possível redução dos problemas e fracassos comuns aos projetos de software da indústria, destacados no início deste capítulo. O domínio de gerenciamento de projetos de software, entretanto, apresenta uma dinâmica complexa, o que pode dificultar a transferência do conhecimento segundo as formas tradicionais de ensino (aulas expositivas e livros-texto). Assim como futuros pilotos de avião são treinados em simuladores de vôo além das aulas tradicionais, no domínio do gerenciamento de projetos de software também é preciso exercitar a vivência prática do estudante.

Este trabalho tem como objetivo propor um mecanismo de treinamento prático, em um ambiente virtual, aliado às formas tradicionais de ensino, de forma a permitir ao estudante observar os diversos fatores que influenciam o desenvolvimento de um projeto de software e como suas decisões afetam o comportamento dinâmico do projeto. Incluir este mecanismo de treinamento prático junto ao aprendizado teórico pode permitir ao futuro gerente ganhar experiência e conhecimento, diminuindo os riscos e fracassos em projetos reais.

Na busca deste objetivo, destaca-se uma oportunidade de integração de dois mercados crescentes (treinamento e entretenimento), aparentemente ortogonais, permitindo que a força de trabalho jogue e aprenda ao mesmo tempo. Segundo PRENSKY (2001), algumas das palavras que são mais frequentemente associadas com o conceito de trabalho são pressão, monotonia, prazos, salário, horas-extras, metas, entre outras. As palavras citadas mais associadas ao conceito de jogar, geralmente, são diversão, riso, escolha, espontaneidade, relaxamento, entre outras. Para o autor, as pessoas dão maior preferência para tarefas difíceis quando apresentadas como jogos ao invés de trabalho.

Desfa forma, os jogos são apresentados como uma forma diferenciada de simulação que pode oferecer benefícios adicionais à pura utilização de simuladores computacionais de modelos. Considerada uma atividade divertida e agradável, a inserção de jogos em um contexto de treinamento pode contribuir para a melhoria da

motivação e do engajamento do estudante, aumentando a atratividade, concentração e, finalmente, a transferência de conhecimento segundo as metas educacionais previstas.

Ao longo deste trabalho, são apresentadas em detalhes as suas motivações, no domínio de gerenciamento de projetos, simulações e jogos, que guiaram à proposta de um jogo para o treinamento de gerentes, baseado em simulações de modelos dinâmicos do desenvolvimento de software. A partir desta proposta, é apresentada a construção de um protótipo e um estudo experimental de foco qualitativo sobre sua aplicação, quando inserido em um processo de treinamento de gerentes de projetos de software.

1.4. Organização

Este trabalho é composto por 7 capítulos. Este primeiro capítulo apresentou a motivação do trabalho, baseada na demanda por eficiência e prática no treinamento de gerentes de projetos de software e nos benefícios do crescente mercado de entretenimento e jogos digitais; o objetivo de construção de um jogo de simulação para treinamento em gerenciamento de projetos de software e a forma como este trabalho está organizado ao longo dos capítulos.

O segundo capítulo discute as características do gerenciamento e do treinamento de gerentes de projetos de software, destacando a contribuição de um componente prático por meio de simulações computacionais e jogos digitais.

O terceiro capítulo discute a modelagem do desenvolvimento de projetos de software para fins de simulação e a Dinâmica de Sistemas como disciplina de base para a construção dos modelos de um jogo de simulação.

O quarto capítulo apresenta a proposta de um jogo para treinamento em gerenciamento de projetos de software, baseado em simulação de modelos que representam a dinâmica durante o desenvolvimento de software. A proposta de construção do jogo se baseia em três principais componentes: os modelos, o simulador e a máquina de jogo.

O quinto capítulo descreve a construção e utilização de um jogo de simulação, chamado *The Incredible Manager*, com base na proposta deste trabalho.

O sexto capítulo apresenta a elaboração e execução de um estudo qualitativo sobre a aplicação do jogo de simulação construído inserido em um contexto de treinamento composto por sessões de simulação e sessões de discussão.

Finalmente, o sétimo capítulo apresenta as considerações finais do trabalho, destacando suas principais contribuições, limitações e trabalhos futuros.

Capítulo

2

Treinamento em gerenciamento de projetos de software

Apesar do gerenciamento ser um conceito universal, a sua aplicação nos projetos de engenharia de software ainda é muitas vezes deficiente e inadequada, o que pode ser um fator de influência no grande número de projetos problemáticos ou que fracassam. Neste sentido, a formação e o treinamento de gerentes de projetos pode ser um dos ingredientes para o sucesso dos projetos de software, reduzindo os riscos associados à falta de conhecimento e experiência na área de gerenciamento.

O domínio de projetos de desenvolvimento de software, por sua vez, exibe aspectos dinâmicos que podem ser melhor explorados no treinamento quando componentes práticos são aliados aos meios tradicionais de ensino, assim como no treinamento de pilotos de aviões, por exemplo. Simuladores são ferramentas que podem caracterizar este componente prático. Por outro lado, os jogos apresentam algumas características que podem favorecer o treinamento de adultos com o aumento da motivação e do engajamento.

Neste capítulo, são discutidos os princípios do gerenciamento de projetos de software e o mecanismo de tomada de decisões, apresentando a importância da formação e da experiência dos gerentes na condução e sucesso dos projetos. Posteriormente, é analisado o uso de um componente prático no treinamento e as vantagens do uso de jogos. Por fim, é discutida a aplicação de jogos e simulações no treinamento em gerenciamento de projetos e, ainda, questões sobre a efetividade da sua utilização.

2.1. Gerenciamento de projetos e tomada de decisões

Segundo o PMBOK (1996), um projeto é definido como um esforço temporário (com início e fim determinados) realizado com o objetivo de criar um produto ou

serviço único. Projetos envolvem a participação de pessoas com características diferentes. Organizações e indivíduos, incluindo patrocinadores, clientes, a própria organização e a equipe de desenvolvimento, podem ser afetados positivamente ou negativamente pelo projeto. Para permitir um melhor controle de sua execução, em geral, os projetos são divididos em várias fases que, em conjunto, formam o ciclo de vida do projeto. O ciclo de vida serve como uma definição do início e fim do projeto, estabelecendo que trabalho técnico deve ser feito e quem deve estar envolvido em cada uma das fases. Cada fase é identificada por um ou mais subprodutos que representam um marco tangível e verificável de trabalho.

O gerenciamento de projetos é a aplicação de conhecimento, experiências, habilidades, ferramentas e técnicas nas diversas atividades de um projeto com o propósito de planejamento e controle destas atividades, de forma a atender ou exceder as necessidades e expectativas das pessoas envolvidas com o projeto. Desta forma, o gerenciamento envolve um balanceamento entre demandas competitivas relacionadas a recursos limitados, escopo, cronograma, custo e qualidade, além da resolução de conflitos entre os participantes do projeto, que podem possuir diferentes necessidades e expectativas (PMBOK, 1996).

THAYER (1997) apresenta as responsabilidades de um gerente de projetos segundo o modelo clássico de gerenciamento. Por este modelo, o gerenciamento é dividido em cinco grandes funções: planejamento, organização, gerência de recursos humanos, direção e controle. O planejamento é composto por atividades que determinam o curso de ações para se atingir os objetivos do projeto, decidindo o que será feito, como, quando e por quem. As funções de organização envolvem o desenvolvimento de uma estrutura organizacional efetiva e eficiente para a realização das tarefas e distribuição de responsabilidades e autoridade. A gerência de recursos humanos envolve a seleção e treinamento das pessoas para os papéis estabelecidos pela organização do projeto, garantindo que eles serão ocupados por pessoas qualificadas e motivadas. As funções de direção e controle procuram criar um ambiente de supervisão e motivação da equipe, visando garantir que o projeto se desenvolva de acordo com o plano, medindo resultados e tomando as atitudes corretivas necessárias.

O PMBOK (1996), por sua vez, divide o gerenciamento de projetos em diversas áreas de conhecimento. Cada área de conhecimento é descrita por processos que interagem de forma a garantir o desenvolvimento e a execução do plano do projeto, assim como o controle de mudanças durante o seu curso. As áreas que contêm funções essenciais para o gerenciamento de projetos estão relacionadas ao gerenciamento de escopo, de cronograma, custos e controle de qualidade. As demais

áreas contêm funções facilitadoras e são relacionadas ao gerenciamento de recursos humanos, comunicações, riscos, aquisição de bens e serviços externos.

Apesar das abordagens propostas para o gerenciamento de projetos de software, este ainda é uma tarefa complexa, com decisões por vezes baseadas em dados incompletos, insuficientes ou mutáveis. Segundo BARROS (2001), o gerenciamento de projetos é uma atividade intensamente baseada em conhecimento. Os gerentes mais experientes, geralmente, obtêm mais sucesso no planejamento e controle de projetos do que os inexperientes, uma vez que utilizam conhecimento e habilidades adquiridos durante diversos projetos realizados no passado. Este conhecimento tácito permite que os gerentes experientes possam reagir melhor quando uma tomada de decisão for necessária durante o processo de desenvolvimento de software.

Segundo STERMAN (1988), as decisões e ações que tomamos são baseadas em imagens mentais que fazemos do mundo real e dos relacionamentos entre suas partes. Estes modelos mentais atuam como filtros na interpretação de experiências, avaliação de planos e escolha de possíveis cursos de ação. De acordo com DOYLE *et* al. (2000), um modelo mental é uma representação conceitual limitada que um indivíduo armazena na memória a partir da sua percepção de um sistema dinâmico real ou hipotético. Como é relacionada à percepção individual, a relação entre o sistema e a sua representação mental depende de cada indivíduo e da acurácia da sua percepção.

O gerenciamento de projetos de software é uma área em que são necessárias tomadas de decisões dinâmicas, ou seja, a avaliação e escolha de alternativas ocorre durante o curso de desenvolvimento de um sistema que se modifica ao longo do tempo. Para DOYLE *et al*. (2000), na busca por uma regra de decisão para uma ação a ser tomada, um gerente pode proceder de duas formas: simulações mentais ou busca por heurísticas. De acordo com a primeira alternativa, o gerente usa sua percepção atual da realidade do projeto, altera seu modelo mental e conduz uma simulação mental para predizer o estado futuro que será obtido a partir da sua decisão. Na segunda alternativa, o gerente pode buscar na memória alguma regra de decisão baseado em experiências e heurísticas de circunstâncias similares anteriores que possam se adequar na realidade atual do projeto. Enquanto a segunda abordagem depende da quantidade e qualidade das experiências vividas anteriormente, a primeira abordagem é limitada pelas próprias restrições dos modelos mentais. Segundo STERMAN (1988) e DOYLE *et al.* (2000), os modelos mentais são instáveis ao longo do tempo e seus limites são difusos. A mente humana tem limitações em lidar com um grande número de fatores distintos simultaneamente,

fazendo com que os modelos possam ser ambíguos e propensos a erros, não conseguindo capturar bem relações que não sejam lineares nem relações de ações e reações distantes no tempo. Além disto, pode ser difícil compartilhar e examinar modelos mentais com outras pessoas (apenas uma fração do conhecimento que uma pessoa obtém ao longo da vida é escrito explicitamente e uma fração menor ainda é expressa de forma quantitativa). Desta forma, modelos mentais usados para tomar decisões são simplificados e podem não ser simulados mentalmente com precisão.

Muitos problemas enfrentados pelos gerentes de projetos de software não são intuitivos e a confiança das decisões baseadas apenas na experiência do gerente e sua equipe não é uma política que tem se mostrado eficiente, tendo em vista os relatos de fracassos de projetos de software na indústria (BARROS, 2001). Nem sempre as medidas tomadas pelos gerentes, com base em sua intuição, para garantir o sucesso de um projeto fora de controle representam a solução mais adequada. Em decorrência da adoção destas medidas, o comportamento mais comumente observado em projetos de software é o de "apagar incêndios". Dentre as medidas tomadas com maior freqüência, podemos destacar o aumento do cronograma, da equipe ou orçamento, pressão nos fornecedores, redução do escopo do projeto, mudança nas metodologias de desenvolvimento e da tecnologia utilizada, ou até mesmo o cancelamento do projeto (PFAHL *et al*., 2000).

Para o PMBOK (1996), dentre as habilidades do gerente de projeto destacam-se a liderança, a comunicação, a negociação, a solução de problemas e a tomada de decisões. Tornar-se um gerente de projetos experiente, portanto, pode custar muitas decisões erradas em projetos reais. A origem técnica da maioria dos gerentes e o treinamento inadequado podem aumentar ainda mais o custo dos erros nos projetos.

2.2. Treinamento e prática

Treinamento está relacionado com aprendizado. Aprender significa tornar-se apto ou capaz de realizar alguma coisa, tomar conhecimento de algo, retê-lo na memória, em conseqüência de estudo, observação, experiência ou advertência (FERREIRA, 1993).

Há na literatura psicológica diversas teorias que procuram estudar o desenvolvimento e a aprendizagem humana. Para a psicologia do desenvolvimento de Jean Piaget (FLAVELL, 1988), o funcionamento intelectual é uma forma especial de atividade biológica baseada em duas dimensões hereditárias: uma hereditariedade específica de estruturas anatômicas inatas, que impedem ou facilitam o funcionamento intelectual mas não o explicam, e uma hereditariedade geral, representando um modo

de funcionamento intelectual a partir do qual são construídas todas as aquisições cognitivas. O funcionamento intelectual, acima de tudo, trata-se também de um processo de adaptação ao meio, que ocorre por meio de intercâmbio entre o organismo e o ambiente, tendo como efeito a modificação do organismo. Este modelo tem duas características importantes: as invariantes funcionais e o construtivismo. As invariantes funcionais indicam que o modo de funcionamento intelectual permanece essencialmente constante durante toda a vida. Por outro lado, a teoria piagetiana é construtivista, uma vez que prevê que as estruturas cognitivas são construídas por meio da ação do organismo com o ambiente.

Uma outra teoria, a andragogia (KNOWLES, 1984), procura desenvolver uma teoria específica para o aprendizado adulto. A andragogia presume que adultos são auto-direcionados e esperam ter responsabilidade para tomar decisões. Desta forma, seguem-se quatro princípios básicos: adultos precisam saber porque têm de aprender alguma coisa, precisam aprender de forma experimental, abordam o aprendizado como resolução de problemas e aprendem melhor quando o tópico é de valor imediato. Portanto, o treinamento adulto deve enfatizar mais o processo e menos o conteúdo e o instrutor.

Para muitos instrutores, segundo PRENSKY (2001), aprendizagem, treinamento e educação não ocorrem em função do aprendiz, mas sim em função do conteúdo: o que ensinar, ao invés de por que e como ensinar. Até os dias atuais, as estratégias de ensino mais comuns são focalizadas no conteúdo e no instrutor, que é a pessoa que detém o conhecimento e assume a responsabilidade por decisões sobre o que será aprendido, como e quando será aprendido. O instrutor deve então tentar transferir o conhecimento sobre o conteúdo por meio de aulas expositivas, livros e testes.

Muitas vezes, grande parte do que as pessoas aprendem são habilidades e comportamentos, e uma das principais formas deste aprendizado é por meio de prática. Os adultos, portanto, aprendem mais efetivamente quando o aprendizado contém um componente prático, pois aprender experimentando é mais divertido e testar o que foi aprendido oferece às pessoas a chance de participarem, cometerem erros e desafiarem a si mesmas. Na área de gerenciamento de projetos de software, por exemplo, as formas de ensino tradicionais, por meio de aulas expositivas e livros, são fundamentais para o aprendizado das técnicas e práticas que formam o corpo de conhecimento da área, porém podem não ser suficientes para a motivação e formação de um bom gerente de projetos.

Para DRAPPA e LUDEWIG (2000), uma das chaves para o sucesso na educação é a motivação e a melhor motivação para a aprendizagem em gerenciamento é a experiência em projetos que falharam por falta de gerenciamento.

As falhas devem ser consideradas fatores críticos e naturais para a inovação e o progresso. A análise dos erros e lições aprendidas é um dos ingredientes fundamentais para uma receita de sucesso no gerenciamento de projetos de software (STANDISH GROUP, 1999). É importante que as características dinâmicas destes projetos sejam experimentadas como elas ocorrem na realidade. Uma possibilidade para o gerente ter este aprendizado é testar seus conhecimentos e hipóteses na realização de projetos pilotos. Entretanto, o tempo e o custo desta solução para os gerentes e para as organizações nem sempre são adequados, uma vez que podem ser necessários vários projetos para que o gerente adquira a experiência necessária, vindo a fracassar em diversos deles.

Outra alternativa para o treinamento dos gerentes que ameniza o custo dos projetos piloto e permite a experimentação é a simulação de modelos que representam o processo de desenvolvimento de software. Uma simulação é uma experiência realizada com modelos que permite reproduzir o comportamento de um sistema por meio de operações numéricas realizadas pelo computador sobre as informações do modelo, facilitando a compreensão das interações entre o sistema como um todo e seus componentes (BELLINGER, 2003). A simulação pode ser vista como um estágio intermediário entre a limitada instrução teórica e a vida real, em que a experimentação pode ser muito cara ou perigosa. Por meio de simulações, é possível o acúmulo de experiências a partir da introdução e prática de situações desconhecidas. Diferente de projetos reais, não há a necessidade da obtenção de produtos tangíveis nem o risco de conseqüências dramáticas no caso de fracassos. Outra vantagem do uso de simulações é a redução do tempo gasto no treinamento. Os modelos simulados podem ser rapidamente avaliados e repetidos com diversos cenários, enquanto projetos reais podem se estender por semanas ou meses, tornando impraticável a exploração de várias alternativas.

Um aspecto importante no uso de simulações no treinamento é a modelagem da realidade, conforme destaca PRENSKY (2001). A realidade pode levar à monotonia, por isso, a fidelidade da simulação pode variar entre baixa e alta. Em simulações de baixa-fidelidade, um ou poucos elementos são abstraídos da realidade para serem enfatizados, o que resulta em um aprendizado de princípios gerais e idéias que podem ser usadas em uma variedade de situações. Em simulações de alta-fidelidade, há a tentativa de modelar o máximo possível da realidade, o que pode levar a um treinamento mais efetivo.

Um simulador, em geral, está relacionado a propósitos de predição, uma vez que procura modelar uma situação específica, com considerável precisão, de forma a oferecer informações úteis sobre os possíveis resultados obtidos a partir de uma

decisão particular (ELGOOD, 1988). Há na literatura e na indústria ferramentas para apoiar o uso de simulação como suporte ao aprendizado, mas para MAIER e GRÖßLER (1998) é preciso uma uniformização nos termos utilizados para definir estas ferramentas. Para AHDELL e ANDRESEN (2001), um simulador atua sobre um modelo operacional, utilizando componentes selecionados de um processo, mecanismo, sistema real ou hipotético. Um jogo, por sua vez, é uma atividade estruturada em que dois ou mais participantes (podendo um deles ser um computador) competem dentro de determinadas regras de forma a atingir um objetivo.

2.3. Treinamento e jogos

Simuladores, em geral, produzem, como resultado da execução de seus modelos, números ou gráficos que são representações muito abstratas do que realmente acontece durante a simulação. Além da dificuldade de interpretar os resultados gerados pelo simulador, para os aprendizes, o simples acompanhamento de números não relacionados perceptualmente com acontecimentos reais pode acabar tornando o treinamento com simulação tão monótono quanto as tradicionais aulas expositivas e testes.

Para PRENSKY (2001), em uma situação artificial de treinamento, prover motivação é um papel tradicional do instrutor. Algumas potenciais motivações incluem satisfação pessoal, vitória, prazer e diversão. Os jogos são um subconjunto de diversão que criam motivação e relaxamento e contêm algumas qualidades especiais que os tornam bem apropriados para o processo de aprendizagem.

Um jogo contém características próprias que o diferem de um simulador. Pode haver um jogo que não é uma simulação e uma simulação que não é um jogo, mas também um jogo de simulação. RIEBER (1996) comenta que o uso do termo jogar aplicado ao ensino ainda desperta interpretações equivocadas. Para muitos, jogar é algo contrário a trabalhar e trabalhar é respeitável e produtivo, enquanto jogar não. Jogar é considerado uma atividade fácil e irrelevante, sem conseqüências para o aprendizado formal ou informal.

Alguns dos principais elementos que compõem os jogos são interação, flexibilidade, competição, *feedback* e efeitos dramáticos, usabilidade e realidade (PRENSKY, AHDELL e ANDRESEN, 2001). Uma simulação, portanto, não é um jogo sem seus elementos estruturais e especialmente diversão e motivação, de forma a garantir o engajamento do participante no aprendizado.

A interatividade é uma característica fundamental, pois o jogador pode facilmente se entediar sem uma participação ativa. A flexibilidade permite ao jogador

um nível de controle e investigação do conteúdo seguindo seu próprio caminho, de acordo com regras bem definidas. A competição representa o desafio, o conflito, a solução de problemas que requerem exploração e habilidades estratégicas que motivam o jogador na busca dos objetivos definidos no jogo.

O jogo, por meio de *feedback* às ações do jogador, mede e apresenta o progresso deste na busca dos objetivos. Este *feedback* é feito por meio de representação e efeitos dramáticos, ou seja, a história do jogo. Os efeitos dramáticos são os fatores que relacionam entretenimento ao conteúdo do aprendizado, por meio de sons, música, humor e boas histórias. Estes fatores adicionam contexto, personagens e dramatização ao conteúdo, permitindo que o aprendiz tenha a chance de vivenciar um papel e atuar dentro do jogo. Para tanto, a usabilidade e realidade do jogo também são fatores importantes e representam o quão fácil e intuitivo é o jogo e como os diferentes elementos de mídia se relacionam para oferecer uma situação de proximidade com o mundo real nele representado.

Para RIEBER (1996), os jogos são atividades em que as pessoas são absorvidas e parecem fluir na sua realização de forma automática e espontânea, se sentindo livres no controle da atividade enquanto sentem que o tempo se transforma, ou seja, as horas passam sem serem percebidas. Esta descrição de um estado de satisfação e engajamento é resultado de atividades que oferecem prazer e diversão, o que ocorre quando a atividade oferece desafios, tem objetivos claros, absorve a atenção do participante e oferece *feedback* claro e consistente na busca dos objetivos.

AHDELL e ANDRESEN (2001) explicam que há dois tipos de motivação: a intrínseca e a extrínseca. A motivação intrínseca provém da motivação pessoal, do relacionamento natural entre a pessoa e a razão pela qual ela realiza uma atividade. A motivação extrínseca provém de estímulos externos, tais como recompensas verbais, elogios ou até mesmo recompensas tangíveis, como dinheiro. Para RIEBER (1996), o estudante deve considerar o aprendizado intrinsecamente motivante. AHDELL e ANDRESEN (2001) e RIEBER (1996) citam uma relação entre a motivação intrínseca e o uso de jogos para o aprendizado, criada por três qualidades que são o desafio, a fantasia e a curiosidade.

A diversidade de jogos e suas características nos dias atuais torna difícil uma classificação rígida e organizada. Há diferentes tipos de conteúdo a serem aprendidos quando se deseja aplicar jogos em educação e treinamento e isto deve ser levado em consideração na escolha da categoria de jogo que melhor se aplica ao conteúdo-alvo. PRENSKY (2001) apresenta uma taxonomia sobre jogos a partir de 5 categorias: jogos de tabuleiro, jogos de cartas, jogos atléticos, jogos infantis e jogos digitais. Estes últimos, por sua vez, apresentam diversas outras categorias, podendo citar algumas

principais tais como jogos de ação, quebra-cabeça (*puzzle*), aventura, estratégia e simulação. Estes três últimos estilos podem ser destacados. Em jogos de aventura, o jogador deve achar seu caminho por um mundo desconhecido, interagir com objetos e pessoas e resolver diversos problemas. Em jogos de simulação, o jogador é requisitado a pilotar coisas, construir mundos ou gerenciar companhias. Em jogos de estratégia, o jogador torna-se responsável por algo como um exército ou uma civilização e deve fazê-la evoluir segundo um conjunto de regras e de acordo com suas decisões, sozinho ou contra diversos oponentes.

Para um conteúdo que privilegie o julgamento, raciocínio e compreensão de sistemas (treinamento de pensamento estratégico e tático, análise qualitativa e decisões gerenciais, por exemplo), são mais adequados os jogos que apresentem exemplos, resolução de problemas e prática de realização de escolhas. Jogos de quebra-cabeça, estratégia, aventura e simulações representam alternativas apropriadas para estes propósitos.

O uso de jogos digitais também oferece muito mais recursos tecnológicos do que jogos comuns de tabuleiro. Eles podem apresentar maiores e variados recursos sonoros e visuais que ofereçam maior *feedback* e diversão aos usuários. O processamento das regras e respostas do modelo às ações do jogador são realizadas computacionalmente, mais rápidas e oferecendo um grande número de variações de opções e cenários, inclusive permitindo que o usuário jogue contra o próprio computador e não necessariamente oponentes reais. Os jogos digitais podem ser personalizados por cada jogador e facilmente atualizados ou modificados de acordo com níveis de desafio ou cenários desejados, lidando com uma enorme quantidade de conteúdo.

2.4. Treinamento com simulação em gerenciamento de projetos

Utilizando simulações em projetos de software, um gerente de projetos pode tirar proveito do treinamento sem correr os riscos de fracassos em projetos reais. Entretanto, os estudantes não devem apenas aprender a reconhecer um conjunto finito de padrões de comportamento, mas atender a alguns objetivos, conforme avalia MANDL-STRIEGNITZ (2001):

- Aprender e aplicar de forma prática importantes funções e técnicas do gerenciamento de projetos, como o planejamento ou o controle do projeto, modelos e ferramentas de estimativas, técnicas de decomposição, etc;

- Experimentar as conseqüências positivas e negativas de executar ou negligenciar importantes funções do gerenciamento de projetos, como análise de riscos ou atividades de controle de qualidade em função do custo e cronograma;
- Confrontar-se com aspectos complexos que devem ser resolvidos no desenvolvimento de software durante o planejamento e controle do projeto, tais como a perda de um membro importante da equipe e a realização de estimativas realistas para o projeto;
- Experimentar as relações complexas de projetos de software, analisando as diversas influências que existem sobre o projeto antes de chegar a uma decisão. Por exemplo, nem sempre adicionar mais desenvolvedores em um projeto atrasado aumenta a produtividade da equipe, conforme indica a “lei de Brooks” (BROOKS, 1974);
- Ter oportunidade de testar diferentes abordagens de gerenciamento e aprender sobre suas conseqüências, ou seja, estar livre para cometer erros e aprender com os seus resultados;
- Aprender a se concentrar no seu papel de gerente de projetos, mesmo que sua origem seja técnica.

