PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 34/06

SimulES: Um Jogo para o Ensino de Engenharia de Software

Eduardo Magno Lages Figueiredo
Cidiane Aracaty Lobato
Klessis Lopes Dias
Julio Cesar Sampaio do Prado Leite
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 34/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Outubro/October, 2006

SimulES: Um Jogo para o Ensino de Engenharia de Software

Eduardo Magno Lages Figueiredo, Cidiane Aracaty Lobato, Klessis Lopes Dias, Julio Cesar Sampaio do Prado Leite, Carlos José Pereira de Lucena

{emagno, cidiane, klessis, julio, lucena}@inf.puc-rio.br

Abstract. Game technology has been used for learning in many educational areas, but it is not very popular in software engineering. To address this problem we present SimulES, an educational card game that simulates the software engineering process. This game is based on a previous version named Problems and Programmers. Using our game, a student can take the role of a software project manager and deal with problems which are not sufficiently highlighted by traditional lectures. We also compare SimulES to Problems and Programmers and the results are reported.

Keywords: Educational Card Game, Software Engineering Simulation, Software Process.

Resumo. Tecnologia de jogos tem sido usada para o ensino em muitas áreas, mas em engenharia de software ela não é muito comum. Para minimizar este problema, este artigo apresenta SimulES, um jogo educacional de cartas que simula o processo de desenvolvimento de software. Este jogo é baseado em uma versão preliminar chamada "*Problems and Programmers*". O jogo proposto neste artigo permite que um estudante assuma o papel de gerente de projeto e depare com problemas que não são bem cobertos em aulas tradicionais. Este artigo também compara o jogo SimulES com "*Problems and Programmers*" e os resultados são reportados.

Palavras-chave: Jogo Educacional de Cartas, Simulação de Engenharia de Software, Processo de Software.

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Sumário

1 Introdução

O uso de jogos para estimular a curiosidade e prover uma motivação para o aprendizado é um tema já amplamente pesquisado e discutido [Oh e Hoek 2001] [Virvou *et al*. 2005]. No entanto, na área de engenharia de software essa estratégia é pouco estudada. Um curso típico de engenharia de software consiste de aulas em que conceitos teóricos são passados aos alunos e exercitados por atividades em pequenos exemplos práticos. Apesar de o professor poder explicar assuntos relacionados à gerência de projeto (tal como características humanas), certos problemas de grandes projetos, com elevado número de pessoal, não são satisfatoriamente cobertos nas atividades práticas.

A motivação desta pesquisa partiu de um estudo no contexto de evolução de software[1]. Ao definirmos um artefato para aplicar os conceitos de evolução de software, escolhemos o jogo "*Problems and Programmers*" (PnP) [Baker *et al*. 2005] [Navaro *et al*. 2004]. Ao decidirmos por essa estratégia, partimos para jogarmos o referido jogo conforme as instruções e os recursos disponíveis no sítio [Problems and Programmers 2006]. Jogamos PnP três vezes e identificamos problemas que podem ser classificados em duas categorias: (i) as técnicas de ensino são limitadas e não permitem aos jogadores adquirirem conhecimento externo ao jogo e (ii) muitos conceitos de engenharia de software utilizados são vagos ou obsoletos. Além disso, o conceito de evolução de software praticamente não está presente no jogo e este foi o ponto de partida para decidirmos pela alteração das regras e estrutura do jogo.

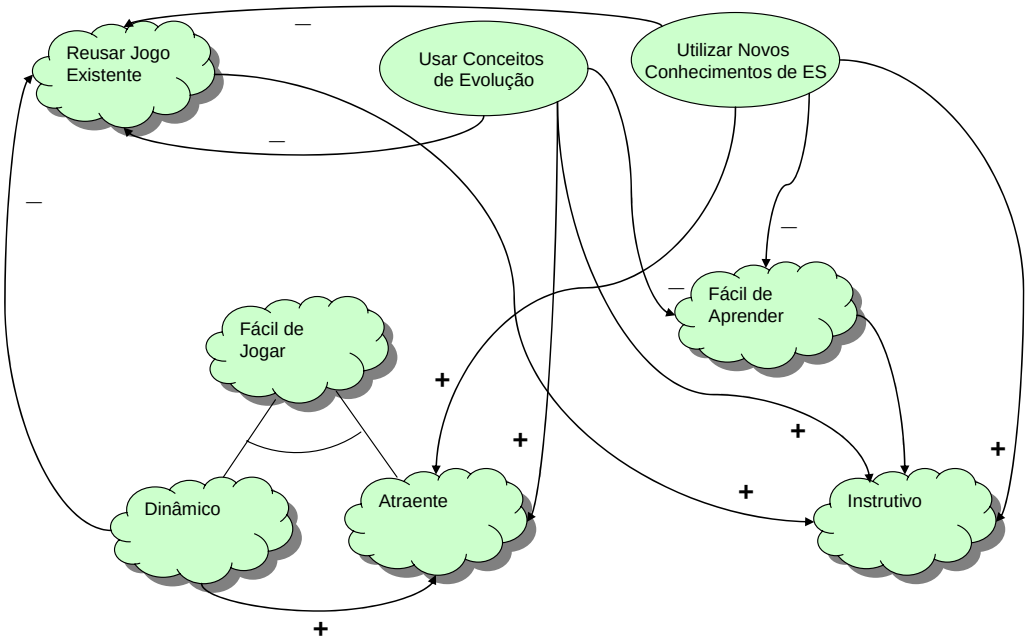


Figura 1: Principais objetivos do jogo

1 O trabalho foi realizado durante o semestre de 2006-1 no contexto de uma disciplina de pós-graduação do Departamento de Informática da PUC-Rio. Informações parciais sobre esse trabalho podem ser encontradas em: http://evolsoftware.blogspot.com/