PFAHL *et al*. (2000) apresentam um experimento controlado para investigar a efetividade de um programa de treinamento em gerenciamento de projetos de software utilizando um módulo de simulação. Os participantes foram divididos em dois grupos. Um dos grupos pode contar apenas com uma técnica de estimativa tradicional, o COCOMO, enquanto os demais participantes contaram com um modelo de simulação. A efetividade do treinamento foi avaliada por meio de questionários que procuravam capturar o interesse em questões de gerenciamento de projetos, conhecimento sobre padrões de comportamento típicos de projetos de desenvolvimento de software e a compreensão de algumas dinâmicas de projeto, tais como questões que envolvem desde a aplicação de modelos de planejamento até conflitos entre variáveis do processo de desenvolvimento envolvendo restrições de força de trabalho, cronograma ou mudança de requisitos, por exemplo. A análise dos resultados obtidos no estudo experimental não demonstrou suporte para garantir a influência do treinamento sobre a compreensão das dinâmicas de projetos, comparando os participantes antes e após a sessão de treinamento e comparando os resultados dos dois diferentes grupos.

Algumas propostas de simuladores aplicados ao treinamento em engenharia de software podem ser encontradas na literatura. SYCAMORE (1996) desenvolveu uma ferramenta para o gerenciamento de projetos de software que utiliza equações

matemáticas para representar o efeito de diversas condições e ações gerenciais sobre o projeto, como pressão de cronograma, re-trabalho e excesso de comunicação. O gerente de projetos pode utilizar a ferramenta para simular hipóteses de decisões gerenciais sobre o escopo do projeto, recursos, produtividade e seqüenciamento de trabalho antes de fazer modificações no projeto, variando aspectos tais como o perfil, quantidade e experiência da equipe.

MERRIL (1995) propõe a criação e validação de um simulador de projetos de software e também um processo de treinamento que tem como foco a utilização de um simulador. O objetivo é criar uma forma de treinamento para ensinar lições de gerenciamento de projetos de software que demandem que o estudante planeje e reaja a cenários específicos, como a "lei de Brooks". O modelo utilizado pelo simulador permite a inclusão de aspectos sociais e industriais do processo de desenvolvimento que afetam diretamente ou indiretamente o cronograma, a qualidade, o custo e esforço do desenvolvimento.

MAIER e STROHHECKER (1996) desenvolveram um simulador chamado LEARN! para um jogo de gerenciamento competitivo chamado LOBSTER. No jogo, quatro grupos de estudantes competem entre si, tomando decisões sobre investimentos em equipamentos, preços, propaganda, pesquisa e desenvolvimento, entre outros. O simulador gerencia três grupos competitivos e as decisões do jogo, de forma que um jogador possa treinar e investigar suas decisões. Apesar da utilização do LEARN! para investigar se há contribuições para a efetividade das decisões tomadas pelo jogador, os resultados não foram estatisticamente válidos, porém ofereciam indícios de que o uso do simulador causa melhorias nas decisões dos jogadores.

Oh e van der Hoek (2002) propõe um jogo para simulação de regras da engenharia de software, chamado SimSE, descrevendo os benefícios do uso de simulação para o treinamento e apontando limitações na maioria das simulações de processos de software. A filosofia básica destas simulações consiste em criar um modelo que represente o mundo real, escolher um conjunto de valores para alguns parâmetros iniciais do modelo, executar o modelo e examinar os resultados e os rastros do modelo durante a simulação para tentar compreender o funcionamento e analisar oportunidades de melhoria do processo. As simulações são, em grande parte, executadas sem interrupção e a visualização de seus resultados utiliza recursos de interface com usuário simples (tais como gráficos, tabelas, botões e textos), colocando o usuário em uma posição passiva e o desenvolvimento de software como linear e não-interativo. Os jogos, por sua vez, oferecem diversão, estimulam experimentação,

apresentam interfaces gráficas que permitem interação imediata do usuário e a percepção de relações de causa e efeito e relações distantes no tempo.

A proposta do SimSE é a criação de um jogo mono-usuário em que o jogador assume o papel de gerente de projetos e deve gerenciar uma equipe na realização de um projeto de engenharia de software. As atividades de gerenciamento incluem contratações, demissões, alocação de pessoal, monitoramento de progresso e compra de ferramentas. A interface com o usuário proposta visa representar um ambiente virtual de escritório onde ocorre o desenvolvimento do projeto, provendo o jogador de informações sobre os personagens, artefatos e informações pertinentes ao projeto para que o jogador guie suas decisões e complete o projeto dentro de um orçamento, cronograma e qualidade requisitados inicialmente. Uma ferramenta separada, o construtor de modelos, visa permitir a criação de modelos de projetos de software específicos, de acordo com as metas educacionais previstas. Uma ferramenta pertencente ao jogo, a ferramenta de explicação, visa oferecer ao estudante um rastro de eventos com o histórico de seu progresso e ações ao longo do tempo, de forma a indicar claramente as regras e relações de causa e efeito por trás da execução do jogo que impactam no desempenho e no sucesso ou fracasso do jogador. No estado atual do desenvolvimento do SimSE, as regras conhecidas na literatura da engenharia de software que irão compor o modelo do jogo já foram catalogadas, porém, a ferramenta ainda não dispõem de uma interface gráfica com efeitos multimídia.

A equipe do laboratório de engenharia de software da Universidade de Stuttgart desenvolve o projeto SESAM ("*Software Engineering Simulation by Animated Models*"), visando satisfazer as lacunas dos problemas de treinamento de gerentes de projetos de software e das simulações contínuas (DRAPPA e LUDEWIG, 2000). No SESAM, o jogador controla o simulador por meio de uma interface puramente textual. O estudante recebe uma descrição do projeto e seu objetivo é fazer dele um sucesso, podendo interagir com os funcionários, associá-los a tarefas do processo de desenvolvimento e questioná-los sobre suas qualificações e o andamento das suas atividades. Ao final, o jogador recebe sua pontuação com base nos resultados obtidos face às expectativas iniciais para o projeto. O simulador é baseado em um modelo para projetos de software, chamado “QA Model” por enfatizar questões de garantia de qualidade. A construção do modelo foi baseada, principalmente, na análise da literatura de engenharia de software com foco em investigações empíricas.

DRAPPA e LUDEWIG (2000) descrevem um estudo experimental para avaliar se haveria algum efeito constatado sobre o conhecimento e habilidades em engenharia de software e gerenciamento de projetos dos estudantes após a utilização do sistema SESAM. O estudo foi baseado em pré-testes, simulação e pós-testes, porém os

resultados não foram conclusivos na indicativa de que o uso do SESAM causou algum efeito no aprendizado. DRAPPA e LUDEWIG (2000) e MANDL-STRIEGNITZ (2001) sugerem que o simulador é apenas parte de um processo de aprendizagem que deve englobar aulas e simulação. Neste sentido, MANDL-STRIEGNITZ (2001) propõe uma sessão de treinamento para o gerenciamento de projetos que visa mesclar sessões de simulações com sessões de discussões teóricas. Inicialmente, os estudantes recebem as informações sobre o projeto que devem gerenciar durante a simulação, utilizando o sistema SESAM. A partir destas informações, que incluem características de tamanho do projeto e requisitos de custo, tempo e qualidade, o estudante deve planejar e simular o projeto de acordo com seu plano. Ao final da simulação, há a primeira sessão de discussão em que um tutor e os estudantes devem analisar os resultados da simulação. Desta forma, eles podem reconhecer a influência e as relações das diferentes decisões e resultados obtidos, identificando erros a serem evitados em ocasiões futuras. Posteriormente, há uma nova sessão de simulação em que os estudantes têm a chance de repetir o projeto, aplicar suas descobertas e comparar seus novos resultados com os anteriores. Uma última sessão de discussão é feita para discutir em que aspectos os estudantes obtiveram melhores resultados, assim como os possíveis problemas que ainda possam restar.

Dentre os motivos pelos quais os estudos não apresentaram resultados positivos estatisticamente validados, DRAPPA e LUDEWIG (2000) observam que os estudantes não refletem sobre seus resultados, não sabendo o que houve de errado no projeto. Uma melhor análise é necessária e, para isto, os resultados devem ser apresentados de forma mais clara, diminuindo o esforço da compreensão do estudante. PFAHL et al. (2000) consideram que os modelos utilizados para a simulação podem ser melhor explorados não só na exibição dos relacionamentos causais e suas forças entre os componentes do modelo, mas também devem oferecer meios de modificar a estrutura destes relacionamentos para tornar os efeitos destas mudanças sobre o projeto visíveis durante a simulação.

2.5. Efetividade de simulações no treinamento

Para GRÖßLER (2000), o uso de simulações é adequado no treinamento uma vez que as pessoas não são apenas recipientes para serem preenchidas com conhecimento. À luz da teoria piagetiana (FLAVELL, 1988), o conhecimento deve ser construído por meio de uma participação do estudante a partir de uma exploração ativa e auto-controlada dos recursos da aprendizagem juntamente com o conhecimento já adquirido, permitindo assim a construção de mais conhecimento.

Entretanto, apesar das aparentes vantagens da utilização de simulações no processo de aprendizado do gerenciamento de projetos de software, poucos estudos têm sido realizados para tentar demonstrar a efetividade do uso de simulações neste contexto. Esta efetividade é uma questão difícil de ser avaliada, uma vez que os efeitos do treinamento são difíceis de serem medidos e isolados para serem relacionados ao uso da simulação.

Uma grande discussão em torno da busca da efetividade dos simuladores consiste na sua comparação com os outros instrumentos de ensino, como leitura de livros e aulas expositivas. Alguns métodos podem ser mais adequados para certas situações do que outros e há muitos fatores que interferem nesta comparação (qualidade didática do professor ou conteúdo do livro lido). Em situações particulares, simuladores podem ser mais adequados, porém resultados sobre sua efetividade e eficiência sobre as demais formas de ensino e situações não devem ser generalizados (GRÖßLER *et al*., 1999).

Para AHDELL e ANDRESEN (2001), a efetividade das simulações deve ser medida pelos resultados de aprendizado dos estudantes (como o aumento de conhecimento e a melhoria de habilidades) divididos pelos custos do treinamento. Entretanto, medir a influência direta do treinamento sobre os estudantes é uma tarefa complicada e a maioria das pesquisas nesta área tem sido conceitual, ao invés de empírica e quantitativa. A maioria das medições é feita qualitativamente para capturar se o aprendiz se divertiu e sentiu que aprendeu. Medições também são realizadas utilizando-se pré-testes e pós-testes, porém pode ser difícil capturar os efeitos a longo prazo ou separá-los de outros fatores que podem influenciar o aprendizado.

Para GRÖßLER *et al*. (1999), a validade do uso de simulações não foi ainda provada, mas há alguns indícios de que pode promover o aprendizado. Geralmente, os simuladores não são feitos de forma a favorecer a experimentação, não oferecendo formas de adaptação para permitir diferentes questões de pesquisa. Desta forma, ao invés de investir em pesquisas contrapondo o aprendizado baseado em livros e aulas expositivas com o baseado em simuladores, a variação sistemática de características dos simuladores permite encontrar características que os tornam mais efetivos em situações de aprendizagem particulares. GRÖßLER *et al*. (1999) apresentam um projeto e implementação de um protótipo de um ambiente de aprendizado interativo que contém algumas características de um simulador favorável à experimentação.

Na condução de pesquisas de avaliação de ambientes de aprendizado, não apenas as características da ferramenta de simulação devem ser levadas em consideração, segundo GRÖßLER (2000), que enumera três fatores que influenciam nos estudos: o instrumento (ou seja, o simulador), o usuário e a situação do

aprendizado. Com base nesta estrutura, foi possível identificar algumas características problemáticas no estudo dos simuladores para cada um dos três fatores.

Das características da situação do aprendizado, destacam-se a necessidade de integração com outras formas de ensino e a duração da atividade, de forma a não cansar ou entediar os usuários, mas também permitir tempo para que situações complexas sejam compreendidas e gerenciadas.

Das características dos participantes, destacam-se a existência de várias informações relevantes das pessoas que são dificilmente controladas completamente, tais como conhecimentos prévios sobre o domínio, motivação, experiência e inteligência. Além disto, no uso de simulação, não é possível simular riscos reais e isto pode levar os participantes a terem uma postura mais audaciosa, assumindo mais riscos do que em uma situação real. As medidas resultantes da participação do usuário na simulação, geralmente, não contêm informações sobre o processo de executar a simulação, ou seja, a estratégia seguida pelo participante. Estas medidas são sujeitas a múltiplas interpretações e em problemas complexos é difícil encontrar uma solução ótima que seja tomada como base para comparação do desempenho dos participantes ao usar o simulador.

Dentre as características do instrumento usado, destacam-se a complexidade, a validade do modelo e o nível de abstração. Apesar dos simuladores serem baseados em modelos formais, os modelos nem sempre representam bem a realidade e a validade da simulação passa a depender também da validade do modelo. É difícil determinar o nível de detalhe e que informações são estritamente necessárias para a compreensão dos cenários. Simuladores muito complexos para serem compreendidos e utilizados levam os participantes a utilizar apenas estratégias de tentativa e erro.

Uma outra característica importante é a manipulação do tempo. A compressão e a expansão do tempo pode ser vista como uma das grandes vantagens da simulação e um fator influente no desempenho do usuário. No mundo real, o tempo é uma variável contínua e independente. Apesar da tomada de decisões em intervalos de tempo delimitados ser mais próxima da realidade, esta não é adequada para objetivos de aprendizado. O tempo deve permitir que os participantes possam analisar a situação e tomar decisões de forma a aprender sobre a estrutura dinâmica do domínio. Entretanto, é importante também avaliar os efeitos da pressão do tempo em simuladores sobre o desempenho dos participantes não só como uma forma de aprendizado mas também na avaliação de estratégias, ação e reação e tomada de decisões com base no tempo (GRÖßLER, 1999).

2.6. Conclusões

O gerenciamento de projetos de software pode ser um fator importante para o sucesso ou fracasso do projeto. Muitas vezes, o gerenciamento aplicado na indústria é deficiente e inadequado em função da origem técnica dos gerentes e da falta de compreensão e aplicação efetiva das principais práticas e técnicas do gerenciamento. Sendo uma atividade baseada em conhecimento, a experiência é determinante para as tomadas de decisão necessárias durante o curso do projeto, ao invés da simples aplicação de medidas corretivas aparentemente intuitivas que podem piorar ainda mais um projeto problemático. Entretanto, para adquirir experiência, os gerentes devem ter treinamento para atuarem como gerentes, além de vivenciar e falhar em vários projetos para aprender na prática como funciona a dinâmica dos processos de desenvolvimento de software.

O treinamento baseado em aulas expositivas e livros pode ser complementado com experiências práticas, análise das falhas e lições aprendidas. Componentes práticos como simuladores são uma alternativa versátil e segura em comparação ao uso de projetos piloto reais. O uso destas simulações, entretanto, não é uma questão bem documentada nem estabelecida, pois é muito difícil generalizar os resultados sobre a efetividade dos simuladores em comparação com as outras formas de ensino, sendo difícil medir a influência isolada do simulador no treinamento.

Algumas características encontradas nos jogos favorecem o suporte à simulação, por meio de interatividade, fantasia, desafio e curiosidade que influenciam a motivação e o engajamento dos participantes no treinamento. Desta forma, os jogos se aproximam dos princípios das teorias de aprendizado adulto e podem contribuir de forma mais eficiente na formação de gerentes com conhecimento e habilidades necessárias para guiar um projeto de desenvolvimento de software.

Capítulo

3

Modelos para projetos de desenvolvimento de software

Segundo ELGOOD (1988), um jogo de gerenciamento, na sua forma genérica, é um exercício de tomada de decisões sobre aspectos relacionados aos processos produtivos de uma empresa em que os jogadores são convocados para atuar no controle de determinadas questões do negócio. Estas decisões são geralmente quantitativas e, no domínio da gerência de projetos de software, podem incluir questões como quantas horas diárias um desenvolvedor irá trabalhar ou quanto tempo será investido nas atividades de controle de qualidade. Nestes jogos, o mundo e suas regras são descritos por um modelo, do qual os jogadores não têm conhecimento. O modelo descreve como as decisões tomadas pelos jogadores afetam o comportamento do sistema sob análise.

Neste capítulo, é discutido como pode ser criado um modelo para representar os projetos de desenvolvimento de software para que seja utilizado em simulações e em jogos. A simulação destes modelos pode atender a diversos objetivos, dentre os quais a aprendizagem e o treinamento. Uma das disciplinas de modelagem analisada é a Dinâmica de Sistemas, que procura atender às principais características dos processos de desenvolvimento de software. Posteriormente, é apresentado um conjunto de técnicas com adaptações para algumas das limitações da Dinâmica de Sistemas e são discutidas suas limitações para aplicação em jogos.

3.1. Modelos para sistemas de dinâmica complexa

Um modelo é uma representação simplificada, uma abstração de um sistema real ou conceitual que procura exibir características significativas (e não todos os aspectos) do sistema que se deseja estudar, prever, modificar ou controlar (KELLNER *et al*., 1999).

Os modelos podem ser explícitos ou mentais. Os modelos mentais, conforme visto no capítulo 2, são aqueles formados por nossas imagens do mundo, os relacionamentos entre suas partes e a influência de nossas decisões sobre eles (STERMAN, 1988). Os modelos mentais são comumente utilizados na tomada de decisões e podem, também, ser usados em jogos de gerenciamento não baseados em computador, em que os participantes são responsáveis por simular mentalmente as decisões recebidas e informar seus resultados para os demais jogadores. Os modelos explícitos são representações de sistemas por meio de uma linguagem formada por um conjunto de símbolos e conexões entre eles. De acordo com a linguagem de modelagem, os modelos podem apresentar diferentes graus de formalismo. Os modelos formais são formados por símbolos lógicos e matemáticos que permitem definir precisamente a semântica de cada elemento e conexão do modelo, além da realização de inferências sobre o comportamento do sistema. Entretanto, apenas os modelos mentais determinam a ação das pessoas. Os modelos explícitos, para serem efetivos, devem atuar sobre os modelos mentais de forma a modificá-los, aprimorando a capacidade do analista para a tomada de decisões (FORRESTER, 1991).

Segundo BARROS (2001), os modelos formais podem ser classificados como estáticos ou dinâmicos. Os modelos estáticos permanecem sempre no mesmo estado, independente da passagem do tempo. Nos modelos dinâmicos, ao contrário, o estado pode mudar com o tempo, por meio da alteração dos símbolos do modelo. A forma como ocorre a mudança de estado nos modelos dinâmicos permite classificá-los ainda como discretos ou contínuos. Nos modelos discretos, o tempo é discretizado, ou seja, as mudanças são resultado de eventos que ocorrem ao longo do tempo, em intervalos não constantes e que não podem ser previamente determinados, modificando os símbolos e criando novos eventos a serem processados no futuro. Nos modelos contínuos, os símbolos mudam em intervalos de tempo constantes, infinitesimais e previamente determinados.

Um outro fator de classificação para os modelos formais é a capacidade de representação de incerteza. De acordo com esta classificação, os modelos podem ser estocásticos ou determinísticos. Nos modelos determinísticos, não há expressão de incerteza, ou seja, eles apresentam sempre o mesmo comportamento em diferentes avaliações. Nos modelos estocásticos, há a expressão de incerteza, não sendo possível garantir os mesmos resultados para diferentes avaliações do comportamento do modelo.

Os modelos explícitos formais podem ser simulados computacionalmente. Simulações de modelos são utilizadas, geralmente, quando a complexidade do sistema é maior do que os modelos estáticos conseguem representar e quando não

existe uma solução analítica para a resolução do modelo. Desta forma, é possível fazer importantes observações sobre um sistema ou fenômeno dinâmico quando os custos, riscos ou a própria manipulação do sistema real são proibitivos.

No caso de projetos de software, a simulação computacional pode auxiliar a tomada de decisões. KELLNER *et* al. (1999) e BARROS (2001) destacam categorias de propósitos para a simulação de processos de desenvolvimento de software:

- **Planejamento, controle e gerenciamento estratégico** – Diversos fatores podem ser previstos, tais como esforço, custo, qualidade e cronograma, levando em conta, por exemplo, riscos e restrições de recursos. Gerentes podem comparar os resultados simulados de diversos cenários alternativos quando uma decisão se fizer necessária. Parâmetros importantes para o acompanhamento de um projeto podem ser previstos e comparados com valores monitorados, facilitando a determinação de ações corretivas, quando necessárias;
- **Análise e aprimoramento de processos** - Simulações podem permitir um melhor entendimento de diversos aspectos do processo de desenvolvimento de software e ajudar em decisões de aprimoramento, prevendo o impacto de mudanças no processo antes de colocá-las em prática;
- **Treinamento e Aprendizado** - Simulações podem ser úteis para prática e aprendizado de gerenciamento de projetos, permitindo que estudantes aprendam o impacto de suas decisões e erros comuns, modificando a interpretação das pessoas em função das expectativas iniciais sobre os resultados de determinadas ações.

Os benefícios da simulação de modelos explícitos para o desenvolvimento de software dependem, porém, da capacidade que as técnicas de modelagem oferecem para representar com fidelidade os aspectos desejados do mundo real, assim como a habilidade do modelador em capturar estes aspectos. Os projetos de software, principalmente os de larga escala, são de difícil modelagem. Eles são classificados como sistemas de dinâmica complexa e, geralmente, apresentam algumas das características abaixo (STERMAN, 1992):

- **São formados por componentes inter-relacionados** – Desta forma, uma mudança em uma parte do sistema pode causar implicações em outras partes remotas e com um certo atraso, ou seja, sistemas desta categoria apresentam relações de causa e efeito distantes no tempo e espaço;

- **São dinâmicos** – Muitas informações importantes, como produtividade e taxa de geração de erros, variam ao longo do tempo. Além disso, as respostas a curto e longo prazo do sistema a uma perturbação podem ser distintas. Aumentar a equipe em um projeto atrasado, por exemplo, pode ser uma decisão pior a curto prazo, mas não a longo prazo;
- **Apresentam ciclos de realimentação** – O resultado de uma decisão pode afetar as condições que motivaram a própria decisão, de forma a reforçar ou corrigir as condições que a exigiram. Por exemplo, acrescentar mais desenvolvedores em um projeto atrasado pode tornar o projeto ainda mais atrasado, uma vez que é necessário deslocar parte da equipe produtiva para o treinamento dos novos desenvolvedores, reduzindo a produtividade e atrasando a conclusão de tarefas;
- **Apresentam relacionamentos não-lineares** – Causas e efeitos nem sempre apresentam relações simples e proporcionais. Um aumento da carga diária de trabalho, por exemplo, não irá necessariamente representar um aumento proporcional de produtividade, em função de uma possível exaustão dos desenvolvedores e do aumento do número de erros que deverão ser corrigidos;
- **Apresentam aspectos qualitativos** - Um projeto de software não pode ser visto apenas em função dos componentes técnicos e suas relações, mas também deve considerar os aspectos sócio-culturais e qualitativos, como a existência de diversos patrocinadores, objetivos múltiplos e conflitantes, motivação da equipe e características da cultura da organização. Muitos destes aspectos não podem ser facilmente representados por números, mas podem influenciar no sucesso do projeto e não devem ser negligenciados.

Em um sistema de dinâmica complexa, conforme destaca STERMAN (1992), o comportamento não pode ser previsto eficientemente apenas por modelos mentais, mas sua análise pode se beneficiar de uma abordagem sistêmica, ou seja, a análise do comportamento do sistema a partir das relações entre seus componentes.

3.2. Dinâmica de Sistemas

A Dinâmica de Sistemas (FORRESTER, 1961) é uma disciplina de modelagem criada em 1960, no Massachusetts Institute of Technology, que é amplamente utilizada em diversas áreas do conhecimento humano como sistemas sociais, ecológicos, econômicos e de engenharia. Esta disciplina é uma abordagem para avaliação de problemas complexos que procura enfatizar seus aspectos estruturais na análise do

seu comportamento. Muitas das características de sistemas de dinâmica complexa podem ser expressas pela Dinâmica de Sistemas e não são facilmente descritas por outros tipos de modelagem, tais como as relações de causa e efeito distantes no tempo e os ciclos de realimentação. Em um modelo da Dinâmica de Sistemas, o comportamento visível de um sistema é determinado pela sua estrutura de componentes e suas conexões.

Segundo HOUSTON (1996), a Dinâmica de Sistemas se baseia em dois tipos de modelos: diagramas causais e diagramas de repositórios e fluxos. Os diagramas causais caracterizam um sistema em função dos seus relacionamentos de causa e efeito e seus ciclos de realimentação. Não é sempre necessário utilizar diagramas causais para descrever um sistema, mas o seu refinamento pode auxiliar na criação dos diagramas de repositórios e fluxos, refletindo a estrutura do sistema com mais detalhes.

Um diagrama causal utiliza caixas para representar os elementos do sistema. Entre os elementos há setas que representam os relacionamentos de causa e efeito de um elemento sobre o outro. As setas devem ser sinalizadas positivamente ou negativamente, indicando se a presença do elemento origem da seta, respectivamente, aumenta ou diminui a presença do elemento destino. Por meio destes relacionamentos, é possível observar ciclos de realimentação, que também podem ser positivos e negativos. Nos ciclos positivos, um aumento em um elemento provoca modificações em outros elementos, que irão aumentar a presença do primeiro. Nos ciclos negativos, o aumento de um elemento provoca modificações em outros elementos, que irão fazer com que o primeiro diminua.

A Figura 1 apresenta um trecho de um diagrama causal para o desenvolvimento de software retirado de (BARROS, 2001). Segundo o diagrama, o aumento da pressão no cronograma faz com que haja um aumento na taxa de trabalho diária da equipe, conseqüentemente aumentando sua produtividade. O aumento de produtividade faz com que a quantidade de tarefas pendentes diminua. Entretanto, o aumento da carga diária de trabalho pode fazer com que os desenvolvedores cansados realizem um trabalho de menor qualidade, implicando um aumento na taxa de geração de erros. O aumento do número de erros ocasiona um aumento da demanda de correção e esta demanda representa um aumento na quantidade de tarefas não concluídas. Conseqüentemente, novas tarefas de teste e correção se farão necessárias. Como podemos observar, o pequeno trecho de diagrama causal apresenta um ciclo de realimentação positiva (pressão do cronograma, taxa de trabalho diária, taxa de erros e tarefas não concluídas) e um ciclo de realimentação negativa (pressão do cronograma, taxa de trabalho diária, produtividade diária e tarefas não concluídas).

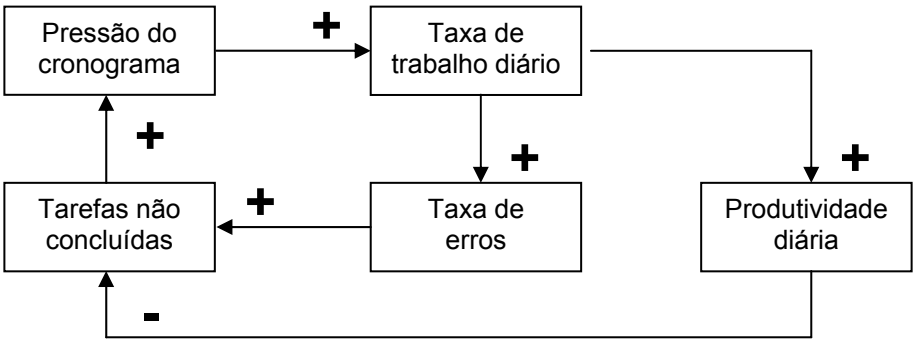

Figura 1 - Diagrama causal (BARROS, 2001)

Os diagramas de repositórios e fluxos são mais refinados do que os diagramas causais e podem ser interpretados matematicamente. Seus componentes são os repositórios, fluxos, processos e conectores. A Figura 2 ilustra um exemplo deste tipo de diagrama que representa, de maneira simplificada, aspectos da gerência de recursos humanos para projetos de desenvolvimento de software.

Repositórios são acumuladores que descrevem o estado do sistema em função de elementos que são consumidos ou acumulados ao longo do tempo. Eles são representados por caixas retangulares. Na Figura 2, há dois repositórios que representam a quantidade de desenvolvedores experientes e inexperientes no projeto. Estas quantidades podem variar de acordo com contratações, demissões e treinamento. Os fluxos são responsáveis pela definição de como ocorre a variação dos repositórios ao longo do tempo, por meio de equações matemáticas.

Os fluxos são representados por setas com válvulas e devem estar conectados a dois repositórios ou a um repositório e um produtor, ou consumidor, infinitos. O fluxo aumenta ou reduz o nível de um repositório a cada unidade de tempo. Os produtores e consumidores infinitos (ilustrados por nuvens) são utilizados para fornecer e absorver, respectivamente, unidades necessárias para a variação do nível de um repositório.

Desta forma, sistemas que se modificam ao longo do tempo podem ser representados por repositórios e fluxos. No exemplo da Figura 2, o fluxo de contratação é responsável pela taxa de aumento da quantidade de novos desenvolvedores inexperientes, enquanto o fluxo de demissão representa a taxa de diminuição de desenvolvedores experientes no projeto. O fluxo de treinamento indica a taxa com que os desenvolvedores inexperientes se transformam em desenvolvedores experientes.

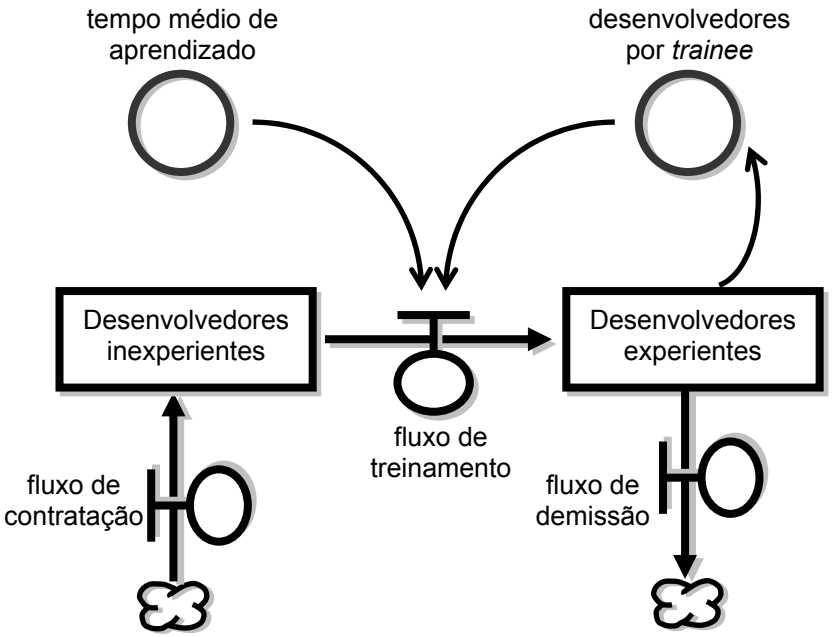

Figura 2 - Diagrama de repositórios e fluxos

Os demais construtores dos diagramas de repositórios e fluxos são os processos e conectores. Um processo é representado por um círculo e é utilizado para fatorar as equações que descrevem os fluxos. Desta forma, um processo recebe um conjunto de parâmetros (nível de um repositório, valor de um fluxo ou resultado de outro processo) para calcular uma informação que pode ser utilizada em outros processos ou fluxos. Os conectores, representados por setas, permitem a transmissão de informações entre os repositórios, processos e fluxos. O fluxo de treinamento no exemplo da Figura 2 se baseia em dois processos para calcular a variação com que os desenvolvedores se tornam experientes: o tempo médio de treinamento e o número de desenvolvedores por *trainee*. O número de desenvolvedores por *trainee*, por sua vez, é baseado na quantidade de desenvolvedores experientes existentes no projeto.

Os fluxos, portanto, representam as equações que são fundamentais para o diagrama. Quando elas se tornam muito complexas ou resultados parciais devem ser analisados, torna-se necessário o uso de processos. A relação entre fluxos e repositórios pode ser vista pela variação do nível de um repositório em um intervalo de tempo, que é igual ao somatório dos fluxos de entrada menos o somatório dos fluxos de saída do repositório.

ABDEL-HAMID e MADNICK (1991) utilizaram a Dinâmica de Sistemas para criar um modelo para projetos de desenvolvimento de software, que se tornou uma base para críticas e extensões por outros autores. A estrutura e o desenvolvimento do modelo envolveram três passos de captura de informação. Inicialmente, foram conduzidas entrevistas com gerentes de projetos de desenvolvimento de software. As informações coletadas nesta fase, junto à experiência dos próprios autores, formaram

a base para formulação do modelo. O segundo passo foi uma extensa revisão da literatura. Finalmente, o modelo foi exposto para críticas e revisões em um processo iterativo para verificar sua validade.

O modelo de ABDEL-HAMID e MADNICK (1991) é composto por quatro seções: planejamento, controle, gerenciamento de recursos humanos e produção de software. Na seção de planejamento, as estimativas iniciais do projeto (como tempo de término e tamanho desejado de equipe) são realizadas. Estas estimativas são revisadas posteriormente, uma vez que diversas variáveis influentes para decisões de planejamento podem mudar dinamicamente durante o curso do ciclo de vida do projeto. Na seção de controle, o progresso do projeto é comparado com o plano original, o que pode acarretar em mudanças no planejamento. A seção de gerenciamento de recursos humanos descreve a contratação, treinamento, assimilação e transferência de recursos humanos. Finalmente, a seção de produção de software modela o processo de desenvolvimento de software, incluindo as atividades de projeto, codificação e testes. A fase de definição de requisitos foi excluída, uma vez que, em muitos ambientes, ela não está totalmente sob controle do grupo de desenvolvimento de software.

3.3. Limitações e extensões da Dinâmica de Sistemas

Apesar de se adequarem a algumas das características da dinâmica complexa dos projetos de desenvolvimento de software, os modelos da Dinâmica de Sistemas apresentam algumas limitações para que sejam utilizados na gerência operacional de projetos de software. Segundo BARROS (2001), por serem formados puramente por equações matemáticas, torna-se difícil a identificação dos conceitos e relacionamentos do mundo real, fazendo com que a construção e compreensão dos modelos seja dificultada. Por outro lado, o conhecimento do domínio sendo modelado é genérico e reutilizável, devendo ser descrito de uma forma mais natural e próxima dos usuários do que apenas equações matemáticas.

A representação do conhecimento por meio de equações também inibe o detalhamento dos elementos de um modelo que pertencem a um mesmo conceito. Nos modelos de projeto de desenvolvimento de software, por exemplo, geralmente são considerados valores médios das propriedades de um desenvolvedor, classificando-o como experiente ou inexperiente. Desenvolvedores distintos, porém, têm experiência e produtividade diferentes e podem não estar igualmente motivados. No caso da quantificação separada das propriedades dos elementos, seria necessária pelo menos uma variável independente para cada atributo de cada elemento sendo

modelado, o que representaria um número muito grande de equações, principalmente em um modelo complexo.

Outras limitações dos modelos dizem respeito à representação de incertezas, a expansão e a integração dos modelos. Segundo BARROS (2001), em um modelo de projeto de software devem estar representadas incertezas sobre o valor assumido por um parâmetro (tempo de conclusão de uma tarefa, por exemplo) e sobre a ocorrência de eventos futuros. As incertezas acerca do valor de um parâmetro podem ser representadas pela utilização de distribuições de probabilidade. Para a representação de incertezas sobre eventos futuros, todos eles deveriam ser previstos e incluídos no modelo. Uma outra alternativa seria a representação dos eventos em modelos complementares, sem que fosse necessário alterar o modelo original para avaliar o impacto de um novo evento sobre o comportamento do modelo. Apesar da integração de modelos ser necessária para facilitar a representação de novo conhecimento sobre o modelo, nas ferramentas e modelos da Dinâmica de Sistemas para projetos de software não há auxílio automatizado para esta integração, sem exigir que as equações do modelo sejam compreendidas e alteradas manualmente.

Com base nas deficiências da Dinâmica de Sistemas para a construção dos modelos, representação, detalhamento do conhecimento e de incertezas, BARROS (2001) propõe algumas adaptações. O conjunto de técnicas que envolvem estas extensões torna possível modelar um projeto a partir do comportamento inicialmente esperado para ele. Os modelos complementares (cenários) propostos por BARROS (2001) podem ser integrados ao modelo do projeto original para representar diversos tipos de conhecimento gerencial e incertezas que podem ocorrer ao longo do desenvolvimento do projeto. Por meio de simulação, é possível avaliar os impactos de diversos eventos representados pelos cenários sobre o comportamento original do projeto.

A aplicação destas técnicas para o gerenciamento de projetos de software é conhecida como “gerenciamento de projetos baseado em cenários” (BARROS, 2001). O processo de modelagem é composto por três etapas: a modelagem de domínio, a instanciação de modelo e a compilação de modelo. A Figura 3 exibe o esquema deste paradigma.

Um modelo de domínio representa uma definição genérica de um domínio, como o desenvolvimento de projetos de software, criado a partir de um meta-modelo definido por BARROS (2001). O meta-modelo permite a construção de um modelo de domínio, que define as classes de elementos pertencentes ao domínio do problema, suas propriedades, seu comportamento e os relacionamentos que podem existir entre as instâncias destas classes. O comportamento de cada classe é especificado através de

construtores tradicionais da Dinâmica de Sistemas (repositórios, fluxos, processos e conectores).

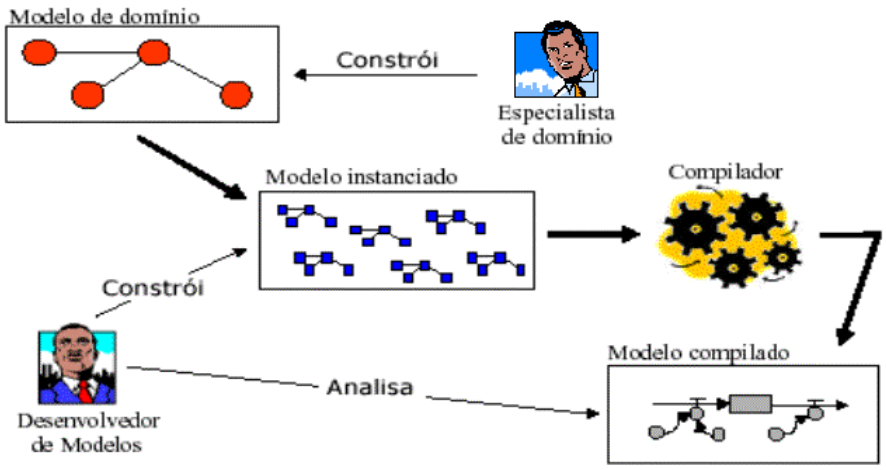

Figura 3 - Gerenciamento de projetos baseado em cenários (BARROS, 2001)

Para projetos de software, por exemplo, atividades e desenvolvedores podem ser representados por classes, cada qual com suas propriedades e seus comportamentos. Um desenvolvedor pode possuir propriedades como experiência e custo por hora. Uma atividade pode ter um comportamento que calcule, por exemplo, o seu custo e outro que determine se ela já foi concluída. O Quadro 1 ilustra um trecho simplificado de modelo de domínio para o gerenciamento de projetos de software.

```
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;
};
```

Quadro 1 - Exemplo de modelo de domínio

O modelo de domínio é abstrato, não pode ser simulado e apenas contém as definições dos elementos do sistema sob análise. Com base neste modelo, é possível definir um modelo operacional (instanciação), o modelo de projeto, em que são criadas

instâncias das classes definidas no modelo de domínio. Os valores das propriedades de cada instância são inicializados e os relacionamentos entre elas definidos. Por exemplo, uma instância de desenvolvedor chamado Jimmy pode ter uma hora de trabalho custando 20 dólares e ser pouco experiente, enquanto uma outra instância chamada Bobby pode ser bastante experiente e custar 30 dólares por hora. O Quadro 2 ilustra um modelo de projeto para o modelo de domínio do Quadro 1.

```
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;
};
```

Quadro 2 - Exemplo de modelo de projeto

O modelo instanciado é, então, traduzido para os construtores básicos da Dinâmica de Sistemas (compilação). O modelo compilado, enfim, pode ser simulado computacionalmente como qualquer modelo da Dinâmica de Sistemas.

Modelos complementares, os modelos de cenários, podem ser integrados ao modelo de projeto. O conhecimento representado em um modelo de cenário pode ser gerado de diversas formas, como através de experimentos, entrevistas, consenso e revisão da literatura (BARROS, 2001). O Quadro 3 apresenta um cenário que, se aplicado a um desenvolvedor, determina o efeito do tamanho da equipe do projeto sobre sua produtividade individual. As últimas linhas do modelo apresentado no Quadro 2 ativam este cenário sobre o desenvolvedor Jimmy, que terá sua produtividade afetada pelo tamanho da equipe, ao contrário do desenvolvedor Bobby.

Além da definição das técnicas que compõem o paradigma do gerenciamento de projetos baseado em cenários, BARROS (2001) também apresenta um modelo de projeto e modelos de cenários para projetos de desenvolvimento de software de média escala, abordando atividades de projeto, codificação, controle de qualidade e testes. Os dados para a construção dos modelos foram extraídos a partir do modelo de

projetos de desenvolvimento de software de ABDEL-HAMID e MADNICK (1991) e dos dados sobre produtividade e qualidade em projetos de software apresentados em (JONES, 2000).

```
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);
   };
};
```

Quadro 3 - Exemplo de modelo de cenário

3.3.1. Limitações da Dinâmica de Sistemas para os jogos

O uso de simulações de modelos baseados na Dinâmica de Sistemas mostra-se útil na representação e análise das dinâmicas complexas do comportamento de projetos de desenvolvimento de software. Considerando suas aplicações em treinamento, os estudantes podem rodar simulações predefinidas que expressam decisões de gerenciamento e são capazes de determinar os impactos das decisões tomadas sobre o projeto. Entretanto, para cada rodada de simulação, os parâmetros relacionados com as decisões devem ser ajustados inicialmente e os resultados são analisados ao final de todos os passos de simulação, sem a intervenção do estudante no processo. Além disso, pode ser difícil para o estudante interpretar os resultados sem a ajuda de algum supervisor que tenha conhecimento do modelo utilizado na simulação (BARROS *et al.*, 2001).

Como parte das características que beneficiam os aspectos motivacionais e o engajamento do participante no treinamento, os jogos demandam interatividade. Para atender a esta característica, a simulação de modelos da Dinâmica de Sistemas deve permitir que o modelo seja modificado pelo estudante ao longo do tempo, sem que os dados previamente calculados pela simulação sejam perdidos. A interpretação dos resultados também deve ser facilitada para refletir estados e eventos relevantes para expressar o comportamento do modelo durante a simulação, sem que o estudante tenha conhecimento do próprio modelo.

3.4. Conclusões

Os modelos explícitos são os principais componentes de uma simulação e um jogo baseado em simulação. Nos jogos, o modelo é responsável por representar o mundo real, recebendo as interações dos jogadores e produzindo respostas de acordo

com o conjunto de regras nele definidas. O desenvolvimento de projetos de software é uma área que pode ser modelada por meio de diversas técnicas e a simulação computacional destes modelos pode ser aplicada para o treinamento e aprendizado. Projetos de software muitas vezes apresentam dinâmica complexa, o que pode não ser bem representado por algumas técnicas de modelagem. A Dinâmica de Sistemas é uma disciplina de modelagem que beneficia uma visão sistêmica e atende aos problemas da modelagem de sistemas de dinâmica complexa, avaliando o comportamento do sistema a partir da estrutura.

O paradigma do gerenciamento de projetos baseado em cenários é uma adaptação da modelagem com a Dinâmica de Sistemas que procura atender a algumas das limitações desta para projetos de software. O conjunto de técnicas que compõem o paradigma permite a definição de modelos para projetos de desenvolvimento de software, que podem ser mais facilmente criados, modificados e expandidos. Estes modelos podem ser aplicados na construção de jogos, visando a representação do conhecimento relacionado com o gerenciamento de projetos de software para o treinamento e aprendizagem do domínio. Entretanto, algumas características inerentes aos jogos como a interatividade e a facilidade de interpretação de resultados não são encontradas em simulações de modelos de Dinâmica de Sistemas.

No próximo capítulo, apresentamos uma proposta para o treinamento em gerenciamento de projetos de software utilizando jogos com base em simulação de modelos da Dinâmica de Sistemas, construídos segundo o paradigma do gerenciamento de projetos baseado em cenários.

Capítulo

4

Aplicação de jogos de simulação no treinamento de gerentes de projetos de software

Conforme visto no capítulo 2, o gerenciamento de projetos é uma área em que é necessária a avaliação e escolha de alternativas durante o desenvolvimento de um projeto. O treinamento em gerenciamento de projetos por meio de um componente prático é importante para a exploração de alternativas, análise de erros e lições aprendidas, permitindo ao estudante vivenciar a dinâmica dos projetos de software sem os riscos da vida real.

Neste capítulo, são apresentados estudos experimentais que motivaram a adoção de jogos de simulação como instrumento para o treinamento com prática. Posteriormente, são propostos os principais componentes para a criação de um jogo de simulação (modelo, simulador e máquina de jogo) para treinamento em gerenciamento de projetos de software.

4.1. Estudos experimentais iniciais com técnicas de simulação

As técnicas de desenvolvimento, integração e simulação de modelos de projeto e cenários, de acordo com o paradigma do gerenciamento de projetos baseado em cenários e conforme visto no capítulo 3, são formas de facilitar a representação formal do conhecimento gerencial e a análise do impacto deste conhecimento sobre o comportamento de um projeto de software. Segundo BARROS (2001), as técnicas relacionadas com o gerenciamento baseado em cenários podem ser classificadas como técnicas de construção e de utilização. As técnicas de construção são voltadas para a criação de modelos de projetos de software e modelos de cenários, que representam o conhecimento gerencial. As técnicas de utilização são voltadas para

aplicação, por meio da integração do conhecimento gerencial com projetos na indústria para avaliação de seu comportamento.

A aplicação destas técnicas pode ser útil no suporte a tomada de decisões, planejamento e controle de um projeto, como também no treinamento de gerentes menos experientes face às incertezas inerentes do desenvolvimento de software. A avaliação destas técnicas, portanto, pode oferecer indícios dos benefícios e dificuldades envolvidos no uso das simulações e sua aplicação na construção de jogos para treinamento de gerentes de projetos.

De forma a avaliar a viabilidade e a continuidade do desenvolvimento das técnicas, dois estudos experimentais foram elaborados e executados. Os estudos se concentraram na avaliação da eficácia da aplicação dos modelos de cenários no gerenciamento de projetos, ao invés da construção de novos cenários. Esta eficácia é medida pelos ganhos e pelas dificuldades obtidas por meio da utilização das técnicas de integração e simulação de modelos de projeto e cenários.

O primeiro estudo (BARROS *et al*., 2002) foi realizado com 18 desenvolvedores de software em ambiente acadêmico (três alunos de doutorado, 14 alunos de mestrado e um aluno de graduação). Nove participantes aplicaram as técnicas propostas e nove outros participantes realizaram o estudo sem a aplicação das técnicas. Dentre os participantes, oito foram líderes em projetos industriais, três foram líderes em projetos acadêmicos e sete nunca foram líderes de projetos. Destes últimos, seis possuíam experiência em desenvolvimento de software em ambiente industrial ou acadêmico. O segundo estudo foi uma replicação do primeiro em um ambiente de cooperação entre universidade e indústria, contando com alunos da Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS) alocados a um laboratório de desenvolvimento de projetos para a indústria. Deste ambiente, fizeram parte do estudo 31 participantes (12 alunos de mestrado, 15 alunos de graduação e quatro graduados), sendo que seis indivíduos eram de áreas não relacionadas diretamente com computação. No total, nove participantes possuíam experiência como gerentes em projetos na indústria, sete como gerentes de projetos de curso ou de pesquisa e 18 não possuíam experiência como gerentes. Destes, 12 possuíam experiência no desenvolvimento de software em ambiente industrial ou acadêmico-industrial.

Os estudos visaram avaliar se os participantes que utilizaram as técnicas de integração e simulação de cenários obtiveram melhores resultados daqueles que apenas contaram com a sua experiência pessoal. Os participantes foram divididos em dois grupos. Um grupo utilizou as técnicas enquanto o outro grupo não (grupo de controle). O tempo de conclusão e o custo de desenvolvimento de um projeto de

software foram as medidas quantitativas levadas em consideração para avaliar o desempenho dos participantes. Além desta avaliação quantitativa sobre os ganhos obtidos pelo uso das técnicas, também foi feita uma avaliação qualitativa sobre a viabilidade e utilidade das técnicas, além das dificuldades encontradas na sua aplicação.

Durante a execução dos estudos, cada participante foi requisitado a gerenciar um projeto de desenvolvimento de software hipotético de pequeno porte. O sistema a ser desenvolvido é parte de um sistema de controle acadêmico, que gerencia informações sobre professores, alunos, áreas de pesquisa, disciplinas e inscrições em turmas. O sistema foi medido em aproximadamente 65 pontos de função ajustados. Os participantes do primeiro grupo foram treinados para a utilização das técnicas propostas e no uso do ambiente de simulação, enquanto o segundo grupo apenas recebeu uma descrição do projeto.

Dada a inviabilidade de trabalhar com projetos reais similares para cada participante, todos receberam um emulador, ou seja, um sistema que controla e apresenta graficamente o comportamento de um projeto, de acordo com as decisões tomadas pelo usuário, a medida que ele avança o tempo do projeto. Dentre as decisões que poderiam ser tomadas pelos participantes no emulador, estão a determinação dos desenvolvedores responsáveis pela execução das tarefas, o grau de esforço em tarefas de controle de qualidade e o número de horas diárias de trabalho dos desenvolvedores. Os participantes do primeiro grupo podiam testar suas decisões no ambiente de simulação antes de aplicá-las no emulador. Os participantes do segundo grupo deveriam recorrer apenas ao seu conhecimento tácito. Ao final do projeto, foram armazenados os valores obtidos para o tempo de conclusão e o custo do projeto.

Os resultados resumidos do primeiro estudo são apresentados na Tabela 1 e os do segundo estudo na Tabela 2. Os mecanismos utilizados na análise dos resultados foram o teste-T (WOHLIN *et al.*, 2000) com 95% de certeza e o teste de Mann-Whitney (WOHLIN *et al.*, 2000). Para o primeiro estudo, apesar do pequeno número de participantes, houve sucesso dos testes para a hipótese de que o tempo de conclusão do projeto dos participantes que utilizaram as técnicas de simulação seria menor do que os que não utilizaram as técnicas, embora os testes estatísticos tenham sido inconclusivos para a hipótese de que o custo do projeto dos participantes que utilizaram as técnicas seria menor do que os que não as utilizaram. No segundo estudo, os testes não foram conclusivos para o tempo de conclusão nem para o custo do projeto. Dados qualitativos nos levam a crer que dificuldades encontradas durante a

execução, e apresentadas na avaliação qualitativa do estudo, tiveram influência no resultado.

Tabela 1 - Resultados do primeiro estudo experimental

Uso das técnicas	Participantes	Custo médio	Tempo médio
Sem uso das técnicas	9	14.957,70 $	37,67 dias
Com uso das técnicas	9	14.537,98 $	31,33 dias

Tabela 2 - Resultados do segundo estudo experimental

Uso das técnicas	Participantes	Custo médio	Tempo médio
Sem uso das técnicas	15	17.032,73 $	27,73 dias
Com uso das técnicas	16	17.046,81 $	29,63 dias

A observação dos participantes durante a execução do estudo experimental se mostrou tão importante quanto os resultados obtidos através dos questionários e sistemas utilizados no estudo. Os principais problemas observados relacionaram-se ao treinamento e engajamento dos participantes para a execução dos estudos, o que refletia em dificuldades no uso das ferramentas e na interpretação dos resultados. Nos dois estudos, o tempo de treinamento nas ferramentas de simulação mostrou-se insuficiente, o que aumentou a necessidade de intervenção do organizador para resolução de dúvidas.

A partir dos dados obtidos nos estudos, foram levantadas hipóteses sobre a influência da facilidade de interpretação dos resultados, da formação e experiência dos participantes nos resultados. Os participantes que relataram maiores dificuldades de interpretação, em média, apresentaram pior desempenho (vide Tabela 3). De forma semelhante, os participantes de menor formação acadêmica e experiência em gerenciamento de projetos apresentaram piores desempenhos (vide Tabela 4 e Tabela 5).

Em geral, as técnicas de simulação são propostas para servir como mecanismos de apoio para gerentes menos experientes na compreensão e controle do comportamento de projetos ao longo do tempo (BARROS *et al.*, 2001). Entretanto, pelo observado, a obtenção de melhores resultados requer maior experiência e melhor formação. Diversos participantes conseguem executar com sucesso as simulações de modelos, mas não conseguem interpretar os resultados sem a ajuda de supervisores que os conheçam detalhadamente. Saber analisar eficientemente o modelo, acompanhando determinadas variáveis e eventos relevantes durante a execução da simulação sem a necessidade de supervisores mais experientes, é um passo valioso para a compreensão do comportamento do sistema.