PnP baseia-se numa visão clássica de engenharia de software, na qual prevalece a visão de um processo de produção bastante linear. Portanto, nossa contribuição não estava apenas centrada em fazer um jogo mais fácil de jogar, mas também de fazer refletir no jogo, práticas mais modernas de produção de software. Desta forma, tínhamos duas meta bem definida: (i) aplicar os conceitos de evolução e (ii) utilizar fundamentos mais modernos de engenharia de software. Além disso, tínhamos algumas metas do tipo "*soft*" [Chung *et al*. 2000], por exemplo, jogo mais atraente, instrutivo, dinâmico, fácil de usar e fácil de aprender. A Figura 1 mostra as metas assim como interferências (positivas ou negativas) entre elas. Aqui utiliza-se a notação base de modelagem orientada a metas de Mylopoulos, Chung e Yu [1999].

Uma vez colocados os grandes objetivos, traçamos uma estratégia para concluirmos o projeto. Nesse artigo é apresentada a maneira com que as metas foram operacionalizadas e uma primeira versão do novo jogo, batizado de SimulES (Simulador de Uso da Engenharia de Software). É mostrada também uma avaliação comparativa, baseado no uso deste jogo e em observações referentes ao jogo original.

O restante deste artigo está organizado da seguinte forma. Na Seção 2 o jogo PnP é brevemente apresentado destacando seus principais pontos positivos e negativos. O novo jogo, originado de PnP, é proposto na Seção 3. Uma argumentação sobre como as metas foram atingidas é feita na Seção 4, inclusive com uma avaliação comparativa entre os dois jogos. Na seção 5 destacamos as interfaces deste artigo com alguns trabalhos existentes na literatura. Concluímos (Seção 6) mencionando as principais contribuições e futuros desdobramentos que este projeto possa vir a ter.

2 A Origem do Jogo

O jogo proposto neste artigo é baseado em um jogo educacional chamado "*Problems and Programmers*" (PnP) [Baker *et al*. 2005]. PnP é um jogo de cartas direcionado para a área de engenharia de software, desenvolvido no Departamento de Informática da Universidade da Califórnia. Seu objetivo é simular o processo de desenvolvimento de sistemas desde a concepção até a fase de entrega do software. Este jogo é dirigido para estudantes com nível básico de conhecimento em disciplinas de engenharia de software, mas conceitos avançados são apresentados no decorrer do jogo. A idéia dos autores é utilizar o jogo como uma ferramenta de apoio durante um curso tipicamente semestral.

2.1 Contribuições e Limitações de PnP

O exercício de simulação adotado pelo jogo PnP ajuda os estudantes a aprenderem lições sobre boas práticas de engenharia de software que serão refletidas quando estes se depararem com um projeto real de sistema de software. Uma característica interessante deste jogo é que ele ensina a partir de decisões tomadas de forma equivocada pelos jogadores. Assim, no fim de uma série de jogadas, os participantes aprendem quais decisões e caminhos devem ser seguidos em seus projetos.

Após o estudo do jogo PnP e dos conceitos que o sustenta, foram observadas algumas questões que poderiam ser evoluídas melhorando o jogo tanto em sua dinâmica quanto em seu propósito de simulação do mundo real. As principais limitações do jogo PnP identificadas durante este estudo são listadas a seguir:

- *Muito amarrado a um único processo de software*. O jogo segue um processo de desenvolvimento de software chamado Modelo Cascata que, apesar de bem conhecido, não é muito utilizado atualmente. Novas abordagens têm sido propostas recentemente como o Modelo Espiral [Boehm 1986] e Programação Extrema (XP) [Beck 1999]. Entretanto, o jogo não oferece liberdade ao jogador para escolha de seu próprio processo de desenvolvimento.
- *Utilização de cartas muito abstratas e sem informações adicionais*. No geral, as cartas do jogo não permitem aos jogadores uma clara interpretação, talvez pelo fato destas se limitarem a um título e seu efeito (como as cartas de problemas e conceitos). Desta forma, caso o jogador não entenda a mensagem educativa da carta, ele não possui nenhuma fonte extra de informação no jogo.
- *Jogo não desperta entusiasmo dos jogadores nas primeiras rodadas*. Um jogo educativo, bem como qualquer categoria de jogo, deve motivar os participantes para alcançar seu objetivo. Entretanto, o jogo PnP possui uma dinâmica inicial pouco interessante, pois não permite jogar cartas de problemas enquanto os jogadores não possuírem carta de código. Desta forma, os jogadores se limitavam a comprar cartas de documentação sem nenhuma interação com os adversários.
- *O jogo não limita o número de jogadores, mas suas regras dificultam que mais pessoas o joguem*. Em nossa experiência, 6 pessoas jogaram e um problema é que muitos problemas são lançados para cada jogador (até 5). Com este elevado número de problemas, torna-se difícil para qualquer participante evoluir e ganhar no jogo.
- *Ausência de um tabuleiro para organização da área de projeto do jogador*. Em nossa experiência jogando, percebemos ser difícil manter as cartas dispostas de forma organizada na mesa. Em poucas rodadas, não era possível saber, por exemplo, se determinada carta era de requisitos ou de desenho.
- *Ausência de um mapeamento claro entre os artefatos do jogo e os conceitos de engenharia de software*. PnP é intencionado para ser usado de forma complementar a aulas tradicionais. Entretanto, o professor que utiliza o jogo não possui um elo que conecte determinado conceito utilizado em aula a um conjunto de artefatos ou cenários do jogo.

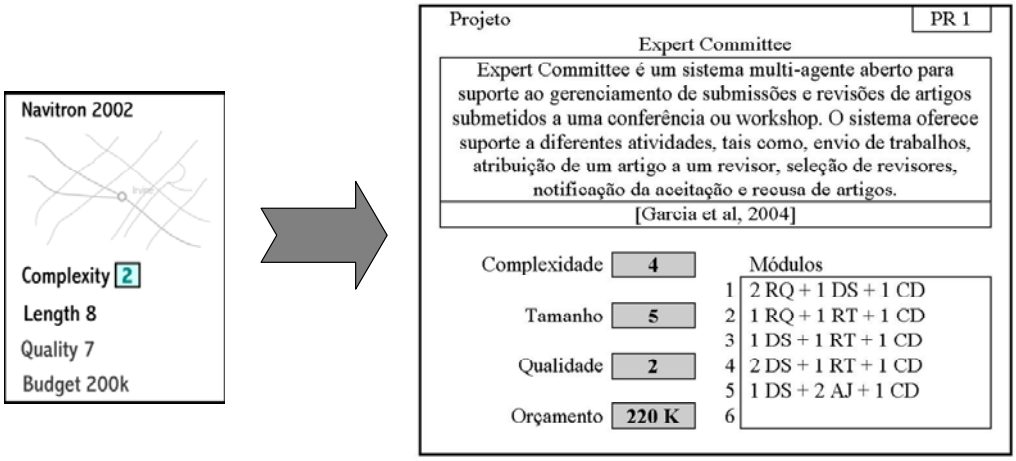
3 SimulES: O Jogo de Simulação

Nesta seção é apresentado um jogo para ensino de engenharia de software, batizado de SimulES (Simulador de Uso da Engenharia de Software), que procura resolver os problemas listados na seção anterior. SimulES pode ser jogado entre 4 e 8 jogadores e é apresentado nesta seção de forma comparativa com PnP (seção 2) de tal forma a explicar suas principais diferenças tanto na estrutura quanto na maneira de jogar. Como em PnP, o objetivo de SimulES é que jogadores disputem para terminar um projeto de software e o vencedor será quem implantar o projeto primeiro. Através deste jogo, seus participantes, preferencialmente alunos, aprendem importantes conceitos de computação e especialmente de engenharia de software. Os recursos do jogo são: cartões de projeto (subseção 3.1), um tabuleiro (subseção 3.2), cartas (subseção 3.3) e um dado. Em relação o jogo PnP, estes recursos foram totalmente reformulados. O tabuleiro e o dado não existiam no jogo anterior e foram incluídos em SimulES, as cartas foram atualizadas para conceitos e práticas mais modernas de engenharia de software e os cartões de projeto foram estendidos para conterem mais informações.

3.1 Cartões de Projetos

Na verdade, os cartões de projetos no jogo PnP são muito simples – Figura 2 (a) – o que motivou a criação de projetos mais bem elaborados de tal forma a deixar a simulação mais realística. Três novos tipos de informações foram adicionados a estes cartões: uma breve descrição textual do projeto, a forma de composição de módulos e referências para bibliografia relacionada. A Figura 2 (b) ilustra um exemplo de cartão de projeto no jogo SimulES em que é descrito um sistema multi-agentes para gerência e automação do processo de revisão de eventos científicos [Garcia *et al*. 2004]. Nesta figura também pode ser visto as seguintes informações presentes neste tipo de cartão:

- **Descrição**: texto em linguagem natural que descreve as principais características do projeto. Esta descrição torna o jogo mais realista e ajuda o jogador a compreender os principais requisitos do sistema. Abaixo da descrição são colocadas referências para artigos e livros relacionadas ao projeto ou seus principais conceitos.
- **Complexidade**: indica quantos pontos de tempo um engenheiro de software precisa gastar para completar um bom artefato. Artefatos de má qualidade custam metade deste valor. A complexidade do projeto pode ter valor 2 ou 4.



(a) PnP (b) SimulES

Figura 2: Exemplo de cartões de projeto no (a) jogo original e (b) SimulES

- **Tamanho**: indica quantos módulos integrados devem ser completados para integrar e terminar o projeto. O número máximo de módulos que um projeto pode ter é 6. O cartão indica também a forma que os módulos devem ser construídos. Por exemplo, o sistema da Figura 2 deve ser composto de 5 módulos e o primeiro deve conter duas cartas de requisitos (RQ), uma de desenho (DS) e uma de código (CD).
- **Qualidade**: representa o quão livre de defeitos deve estar o produto final. O valor varia entre 1 e 5 indicando o número mínimo de módulo sem defeitos necessários para se vencer o jogo. A qualidade do sistema é verificada na fase de entrega do produto.
- **Orçamento**: quantidade de dinheiro disponível para gastar com o projeto e será uma restrição para contratação de engenheiros de software bem como para o uso de cartas de conceitos. Durante qualquer momento do jogo o valor dos salários de todos os engenheiros somado com os custos dos conceitos (caso estes tenham valor) não deve ultrapassar o orçamento do projeto.