Tabela 3 - Resultados segundo dificuldade de interpretação

Dificuldade de interpretação	Participantes	Custo médio	Tempo médio
Com dificuldade de interpretação	3	17.842,67 $	31,67 dias
Sem dificuldade de interpretação	13	16.863,15 $	29,15 dias

Tabela 4 - Resultados segundo formação dos participantes

Formação	Participantes	Custo médio	Tempo médio
Alunos de graduação e graduados	9	17.132,78 $	29,89 dias
Alunos de mestrado	7	16.936,29 $	29,29 dias

Tabela 5 - Resultados segundo experiência dos participantes

Experiência	Participantes	Custo médio	Tempo médio
Não gerentes, gerentes projetos de curso	11	17.637,09 $	30,36 dias
Gerentes na indústria	5	15.748,20 $	28,00 dias

Um outro fator observado na execução do segundo experimento, está relacionado ao engajamento dos participantes na realização do estudo. Quanto maior o tempo gasto na atividade, somado às dificuldades de uso e interpretação, menor o engajamento dos participantes pela atividade, não se tornando uma atividade intrinsecamente motivante. A falta de engajamento influencia negativamente o desempenho dos participantes, já que eles não atuam no uso das ferramentas e nas tomadas de decisão como se estivessem vivenciando o papel de gerente como o fariam no mundo real. O ambiente de execução do estudo também mostrou-se um fator importante para a aplicação das técnicas em treinamento, sendo necessários recursos tecnológicos (estações individuais de trabalho, ferramentas de apresentação, entre outros) e espaço físico que facilitem o treinamento nos instrumentos, o acompanhamento individual dos participantes e aumento da concentração de cada um na realização da atividade, sem que haja comparação e interferência de outros participantes ou pressão com o tempo transcorrido.

4.2. Proposta de um jogo de simulação

Conforme visto no capítulo 2, os jogos são ferramentas que parecem atuar beneficamente sobre algumas das limitações observadas no treinamento baseado em simulações. As principais características dos jogos, como interatividade, solução de problemas e *feedback* multimídia, parecem adequadas para uma melhor motivação e o engajamento dos estudantes, permitindo maior facilidade de utilização e interpretação durante a execução de simulações.

Desta forma, é proposto um jogo para treinamento de gerentes de projetos de software, fazendo com que o estudante atue como gerente, planejando e controlando projetos de software para que sejam finalizados com sucesso. No contexto do jogo

proposto, será considerado um projeto de sucesso aquele que foi desenvolvido dentro das estimativas previstas para ele pelo próprio gerente. Este sucesso pode ser medido quantitativamente pelo tempo de conclusão e o custo do projeto.

A interação do estudante com o jogo, tendo em vista o treinamento aplicado no gerenciamento de projetos de software, deve abordar algumas das principais etapas do gerenciamento de projetos vistas no capítulo, tais como as descritas por THAYER (1997): planejamento, gerência de recursos humanos, direção e controle. Neste sentido, o estudante tem um papel ativo fundamental no andamento do jogo, enriquecendo as lições aprendidas para cada etapa. Da mesma forma, a tomada de decisões sob pressão (passagem de tempo de forma contínua, limitando o tempo de raciocínio e a existência de recursos disponíveis limitados) e a não-linearidade permitem que diferentes jogadores vivenciem a experiência de gerenciar um mesmo projeto de formas diferentes.

4.2.1. Os personagens

Os personagens previstos no jogo devem representar, de forma simplificada, um ambiente de desenvolvimento de software. São eles:

- **O gerente** – O papel assumido pelo jogador, que recebe a descrição do projeto e é responsável pelo seu planejamento e execução, vivenciando diversas situações de tomada de decisões ao longo do desenvolvimento do projeto de forma a garantir que ele será completado dentro das estimativas previstas.
- **Os desenvolvedores** - Os personagens que compõem a equipe de desenvolvimento com que o gerente vai contar para o projeto. Cada desenvolvedor possui características pessoais que devem ser consideradas pelo gerente, tais como o custo por hora de trabalho e a experiência em cada tipo de tarefa a ser feita durante o projeto. Os desenvolvedores são responsáveis pela realização das tarefas. Eles podem ser incluídos ou retirados da equipe, ou ter sua carga diária de trabalho alterada ao longo do projeto, de acordo com as decisões do gerente.
- **O patrocinador** – O personagem a quem o gerente deve obedecer. O patrocinador é quem faz a aceitação do plano de projeto e as estimativas de cronograma e custos propostas pelo gerente.

4.3. Estrutura para um jogo de simulação

Segundo VERONESE et al. (2003) , geralmente, os jogos para treinamento são projetados para a estrutura de modelos específicos, dificultando a sua adequação a

diferentes modelos. Na indústria de jogos de entretenimento, por sua vez, há uma diversidade de ferramentas que auxiliam na geração de jogos (*game authoring tools* ou *game engines*), porém sem suporte adequado ao desenvolvimento de jogos baseados em modelos explícitos que ditem as regras do jogo. Na falta de um consenso sobre uma estrutura bem estabelecida para a criação de jogos de simulação aplicados ao treinamento, esta proposta destaca três componentes principais para a criação deste tipo de software: o modelo, o simulador e a máquina de jogo, conforme ilustrado na Figura 1.

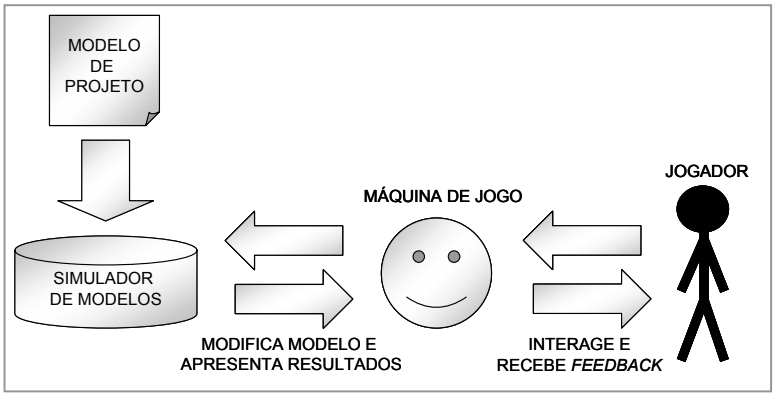

Figura 1 - Estrutura para criação de jogos de simulação

O modelo é o principal componente para o jogo de simulação. Ele representa o mundo e as características que serão simuladas e apresentadas ao jogador.

O simulador é o componente responsável por receber e traduzir um modelo de projeto e realizar a passagem de turnos, ou seja, o avanço do tempo e a simulação dos valores que descrevem os elementos do modelo.

A máquina de jogo, por sua vez, é responsável por manter a interação do jogador com a simulação, modificando o modelo em função das ações do jogador e apresentando os resultados da simulação a cada instante de tempo. Esta apresentação deve ser rica em recursos multimídia, valorizando a experiência cognitiva do jogador, sua motivação e engajamento ao treinamento.

4.3.1. O modelo

Um modelo utilizado para o jogo de simulação deve representar a dinâmica dos projetos de desenvolvimento de software e ser um componente flexível, ou seja, não deve ser intrinsecamente preso à estrutura do jogo, podendo ser facilmente trocado por diferentes modelos que capturem situações gerenciais distintas segundo as metas educacionais previstas.

Nesta proposta, a construção de um modelo de projeto é baseada no modelo de domínio para gerenciamento de projetos de software proposto pelo paradigma do

gerenciamento de projetos baseado em cenários, conforme visto no capítulo 3, por apresentar benefícios de construção, reutilização e integração de conhecimento gerencial.

4.3.2. O simulador

A característica mais importante de um simulador para seu uso no jogo é a capacidade de realizar simulação com estrutura dinâmica. Conforme indicado na seção 3.3.1, para a utilização de modelos descritos segundo as técnicas de construção e simulação do paradigma do gerenciamento de projetos baseado em cenários é necessária uma adaptação que permita a modificação do modelo durante sua simulação.

Na simulação de modelos baseados no meta-modelo da Dinâmica de Sistemas o comportamento visível do sistema, representado pelo modelo, é definido pela sua estrutura. A estrutura de um modelo é definida pelas instâncias existentes, seus relacionamentos e os valores de suas propriedades. Ao utilizar uma simulação, o gerente deve preparar um plano (inicializando instâncias, propriedades e relacionamentos) e segui-lo até o final, sem alterar a estrutura do sistema ao longo do tempo (simulação com estrutura estática).

Esta estrutura estática não reproduz fielmente a realidade, pois ao longo de um projeto de software, um gerente muitas vezes precisa tomar atitudes para controlar o andamento do projeto, alterando as relações entre tarefas, desenvolvedores e recursos. Pode ser muito comum, por exemplo, o aumento da pressão do cronograma do projeto a medida que a data de conclusão se aproxima. De forma a tentar manter o cronograma, o gerente pode requisitar um aumento do número de horas de trabalho diárias da equipe ou até contratar novos desenvolvedores. Estas decisões modificam valores de propriedades dos elementos componentes do modelo e alguns relacionamentos entre eles, afetando a estrutura do modelo e, conseqüentemente, o seu comportamento.

No segundo estudo experimental visto na seção anterior, o grupo de participantes que apenas utilizaram o emulador de projetos apresentou desempenho melhor do que os participantes que utilizaram as técnicas de integração e o ambiente de simulação. No uso do emulador, os participantes tinham a oportunidade de tomar decisões sobre o projeto durante seu curso. Os participantes que utilizaram as técnicas de simulação deveriam estudar e preparar um plano inicial para o projeto e aplicá-lo no emulador, porém não tinham como avaliar as incertezas e tomar decisões ao longo do desenvolvimento do projeto.

Portanto, a interatividade no uso de simulações, uma característica fundamental na construção de jogos, depende de flexibilidade para modificação da estrutura do modelo durante a execução da simulação, de acordo com as ações do usuário. Cada decisão que afete a estrutura do modelo pode ser representada como um evento externo que modifique as instâncias do modelo operacional do projeto, suas propriedades e seus relacionamentos, sem que os resultados obtidos anteriormente ao evento sejam perdidos.

No desenvolvimento de software, por exemplo, consideremos que em um determinado instante de tempo esteja sendo executada a tarefa de codificação e o gerente perceba que não irá conseguir cumprir o cronograma do projeto. Ele decide que deve contratar um desenvolvedor mais experiente em codificação e trabalhando horas-extras, de forma a aumentar a produtividade e conseguir terminar o projeto no prazo previsto. Estas decisões do gerente caracterizam três eventos sobre o modelo: a inclusão de um novo elemento (novo desenvolvedor) no modelo, a alteração de propriedade (horas de trabalho) do elemento e a alteração de um relacionamento (desenvolvedor alocado a tarefa de codificação). Uma vez interrompida a simulação e aplicados os eventos, o comportamento do modelo continua a ser simulado segundo a nova estrutura, a partir do ponto de interrupção e preservando todos os resultados obtidos previamente ao evento.

Os eventos podem ser classificados em eventos de criação e remoção de elementos, alteração de propriedades e relacionamentos e ativação e desativação de cenários. Cada tipo de evento é detalhado abaixo:

- Criação ou remoção de uma nova instância de uma classe, como, contratar ou despedir um desenvolvedor, por exemplo;
- Modificação de uma propriedade de uma instância de uma classe, como as horas de trabalho de um desenvolvedor;
- Adição ou remoção de um elemento em um relacionamento múltiplo, como alterar a lista de precedência de uma tarefa, por exemplo;
- Alteração de um elemento em um relacionamento simples, modificando, por exemplo, o desenvolvedor alocado a uma tarefa;
- Ativação ou desativação de um cenário sobre uma instância de uma classe, por exemplo, fazendo com que a produtividade de um desenvolvedor sofra influência da necessidade de comunicação em função do tamanho da equipe.

Para que durante a execução da simulação e de um jogo o modelo seja transparente ao usuário, a integração dos modelos de projeto e cenários, o tratamento

de eventos e a simulação das equações matemáticas do modelo devem ser feitas sem que o usuário tenha conhecimento das transformações e controles necessários. Desta forma, por meio da simulação de modelos com estrutura dinâmica, é possível a utilização dos modelos para projetos de desenvolvimento de software definidos pelo paradigma do gerenciamento de projetos baseado em cenários na construção de jogos de simulação, em que o jogador pode atuar interativamente modificando o modelo durante a execução do jogo.

4.3.3. A máquina de jogo

Conforme visto anteriormente, a máquina de jogo é o componente responsável pela interação do jogador com a simulação do modelo. Esta interação deve ocorrer de forma multimídia: elementos gráficos, sonoros e efeitos dramáticos que dão ao jogo uma história motivante, com objetivos e regras bem definidas.

A máquina de jogo pode permitir que sejam utilizadas diversas fases. Uma fase é considerada como a execução do jogo para um único modelo de projeto, segundo a estrutura da Figura 1. A utilização de múltiplas fases permite que várias situações e políticas gerenciais possam ser apresentadas sequencialmente ao jogador em diferentes configurações de projetos de software, tornando o jogo um instrumento mais flexível para treinamentos com metas distintas.

Para a execução de cada fase, a máquina de jogo é responsável pelo processamento de um conjunto de etapas funcionais pelas quais o jogador deve passar. Estas etapas descrevem o início do projeto, a criação de um plano de projeto, sua aceitação pelos patrocinadores, a execução e o controle da rede de tarefas até que o projeto chegue ao seu fim. Estas etapas estão ilustradas na Figura 2 e são apresentadas em detalhes nas seções a seguir.

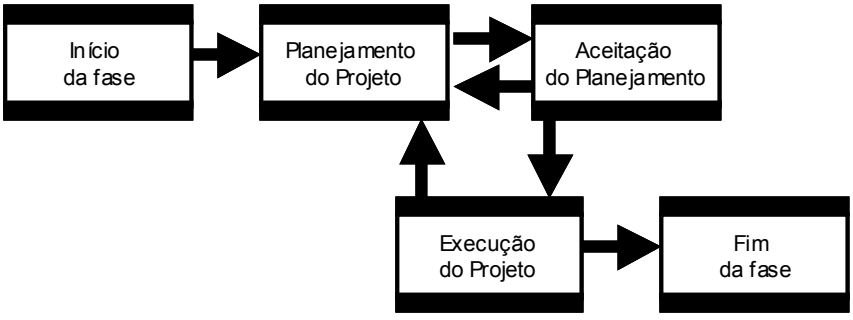

Figura 2 - As etapas de uma fase do jogo

4.3.3.1. Início da fase

O início de uma fase apresenta o projeto que será gerenciado pelo jogador. Esta apresentação ocorre por meio de um documento com a descrição do produto a ser desenvolvido e as características do projeto: suas tarefas e o número de pontos por

função de cada uma delas, demandas de qualidade para o produto (por exemplo, os testes só serão encerrados quando 95% dos erros forem corrigidos) e demandas de cronograma e recursos disponíveis para o gerente utilizar. O documento também descreve os cenários específicos que serão encontrados para este projeto, permitindo que várias situações gerenciais possam ser treinadas em diversas fases. A Figura 3 apresenta um resumo desta etapa.

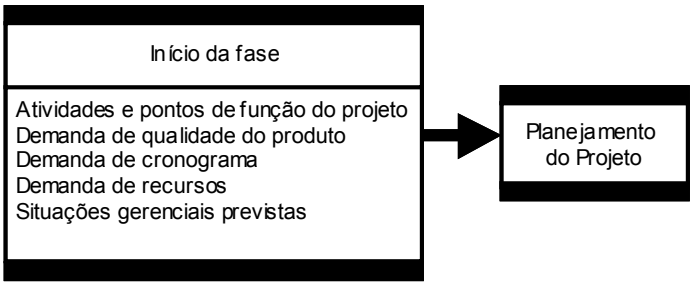

Figura 3 – O início da fase

4.3.3.2. Planejamento do projeto

Nesta etapa, o jogador deverá produzir o plano de projeto que será executado durante o desenvolvimento do produto. Na composição do plano, ele deverá analisar as tarefas que serão realizadas, seu seqüenciamento, o grau de esforço em tarefas de controle de qualidade e a equipe de trabalho alocada. O plano de projeto é concluído com as estimativas de cronograma e custos para o projeto.

O jogador deve ser capaz de analisar a experiência de cada desenvolvedor, retirá-los da equipe, modificar sua carga de horas de trabalho diárias e contratar novos desenvolvedores. As estimativas para o projeto devem ser calculadas e apresentadas dinamicamente à medida em que o jogador modifica o plano de projeto. A Figura 4 apresenta o resumo da etapa de planejamento.

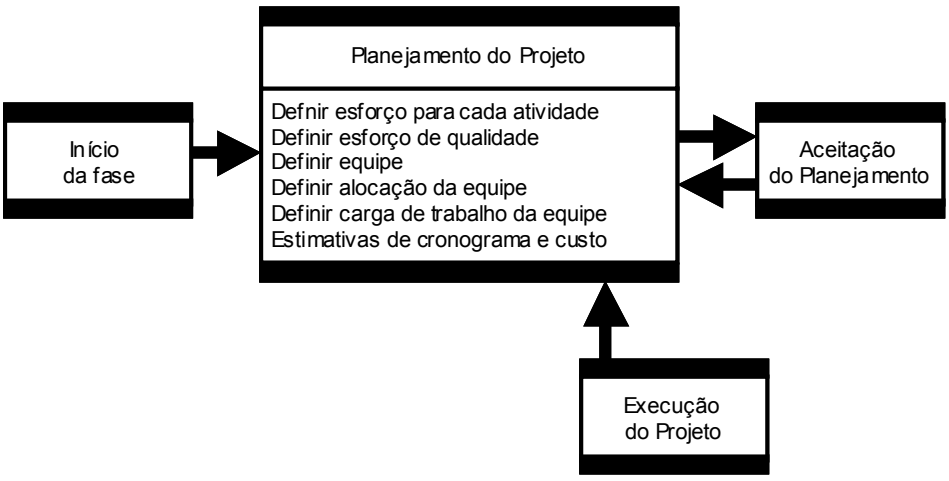

Figura 4 - A etapa de planejamento

As tarefas do projeto devem estar relacionadas entre si por meio de listas de precedências, ou seja, uma tarefa só pode ser executada se todas as que forem suas precedentes tiverem sido concluídas. Desta forma, a rede de tarefas deverá seguir a ordem de análise, projeto, codificação e testes. As tarefas de inspeção, quando houver, serão realizadas após o projeto e antes da codificação[1].

As tarefas de controle de qualidade, ou seja, inspeções, podem ou não fazer parte da rede de tarefas realizadas no contexto do projeto, de acordo com a decisão do gerente. O grau de esforço nas tarefas de inspeção e nas de testes é importante para que sejam cumpridas as demandas de qualidade exigidas na descrição do projeto, porém, ao mesmo tempo, pode comprometer as estimativas de custos e cronograma.

Tendo as tarefas definidas e seqüenciadas, o jogador deve apresentar uma estimativa para o cronograma e para o custo do projeto. Para isto, é preciso definir a equipe de trabalho e o tempo de execução para cada tarefa. A definição da equipe consiste em alocar um desenvolvedor para cada tarefa, sendo que não pode haver mais de um desenvolvedor alocado em uma mesma tarefa[2]. O jogador também deve poder modificar dados sobre a equipe, alterando a carga diária de trabalho de um desenvolvedor. A carga normal de trabalho é de oito horas diárias, podendo ser de até 12 horas. A estimativa de cronograma deve ser feita com base nos tempos das tarefas e a estimativa de custo em função das horas de trabalho realizadas e do custo por hora de trabalho de cada desenvolvedor alocado a uma tarefa.

4.3.3.3. Aceitação do planejamento

O plano criado pelo jogador poderá ser aceito ou recusado pelo patrocinador do projeto. O plano será recusado caso haja alguma tarefa da rede com duração ou desenvolvedor responsável indefinidos. O plano também é recusado se as estimativas ficarem além de uma determinada margem (como, por exemplo, 20%[3]) sobre os valores determinados nas demandas descritas na apresentação do projeto. Ao ter o plano recusado, o jogador deve refazê-lo.

4.3.3.4. Execução do projeto

Esta etapa está resumida na Figura 5. Com o plano preparado e aceito, é iniciada a execução da rede de tarefas do projeto. O tempo e o custo disponível para o gerente serão iniciados com os valores estimados e aceitos no plano de projeto. Estes valores diminuem durante a execução do projeto, representando o consumo de tempo

[1] Outros tipos de inspeção, além da inspeção de projeto, elas não serão considerados no jogo.
[2] A restrição de um desenvolvedor por tarefa se deve à falta de dados quantitativos na literatura sobre produtividade, geração e correção de erros na realização de uma tarefa por mais de um desenvolvedor.
[3] Esta margem foi arbitrada no modelo, mas pode ser facilmente alterada em outras aplicações do jogo.

e recursos para a construção. A passagem de tempo deve ser automática, ou seja, um dia transcorrerá em um determinado intervalo de tempo, que pode ser configurado pelo jogador.

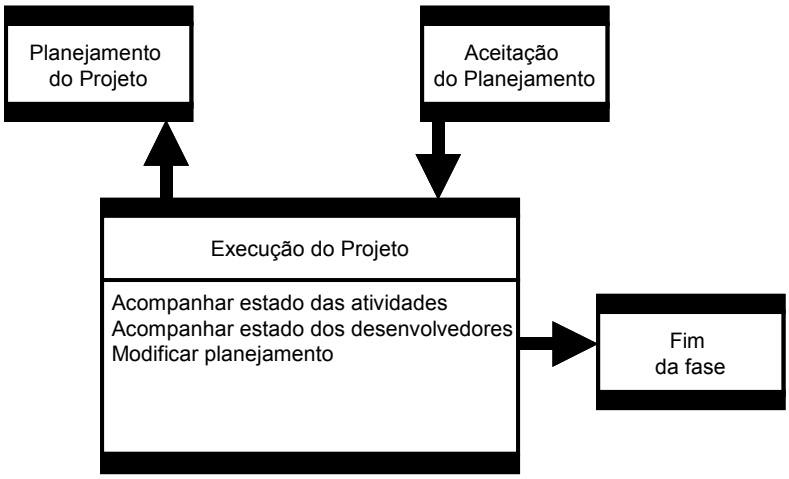

Figura 5 - A etapa de execução

Nesta etapa, o jogador deve se preocupar em analisar se os objetivos estão sendo atingidos, monitorando, medindo o progresso e tomando ações corretivas que forem necessárias, modificando, desta forma, o seu plano de projeto original. A monitoração deve ser feita por meio do acompanhamento das transições dos estados dos desenvolvedores, das tarefas e do projeto sendo executado. As transições de estados devem ser indicadas por envio de relatórios de projeto informando ao jogador algum acontecimento ou situação importante (por exemplo, a percepção de que o projeto está atrasado ou a notificação de que uma das tarefas começou a ser executada). Uma outra forma de indicar as transições de estados é por meio de mudanças na apresentação gráfica do jogo e dos personagens que permitam ao jogador perceber e compreender o desempenho do seu plano de projeto e ações tomadas por ele. A Figura 6 apresenta os estados para o projeto, as tarefas e os desenvolvedores.

Para refazer o plano de projeto, o jogador deve poder modificar as estimativas de tempo das tarefas e dados da equipe (horas de trabalho diárias), alterar o desenvolvedor responsável por uma tarefa, contratar e retirar um desenvolvedor da equipe. O plano modificado deve ser novamente submetido a aceitação pelo patrocinador.

O gerente deve estar atento ao comportamento do projeto para que suas estimativas não sejam ultrapassadas e ele perca a fase. O acompanhamento do projeto depende do próprio gerente. Ele deve estar atento para observar, por exemplo, que se o trabalho aumenta e a data de conclusão permanece a mesma, aumenta a pressão no cronograma.

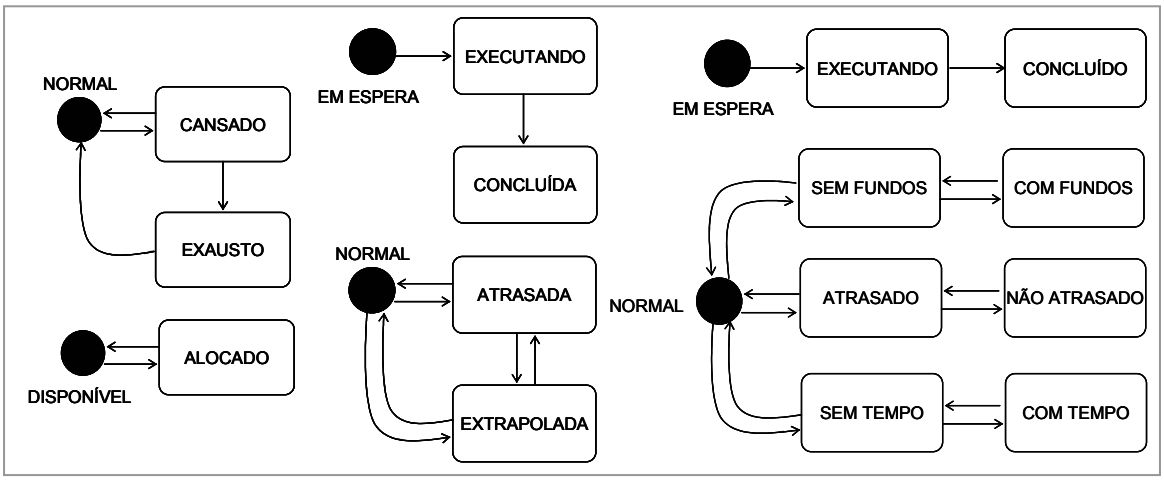

Figura 6 - Estados do desenvolvedor, tarefa e projeto, respectivamente na vertical

4.3.3.5. Fim da fase

O jogador chega ao final de uma fase com sucesso caso todas as tarefas do projeto tenham sido executadas dentro das estimativas de recursos planejadas. No caso dos recursos terminarem antes do final da execução de toda a rede de tarefas, o jogador chega ao final da fase com fracasso. Para evitar isto, ele deve estar atento ao andamento do projeto e refazer o plano caso perceba que os recursos inicialmente requisitados serão insuficientes. A Figura 7 resume o fim da fase.

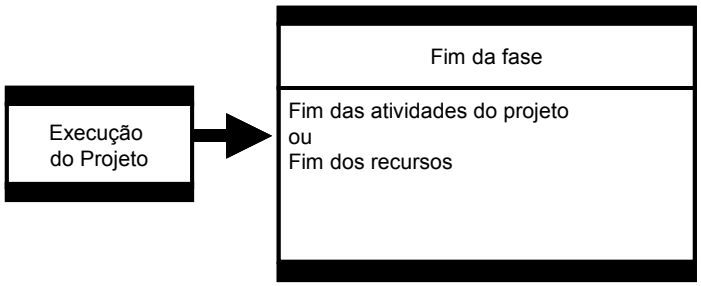

Figura 7 - O fim da fase

4.4. As lições para o jogador

Durante as duas principais etapas de cada fase do jogo (planejamento e execução), o jogador pode vivenciar diversas situações em que suas decisões devem estar baseadas em cuidadosas análises de custo-benefício.