Tabela 1: Agrupamento das cartas por categorias

Categorias	Subcategorias	Nº de Cartas
Gerenciamento	Gerência Técnica, Gerência de Versão e Rastros, Treinamento	11
Recursos Humanos	Personalidade, Recursos Financeiros, Educação	11
Requisitos	Contexto, Problemas ou Alteração, Requisitos Não-Funcionais	12
Desenho	Padrões, *Frameworks*, Qualidade de Desenho	12
Código	Testes e Assertivas, Refatoração	12
Comunicação	Interface com Usuário, Documentação de Ajuda	10

Durante a evolução do jogo PnP para se chegar ao SimulES foi feita gerência de configuração pela equipe. Esta gerência inclui códigos, que não existiam no jogo original, para controle de versão das cartas de SimulES. Além disso, o controle também foi feito com a criação de grupos de cartas que tratam de um mesmo tema. Por exemplo, cartas referentes a requisitos receberam o código "RQ", desenho "DS", recursos humanos "RH" e outras seguem esse padrão. Com o agrupamento foram identificadas seis grandes categorias e subdivisões. A Tabela 1 mostra as categorias, suas divisões e o número total de cartas em cada categoria. Todos os grupos têm pelo menos 10 cartas para que seu conceito possa ser bem explorado no jogo. Além da classificação, novas cartas foram criadas para atingir o número mínimo.

3.2 O Tabuleiro

O tabuleiro é uma área na qual cada jogador coloca seus engenheiros de software em colunas e os artefatos em linhas. Os artefatos podem ser dos seguintes tipos: requisitos, desenhos, códigos, rastros e ajuda aos usuários (ajudas). Note que estes dois últimos não existiam na versão original do jogo. As cartas de rastros têm o intuito de interligar artefatos e contribuir para a gerência por requisitos, que é apoiada na rastreabilidade. Na Figura 3 é representado o tabuleiro em um cenário de jogo com a contratação de dois engenheiros. As cartas de artefatos são colocadas nas células do tabuleiro, abaixo do engenheiro que as produziram e nas linhas referentes aos seus tipos. Por exemplo, na linha de requisitos da Figura 3 estão dois artefatos feitos por Janaína e um por Carlos.

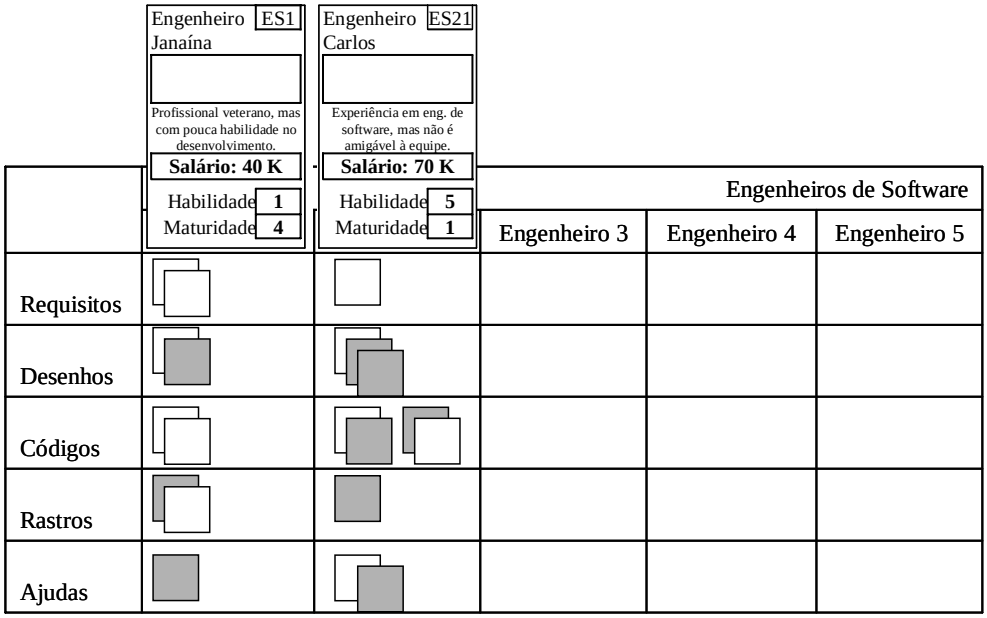


Figura 3: Tabuleiro de jogo com uma configuração de dois engenheiros

3.3 As Cartas

Os principais recursos do jogo SimulES são as cartas que se dividem em quatro tipos: problemas, conceitos, engenheiros de software e artefatos. Os artefatos podem ou não conter problemas (bug) e todas as cartas, exceto as de artefatos, possuem um nome e um código de identificação. Os quatro tipos de cartas são apresentados na Figura 4.

1. **Problemas**: descrevem problemas clássicos de engenharia de software resultantes de falhas no processo de produção. Essas cartas são utilizadas, para criar obstáculos ao progresso dos jogadores adversários. Além de um nome e um código, as cartas de problemas possuem os seguintes atributos (Figura 4, a): (i) literatura de apoio para referências que melhor caracterizam o referido problema; (ii) critério que descreve as condições a serem satisfeitas por um jogador para que seu oponente lhe jogue o problema; e (iii) efeito no jogador (ou em seus engenheiros) quando a carta é jogada. As cartas de problemas são as principais fontes de retorno negativo para o jogador quando este toma más decisões do ponto de vista da engenharia de software. Desta forma, o jogador pode reconhecer seus erros e associá-los a eventos que ocorrem no mundo real.
2. **Conceitos**: descrevem boas práticas de engenharia de software. Essas cartas podem ser utilizadas pelos jogadores para avançarem face ao seu objetivo. Os principais atributos das cartas de conceitos são (Figura 4, b): (i) uma literatura de apoio como nas cartas de problemas; (ii) efeito na configuração do jogo ou tabuleiro do jogador; e (iii) custo (quando presente na carta) que incorre em gastos após o conceito ser aplicado.

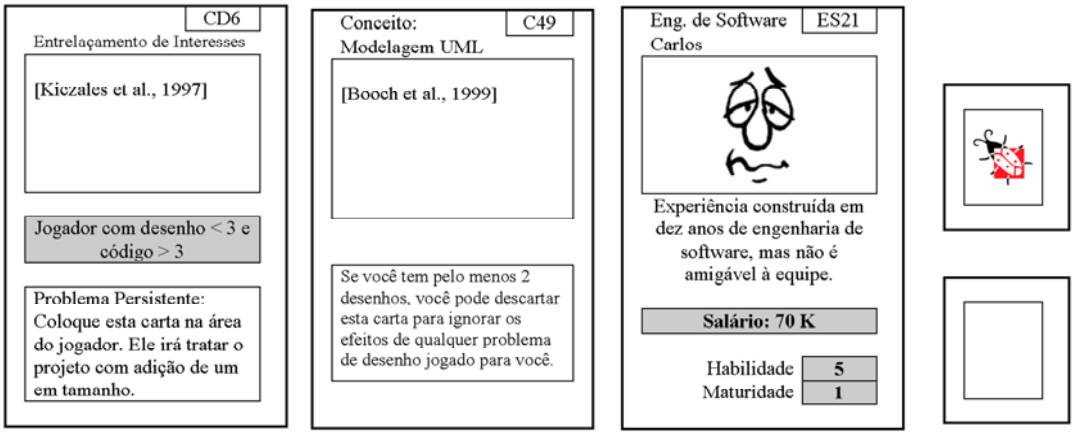


Figura 4: Cartas do jogo: (a) problema, (b) conceito, (c) engenheiro de software e (d) artefatos com e sem bug

3. **Engenheiros de Software**: este tipo de carta é o principal recurso que o jogador terá para progredir no jogo. Os engenheiros produzem artefatos que são necessários para cumprir o projeto. Estas cartas apresentam as seguintes informações (Figura 4, c): (i) nome e descrição pessoal do engenheiro; (ii) salário a ser pago ao funcionário que deve respeitar o limite de orçamento do projeto; (iii) habilidade ou número de "pontos de tempo" que um engenheiro possui a cada rodada para desempenhar ações no jogo; e (iv) maturidade que reflete a tendência do engenheiro de software em ser um bom trabalhador. A habilidade e a maturidade são medidas em uma escala de 1 a 5.

4. **Artefatos**: simbolizam os produtos produzidos pelos engenheiros de software que podem ou não conter problemas (Figura 4, d). Além disso, os artefatos podem ter duas cores, branco ou cinza, dependendo de sua qualidade. Apesar de gastarem o dobro dos "pontos de tempo" para serem produzidas, as cartas de cor branca contêm defeitos na proporção de 5 cartas para 1 defeito enquanto nas cartas de cor cinza esta proporção é de 3 para 2. Ou seja, com a prática o jogador aprende que, geralmente, é melhor desenvolver um artefato branco, mesmo que este seja mais trabalhoso. Para vencer no jogo, é preciso completar os módulos do projeto através da composição dos artefatos.

3.4 A Dinâmica do Jogo

Antes do início do jogo, um cartão de projeto é escolhido aleatoriamente de uma série de projetos disponíveis. As informações deste cartão devem ficar visíveis a todos os jogadores. Em seguida, cada jogador monta seu tabuleiro de jogo e as cartas são separadas em quatro montes: um para engenheiros de software, outro para problemas e conceitos, um para artefatos branco e outro para artefatos cinza. Com o dado escolhe-se quem começa o jogo e o jogo deve prosseguir no sentido horário.

A cada jogada, o jogador da vez lança o dado e de acordo com o número tirado, retira cartas nos montes dependo do valor obtido. Se o jogador tirar entre 1 e 3 no dado ele terá direito de pegar no monte de problemas e conceitos o número de cartas indicadas pelo dado, mas não poderá pegar nenhuma carta de engenheiro. Se o jogador tirar entre 4 e 6 no dado, ele irá pegar 3 cartas de problemas e conceitos e a diferença (dado – 3) no monte de engenheiros de software. Por exemplo, se na vez da jogadora Maria o número tirado for 2, ela compra duas cartas no monte de problemas e conceitos. Por outro lado, se o número tirado for 5, Maria retira 3 cartas no monte de problemas e conceitos e 2 (5 – 3) engenheiros de software. As cartas de problemas e conceitos são guardadas na mão do jogador até o momento que ele achar oportuno para jogá-las. As cartas de engenheiros também podem ser guardadas nas mãos ou então podem ser colocadas imediatamente no tabuleiro, o que indica a contratação do funcionário. Entretanto, o jogador pode ficar com no máximo seis cartas nas mãos, incluindo conceitos, problemas e engenheiros de software. Caso haja mais de seis cartas com o jogador, este deverá descartar as cartas excessivas antes do final de sua jogada.

Cartas de artefatos são retiradas dos montes de acordo com os engenheiros de software já presentes no tabuleiro do jogador. O jogador pode retirar tantos artefatos quanto lhe convier, desde que estes estejam dentro da produtividade de sua equipe de engenheiros de software. As cartas de artefatos brancas requerem pontos de habilidade padrão enquanto as cinzas requerem a metade dos pontos de habilidade. Ou seja, com o mesmo esforço um engenheiro pode produzir duas vezes mais artefatos de má qualidade do que artefatos de boa qualidade.