A produtividade da equipe depende da carga horária diária de trabalho, do tempo de permanência no projeto, da experiência dos desenvolvedores para as tarefas a que estão alocados e também dos custos de comunicação em função do tamanho da própria equipe. Desenvolvedores mais baratos podem ser menos experientes, o que irá atrasar o cronograma. Porém, ao optar por aqueles mais hábeis, é preciso avaliar o impacto no custo. Quanto mais tempo um membro da equipe faz parte do projeto,

melhor sua produtividade. Portanto, a rotatividade e a multiplicidade de pessoas pode ser um fator benéfico, mas também custoso. Os desenvolvedores podem ser requisitados a trabalharem além das oito horas diárias. Ao trabalhar mais, entretanto, mais erros podem ser introduzidos nos artefatos sendo produzidos, em razão do cansaço da equipe. Caso os desenvolvedores cheguem ao estado de exaustão, eles irão se recusar a trabalhar além das oito horas diárias até que se estejam recuperados.

A meta de qualidade é um aspecto importante para o sucesso do projeto. Ignorar tarefas de inspeção pode ser uma falha para atingir a meta de qualidade, sobrecarregando as tarefas de testes, que podem tornar-se mais longas do que o esperado. A qualidade também é um fator que depende da geração de erros ao longo do desenvolvimento do projeto. Quanto menos erros forem introduzidos, menor o esforço para corrigi-los. A geração de erros ocorre naturalmente, mas depende também da experiência dos desenvolvedores que constroem os artefatos ao longo da rede de tarefas e do cansaço da equipe.

O gerente deve analisar as vantagens e desvantagens de cada decisão, avaliando o impacto que isso pode causar no desenvolvimento do projeto e nas suas estimativas de cronograma e custos. Desta forma, o jogo ocorre de forma não-linear, ou seja, oferece escolhas a serem feitas pelo jogador que influenciam o rumo do próprio jogo. Não existe uma linha reta e sim diferentes caminhos que podem ser traçados do início até o fim do jogo. Quanto maior a não-linearidade, as experiências dos jogadores tornam-se mais únicas, uma vez que cada um dos desafios encontrados pelo gerente para a tomada de decisões tem múltiplas soluções.

4.5. Conclusões

Neste capítulo foi apresentada uma proposta de um jogo para treinamento em gerenciamento de projetos. Os estudos experimentais sobre as técnicas de integração e simulação de modelos de projeto e cenário do paradigma do gerenciamento de projetos baseado em cenários ressaltaram a importância da interpretação dos resultados obtidos durante uma simulação e da motivação dos participantes para o treinamento.

No caminho do desenvolvimento de um jogo de simulação, três componentes foram destacados: o modelo, o simulador e a máquina de jogo. O simulador proposto deve permitir interatividade. Desta forma, foi proposta a simulação dos modelos com estrutura dinâmica, permitindo que o jogador atue como gerente de um projeto de software alterando as relações entre os desenvolvedores, as tarefas e as

características destes elementos, mesmo após o início da execução da rede de tarefas que descreve o projeto. Por fim, foi apresentada a máquina de jogo, o componente que regula a interação do jogador com o simulador e apresenta a história do jogo.

Em comparação com a maioria das propostas de ferramentas de simulação descritas no capítulo 2, podemos destacar a flexibilidade na criação e utilização de modelos, a interatividade e a apresentação multimídia do jogo de simulação proposto neste trabalho como os aspectos diferenciais e importantes para a motivação, engajamento dos estudantes e, conseqüentemente, para a eficiência do treinamento. Oh e van der Hoek (2002), por sua vez, apresentam a proposta que mais se aproxima da proposta deste trabalho (SimSE), embora o desenvolvimento ainda não tenha sido concluído com a criação da ferramenta com recursos multimídia. Podemos destacar a ferramenta de explicação do SimSE (que oferece ao jogador, durante o decorrer do jogo, rastros e indicações claras de regras e relações causais a partir das suas ações) como um requisito benéfico ao treinamento que não foi considerado na proposta deste trabalho. A flexibilidade, facilidade de criação e integração de modelos e cenários gerenciais não presos à máquina de jogo, entretanto, é um traço importante não encontrado nas propostas de jogos de simulação para treinamento analisadas.

No próximo capítulo, é apresentada a implementação de um jogo para que o jogador vivencie um gerente de projetos de software segundo as propostas apresentadas neste capítulo.

Capítulo

5

The Incredible Manager

No capítulo anterior, foi apresentada a proposta de um jogo de simulação para treinamento de gerentes de projetos de software. Com base nos três principais componentes apresentados nesta proposta – modelo, simulador e máquina de jogo –, neste capítulo é discutida a construção de um jogo de simulação chamado *The Incredible Manager*.

5.1. O modelo

Conforme descrito no capítulo anterior, o jogo de simulação proposto apresenta diversas fases, sendo cada uma a representação de um modelo de desenvolvimento de projeto de software. Os modelos que constituem estas fases são criados segundo o meta-modelo da Dinâmica de Sistemas proposto por BARROS (2001). No protótipo implementado, há apenas uma fase, ou seja, um modelo de projeto.

O modelo representa o desenvolvimento de um sistema de controle acadêmico para uma universidade e foi baseado no sistema CtrlPESC, desenvolvido para o Programa de Engenharia de Sistemas e Computação da Coordenação dos Programas de Pós-Graduação em Engenharia da Universidade Federal do Rio de Janeiro (COPPE/UFRJ). O sistema CtrlPESC foi avaliado em aproximadamente 160 pontos de função ajustados e suas funcionalidades incluem 8 cadastros e consultas sobre linhas de pesquisa, professores, alunos, administradores do sistema, disciplinas, turmas, inscrições em disciplinas e regras personalizadas para consultas de acompanhamento acadêmico. O sistema tem três tipos de usuários: administradores, professores e alunos. Os administradores têm acesso a todas as funcionalidades e dados. Os professores e alunos têm acesso limitado aos seus dados e aos dados genéricos. Os professores e alunos podem visualizar seus dados pessoais de cadastro, listas de professores, linhas de pesquisa, disciplinas e turmas oferecidas. Aos professores também é permitido efetuar modificações em seus dados pessoais, visualizar alunos

inscritos nas turmas em que é responsável e lançar notas. Aos alunos é também permitido efetuar modificações em seus dados pessoais, realizar e cancelar inscrições em disciplinas durante um determinado período.

As tarefas para o desenvolvimento do sistema CtrlPESC são organizadas em 8 ramos (um para cada área de funcionalidade), sendo que cada ramo é composto por 4 tarefas essenciais: análise, projeto, codificação e testes. Uma quinta tarefa (inspeção) existe para cada ramo entre o projeto e a codificação, mas sua realização é opcional.

Na construção do modelo foram levados em conta um modelo de projeto e todos os cenários previamente criados por BARROS (2001), somados a dois novos cenários desenvolvidos a partir do trabalho de ABDEL-HAMID e MADNICK (1991): *bad fixes* e *communication overhead*. O primeiro cenário caracteriza a geração de novos erros durante as tarefas de depuração e correção de erros. O segundo cenário caracteriza a influência do tamanho da equipe alocada ao projeto na produtividade individual de cada membro da equipe, em função de maior necessidade de comunicação. Os novos cenários e a listagem completa do modelo utilizado no protótipo do sistema CtrlPESC podem ser encontrados no apêndice A.

Cada modelo deve estar associado a um arquivo texto que contém uma descrição detalhada do projeto de desenvolvimento que ele representa, as peculiaridades que o jogador deverá encontrar e os cenários previstos. A configuração do conjunto de fases que será utilizado no jogo é feita por meio de um arquivo em formato XML. A Figura 1 exibe o conteúdo deste arquivo para o protótipo implementado. O conjunto de fases é identificado pela marcação `<fases>`. Cada fase é identificada pela marcação `<fase>` e contém dois atributos: `modelo`, que indica o nome do modelo, e `descricao`, que indica o arquivo texto com a descrição da fase em questão.

```
- <fases>
    <fase modelo="fase1.msd" descricao="fase1.txt" />
    <fase modelo="fase2.msd" descricao="fase2.txt" />
  </fases>
```

Figura 1 - Arquivo de configuração de fases

5.2. O simulador

No capítulo 4 foi discutida a necessidade de um simulador com capacidade de simulação com estrutura dinâmica, permitindo a interação do jogador com o modelo durante o jogo. O simulador utilizado pelo protótipo foi implementado como uma união de duas ferramentas previamente desenvolvidas por BARROS (2001) para o paradigma do gerenciamento baseado em cenários: o compilador e o simulador de

modelos da Dinâmica de Sistemas. O compilador recebe modelos escritos segundo o meta-modelo da Dinâmica de Sistemas e o traduz para os construtores tradicionais da linguagem. O simulador é capaz de simular computacionalmente os modelos baseados nos construtores tradicionais da Dinâmica de Sistemas.

A Figura 2 mostra os passos que representam o pseudo-código para a simulação. Após a compilação do modelo, é preciso ordenar topologicamente as equações, de forma a respeitar as dependências entre elas quando forem avaliadas uma a uma pelo simulador. Cada equação se torna uma variável do modelo. Equações redundantes são removidas e os valores das equações que representam os repositórios são inicializados. A cada passo de simulação executado, o simulador calcula e guarda o valor de cada equação do modelo. Todos os valores de cada instante de simulação podem ser consultados pelo nome da equação, que é único no modelo.

```
1. Leitura dos modelos de projeto e cenários
      a. Verificação semântica dos modelos
2. Compilação do modelo
      a. Tradução da estrutura baseada no meta-modelo
      b. Geração das equações da Dinâmica de Sistemas tradicional
      c. Otimização do modelo (equações redundantes)
3. Preparação para simulação
      a. Ordenação topológica das equações
      b. Inicialização de repositórios
4. Passo de simulação (executado repetidas vezes)
      a. Resolver cada equação
      b. Guardar o valor de cada equação
5. Disponibilizar valores para das equações de interesse
```

Figura 2 - Pseudo-código para a simulação

Conforme discutido no capítulo 3, as duas ferramentas, porém, não permitem realizar modificações no modelo durante a simulação. Desta forma, foi feita uma união das duas ferramentas e uma adaptação para permitir a ocorrência e processamento dos eventos descritos no capítulo 4 e que caracterizam a simulação com estrutura dinâmica. A nova ferramenta foi escrita em linguagem C e empacotada como uma biblioteca de vínculo dinâmico.

A Figura 3 mostra os passos executados para o tratamento dos eventos apresentados na seção 4.2.1, que devem ocorrer durante as chamadas para o passo de simulação. Como os eventos representam decisões de alto nível de abstração tomadas pelo jogador e não diretamente sobre as equações matemáticas, a modificação na estrutura deve ser feita no nível do meta-modelo da Dinâmica de Sistemas. Isto ocasionará a recompilação do modelo e regeneração de todas as equações. Algumas equações novas podem ser criadas e equações antigas podem

desaparecer, mas os valores já assumidos por elas nos passos de simulação anteriores devem ser mantidos. Quando a simulação recomeçar, no cálculo dos valores de cada equação são levados em conta os valores obtidos no passo anterior.

Os serviços oferecidos por este componente integrado permitem carregar, compilar e gerar código (em construtores tradicionais da Dinâmica de Sistemas) de um modelo, executar passos de simulação, acesso aos valores obtidos pela simulação e acesso a mensagens de erro. O componente oferece também métodos de acesso aos dados do modelo (cenários, classes, propriedades, relacionamentos e instâncias de classes), métodos para o processamento de eventos sobre instâncias de classes.

```
1. Modificar a estrutura do modelo de acordo com o evento
2. Liberar as equações existentes
      a. Não perder os resultados já simulados
3. Recompilação do modelo
      a. Criar novas equações
      b. Otimização do modelo
4. Preparar novamente para simulação
      a. Reordenação topológica das equações
```

Figura 3 - Pseudo-código para a ocorrência de eventos

5.3. A máquina de jogo

A máquina de jogo foi implementada na linguagem JAVA (SUN MICROSYSTEMS, 2003a) e acessa os serviços do componente integrado de compilação e simulação de modelos por meio de chamadas nativas, segundo a API JNI (SUN MICROSYSTEMS, 2003b). Para a manipulação dos desenhos dos personagens, foi utilizada a API GAGE2D (DNSALIAS, 2003).

O projeto de construção da máquina de jogo foi baseado no padrão MVC (*Model-View-Controller*), descrito em (GAMMA *et al.*, 1994). Segundo este padrão, classes de controle (*Controller*) manipulam classes de modelo (*Model*) e são responsáveis pelo recebimento e tratamento de eventos oriundos das classes de apresentação (*View*). Estas apenas apresentam e capturam dados das classes de modelo. Desta forma, a estrutura de controle da máquina de jogo apresenta um controle central (`ControleJogo`) e dois subcontroles (`ControleExecucao` e `ControlePlanejamento`).

O controle do jogo é responsável pelos eventos do ciclo de vida do jogo, enquanto cada subcontrole é responsável pelos eventos da etapa a que estão associados. Desta forma, o controle de planejamento é responsável pelos eventos de mudanças e submissão do plano de projeto, enquanto o controle de execução é responsável pela realização dos passos de simulação, apresentação da animação e

pelos eventos gerados pelo jogador durante a execução. A Figura 4 apresenta as classes de controle.

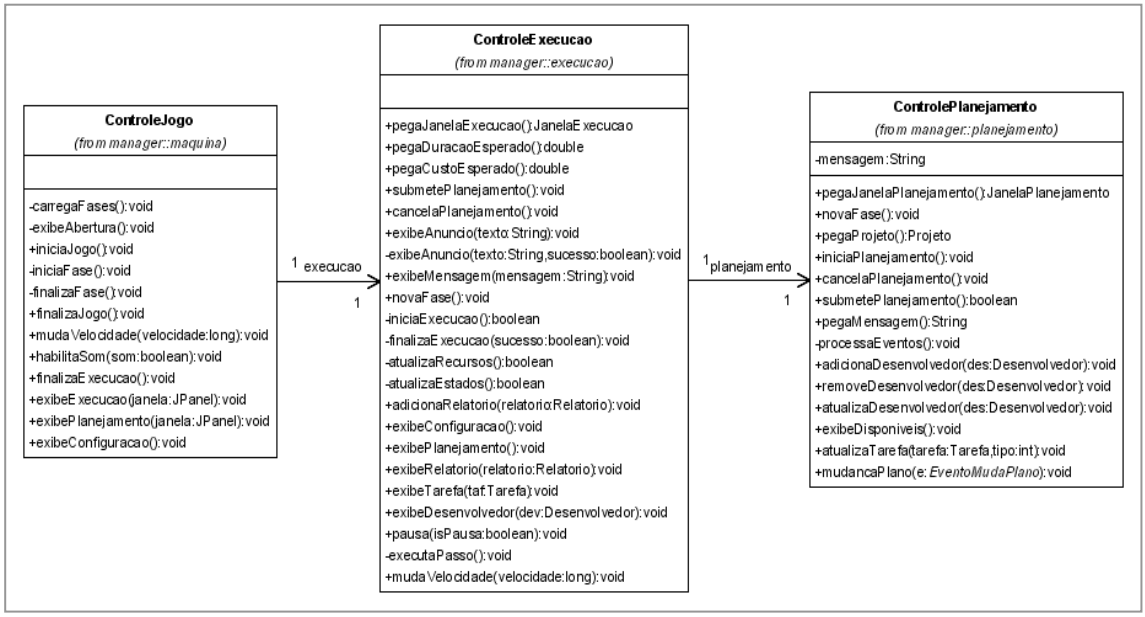

Figura 4 - Diagrama de classes de controle

O controle do jogo é responsável pela janela de abertura (`JanelaAbertura`) do jogo, pela janela de diálogo de configuração (`JanelaConfiguracao`) e pela janela de jogo. A janela de jogo (`JanelaJogo`) é utilizada pelo controle do jogo para apresentar a tela apropriada às etapas da fase (planejamento ou execução). O diagrama do controle do jogo é apresentado na Figura 5. As demais classes de apresentação estão subordinadas ao controle apropriado à etapa da fase a que pertencem (planejamento ou execução), para que seus eventos sejam corretamente tratados.

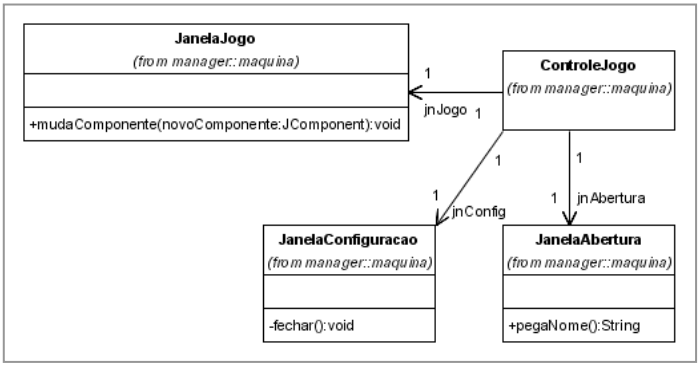

Figura 5 - Diagrama de controle do jogo

Durante a etapa de planejamento, o controle do jogo exibe a tela de planejamento (`JanelaPlanejamento`) na janela de jogo. A tela de planejamento é composta por dois painéis de apresentação e edição de tarefas e desenvolvedores (`PainelEquipe` e `PainelTarefas`). O controle de planejamento é ilustrado pela Figura 6.

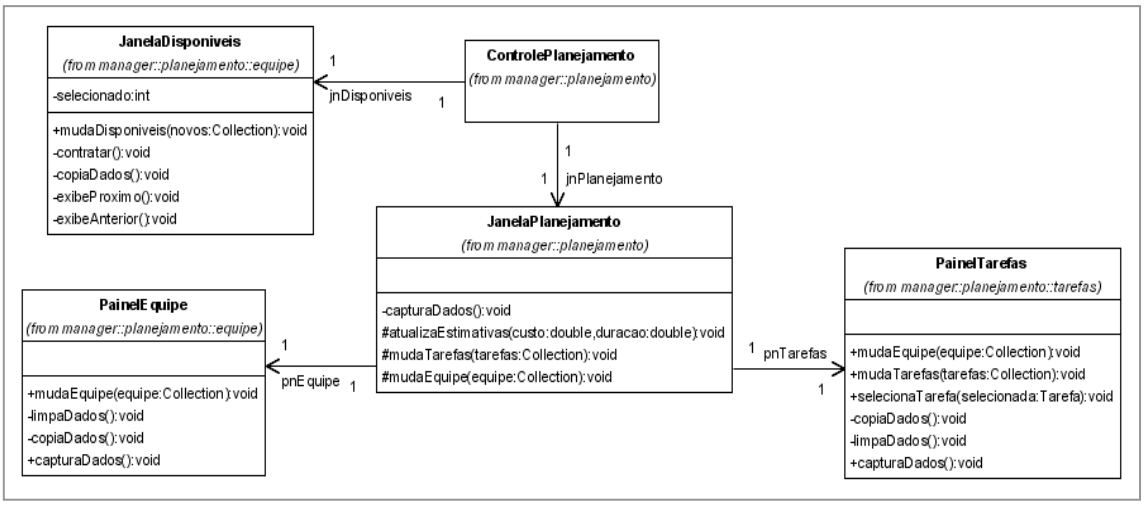

Figura 6 - Diagrama de controle de planejamento

Na etapa de execução, o controle do jogo exibe a tela de planejamento (`JanelaExecucao`) na janela de jogo. A tela de execução é composta por um painel animado (`PainelAnimado`) em que são apresentados os personagens (`Personagem`). Um personagem é a representação gráfica de um desenvolvedor. O diagrama da Figura 7 ilustra o controle de execução.

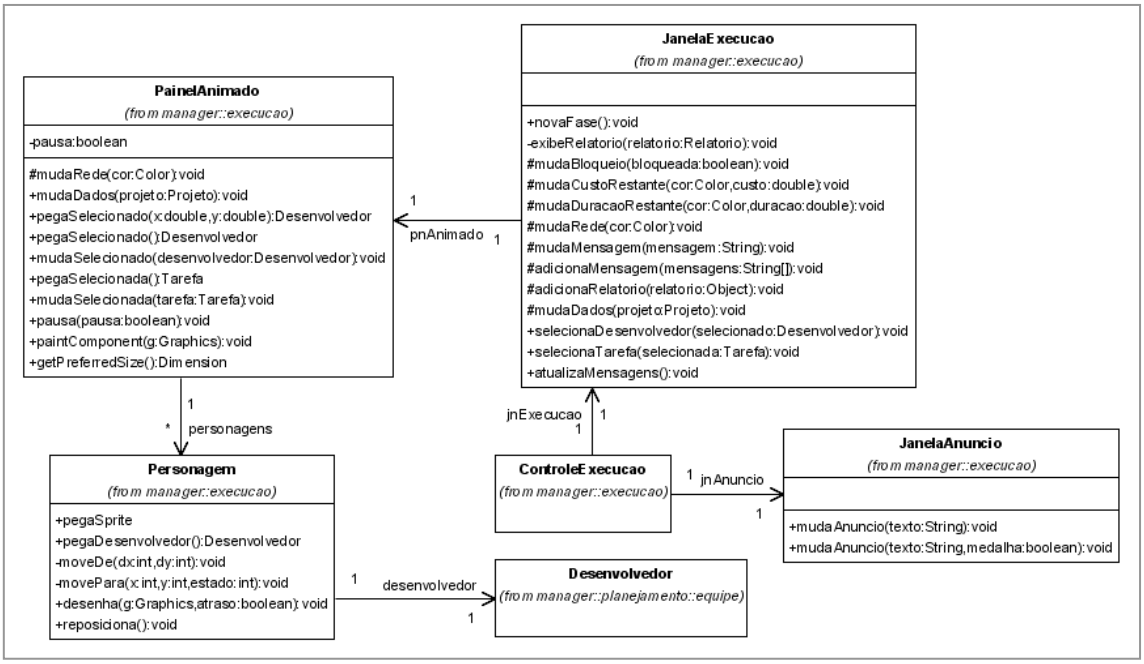

Figura 7 - Diagrama de controle de execução

As três principais classes de modelo (`Projeto`, `Desenvolvedor` e `Tarefa`) estão subordinadas ao controle de planejamento, assim como os eventos de mudança sobre o plano de projeto (`EventoMudaPlano`). A Figura 8 apresenta o diagrama de classes de domínio.

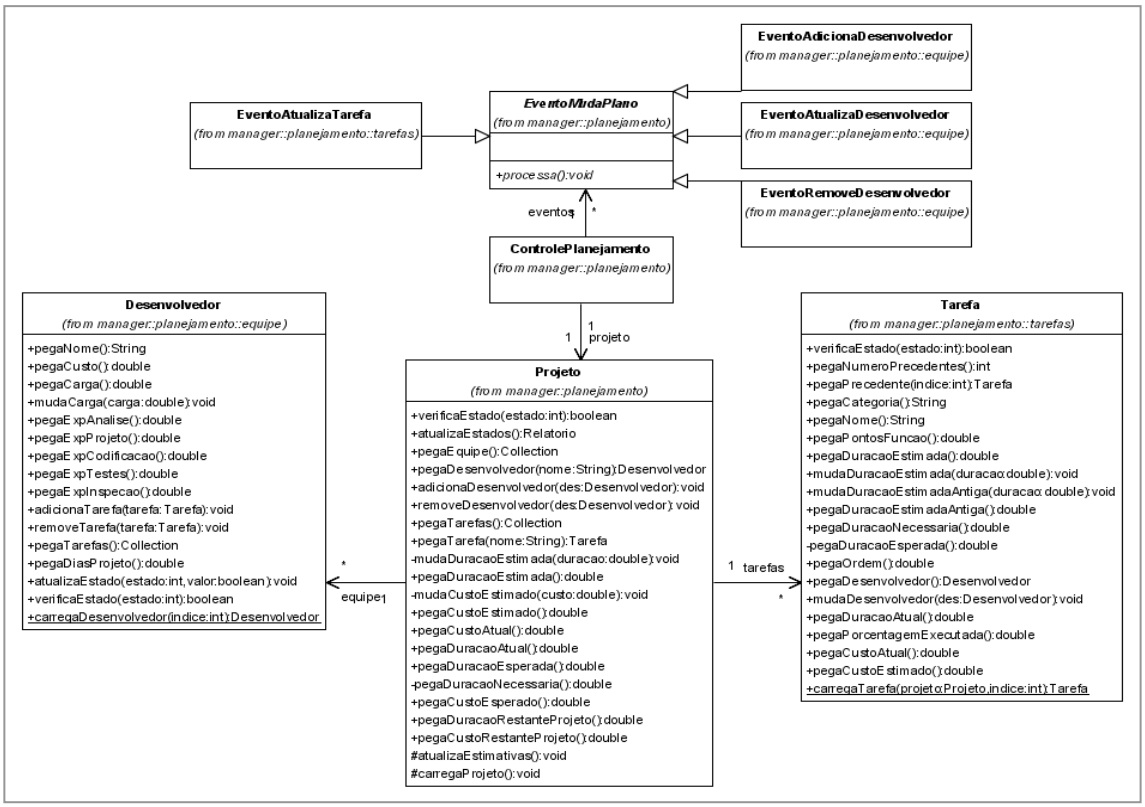

Figura 8 - Diagrama de classes de domínio

5.3.1. Início do jogo

Na tela de abertura, ilustrada pela Figura 9, o jogador encontra três opções: começar um novo jogo (*new game*), alterar configurações (*game options*) ou sair do jogo (*exit the game*). O jogador também pode entrar com seu nome, que será usado para a geração do arquivo com o seu histórico de ações realizadas durante a execução do jogo. Este arquivo pode ser utilizado posteriormente para discussões após o jogo em um contexto de treinamento

Figura 9 - Tela de abertura

Ao selecionar a opção para começar um novo jogo, o arquivo de configuração de fases é carregado e cada fase começará a ser executada seqüencialmente. O início de uma fase apresenta a tela de execução do jogo com uma janela de diálogo (ilustrada na Figura 10) contendo a descrição da fase.