Ao fim de sua jogada, o jogador está apto a receber as cartas de problemas de seus adversários. Ele pode receber problemas dos três jogadores que jogaram imediatamente antes dele. As cartas de problemas recebidas não alteram diretamente o estado do tabuleiro e devem ser guardadas para a rodada seguinte. No inicio de sua jogada na rodada seguinte, se o jogador tem em mãos uma carta de conceito que invalida o problema, então ele diz que a carta problema não se aplica e descarta tanto a carta de problema quanto a carta de conceito. Além disso, durante a jogada, um engenheiro de software pode exercer uma série de tarefas. A habilidade do engenheiro

determina quantos "pontos de tempo" ele tem e, portanto, quantas ações ele pode desempenhar. Os engenheiros têm quatro opções de tarefas como descritas a seguir:

1. **Construir artefato**: o engenheiro pode construir artefatos de cor branca ou cinza gastando o valor de "pontos de tempo" indicado pela complexidade do projeto. É importante lembrar que construindo artefatos de cor cinza o engenheiro paga metade da complexidade do projeto.

2. **Inspeção de artefato**: esta tarefa se caracteriza por desvirar uma das cartas de artefato que estão com a face virada para baixo. Se um artefato apresenta um defeito, esta carta pode potencialmente causar problemas quando o projeto for terminado.

3. **Corrigir defeito**: após inspecionar o artefato e descobrir um defeito, o engenheiro pode corrigi-lo gastando um "ponto de tempo". Para isso ele substitui a carta com defeito por uma nova carta de mesma qualidade (branca ou cinza) do monte de artefatos.

4. **Integrar artefatos em um módulo**: esta tarefa pode ser feita quando houver artefatos suficientes no tabuleiro do jogador para compor um módulo requerido pelo projeto. A integração retira os artefatos do tabuleiro e os coloca separados no monte de integração.

Uma vez terminado os componentes necessários para o termino do projeto, o jogador pode afirmar que completou o projeto. Nesse momento é feita a validação por parte do cliente. Isto significa que alguns dos módulos do projeto são aleatoriamente verificados e devem estar livres de problemas. O número de módulos a serem verificados na validação é igual ao valor da qualidade indicado na carta de projeto. O jogador somente é declarado vencedor se em nenhum dos módulos verificados for encontrado defeito.

4 Avaliação Comparativa

Esta seção faz uma comparação qualitativa entre o jogo PnP e sua versão proposta na Seção 3. O foco desta comparação está nas limitações de PnP identificadas na Seção 2.1 de tal forma a explicar como estas foram tratadas em SimulES. Abaixo os problemas da Seção 2.1 são novamente listados, porém, seguidos pela solução adotada no novo jogo.

- *Muito amarrado a um único processo de software*. Em SimulES é dado liberdade para o jogador escolher a abordagem de desenvolvimento de seu interesse com seus prós e contras. Assim, o jogo não é mais limitado ao modelo cascata uma vez que não obriga a utilização do processo totalmente seqüencial e permite o jogador voltar para corrigir problemas nas fases anteriores. Por exemplo, mesmo que o jogador esteja na fase de codificação, caso ele sinta necessidade, pode voltar a trabalhar nos requisitos ou desenho.
- *Utilização de cartas muito abstratas e sem informações adicionais*. Permitir que as cartas se tornassem mais claras não é tarefa trivial, visto que elas possuem um limite de tamanho e muita informação nas cartas pode tornar o jogo desinteressante. Uma solução adotada para minimizar este problema foi adicionar referências para bibliografia relacionada ao conteúdo da carta de tal forma que qualquer jogador (ou mesmo o professor da disciplina) possa consultar. Além de ser uma fonte de explicação para a carta, a bibliografia tem ainda o papel de ampliar o conhecimento em um determinado tema.

- *Jogo não desperta entusiasmo dos jogadores nas primeiras rodadas*. Com o intuído de deixar o jogo mais dinâmico mesmo em seu início, as regras foram alteradas para permitir que problemas sejam jogados em qualquer artefato (e não apenas em código). Esta nova abordagem é plausível, pois um sistema de software real pode apresentar problemas em todas as fases do processo. Além disso, eliminamos a obrigatoriedade de seguir o modelo cascata.
- *O jogo não limita o número de jogadores, mas suas regras dificultam que mais pessoas joguem*. Em SimulES a quantidade de jogadores deve ser entre 4 e 8. Além disso, alguns pontos da regra foram alterados para evitar que um número maior de jogadores iniba o andamento do jogo. Por exemplo, na nova versão, cada jogador recebe no máximo 3 problemas por rodada (dos três oponentes que jogam antes dele).
- *Ausência de um tabuleiro para organização da área de projeto do jogador*. O tabuleiro foi idealizado para organizar os engenheiros de software e os artefatos pertencentes ao projeto do jogador (vide Figura 3).
- *Ausência de um mapeamento claro entre os artefatos do jogo e os conceitos de engenharia de software*. Este problema é resolvido em SimulES a partir de duas estratégias: (i) utilizando as referências é possível saber os conceitos associados a uma carta; e (ii) os identificadores e as categorias (Tabela 1) permitem chegar ao conjunto de artefatos que atendem ao mesmo conceito.

5 Trabalhos Relacionados

A utilização de jogos como forma de melhorar o aprendizado de crianças e adultos é um tema bem explorado na literatura [Oh e Hoek 2001] [Virvou *et al* 2005]. Mesmo em disciplinas de computação existem algumas iniciativas para utilização de jogos ou simuladores, como por exemplo, para o ensino de Computação Gráfica [Battaiola 2002] e Redes de Computadores [Pinheiro e Ribeiro 2005]. Bittencourt e Giraffa [2003] apresentam um trabalho sobre a utilização de RPG (Role-Playing Games) no processo de ensino-aprendizagem e disponibiliza um framework pare este propósito. Apesar de existirem tais trabalhos explorando a importância de jogos na Computação, poucos são focados no ensino de engenharia de software.

Além do jogo PnP discutido na Seção 2, uma abordagem para ensino de engenharia de software é proposta por Drappa e Ludewig [2000]. Em seu trabalho, Drappa e Ludewig apresentam o projeto SESAM que é constituído principalmente de um simulador para treinamento em engenharia de software. Este simulador permite que um único usuário assuma o papel de gerente e utilize modelos e regras complexas no desenvolvimento de um sistema. Entretanto, o projeto SESAM não possui características de jogos, tais como entretenimento e competitividade. Além disso, tal simulador é destinado principalmente a profissionais e não a estudantes de computação.

A equipe idealizadora de PnP, também disponibiliza uma versão digital do jogo chamada SimSE [Navaro *et al*. 2004]. Em sua versão para computador, o jogo permite que um jogador assuma o papel de gerente de projeto e desempenhe atividades como contratar e demitir programadores, associar tarefas, monitorar o progresso, entre outras. Por ser em sua essência adaptação de PnP, o jogo SimSE recai nos mesmos problemas discutidos na Seção 2.1 como limitação ao Modelo Cascata e falta de ligação explicita dos artefatos do jogo com os conceitos de engenharia de software. Em adição,

o jogo perde muito de seu atrativo por não permitir competição entre múltiplos jogadores.

6 Conclusões e Trabalhos Futuros

Neste artigo é apresentado um jogo educacional que simula o processo de desenvolvimento de software desde a fase de concepção até a fase de entrega do produto. As contribuições deste artigo podem ser vistas sobre dois aspectos: o jogo propriamente dito e uma avaliação dos conceitos envolvidos. Em relação ao jogo, o artigo apresenta melhorias ao jogo original tanto para a parte conceitual quanto para a dinâmica. Tais melhorias têm o objetivo de ensinar práticas mais modernas de engenharia de software, tais como evolução de software, bem como tornar o jogo mais atrativo. Do ponto de vista dos conceitos envolvidos, o artigo apresenta uma discussão dos problemas do jogo original indicando soluções para tais problemas. Maiores informações sobre o jogo SimulES encontram-se disponível na página [SimulES 2006].

Como trabalho futuro, é sugerido uma avaliação mais sistemática do jogo utilizando alunos com diferentes graus de conhecimento em engenharia de software. Além disso, pretende-se estender o jogo de tal forma a atender conceitos não somente da Computação, mas também de outras áreas do conhecimento, tais como Administração, Economia, Sociologia, dentre outras. É proposta ainda a criação de uma versão digital do jogo utilizando recursos tecnológicos avançados e alta interatividade entre jogadores. Este último trabalho é fundamental para que o jogo seja mais bem difundido e alcance o propósito de oferecer novas alternativas para o ensino tradicional, especialmente em engenharia de software.

Referências Bibliográficas

BAKER, A.; NAVARRO, E.; HOEK A. "**An Experimental Card Game for Teaching Software Engineering Processes**". In: Journal of Systems and Software, vol. 75, issue 1-2, 2005, pages 3-16.

BATTAIOLA, A.; ELIAS, N.; DOMINGUES, R.; ASSAF, R.; RAMALHO, G. "**Desenvolvimento da Interface de um Software Educacional com Base em Interfaces de Jogos**". In: Simpósio Brasileiro de Informática na Educação (SBIE), 2002.

BECK, K. "**Extreme Programming Explained**". Addison-Wesley Longman, 1999.

BITTENCOURT, J. R; GIRAFFA, L. M. "**Modelando Ambientes de Aprendizagem Virtuais utilizando Role-Playing Games**". In: Simpósio Brasileiro de Informática na Educação (SBIE), 2003.

BOEHM, B. "**A Spiral Model of Software Development and Enhancement**". ACM SIGSOFT Software Engineering Notes, vol. 11, issue 4, 1986 , pages 14-24.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. "**The Unified Modeling Language User Guide**". Addison-Wesley, 1st edition, 1999.

CHUNG, L. K.; NIXON, B.; YU, E.; MYLOPOULOS, J. "**Non-Functional Requirements in Software Engineering**". Kluwer Publishing, 2000.

DRAPPA, A.; LUDEWIG, J. "**Simulation in Software Engineering Training**". In: International Conference on Software Engineering, 2000, pages 199-208.

GARCIA, A.; SANT'ANNA, C.; CHAVEZ, C.; SILVA, V.; LUCENA, C.; STAA, A. "**Separation of Concerns in Multi-Agent Systems: An Empirical Study**". In: Software Engineering for Multi-Agent Systems II, Springer, LNCS 2940, 2004.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J.-M.; IRWIN, J. "**Aspect-Oriented Programming**". In: European Conference on Object-Oriented Programming (ECOOP), pages 220-242. Finland, 1997.

MYLOPOULOS, J., CHUNG, L., YU, E. "**From Object-oriented to Goal Oriented Requirements Analysis**". In: Communications of the ACM, Vol. 42, No. 1, pages 31-37, Jan. 1999.

NAVARRO, E.; BAKER, A.; HOEK, A. "**Teaching Software Engineering Using Simulation Games**". In: International Conference on Simulation in Education (ICSIE). California, 2004.

OH, E.; HOEK, A. "**Adapting Game Technology to Support Individual and Organizational Learning**". In: International Conference on Software Engineering & Knowledge Engineering (SEKE), pages 347–354. June 13-15, Argentina, 2001.