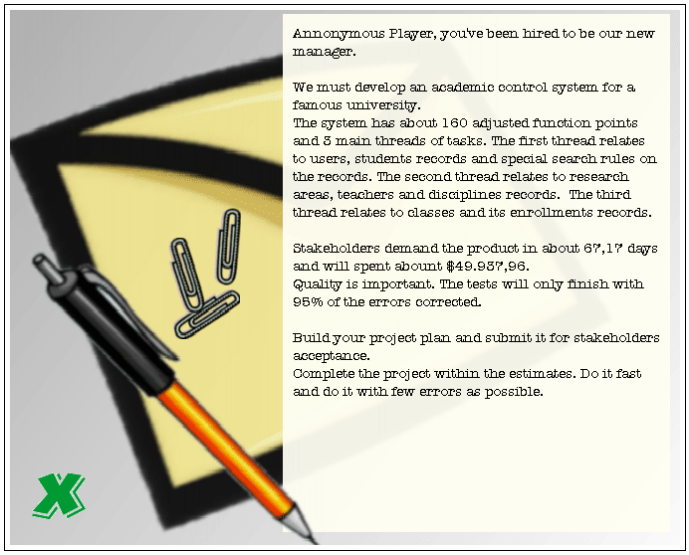

Figura 10 - Janela de diálogo de início de fase

5.3.2. Planejamento e aceitação do projeto

Após o fechamento da janela de diálogo de início de fase, o jogador é direcionado para a tela de planejamento, em que deverá compor o plano do projeto. A tela de planejamento (ilustrada na Figura 11) representa um caderno de anotações para o gerente e tem duas visões: uma visão da equipe e uma visão da rede de tarefas. A esquerda, são exibidos e atualizados constantemente os valores estimados para duração e custo do projeto (*total time* e *total cost*) segundo as escolhas do jogador na criação do plano.

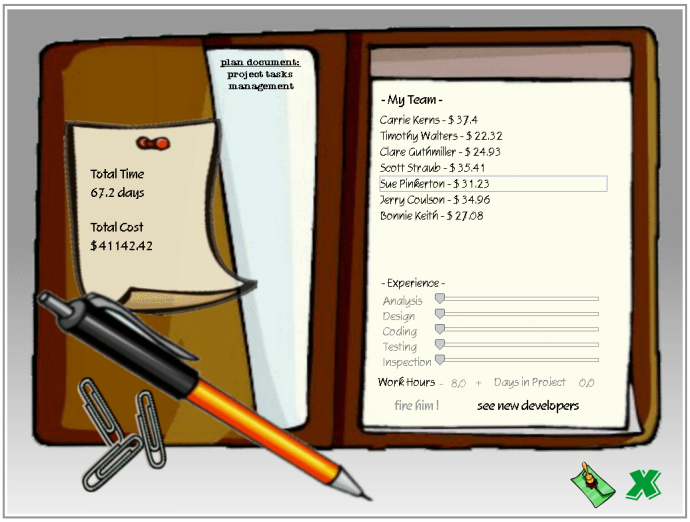

Figura 11 - Visão de equipe na tela de planejamento

Na visão de equipe da tela de planejamento, apresentada pela Figura 11, o jogador encontra uma lista de desenvolvedores (inicialmente vazia) que fazem parte do projeto. Cada desenvolvedor é apresentado pelo seu nome e seu custo por hora. Na parte inferior, são exibidos detalhes sobre um desenvolvedor selecionado na lista: sua experiência para cada tipo de tarefa, o número de horas de trabalho diárias (que pode ser alterado para valores entre 8 a 12 horas) e o número de dias em que o desenvolvedor está participando do projeto. O jogador também pode demitir o desenvolvedor selecionado da lista por meio do botão de demissão (*fire him*) e visualizar desenvolvedores para contratação por meio do botão de visualização de desenvolvedores disponíveis (*see new developers*). Este botão abre uma janela de diálogo, ilustrada na Figura 12, permitindo ao jogador navegar por diversos currículos (experiência nos tipos de tarefas e custo por hora) e contratar algum desenvolvedor de interesse. O custo por hora de cada desenvolvedor é calculado em função da sua experiência em cada tipo de tarefa.

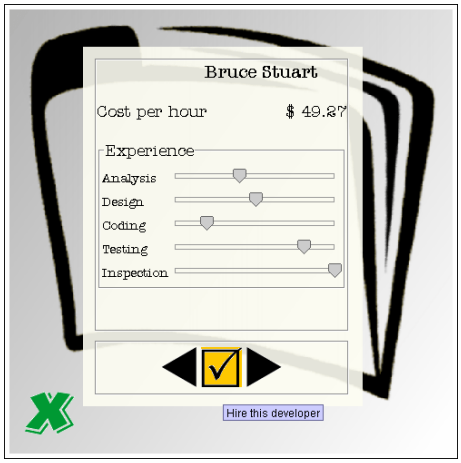

Figura 12 - Janela de diálogo de desenvolvedores disponíveis

Ao selecionar o botão de gerenciamento de tarefas (*project tasks management*), na parte superior da visão de equipe, a tela de planejamento exibe a visão de tarefas, apresentada na Figura 13. Uma vez na visão de tarefas, para retornar à visão de equipes, o jogador deverá selecionar o botão de gerenciamento de equipe (*project team management*), no mesmo local. Na visão de tarefas, o jogador encontra a rede de tarefas do projeto e o andamento da execução de cada uma delas. A parte inferior exibe detalhes de uma tarefa selecionada na rede: o nome, categoria (análise, projeto inicial, projeto detalhado, inspeção, codificação ou testes), pontos de função, duração estimada em dias (que deve ser editada pelo jogador), duração real da tarefa até o momento e o desenvolvedor responsável (que também deve ser definido pelo jogador). Nenhuma tarefa (fora inspeções) pode ficar sem desenvolvedor ou com duração igual a zero para que o plano de projeto seja aceito. Caso o jogador não

queira realizar as tarefas de inspeção, não precisa preencher suas durações ou deixá-las com zero dias de duração. Caso o jogador opte por realizar inspeções, deve definir uma duração estimada e associar um desenvolvedor para executá-las.

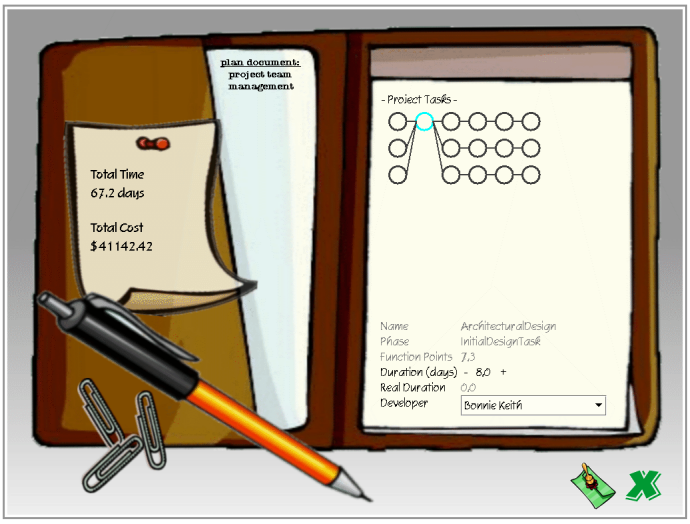

Figura 13 - Visão de tarefas na tela de planejamento

Uma vez tendo preenchido o plano, o jogador deverá submetê-lo para aceitação, clicando no botão em forma de carta na parte inferior da tela de planejamento. Se o plano tiver sido preenchido de forma incompleta, ou se as estimativas de duração e custos ficaram acima do desejado pelos patrocinadores, o plano será recusado e o jogador deverá modificá-lo até que ele seja aceito.

5.3.3. Execução do projeto

O plano aceito é imediatamente posto em execução, com o fechamento da tela de planejamento e a exibição da tela de execução do projeto. Esta tela, apresentada na Figura 14, é composta por uma barra de controle inferior e um painel animado.

Na parte central da barra de controle, há três botões: parar momentaneamente o jogo (*pause*), abrir o diálogo de configurações (botão em forma de engrenagem) e abrir a tela de planejamento do projeto (botão em forma de caderno). A direita, há um espaço para texto em que são exibidos dados de um desenvolvedor ou de uma tarefa selecionada pelo jogador. A esquerda, são contados de forma decrescente os dias e fundos financeiros que restam ao gerente para a execução do projeto. Estes valores são calculados pela diferença entre as estimativas de custos e cronograma previstas no plano de projeto criado pelo jogador e os dias e fundos financeiros consumidos e acumulados a cada turno.

O painel animado, na parte central da tela de execução, ilustra um ambiente de escritório em que são representados o andamento da rede de tarefas (na forma de um quadro branco na parede) e os desenvolvedores alocados ao projeto com suas mesas

de trabalho. Durante a execução do projeto, o jogador deve se concentrar nas mudanças que ocorrem na apresentação do painel animado. Estas mudanças ilustram algumas transições de estados que indicam se o projeto está com problemas e podem levar o jogador a terminar a fase com fracasso.

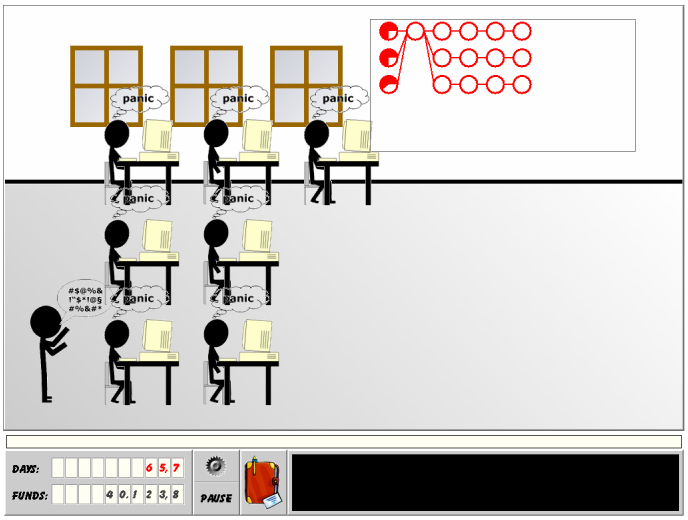

Figura 14 - Tela de execução

As mudanças gráficas das estimativas, das tarefas e da rede de tarefas em função da transição de estados são exibidas por meio de mudanças de tonalidades. No caso do projeto apresentar falta de tempo ou falta de fundos financeiros (ou seja, o tempo ou os fundos restantes são menores do que o necessário), os dígitos dos recursos restantes na barra de controle ganharão tonalidade vermelha. No caso de alguma tarefa estar atrasada, ou ter extrapolado o tempo estimado, ela será preenchida com tonalidade vermelha. Caso contrário, será preenchida com tonalidade verde. No caso de todo o projeto estar atrasado, todas as linhas de contorno da rede de tarefas são desenhadas com tonalidade vermelha.

As mudanças gráficas dos personagens são variações nos desenhos dos desenvolvedores e balões de pensamento. Cada desenvolvedor transita pelo escritório, indo, vindo e sentando na sua mesa de trabalho, durante o decorrer de um dia. Os balões de pensamento indicam se ele está ocioso, cansado ou em pânico. Um desenvolvedor fica em pânico caso o projeto esteja atrasado, sem fundos, sem tempo ou a tarefa que ele está executando esteja atrasada ou tenha extrapolado o tempo estimado. Um personagem especial, o patrocinador, aparece quando o projeto está atrasado, sem fundos e sem tempo. A Figura 15 apresenta os estados dos personagens: a figura do patrocinador com seu balão de ofensas, o desenvolvedor sentado ocioso, em pânico-cansado, o desenvolvedor em pé normal, em pânico e cansado.

Uma outra forma de indicar as transições de estados do modelo é pela emissão de relatórios. Um relatório é emitido quando um desenvolvedor começa ou termina uma tarefa, percebe que a tarefa que ele executa está atrasada, percebe que a tarefa extrapolou o tempo estimado, e quando ele se sente cansado por trabalho excessivo. Caso os patrocinadores percebam que o projeto está atrasado, sem fundos ou sem tempo hábil para conclusão, também são emitidos relatórios para alertar o gerente. Os relatórios são representados por esferas azuis, acumulados em uma lista horizontal acima da barra de controle da tela de execução. Para abrir um relatório, o jogador deverá efetuar um duplo clique sobre uma esfera, abrindo uma janela de diálogo que exibe o conteúdo do relatório, conforme apresenta a Figura 16.

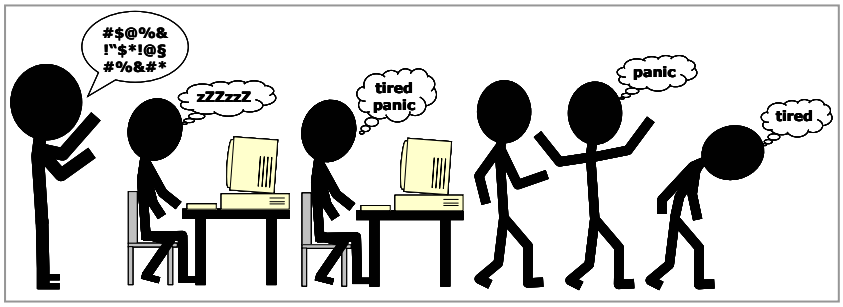

Figura 15 - Estados dos personagens

Durante a execução do projeto, é possível modificar o plano ao clicar no botão de planejamento na barra de controle da tela de execução. O jogador pode modificar sua equipe (contratando, demitindo desenvolvedores ou alterando as horas de trabalho), modificar o desenvolvedor alocado e modificar a duração estimada de uma tarefa. Não é permitido, porém, modificar tarefas já concluídas ou estimar uma duração menor do que já foi executada até o momento do replanejamento. O novo plano deve ser novamente submetido para aceitação e reinicia-se a execução.

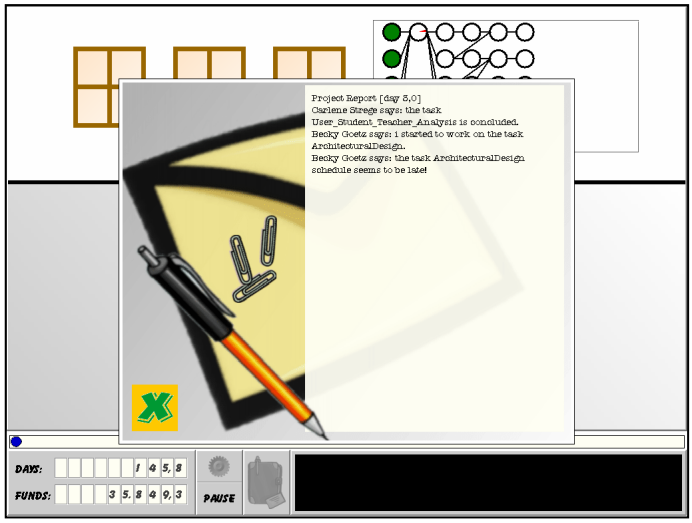

Figura 16 - Janela de diálogo de relatório

A janela de diálogo de configurações, exibida pelo clique no botão de configuração na parte central da barra de controle da tela de execução, permite ao jogador sair do jogo (*exit the game*), habilitar ou desabilitar efeitos sonoros (*enable sound*) e alterar a velocidade com que os turnos de execução são realizados (*game speed*), ou seja, o tempo entre os passos de simulação. Cada turno equivale a um décimo de dia e, inicialmente, ocorre a cada 2 segundos. A Figura 17 apresenta a janela de diálogo de configurações.

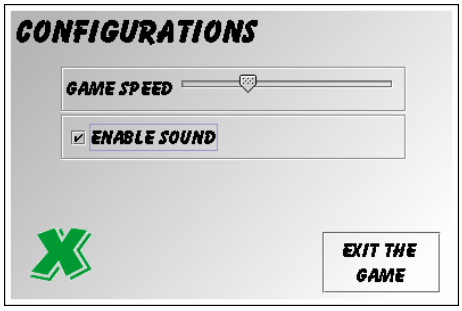

Figura 17 - Janela de diálogo de configuração

5.3.4. Fim da fase

Uma fase chega ao seu fim caso todas as tarefas previstas sejam concluídas dentro das estimativas de cronograma e fundos financeiros previstos no plano do projeto. Caso algum recurso (tempo ou fundos) acabe antes da conclusão da rede de tarefas, o jogador chegou ao fim da fase sem sucesso. A Figura 18 apresenta a janela de diálogo de fim de fase.

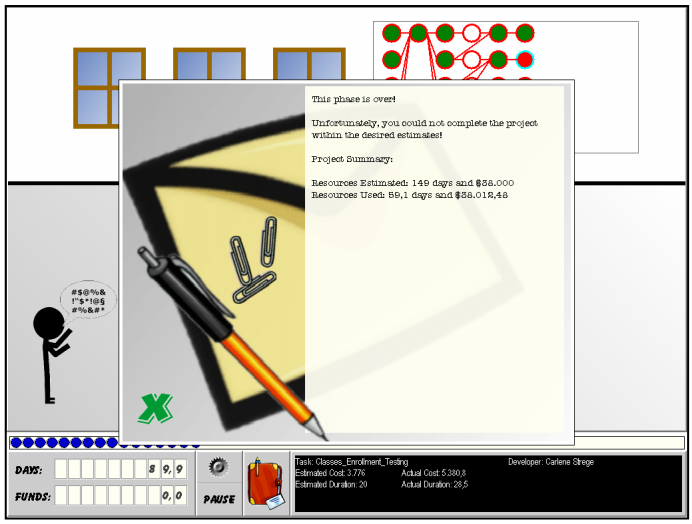

Figura 18 - Janela de diálogo de fim de fase

5.4. Conclusões

Neste capítulo foi apresentado o jogo de simulação *The Incredible Manager* com base na proposta para aplicação de jogos no treinamento de gerentes de projetos de software. As limitações da fidelidade do jogo em relação ao mundo real são, em grande parte, limitações resultantes do modelo sendo simulado, além das limitações nos efeitos visuais.

A pura utilização do jogo de simulação, porém, não deve ser vista como único mecanismo de treinamento que ofereça garantias de efetividade e eficiência da aplicação de jogos sobre treinamento. Desta forma, no próximo capítulo, é apresentado um estudo qualitativo visando avaliar a utilização do jogo *The Incredible Manager* em um contexto de treinamento.

Capítulo

6

Um estudo da aplicação de jogos de simulação em gerenciamento de projetos de software

Com base na hipótese de que projetos de software de sucesso são caracterizados por planejamento, controle de progresso e qualidade efetivos e, acima de tudo, gerentes de projeto qualificados (BROWN, JONES, 1996, BARROS, 2001), foi proposto e construído um jogo de simulação para treinamento de gerentes de projetos de software, visando uma melhor forma de educação de futuros gerentes. Por outro lado, a existência do jogo não garante o aprimoramento do treinamento, pois, segundo GRÖßLER et al. (1999), a efetividade no ensino com uso de simulações não foi provada e não é bem documentada.

Neste capítulo, é apresentado um estudo com foco qualitativo sobre a utilização do jogo de simulação apresentado no capítulo 5, como parte integrante de um modelo de treinamento de gerentes de projetos de software. Ao final, são apresentados resultados da execução do estudo em dois ambientes.

6.1. Planejamento

Na busca de indícios sobre a efetividade do uso de simulações aplicadas ao treinamento, os estudantes não devem ser apenas capazes de utilizar melhor o simulador, mas também de apresentar maior controle sobre certas situações na vida real. Esta transferência de conhecimento pode acontecer quando o indivíduo apresenta a aquisição de conhecimento explícito e independente de domínio. Uma vez que esta aquisição de conhecimento é difícil de ser comprovada em muitos casos, a capacidade de transferência de conhecimento atribuída às simulações também torna-se difícil de ser confirmada. Outras questões na avaliação da efetividade das simulações estão relacionadas a melhoria de desempenho dos estudantes a longo

prazo e a comparação de simulações com outras formas de treinamento. Em estudos longitudinais (realizados sobre os mesmos participantes ao longo do tempo), é difícil avaliar a influência apenas do uso de simulações na atuação profissional dos estudantes. Da mesma forma, é difícil medir vantagens de uma forma de ensino sobre outra, em função da interferência de muitas variáveis que não podem ser controladas (como a qualidade do professor, do livro ou do simulador).

Conforme visto no capítulo 2, MANDL-STRIEGNITZ (2001) propõe um conceito de treinamento unindo sessões de simulação com sessões de discussão, além do aprendizado baseado em livros-texto. Em um estudo realizado com base neste conceito de treinamento, os participantes apresentaram melhora de desempenho na segunda sessão de simulação, em relação aos aspectos capturados na primeira sessão de discussão, mesmo ainda apresentando problemas no controle dos projetos simulados. Em uma avaliação qualitativa sobre o treinamento, os participantes foram positivos sobre o aprendizado de funções de gerenciamento de projetos e das dificuldades de gerenciar um projeto de software com sucesso. Eles consideraram as sessões de discussão como o ponto crucial do conceito de treinamento estabelecido. A chance de repetição do projeto com diferentes estratégias também foi considerada um ponto positivo pelos participantes, que puderam imediatamente perceber os efeitos da aplicação do que foi discutido.

De forma a avaliar a influência da aplicação de um jogo de simulação no treinamento em gerenciamento de projetos, foi elaborado um estudo de caso com foco qualitativo baseado no conceito de treinamento proposto por MANDL-STRIEGNITZ (2001). A estrutura a seguir, baseada no método GQM (BASILI et al., 1994), descreve o objetivo do estudo experimental.

Analisar *a utilização de um jogo de simulação dentro de um conceito de treinamento*
Com o propósito de *caracterizar a utilidade do jogo e continuidade do desenvolvimento*
Referente *aos* *ganhos obtidos por sua utilização e as percepções sobre sua utilização*
Do ponto de vista do *pesquisador*
No contexto do *gerenciamento de projetos de desenvolvimento de software.*

6.2. Participantes

Foram selecionadas para o estudo duas amostras de alunos. A primeira amostra foi selecionada a partir de uma população de alunos de pós-graduação em engenharia de sistemas e computação da Coordenação dos Programas de Pós-Graduação em Engenharia da Universidade Federal do Rio de Janeiro. A segunda amostra foi selecionada em um ambiente de desenvolvimento de cooperação acadêmico-industrial a partir de uma população de alunos de graduação e pós-graduação em ciência da

computação da Pontifícia Universidade Católica do Rio Grande do Sul, em Porto Alegre.

6.3. Instrumentos

Na execução do estudo foi utilizado o jogo de simulação para gerenciamento de projetos de software *The Incredible Manager*. O objetivo do jogo é fazer com que o jogador, a cada fase, seja responsável por planejar e controlar o desenvolvimento de um projeto de software até sua conclusão ou fim dos recursos disponíveis para seu desenvolvimento, estimados pelo planejamento realizado pelo gerente.

Ao final da execução, o jogo disponibiliza o histórico com as ações do jogador em cada fase. A partir dos históricos de cada participante nas duas sessões de simulação, foram coletados dados quantitativos sobre o desempenho de cada jogador.

Os participantes também foram requisitados a preencher dois questionários. O primeiro questionário foi preenchido no início do estudo e coletou informações acerca da formação acadêmica, experiência em desenvolvimento de software, interesse e habilidade em gerenciamento de projetos. O segundo questionário foi preenchido ao final do estudo e coletou informações sobre a percepção da experiência de utilização do jogo pelo participante e as contribuições que ele teria a oferecer. Todos os questionários encontram-se no apêndice B.

6.4. Procedimento

O procedimento a ser adotado para execução do estudo é resumido abaixo. Cada passo é apresentado em detalhes a seguir.

- Sessão de treinamento;
- Preenchimento do questionário sobre o participante;
- Primeira sessão de simulação;
- Captura dos dados quantitativos da execução;
- Primeira sessão de discussão;
- Segunda sessão de simulação;
- Captura dos dados quantitativos da execução;
- Segunda sessão de discussão;
- Preenchimento do questionário qualitativo.

6.4.1. Sessão de treinamento

Os estudantes são convidados a participar de um treinamento de tópicos em gerenciamento de projetos de software, um treinamento sobre o jogo utilizado no estudo e as ações de gerenciamento oferecidas por ele.

6.4.2. Primeira sessão de simulação

Os estudantes devem executar o jogo, constituído de uma única fase (ou seja, um único projeto de software). Inicialmente, cada estudante recebe informações sobre o projeto a ser desenvolvido, incluindo características do projeto (atividades e estimativas de pontos de função) e objetivos do cliente (cronograma e recursos disponíveis, especificações de qualidade). A seguir, cada estudante deve criar um plano de projeto guiado pelo jogo. Para criar um plano de projeto, é preciso definir uma equipe com base nas suas qualificações, a carga de trabalho de cada desenvolvedor, a análise da rede de tarefas, a duração estimada de cada tarefa, o grau de esforço investido em atividades de controle de qualidade e os desenvolvedores responsáveis por cada tarefa. Ao final do planejamento, o estudante terá uma estimativa de custos e cronograma para seu projeto que começará a ser executado. Ele deve gerenciar o desenvolvimento do projeto de acordo com suas estimativas até que seja concluído com sucesso, mesmo que sejam necessários diversos ajustes ao planejamento original. As decisões do estudante devem ser tomadas sem o auxílio ou participação de nenhuma outra pessoa.

6.4.3. Primeira sessão de discussão

Na primeira sessão de discussão, são apresentadas diversas abordagens e cenários comuns no gerenciamento de projetos de software, demonstrando exemplos positivos e negativos de decisões gerenciais em situações particulares, o que permite aos estudantes reconhecer a interação de diferentes influências entre as suas decisões e os resultados do projeto simulado. Exemplos de aspectos levantados para discussão são o gerenciamento de qualidade e o gerenciamento de recursos humanos.

6.4.4. Segunda sessão de simulação

Após a discussão, os estudantes são convidados a repetir a execução do jogo, mantendo as mesmas características da primeira sessão de simulação, fora as variações inerentes a estocasticidade do modelo. Desta forma os estudantes têm a chance de implementar o conhecimento adquirido durante a sessão de discussão,

avaliar imediatamente os novos resultados e compará-los com a primeira sessão de simulação.

6.4.5. Segunda sessão de discussão

Por fim, na segunda sessão de discussão, são apresentadas as lições aprendidas e os resultados observados na comparação entre as sessões de simulação. Exemplos de questões podem ser baseados nos aspectos em que os estudantes melhoraram na segunda simulação, como as melhorias afetaram os resultados obtidos e a existência de problemas restantes.

6.5. Resultados

Na aplicação do estudo não puderam ser contemplados todos os passos previstos no planejamento devido à indisponibilidade de tempo. Participaram do estudo, no Rio de Janeiro, 7 alunos (6 de mestrado e 1 de doutorado) matriculados no curso de gerência de projetos do terceiro trimestre do ano de 2003, no PESC da COPPE/UFRJ. Participaram, em Porto Alegre, 8 alunos (6 de mestrado e 2 de graduação) pertencentes a um ambiente de desenvolvimento de software em uma parceria indústria-universidade. O tempo despendido para o primeiro passo do estudo (sessão de treinamento) nos dois locais foi de aproximadamente 20 minutos. Na execução da primeira sessão de simulação, o tempo mínimo foi de 50 e 55 minutos, enquanto o tempo máximo foi de 120 e 140 minutos (Rio de Janeiro e Porto Alegre, respectivamente). O modelo utilizado pelo jogo de simulação na execução do estudo em Porto Alegre foi reduzido de 20 para 15 tarefas (mantendo o número total de pontos de função do projeto) compondo a rede de tarefas a serem gerenciadas pelo jogador, na tentativa de reduzir o tempo gasto pelos participantes na execução da simulação, o que não foi comprovado.

Por falta de tempo, os estudantes não participaram de uma segunda sessão de simulação e da segunda sessão de discussão que seguiriam esta simulação. Desta forma, os dados de desempenho coletados referem-se apenas a execução do jogo na primeira sessão de simulação. Dos dados coletados, apenas 5 questionários de desempenho não puderam ser aproveitados por erros na geração do arquivo de histórico do jogador durante a execução do jogo na sessão de simulação. Os demais questionários foram preenchidos corretamente pelos participantes, conforme descrito no planejamento do estudo. Apenas um participante (em Porto Alegre) executou a segunda sessão de simulação, tendo obtido resultado positivo em relação ao seu desempenho na primeira sessão de simulação e após a primeira sessão de discussão.

A Tabela 1 apresenta os resultados quantitativos obtidos antes da primeira sessão de simulação.

Tabela 1 - Características dos participantes antes da primeira sessão de simulação[1]

Característica do participante	média geral	média RJ	média RS
Experiência em desenvolvimento de software	3,5	3,1	3,8
Habilidade em gerenciamento de projetos	1,9	1,7	2,0
Interesse em gerenciamento de projetos	4,3	4,3	4,3

Após a primeira sessão de simulação e a sessão de discussão subsequente, os resultados obtidos a partir do questionário qualitativo podem ser resumidos nas tabelas 2, 3, 4 e 5.

Tabela 2 - Avaliação do treinamento (habilidade e interesse)

Avaliação do treinamento	aumentou	Indiferente	diminuiu
Habilidade em gerenciamento de projetos	100%	0%	0%
Interesse em gerenciamento de projetos	87%	13%	0%

Tabela 3 - Avaliação do treinamento (tipo de treinamento)

Avaliação do treinamento	bom	indiferente	ruim
Treinamento utilizando jogos de simulação	100%	0%	0%

Tabela 4 - Avaliação do treinamento (aprendizado)

Avaliação do treinamento	todos	nenhum	muitos	poucos
Aspectos apresentados aprendidos	0%	0%	100%	0%

Tabela 5 - Avaliação do treinamento (diversão)

Avaliação do treinamento	sim	não	muito	pouco
O treinamento foi divertido	47%	0%	53%	0%

Pelo preenchimento do questionário qualitativo, os participantes, em geral, relataram que puderam perceber que o sistema de fato simula uma situação realizável de gestão de projetos, permitindo uma aplicação prática do conhecimento de gerência de projetos ao abordar questões em curto espaço de tempo. O treinamento permitiu aos estudantes perceber que em um projeto de software há vários fatores relacionados, inclusive pressões psicológicas, fazendo com que se sentissem vivenciando a realidade de um projeto. O treinamento não foi considerado cansativo, foi estimulante, dinâmico, extremamente divertido, prático e proveitoso. A interface foi considerada atrativa e amigável ao usuário, fazendo com que o tempo passasse despercebido durante a execução. O nível de dificuldade alto foi apontado como fator que aumentou a atenção e a competição na busca por melhores resultados na execução jogo de simulação.

[1] Para valores atribuídos de 0 (nenhum) a 5 (muito).

Dentre as desvantagens relatadas, os participantes citaram as limitações do modelo de simulação em capturar todos os aspectos existentes nas dinâmicas de projetos reais (mais de um desenvolvedor por tarefa, variações salariais, aspectos de relacionamento inter-pessoal, entre outros). Alguns consideraram curta a sessão de treinamento antes da simulação (inclusive para familiarização com a ferramenta) e a possibilidade de cenários e implicações decorrentes das ações do jogador poderem passar despercebidos. A emissão e leitura de relatórios durante o jogo foram consideradas deficientes, fazendo com que muitos jogadores deixassem de acompanhá-los. Dentre melhorias sugeridas, além das baseadas nas limitações citadas, os participantes destacaram a possibilidade de adoção de metodologias de ensino a distância, a interação entre mais de um jogador, a atribuição de notas e maiores detalhes relacionados ao desempenho do jogador e uma ferramenta explicativa, que permitisse o retorno de passos e a avaliação de erros, lições aprendidas e caminhos alternativos durante o decorrer da simulação.

Com base nos históricos de ações dos jogadores durante a primeira sessão de simulação, as tabelas 6 e 7 apresentam médias comparativas para estimativa e utilização de recursos, para tomada de decisões e ocorrência de determinados estados durante o jogo. Possíveis discrepâncias entre os valores para Rio de Janeiro e Porto Alegre podem estar relacionados com a diminuição da rede de tarefas de 20 para 15 tarefas, na execução do estudo em Porto Alegre.

Tabela 6 – Médias de recursos estimados e utilizados

Recursos estimados/utilizados	média RJ	média RS
Projeto concluído com sucesso	20%	0%
Duração estimada	64,35 dias	55,64 dias
Duração utilizada	61,34 dias	53,88 dias
Fundos estimados	49.459,63 $	46.227,29 $
Fundos utilizados	45.973,86 $	44.562,81 $

Tabela 7 - Médias de tomada de decisões e estados durante o jogo

Decisões/Estados	média RJ (# vezes)	média RS (# vezes)
Contratação de desenvolvedor	12,4	10,8
Demissão de desenvolvedor	5,6	4,2
Mudança de alocação de desenvolvedor	7,4	8,4
Replanejamentos aceitos	12	18,6
Desenvolvedores cansados	1	2,2
Desenvolvedores exaustos	0,4	0,2
Tarefas extrapolaram o tempo	3,2	3,4
Tempo em tarefas de inspeção (em dias)	4,24	1,7

6.6. Conclusões

A execução do estudo da aplicação do jogo de simulação no treinamento de gerentes de projetos de software em dois ambientes distintos, apesar do pequeno

número de participantes, contribuiu para observarmos a utilidade do jogo e as possibilidades de continuidade da pesquisa. Apesar do estudo não ter sido realizado totalmente conforme o planejamento proposto, em comparação com os estudos experimentais iniciais sobre as técnicas de simulação, apresentados no capítulo 4, os resultados qualitativos e quantitativos deste novo estudo com a aplicação de jogos de simulação indicam que os estudantes se sentiram motivados, engajados no treinamento e sentiram que os aspectos ensinados foram aprendidos de forma divertida e espontânea.

No contexto desta pesquisa, novos estudos devem ser elaborados e executados de forma a guiar melhorias tanto nos instrumentos utilizados quanto no modelo de treinamento utilizando jogos de simulação como mecanismo prático inserido.

Capítulo

7

Considerações finais

7.1. Contribuições

Este trabalho propôs e apresentou a construção de um jogo de simulação como instrumento a ser inserido em um processo de treinamento para gerentes de projetos de software. Ao optar pela utilização de jogos de simulação junto às formas tradicionais de ensino (livros-texto e aulas expositivas), ao contrário de simuladores, observamos indícios de benefícios sobre a motivação e engajamento dos estudantes durante o treinamento, contribuindo assim para sua melhor eficiência. Este trabalho visou criar meios para formação de gerentes de projetos de software mais preparados, o que pode ser um fator importante na diminuição dos problemas e fracassos comuns aos projetos de software na indústria, unindo dois grandes mercados na indústria (treinamento e entretenimento).

O treinamento apoiado por um mecanismo prático virtual permite ao estudante vivenciar as dinâmicas das situações reais de gerenciamento de projetos sem enfrentar os riscos e conseqüências negativas que encontraria na execução de um projeto real. Desta forma, o tempo de treinamento possivelmente poderá ser reduzido e melhor aproveitado, permitindo a avaliação de diversas situações e alternativas gerenciais em um curto período de treinamento. A execução do estudo experimental sobre as técnicas de construção e simulação de modelos de projeto e cenários segundo o meta-modelo da Dinâmica de Sistemas reforçou a motivação para a construção e utilização de jogos de simulação como o mecanismo prático virtual a ser inserido no treinamento de gerentes de projeto.

Com base nestas motivações, foi proposta uma estrutura para a construção de jogos de simulação, baseada em três componentes principais: modelo, simulador e máquina de jogo. A utilização de modelos da Dinâmica de Sistemas que representam projetos de software oferece flexibilidade para a construção e integração de novos modelos de projeto ao simulador e à máquina de jogo. O meta-modelo da Dinâmica de

Sistemas foi a linguagem de modelagem escolhida para a criação dos modelos do jogo, em função das extensões que ela apresenta sobre modelos tradicionais da Dinâmica de Sistemas e da facilidade de integração de conhecimento gerencial em cenários. A partir da evolução de um modelo desenvolvido por BARROS (2001), foram criados dois novos cenários de conhecimento gerencial, de forma a tornar a experiência simulada mais próxima à realidade. A possibilidade de configuração de diversas fases oferece ao jogo e ao treinamento flexibilidade e amplitude ao conjunto de lições e cenários a serem aprendidos, segundo as metas educacionais previstas em cada contexto de treinamento.

A máquina de jogo é o componente que cria o ambiente virtual do jogo. A máquina de jogo desenvolvida garante ao jogador interatividade e *feedback* por meio de elementos multimídia, enriquecendo a experiência cognitiva, a motivação e engajamento do estudante nos aspectos a serem aprendidos na utilização do jogo. A não-linearidade e a tomada de decisões sob pressão (passagem contínua de turnos e restrições sobre utilização de recursos ao jogador) também são aspectos que contribuem para o enriquecimento desta experiência e da eficiência do treinamento.

Para permitir interatividade do jogador durante a execução do jogo, o componente de simulação desenvolvido permite a realização de simulações com estrutura dinâmica, ou seja, permite que a estrutura do modelo (elementos, suas propriedades e relacionamentos) seja modificada durante o decorrer da simulação, interrompendo-a e continuando do ponto de pausa sem que valores obtidos nos passos de simulação anteriores sejam perdidos.

Por fim, a definição e execução de um estudo de caso de foco qualitativo sobre a aplicação do jogo de simulação em um contexto de treinamento de gerentes de projetos de software contribuiu para apresentar indícios da utilidade do jogo e a identificação das possibilidades para continuidade do desenvolvimento da pesquisa.

Podemos, portanto, destacar como principais contribuições deste trabalho:

- Detalhamento da proposta de um jogo de simulação como mecanismo complementar ao treinamento teórico de gerentes de projetos de software;
- Construção do protótipo de um jogo de simulação com base na proposta definida para treinamento de gerentes de projetos de software;
- Realização dos estudos experimentais sobre as técnicas de simulação, que serviram como base das motivações para a proposta do jogo de simulação;
- Definição e realização de um estudo de caso de foco qualitativo sobre a aplicação de um jogo de simulação em um contexto de treinamento de gerentes de projetos de software.

A partir destas contribuições, podemos identificar como principais subprodutos:

- Indícios de melhorias na eficiência no treinamento com a introdução do mecanismo prático, o aumento da motivação, engajamento dos participantes e a redução do tempo de treinamento;
- Definição de uma estrutura flexível que permite uso de uma diversidade de modelos de projetos e situações de treinamento;
- Refinamento do modelo de projeto de software, com a inclusão de novos cenários de conhecimento gerencial;
- Adaptação do componente para simulação com estrutura dinâmica de modelos baseados no meta-modelo da Dinâmica de Sistemas;

7.2. Limitações e trabalhos futuros

As principais limitações da proposta e da ferramenta podem ser direcionadas para as próprias limitações do modelo sendo simulado. A medida em que modelos mais refinados do processo de desenvolvimento de projetos de software forem desenvolvidos, mais próxima de situações do mundo real estará a simulação oferecida pelo jogo. A dificuldade de construção textual de modelos de projetos e cenários utilizando o meta-modelo da Dinâmica de Sistemas também é um aspecto limitador. Neste sentido, há uma demanda para um apoio ferramental gráfico e intuitivo que guie a construção de novos modelos mais complexos.

Acompanhando a evolução dos modelos, a máquina de jogo deve oferecer flexibilidade para apresentar uma maior variedade de transições de estados e *feedback* gráfico, enriquecendo o entretenimento e a percepção do jogador sobre os aspectos educacionais do jogo. Apesar da máquina de jogo construída ser flexível para diversos modelos de projeto, criados a partir do modelo de domínio de projetos de software, ela não apresenta flexibilidade total para suportar evoluções do modelo de domínio e do meta-modelo da Dinâmica de Sistemas. Para contornar este desafio, VERONESE *et al*. (2003) apresentam uma proposta de sistematização para a construção de jogos de simulação aplicados ao treinamento, que enfatiza a separação e reutilização dos elementos que compõem este tipo de software: modelos dinâmicos de simulação, modelos de enredo e apresentação do jogo ao usuário.

As principais evoluções deste trabalho podem ser direcionadas para diversas áreas, como a construção de modelos mais refinados sobre gerenciamento de projetos (ou a criação de modelos de domínio para outras áreas de conhecimento), evoluções pedagógicas sobre o processo de treinamento e aplicação do jogo de simulação (com pesquisas em ensino a distância e o desenvolvimento de novos estudos da aplicação

de jogos no treinamento, por exemplo), evoluções artísticas sobre a apresentação multimídia e usabilidade do jogo, pesquisas sobre o acompanhamento e processamento dos rastros de ações realizadas pelo jogador, o comportamento visível resultante do modelo, os diversos caminhos de ações possíveis e, por fim, pesquisas psicológicas sobre aspectos cognitivos e motivacionais relacionados com a aplicação de jogos no treinamento.

Referências Bibliográficas

ABDEL-HAMID, T.K., MADNICK, S.E., 1991, *Software Project Dynamics - An Integrated Approach*, Prentice-Hall, Englewood Cliffs.

AHDELL, R., ANDRESEN, G., 2001, *Games and Simulation in Workplace eLearning.* M.Sc. Dissertation, Norwegian University of Science and Technology, Department of Industrial Economics and Technology Management, Norway.

BARROS, M.O., 2001, Gerenciamento de Projetos Baseado em Cenários: uma Abordagem de Modelagem Dinâmica e Simulação. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2001, "Explaining the Behavior of System Dynamics Models". In: *Proceedings of the 2001 Conference of the International System Dynamics Society*, Atlanta, USA, July.

BARROS, M.O., WERNER, C.M.L., TRAVASSOS, G.H., 2002, "Um Estudo Experimental sobre a Utilização de Modelagem e Simulação no Apoio à Gerência de Projetos de Software". In: *Anais do XVI Simpósio Brasileiro de Engenharia de Software*, pp. 191-260, Gramado, RS, Brasil, Outubro.

BASILI, V.R., CALDIERA, G., ROMBACH, H.D., 1994, "Goal Question Metric Paradigm", IN: Marciniak, J.J., *Encyclopedia of Software Engineering*, vol. 1, New York, NY: Wiley, pp. 528-532.

BELLINGER, G., 2003, "Modeling & Simulation: an Introduction" [online] [visitado em 01/07/2003]. Disponível em <http://www.outsights.com/systems/welcome.htm>.

BROOKS, F.P.Jr., 1974, *The Mythical Man-Month: Essays on Software Engineering*, Reading, MA, Addison-Wesley Publishing Co.

BROWN, N., 1996, "Industrial-Strengh Management Strategies", *IEEE Software*, v. 13, n. 4 (julho), pp. 94-103.

DANTAS, A.R., BARROS, M.O., WERNER, C.M.L., 2003, “Aplicação de Jogos de Simulação no Treinamento em Gerenciamento de Projetos de Software”, *II Workshop de Jogos e Entretenimento Digital, IX Simpósio Brasileiro de Sistemas Multimídia e WEB*, Salvador, BA, Novembro.

DNSALIAS, 2003, *GAGE 2D API - Genuine Advantage Gaming Engine* [online] [visitado em 01/07/2003]. Disponível em <http://java.dnsalias.com/>.

DOYLE, J.K., FORD, D.N., RADZICKI, M.J., TRESS, W.S., 2003, *Mental Models of Dynamic Systems* [online] [visitado em 01/07/2003]. Disponível em <http://www.wpi.edu/Academics/Depts/SSPS/Faculty/Papers/27.pdf>.

DRAPPA, A., LUDEWIG, J., 2000, “Simulation in Software Engineering Training”. In: *Proceedings of the International Conference on Software Engineering*, pp. 199-208, Limerick, Ireland, June.

ELGOOD, C., 1988, *Handbook of Management Games*, 4 ed., Aldershot, England, UK, Gower Publishing Company Limited.

FERREIRA, A.B.H., 1986, *Novo Dicionário da Língua Portuguesa*, 2 ed., Rio de Janeiro, Editora Nova Fronteira.

FLAVELL, J.H., 1988, *A Psicologia do Desenvolvimento de Jean Piaget*. São Paulo: Editora Pioneira, pp. 41-57.

FORRESTER, J.W., 1961, *Industrial Dynamics*, Cambridge, MA: The MIT Press.

FORRESTER, J.W., 1991, *System Dynamics and the Lessons of 35 Years.* Relatório Técnico D-4224-4, MIT System Dynamics Group, Cambridge, MA, USA.

GAMMA, E., HELM, R., JOHNSON, R., *et al.*, 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Massachusetts, Addison Wesley.

GIBBS, W., 1994, “Software's Chronic Crisis”, *Scientific American*, 9, pp. 86-95.

GRÖßLER, A., 1999, “The Influence of Decision Time on Performance in Use of a Business Simulator”. In: *Proceedings of the 1999 Conference of the International System Dynamics Society*, Wellington, New Zeland, July.

GRÖßLER, A., 2000, “Methodological Issues of Using Business Simulators in Teaching and Research”. In: *Proceedings of the 2000 Conference of the International System Dynamics Society*, Bergen, Norway, August.

GRÖßLER, A., NOTZON, I., SHEHZAD, A., 1999, “Constructing an Interactive Learning Environment (ILE) to Conduct Evaluation Experiments”. In: *Proceedings of the 1999 Conference of the International System Dynamics Society*, Wellington, New Zeland, July.

HOUSTON, D., 1996, *System Dynamics Modeling and Simulation of Software Development: A Tutorial.* Relatório Técnico, Arizona State University, Tempe, AZ, USA.

IDSA - Interactive Digital Software Association, 2003, *The Essential Facts About the Computer and Video Game Industry* [online] [visitado em 01/07/2003]. Disponível em <http://www.idsa.com/EF2003.pdf>.

JONES, C., 1996, *Software Systems Failure and Success*. International Thomson Computer Press, Boston.

JONES, C., 2000, *Software Assessments, Benchmarks, and Best Practices*, In: Addison-Wesley Information Technology Series, Reading, MA: Addison-Wesley Publishing Co.

KELLNER, M.I., MADACHY, R.J., RAFFO, D.M., 1999, “Software Process Simulation Modeling: Why? What? How?”, *Journal of Systems and Software*, v. 46, n. 2-3, pp. 91-105.

KNOWLES, M., 1984, *Andragogy in Action.* San Francisco: Jossey-Bass.

MAIER, F.H, STROHHECKER, J., 1996, “Do Management Flight Simulators Really Enhance Decision Effectiveness”. In: *Proceedings of the 1996 Conference of the International System Dynamics Society*, Cambridge, USA, July.

MAIER, F.H., GROBLER, 1998, “A Taxonomy for Computer Simulations to Support Learning about Socio-Economic Systems”. In: *Proceedings of the 1998 Conference of the International System Dynamics Society*, Quebec, Canada, July.

MANDL-STRIEGNITZ, P., 2001, “How to Successfully Use Software Project Simulation for Educating Software Project Managers”. In: *Proceedings of the 31st Frontiers in Education Conference*, Reno, Nevada, USA, October.

MANDL-STRIEGNITZ, P.; LICHTER, H., 1998, “A Case Study on Project Management in Industry: Experiences and Conclusions”. In: *Proceedings of the European Software Measurement Conference*, pp. 305-313, Antwerp, Belgium, May.

MERRIL, D., 1995, Training Software Development Project Managers with a Software Project Simulator [online] [visitado em 01/07/2003], Proposta de Tese de Mestrado, Arizona State University, Tempe, AZ, USA. Disponível em <http://www.eas.asu.edu/~sdm>.

OH, E., VAN DER HOEK, A., 2002, “Towards Game-Based Simulation as a Method of Teaching Software Engineering”. In: *Proceedings of the 2002 Frontiers in Education Conference*, Boston, MA, USA, November.

PFAHL, D., KOVAL, N., RUHE, G., 2000, An Experiment for Evaluation the Effectiveness of Using a System Dynamics Simulation Model in Software Project Management Education. IESE-Report 057.00/E, Fraunhofer Institut Experimentelles Software Engineering, KaisersLautern, Germany.

PMBOK, 1996, *The Guide to the Project Management Body of Knowledge.* Project Management Institute, Newton Square, PA, USA.

PRENSKY, M., 2001, *Digital Game-Based Learning*, McGraw-Hill.

REEL, J.S., 1999, “Critical Success Factors in Software Projects”, *IEEE Software*, v. 16, n. 3, pp. 18-23.

RIEBER, L. P., 1996, “Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games”, *Educational Technology Research & Development*, v. 44, n. 2, pp. 43-58.

STANDISH GROUP, 2003, *The Chaos Chronicles*. The Standish Group International.

STANDISH GROUP, 1999, *CHAOS: A Recipe for Success* [online] [visitado em 01/07/2003]. Disponível em <http://www.pm2go.com/sample_research/>.

STANDISH GROUP, 1994, *The Chaos Report* [online] [visitado em 01/07/2003]. Disponível em <http://www.pm2go.com/sample_research/>.

STERMAN, J.D., 1988, *A Skeptic's Guide to Computer Models.* Report D-4101-1, MIT System Dynamics Group, Cambridge, MA, USA.

STERMAN, J.D., 1992, *System Dynamics Modeling for Project Management.* Relatório Técnico, MIT System Dynamics Group, Cambridge, MA, USA.

SUN MICROSYSTEMS, 2003a, *Java 2 Standard Edition 1.4.2* [online] [visitado em 01/07/2003]. Disponível em <http://java.sun.com/>.

SUN MICROSYSTEMS, 2003b, *Java Native Interface* [online] [visitado em 01/07/2003]. Disponível em <http://java.sun.com/j2se/1.4.2/docs/guide/jni/>.

SYCAMORE, D.M., 1996, *Improving Software Project Management Through System Dynamics Modeling*. M.Sc. Dissertation, Arizona State University, AZ, USA.

THAYER, R., 1997, "Software Engineering Project Management". In: Thayer, R.; E. Yourdon (eds), *Software Engineering Software Management*, pp. 72-104, Los Alamitos, CA, USA, IEEE Computer Society Press.

VERONESE, G., BARROS, M., WERNER, C.M.L., 2003, "A Caminho de uma Abordagem de Apoio à Construção de Jogos de Simulação para Treinamento baseados em Modelos Dinâmicos". *II Workshop de Jogos e Entretenimento Digital, IX Simpósio Brasileiro de Sistemas Multimídia e WEB*, Salvador, BA, Novembro.

WHITE, D., FORTUNE, J., 2002, "Current Practice in Project Management: An Empirical Study", *International Journal of Project Management*, v. 20, n. 1, pp. 1-11.

WOHLIN, C., RUNESON, P., HÖST, M., OHLSSON, M.C., REGNELL, B., WESSLÉN, A., 2000, *Experimentation in Software Engineering: an Introduction*, Kluver Academic Publishers, Norwell, Massachusets.

Apêndice

A

Modelo para o projeto CtrlPESC

```
#---------------------------------------------------------------------
# Metamodel that represents a software development project
#---------------------------------------------------------------------

MODEL ProjectModel
{

#---------------------------------------------------------------------
#Class that represents a developer participating in a software project
#---------------------------------------------------------------------

CLASS Developer
{
   PROPERTY ExpAnalysis       0;    # range [0, 1]
   PROPERTY ExpDesign         0;    # range [0, 1]
   PROPERTY ExpCoding         0;    # range [0, 1]
   PROPERTY ExpTesting        0;    # range [0, 1]
   PROPERTY ExpInspection     0;    # range [0, 1]
   PROPERTY HourlyCost        0;    # in $

   # Activity to which the developer is associated
   PROC AssociatedTask GROUPMIN(BOUND([Activity],Team),DeveloperNeed);

   # Developer's Productivity for each activity
   PROC Cost HourlyCost * 8;

   # Developer's Productivity for each activity
   PROC  Productivity 1;
   PROC  ProdAnalysis Productivity;
   PROC  ProdDesign Productivity;
   PROC  ProdCoding Productivity;
   PROC  ProdTesting Productivity;
   PROC  ProdInspection Productivity;

   # Developer's error generation rate for each activity
   PROC  ErrorGenerationRate 1;
   PROC  ErrorRateAnalysis ErrorGenerationRate;
   PROC  ErrorRateDesign ErrorGenerationRate;
   PROC  ErrorRateCoding ErrorGenerationRate;
   PROC  ErrorRateTesting ErrorGenerationRate;
};
```

```
#-----------------------------------------------------------------------
# Class that represents an activity in a software project
#-----------------------------------------------------------------------

CLASS Activity
{
   PROPERTY AnalysisTask            0;    # 0 OR 1
   PROPERTY InitialDesignTask       0;    # 0 OR 1
   PROPERTY DetailedDesignTask      0;    # 0 OR 1
   PROPERTY CodingTask              0;    # 0 OR 1
   PROPERTY TestingTask             0;    # 0 OR 1
   PROPERTY InspectionTask          0;    # 0 OR 1
   PROPERTY MinimumDuration         0;    # in days
   PROPERTY ExpectedDuration        0;    # in days
   PROPERTY MaximumDuration         0;    # in days
   PROPERTY Order                   0;    # 0 OR +

   # SET the activity execution time
   PROC ExpectedTime ExpectedDuration;
   PROC MaximumTime MaximumDuration;
   PROC MinimumTime MinimumDuration;

   STOCK ExecutionTime IF(OR(MaximumDuration <=0, MinimumDuration <=0), ExpectedTime,
BETAPERT(MinimumTime, ExpectedTime, MaximumTime));
   PROC ETAjuste (ExpectedTime - ExecutionTime) / DT;
   RATE (ExecutionTime) RTExecutionTime ETAjuste;

   PROC PrecedenceTime GROUPMAX(Precedences, ExpectedTime);
   PROC AccumulatedTime PrecedenceTime + ExpectedTime;

   # Determine IF precedent activities are concluded
   PROC PrecConcluded AND (GROUPMAX (Precedences, PrecConcluded) >= 0, GROUPMAX
(Precedences, RemainingTime) < 0.001);

   # Determine IF the activity is concluded
   PROC Concluded RemainingTime < 0.001;

   # Determine IF the activity is ready to run
   PROC Ready AND (PrecConcluded, NOT(Concluded));

   # Determine IF there are resources available for the activity
   PROC DeveloperNeed IF (Ready, Order, 1000);
   PROC Executing AND (Ready, Team.AssociatedTask = Order);

   # Determine developer producticity
   PROC Productivity ((AnalysisTask * Team.ProdAnalysis)  + (InitialDesignTask *
Team.ProdDesign) + (DetailedDesignTask * Team.ProdDesign) + (CodingTask *
Team.ProdCoding) + (TestingTask * Team.ProdTesting) + (InspectionTask *
Team.ProdInspection)) * DT;

   # Determine activity executed time based on developers productivity
   STOCK ExecutedTime 0;
   RATE (ExecutedTime) RTExecTime IF (Executing, IF (OR(InspectionTask>0,
TestingTask>0), MIN(RemainingTime, DT), MIN(RemainingTime, Productivity)), 0) / DT;
   PROC RemainingTime ExecutionTime - ExecutedTime;

   # Calculates conclusion time for an activity
```

```
   STOCK ConclusionTime 0;
   RATE (ConclusionTime) RTConclusionTime IF(AND(ConclusionTime < 0.01, RemainingTime-
RTExecTime*DT < 0.01), TIME/DT+1, 0);

   # Accumulates activity cost
   STOCK Cost 0;
   RATE (Cost) RTCost IF (Executing, Team.Cost, 0);

   # Accumulates days executing
   STOCK DaysExecuting 0;
   RATE (DaysExecuting) RTDaysExecuting IF (Executing, 1, 0);

   # Errors latent in the activity
   STOCK Errors 0;
   RATE (Errors) RTErrors 0;
};

#---------------------------------------------------------------------
# Class that represents a software project
#---------------------------------------------------------------------

CLASS Project
{
   PROC Concluded GROUPMIN ([Activity], Concluded);
   STOCK ProjectTime 0;
   RATE (ProjectTime) RTProjectTime IF (Concluded, 0, 1);

   PROC ProjectCost GROUPSUM([Activity], Cost);

   PROC ExpectedTime GROUPMAX([Activity], AccumulatedTime);

   PROC AvgDevWorkHours 8;
   PROC AvgDevCost 30.00;

   PROC ExpectedTotalTime GROUPSUM([Activity], ExpectedTime);
   PROC ExpectedCost ExpectedTotalTime * AvgDevCost * AvgDevWorkHours;
};

#---------------------------------------------------------------------
# Relationships among classes
#---------------------------------------------------------------------

MULTIRELATION Precedences Activity, Activity (Successors);

RELATION Team Activity, Developer;

};

#---------------------------------------------------------------------
# Scenario that reflects the variation in productivity due to
# communication overhead
#---------------------------------------------------------------------

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);
   };
};

#-------------------------------------------------------------------
# Scenario that reflects the variation in productivity due to
# experience
#-------------------------------------------------------------------

SCENARIO ProductivityDueExpertise ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      AFFECT ProdAnalysis (0.667 + ExpAnalysis * 0.666) * ProdAnalysis;
      AFFECT ProdDesign (0.667 + ExpDesign * 0.666) * ProdDesign;
      AFFECT ProdCoding (0.667 + ExpCoding * 0.666) * ProdCoding;
      AFFECT ProdTesting (0.667 + ExpTesting * 0.666) * ProdTesting;
      AFFECT ProdInspection (0.667 + ExpInspection * 0.666) * ProdInspection;
   };
};

#-------------------------------------------------------------------
# Scenario that reflects the variation in error generation rates due
# to experience
#-------------------------------------------------------------------

SCENARIO ErrorGenerationDueExpertise ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      AFFECT ErrorRateAnalysis (0.667 + (1 - ExpAnalysis) * 0.666) * ErrorRateAnalysis;
      AFFECT ErrorRateDesign (0.667 + (1 - ExpDesign) * 0.666) * ErrorRateDesign;
      AFFECT ErrorRateCoding (0.667 + (1 - ExpCoding) * 0.666) * ErrorRateCoding;
      AFFECT ErrorRateTesting (0.667 + (1 - ExpTesting) * 0.666) * ErrorRateTesting;
   };
};

#-------------------------------------------------------------------
# Scenario that reflects the gain in productivity due to learning
#-------------------------------------------------------------------

SCENARIO ProductivityDueLearning ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      STOCK DaysInProject 0;
      RATE  (DaysInProject) DaysInProjectCounter 1;

      PROC  DIPFactor MIN (DaysInProject / 20, 1.0);
      PROC  DIPModifier LOOKUP (LearningTable, DIPFactor, 0, 1);
      TABLE LearningTable 1.0, 1.0125, 1.0325, 1.055, 1.09, 1.15, 1.2, 1.22, 1.245,
1.25, 1.25;

      AFFECT Productivity Productivity * DIPModifier;
   };
```

```
};

#---------------------------------------------------------------------
# Scenario that reflects the variation in productivity/error
# generation due to overworking
#---------------------------------------------------------------------

SCENARIO Overworking ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      PROPERTY WorkHours 8;      # 8 to 12 hours

      STOCK DailyWorkHours WorkHours;
      PROC DWHAjuste (WorkHours - DailyWorkHours) / DT;
      RATE (DailyWorkHours) DWHRate1 DWHAjuste;
      PROC  WHModifier 1 + (DailyWorkHours - 8) / (12 - 8);

      PROC  SEModifier LOOKUP (SchErrorsTable, WHModifier-1, 0, 1);
      TABLE SchErrorsTable 0.9, 0.94, 1, 1.05, 1.14, 1.24, 1.36, 1.5;

      AFFECT Cost Cost * DailyWorkHours / 8;
      AFFECT Productivity Productivity * WHModifier;
      AFFECT ErrorGenerationRate ErrorGenerationRate * SEModifier;
   };
};

#---------------------------------------------------------------------
# Scenario that reflects the willingness to overwork due to exhaustion
#---------------------------------------------------------------------

SCENARIO Exhaustion ProjectModel
{
   CONNECTION TheDeveloper Developer
   {
      STOCK Exhaustion 0;
      PROC  MaxExhaustion 50;
      RATE  (Exhaustion) ExhaustionRate IF(OR(Resting = 1, AssociatedTask >= 1000),
MAX(-Exhaustion/DT, -MaxExhaustion / 20.0), EXModifier);

      PROC  EXModifier LOOKUP (ExaustionTable, DedicationFactor, 0, 1.5);
      PROC  DedicationFactor 1 - (1 - Dedication) / 0.4;
      PROC  Dedication 0.6 + (WHModifier - 1) * (1.2 - 0.6);
      TABLE ExaustionTable 0.0, 0.0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.15, 1.3,
1.6, 1.9, 2.2, 2.5;

      STOCK Resting 0;
      RATE  (Resting) RestingRate1 IF (InitResting, 1 / DT, 0);
      RATE  (Resting) RestingRate2 IF (QuitResting, -1 / DT, 0);
      RATE  (DailyWorkHours) DWHRate2 IF (Resting = 1, (8 - DailyWorkHours) / DT, 0);

      PROC  InitResting AND(Resting = 0, Exhaustion > MaxExhaustion);
      PROC  QuitResting AND(Resting = 1, Exhaustion < 0.1);

      AFFECT DWHAjuste IF (Resting = 1, 0, DWHAjuste);
   };

   CONSTRAINT TheDeveloper, Overworking.TheDeveloper;
```

```
};

#-----------------------------------------------------------------------
# Scenario that introduces new erros from error corretion
#-----------------------------------------------------------------------

SCENARIO BadFixes ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC DefaultBadFixes 7.5;
      PROC BadFixes CorrErrors * (DefaultBadFixes / 100.0) * Team.ErrorRateTesting;

      RATE (Errors) RTBadFixes BadFixes;
   };

   CONSTRAINT TheActivity, ErrorCorrection.TheActivity;
};

#-----------------------------------------------------------------------
# Scenario that allows measuring an activity in function points
#-----------------------------------------------------------------------

SCENARIO FunctionPointMeasure ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROPERTY FunctionPoints    0;    # in FP
      PROC   TaskFunctionPoints  0;    # FunctionPoints * TaskContribution
   };
};

#-----------------------------------------------------------------------
# Scenario that estimates activity duration using function points
#-----------------------------------------------------------------------

SCENARIO EstimationFunctionPoints ProjectModel
{
   CONNECTION TheActivity Activity
   {
      # Developer's productivity
      PROC AvgFPMonth 27.80;
      PROC MinFPMonth 15.41;
      PROC MaxFPMonth 40.19;

      # Evaluate the contribution of the activity type to the project
      PROC  ContrAnalysis AnalysisTask * 4.11144;
      PROC  ContrInitialDesign InitialDesignTask * 4.55393;
      PROC  ContrDetailedDesign DetailedDesignTask * 6.07652;
      PROC  ContrCoding CodingTask * 25.31653;
      PROC  ContrTesting TestingTask * 59.94159;
      PROC  Contribution ContrAnalysis + ContrInitialDesign + ContrDetailedDesign +
ContrCoding + ContrTesting;

      # Evaluate the expected activity execution time
      AFFECT ExpectedTime (ExpectedTime * 0.0) + FunctionPoints * (30.0 / AvgFPMonth) *
(Contribution / 100);
```

```
      AFFECT MaximumTime (MaximumTime * 0.0) + FunctionPoints * (30.0 / MinFPMonth) *
(Contribution / 100);
      AFFECT MinimumTime (MinimumTime * 0.0) + FunctionPoints * (30.0 / MaxFPMonth) *
(Contribution / 100);

      AFFECT TaskFunctionPoints (TaskFunctionPoints * 0.0) + FunctionPoints *
(Contribution / 100);
   };

   CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};

#-------------------------------------------------------------------
# Scenario that represents error generation in an activity
#-------------------------------------------------------------------

SCENARIO ErrorGeneration ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC NewErrors (AnalysisTask + InitialDesignTask + DetailedDesignTask +
CodingTask) * ErrorGeneration;
      PROC ErrorGen ((AnalysisTask * Team.ErrorRateAnalysis)  + (InitialDesignTask *
Team.ErrorRateDesign) + (DetailedDesignTask * Team.ErrorRateDesign) + (CodingTask *
Team.ErrorRateCoding));
      PROC ErrorGeneration IF (ExecutionTime > 0, ErrorsFP * FunctionPoints * ErrorGen *
RTExecTime / ExecutionTime, 0);
      PROC ErrorsFP (AnalysisTask * 0.6)  + (InitialDesignTask * 0.18) +
(DetailedDesignTask * 0.57) + (CodingTask * 1.05);
      AFFECT RTErrors RTErrors + NewErrors;
   };

   CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};


#-------------------------------------------------------------------
# Scenario that represents error propagation between two sequential
# activities
#-------------------------------------------------------------------

SCENARIO ErrorPropagation ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC SuccessorCount COUNT(Successors);
      PROC ErrorsToGo IF (SuccessorCount > 0, (Errors + RTErrors*DT) /  SuccessorCount,
0);

      PROC Initializing AND (ExecutionTime > 0.0, ExecutedTime < 0.001, Executing);
      PROC ReadyToInherit OR (AND(ExecutionTime < 0.001, PrecConcluded, Errors < 0.001),
Initializing);
      PROC InheritedErrors IF (ReadyToInherit, GROUPSUM (Precedences, ErrorsToGo), 0);

      AFFECT RTErrors RTErrors + InheritedErrors / DT;
   };

   CONSTRAINT TheActivity.Precedences, ErrorPropagation.TheActivity;
```

```
};

#-------------------------------------------------------------------
# Scenario that represents error regeneration on sequential activities
#-------------------------------------------------------------------

SCENARIO ErrorRegeneration ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC InheritedDensity InheritedErrors / FunctionPoints;
      PROC RegenErrors (AnalysisTask + InitialDesignTask + DetailedDesignTask +
CodingTask) * InheritedErrors * 0.24 * RegenFactor;
      PROC RegenFactor MAX (1, LOOKUP (ActiveErrosDens, InheritedDensity , 0, 10));
      TABLE ActiveErrosDens 1, 1.1, 1.2, 1.325, 1.45, 1.6, 2.0, 2.5, 3.25, 4.35, 6.0;
      AFFECT RTErrors RTErrors + RegenErrors / DT;
   };

   CONSTRAINT TheActivity, ErrorPropagation.TheActivity;

   CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};

#-------------------------------------------------------------------
# Scenario that represents error correction in an activity
#-------------------------------------------------------------------

SCENARIO ErrorCorrection ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROPERTY Target 95.0;      # in %

      # Average errors per function point
      PROC AvgErrorFP 2.4;

      # Errors corrected acummulator for the activity
      STOCK ErrorsCorrected 0;
      RATE (ErrorsCorrected) RTCorrection CorrErrors;

      # Error correction in the activity
      RATE (Errors) RTCorrErrors -CorrErrors;
      PROC CorrErrors (InspectionTask + TestingTask) * RTExecTime * Productivity /
(DetectionCost * DT);

      # Cost to correct an error
      PROC DetectionCost 0.28;

      # Adjustment of time for testng activities
      RATE (ExecutionTime) RTTesting IF (AND(Executing, TestingTask > 0), -
ExecutionTime+ExecutedTime+TestingEffort, 0) / DT;
      PROC TestingEffort TestingDifference * DetectionCost;
      PROC TestingDifference MAX(Errors + InheritedErrors - CorrErrors - TestingTarget,
0);
      PROC TestingTarget FunctionPoints * AvgErrorFP * (1 - Target / 100.0);
   };

   CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
```

```
};

#--------------------------------------------------------------------
# Scenario that represents the density effect over error correction
#--------------------------------------------------------------------

SCENARIO ErrorCorrectionDensity ProjectModel
{
   CONNECTION TheActivity Activity
   {
      PROC ErrorDensityMultiplier MAX (1, LOOKUP (TableErrorDensityMultiplier,
ErrorDensity, 0, 1));
      TABLE TableErrorDensityMultiplier 8, 6.75, 5.25, 4, 3, 2, 1.8, 1.6, 1.2, 1.1, 1;
      PROC ErrorDensity (Errors + RTErrors*DT) / FunctionPoints;
      AFFECT DetectionCost DetectionCost * ErrorDensityMultiplier;
   };

   CONSTRAINT TheActivity, ErrorCorrection.TheActivity;
};

#--------------------------------------------------------------------
# A software project model
#--------------------------------------------------------------------

DEFINE SistemaExemplo ProjectModel
{
   SPEC DT 0.1;

   CtrlPESC = NEW Project

   UserStudentRulesAnalysis = NEW Activity
      SET AnalysisTask = 1;
      SET FunctionPoints = 49.14;
      SET Order = 0;

   AreaTeacherDisciplineAnalysis = NEW Activity
      SET AnalysisTask = 1;
      SET FunctionPoints = 50.70;
      SET Order = 1;

   Classes_Enrollment_Analysis = NEW Activity
      SET AnalysisTask = 1;
      SET FunctionPoints = 60.84;
      SET Order = 2;

   ArchitecturalDesign = NEW Activity
      SET InitialDesignTask = 1;
      SET FunctionPoints = 160.68;
      SET Order = 4;
      LINK Precedences UserStudentRulesAnalysis, AreaTeacherDisciplineAnalysis,
Classes_Enrollment_Analysis;

   UserStudentRulesDesign = NEW Activity
      SET DetailedDesignTask = 1;
      SET FunctionPoints = 49.14;
      SET Order = 5;
      LINK Precedences ArchitecturalDesign;
```

```
AreaTeacherDisciplineDesign = NEW Activity
   SET DetailedDesignTask = 1;
   SET FunctionPoints = 50.70;
   SET Order = 6;
   LINK Precedences ArchitecturalDesign;

ClassesEnrollmentDesign = NEW Activity
   SET DetailedDesignTask = 1;
   SET FunctionPoints = 60.84;
   SET Order = 7;
   LINK Precedences ArchitecturalDesign;

UserStudentRulesInspection = NEW Activity
   SET InspectionTask = 1;
   SET FunctionPoints = 49.14;
   SET Order = 9;
   LINK Precedences UserStudentRulesDesign;

AreaTeacherDisciplineInspection = NEW Activity
   SET InspectionTask = 1;
   SET FunctionPoints = 50.70;
   SET Order = 10;
   LINK Precedences AreaTeacherDisciplineDesign;

ClassesEnrollmentInspection = NEW Activity
   SET InspectionTask = 1;
   SET FunctionPoints = 60.84;
   SET Order = 11;
   LINK Precedences ClassesEnrollmentDesign;

UserStudentRulesCoding = NEW Activity
   SET CodingTask = 1;
   SET FunctionPoints = 49.14;
   SET Order = 13;
   LINK Precedences UserStudentRulesInspection;

AreaTeacherDisciplineCoding = NEW Activity
   SET CodingTask = 1;
   SET FunctionPoints = 50.70;
   SET Order = 14;
   LINK Precedences AreaTeacherDisciplineInspection;

ClassesEnrollmentCoding = NEW Activity
   SET CodingTask = 1;
   SET FunctionPoints = 60.84;
   SET Order = 15;
   LINK Precedences ClassesEnrollmentInspection;

UserStudentRulesTesting = NEW Activity
   SET TestingTask = 1;
   SET FunctionPoints = 49.14;
   SET Order = 17;
   LINK Precedences UserStudentRulesCoding;

AreaTeacherDisciplineTesting = NEW Activity
   SET TestingTask = 1;
   SET FunctionPoints = 50.70;
   SET Order = 18;
```

```
   LINK Precedences AreaTeacherDisciplineCoding;

ClassesEnrollmentTesting = NEW Activity
   SET TestingTask = 1;
   SET FunctionPoints = 60.84;
   SET Order = 19;
   LINK Precedences ClassesEnrollmentCoding;

ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserStudentRulesAnalysis;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaTeacherDisciplineAnalysis;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity Classes_Enrollment_Analysis;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity ArchitecturalDesign;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserStudentRulesDesign;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaTeacherDisciplineDesign;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity ClassesEnrollmentDesign;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserStudentRulesInspection;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaTeacherDisciplineInspection;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity ClassesEnrollmentInspection;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserStudentRulesCoding;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaTeacherDisciplineCoding;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity ClassesEnrollmentCoding;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity UserStudentRulesTesting;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity AreaTeacherDisciplineTesting;
ACTIVATE FunctionPointMeasure
   CONNECT TheActivity ClassesEnrollmentTesting;

ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity UserStudentRulesAnalysis;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity AreaTeacherDisciplineAnalysis;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity Classes_Enrollment_Analysis;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity ArchitecturalDesign;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity UserStudentRulesDesign;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity AreaTeacherDisciplineDesign;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity ClassesEnrollmentDesign;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity UserStudentRulesCoding;
ACTIVATE EstimationFunctionPoints
```

```
   CONNECT TheActivity AreaTeacherDisciplineCoding;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity ClassesEnrollmentCoding;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity UserStudentRulesTesting;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity AreaTeacherDisciplineTesting;
ACTIVATE EstimationFunctionPoints
   CONNECT TheActivity ClassesEnrollmentTesting;


ACTIVATE ErrorPropagation
   CONNECT TheActivity UserStudentRulesAnalysis;
ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaTeacherDisciplineAnalysis;
ACTIVATE ErrorPropagation
   CONNECT TheActivity Classes_Enrollment_Analysis;
ACTIVATE ErrorPropagation
   CONNECT TheActivity ArchitecturalDesign;
ACTIVATE ErrorPropagation
   CONNECT TheActivity UserStudentRulesDesign;
ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaTeacherDisciplineDesign;
ACTIVATE ErrorPropagation
   CONNECT TheActivity ClassesEnrollmentDesign;
ACTIVATE ErrorPropagation
   CONNECT TheActivity UserStudentRulesInspection;
ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaTeacherDisciplineInspection;
ACTIVATE ErrorPropagation
   CONNECT TheActivity ClassesEnrollmentInspection;
ACTIVATE ErrorPropagation
   CONNECT TheActivity UserStudentRulesCoding;
ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaTeacherDisciplineCoding;
ACTIVATE ErrorPropagation
   CONNECT TheActivity ClassesEnrollmentCoding;
ACTIVATE ErrorPropagation
   CONNECT TheActivity UserStudentRulesTesting;
ACTIVATE ErrorPropagation
   CONNECT TheActivity AreaTeacherDisciplineTesting;
ACTIVATE ErrorPropagation
   CONNECT TheActivity ClassesEnrollmentTesting;


ACTIVATE ErrorRegeneration
   CONNECT TheActivity UserStudentRulesAnalysis;
ACTIVATE ErrorRegeneration
   CONNECT TheActivity AreaTeacherDisciplineAnalysis;
ACTIVATE ErrorRegeneration
   CONNECT TheActivity Classes_Enrollment_Analysis;
ACTIVATE ErrorRegeneration
   CONNECT TheActivity ArchitecturalDesign;
ACTIVATE ErrorRegeneration
   CONNECT TheActivity UserStudentRulesDesign;
ACTIVATE ErrorRegeneration
   CONNECT TheActivity AreaTeacherDisciplineDesign;
ACTIVATE ErrorRegeneration
```

```
   CONNECT TheActivity ClassesEnrollmentDesign;
ACTIVATE ErrorRegeneration
   CONNECT TheActivity UserStudentRulesCoding;
ACTIVATE ErrorRegeneration
   CONNECT TheActivity AreaTeacherDisciplineCoding;
ACTIVATE ErrorRegeneration
   CONNECT TheActivity ClassesEnrollmentCoding;


ACTIVATE ErrorGeneration
   CONNECT TheActivity UserStudentRulesAnalysis;
ACTIVATE ErrorGeneration
   CONNECT TheActivity AreaTeacherDisciplineAnalysis;
ACTIVATE ErrorGeneration
   CONNECT TheActivity Classes_Enrollment_Analysis;
ACTIVATE ErrorGeneration
   CONNECT TheActivity ArchitecturalDesign;
ACTIVATE ErrorGeneration
   CONNECT TheActivity UserStudentRulesDesign;
ACTIVATE ErrorGeneration
   CONNECT TheActivity AreaTeacherDisciplineDesign;
ACTIVATE ErrorGeneration
   CONNECT TheActivity ClassesEnrollmentDesign;
ACTIVATE ErrorGeneration
   CONNECT TheActivity UserStudentRulesCoding;
ACTIVATE ErrorGeneration
   CONNECT TheActivity AreaTeacherDisciplineCoding;
ACTIVATE ErrorGeneration
   CONNECT TheActivity ClassesEnrollmentCoding;


ACTIVATE ErrorCorrection
   CONNECT TheActivity UserStudentRulesInspection;
ACTIVATE ErrorCorrection
   CONNECT TheActivity AreaTeacherDisciplineInspection;
ACTIVATE ErrorCorrection
   CONNECT TheActivity ClassesEnrollmentInspection;
ACTIVATE ErrorCorrection
   CONNECT TheActivity UserStudentRulesTesting;
ACTIVATE ErrorCorrection
   CONNECT TheActivity AreaTeacherDisciplineTesting;
ACTIVATE ErrorCorrection
   CONNECT TheActivity ClassesEnrollmentTesting;


ACTIVATE ErrorCorrectionDensity
   CONNECT TheActivity UserStudentRulesInspection;
ACTIVATE ErrorCorrectionDensity
   CONNECT TheActivity AreaTeacherDisciplineInspection;
ACTIVATE ErrorCorrectionDensity
   CONNECT TheActivity ClassesEnrollmentInspection;
ACTIVATE ErrorCorrectionDensity
   CONNECT TheActivity UserStudentRulesTesting;
ACTIVATE ErrorCorrectionDensity
   CONNECT TheActivity AreaTeacherDisciplineTesting;
ACTIVATE ErrorCorrectionDensity
   CONNECT TheActivity ClassesEnrollmentTesting;
```

```
   ACTIVATE BadFixes
      CONNECT TheActivity UserStudentRulesInspection;
   ACTIVATE BadFixes
      CONNECT TheActivity AreaTeacherDisciplineInspection;
   ACTIVATE BadFixes
      CONNECT TheActivity ClassesEnrollmentInspection;
   ACTIVATE BadFixes
      CONNECT TheActivity UserStudentRulesTesting;
   ACTIVATE BadFixes
      CONNECT TheActivity AreaTeacherDisciplineTesting;
   ACTIVATE BadFixes
      CONNECT TheActivity ClassesEnrollmentTesting;

};
```

Apêndice

B

Questionários utilizados no estudo da aplicação do jogo de simulação

B.1. Questionário de caracterização do participante

Nome	
Idade	

Formação acadêmica máxima:

[ ] Graduando	[ ] Graduado	[ ] Mestrando	[ ] Mestre
[ ] Doutorando	[ ] Doutor	[ ] Especialização	

Avaliação de conhecimento:

Experiência em desenvolvimento	[0] [1] [2] [3] [4] [5]
Habilidade em gerenciamento de projetos	[0] [1] [2] [3] [4] [5]
Interesse em gerenciamento de projetos	[0] [1] [2] [3] [4] [5]

[0] = nenhum, [5] = muito

B.2. Questionário de avaliação qualitativa

Avaliação do treinamento:

Habilidade em gerenciamento de projetos	[ ] aumentou [ ] diminuiu [ ] indiferente
Interesse em gerenciamento de projetos	[ ] aumentou [ ] diminuiu [ ] indiferente
O treinamento usando jogos de simulação	[ ] ruim [ ] bom [ ] indiferente
Os aspectos apresentados aprendidos	[ ] todos [ ] nenhum [ ] muitos [ ] poucos
O treinamento foi divertido	[ ] sim [ ] não [ ] muito [ ] um pouco

Questões:

Quais foram as principais vantagens do tipo de treinamento utilizado ?

_____

_____

_____

_____

Quais foram as principais desvantagens do tipo de treinamento utilizado ?

_____

_____

_____

_____

Que sugestões você daria para melhorar o tipo de treinamento ?

_____

_____

_____

_____

Que sugestões você daria para melhorar o jogo de simulação ?

_____

_____

_____

_____

B.3. Dados quantitativos de desempenho

Desempenho final:

[ ] Projeto concluído com sucesso	[ ] Projeto cancelado por falta de recursos

Desempenho em recursos:

Recursos	Tempo	Fundos
Utilizado		
Estimado		

Tomada de decisões e transição de estados:

Tomada de decisão	# vezes	Ocorrência do estado	Tempo
Aumento de carga horária		Desenvolvedor cansado	
Diminuição de carga horária		Desenvolvedore exausto	
Contratação de desenvolvedor		Projeto atrasado	
Demissão de desenvolvedor		Projeto sem fundos	
Aumento do tempo de tarefa		Projeto sem tempo	
Diminuição do tempo de tarefa		Tarefas atrasadas	
Mudança de desenvolvedor		Tarefa extrapolada	
Replanejamentos feitos		Tarefa de inspeção	