"**Problems and Programmers Home Page**". Disponível on-line em http://www.problemsandprogrammers.com. Acessado em Agosto, 2006.

PINHEIRO, C.; RIBEIRO, M. "**LVR Laboratório Virtual de Redes - Protótipo para Auxílio ao Aprendizado em Disciplinas de Redes de Computadores**". In: Simpósio Brasileiro de Informática na Educação (SBIE), 2005.

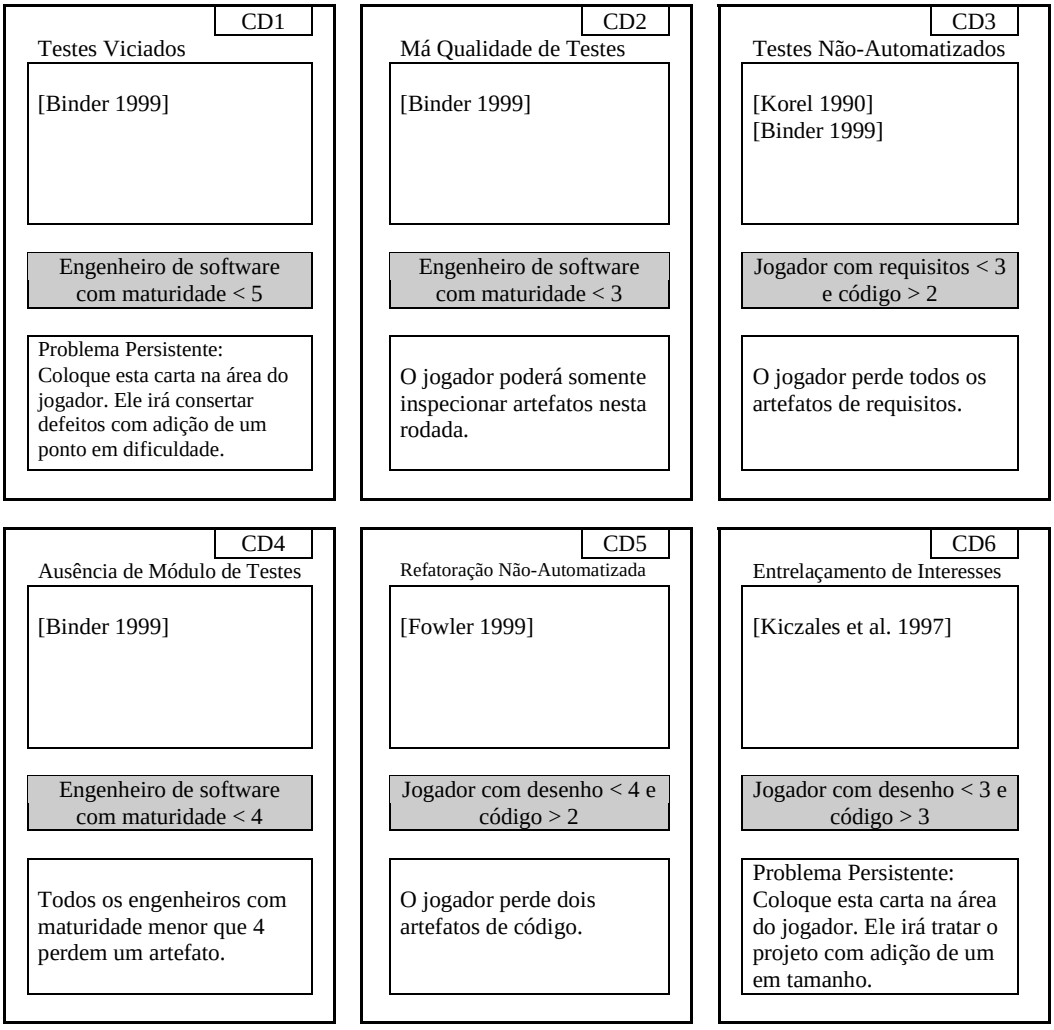
"**SimulES: Simulador de Engenharia de Software**". Disponível on-line em http://www.teccomm.les.inf.puc-rio.br/emagno/simules/. Acessado em Agosto, 2006.

VIRVOU, M.; KATSIONIS, G.; MANOS, K. "**Combining Software Games with Education: Evaluation of its Educational Effectiveness**". In: Educational Technology & Society, 2005, pp 54-65.

Apêndice 1 Cartas de Artefatos

Apêndice 2 Cartas de Problemas

A2.1 Cartas de Problemas de Código



CD7

Espalhamento de Interesses

[Kiczales et al. 1997]

Jogador com desenho < 4 e código > 3

O jogador perde um artefato de desenho e um de código.

CD8

Código Não-Padronizado

[Staa 2000]

Engenheiro de software com maturidade < 2

Os engenheiros com maturidade menor que 2 são demitidos.

CD9

Inconsistência entre Artefatos

[Gotel and Finkelstein 1994]

Jogador com rastro < 3 e código > 4

Problema Persistente: Coloque esta carta na área do jogador. Ele irá consertar defeitos com adição de um ponto em dificuldade.

CD10

Linguagem Inapropriada

[Sethi 1996]

Jogador com requisitos < 4 e código > 3

O jogador perde três artefatos de código.

CD11

Impossibilidade de Reuso

[Jacobson et al. 1997]
[Krueger 1992]
[Sommerville 2000, cap. 20]

Jogador com requisitos < 2 e código > 3

Problema Persistente: Coloque esta carta na área do jogador. Ele irá tratar o projeto com adição de um em tamanho.

CD12

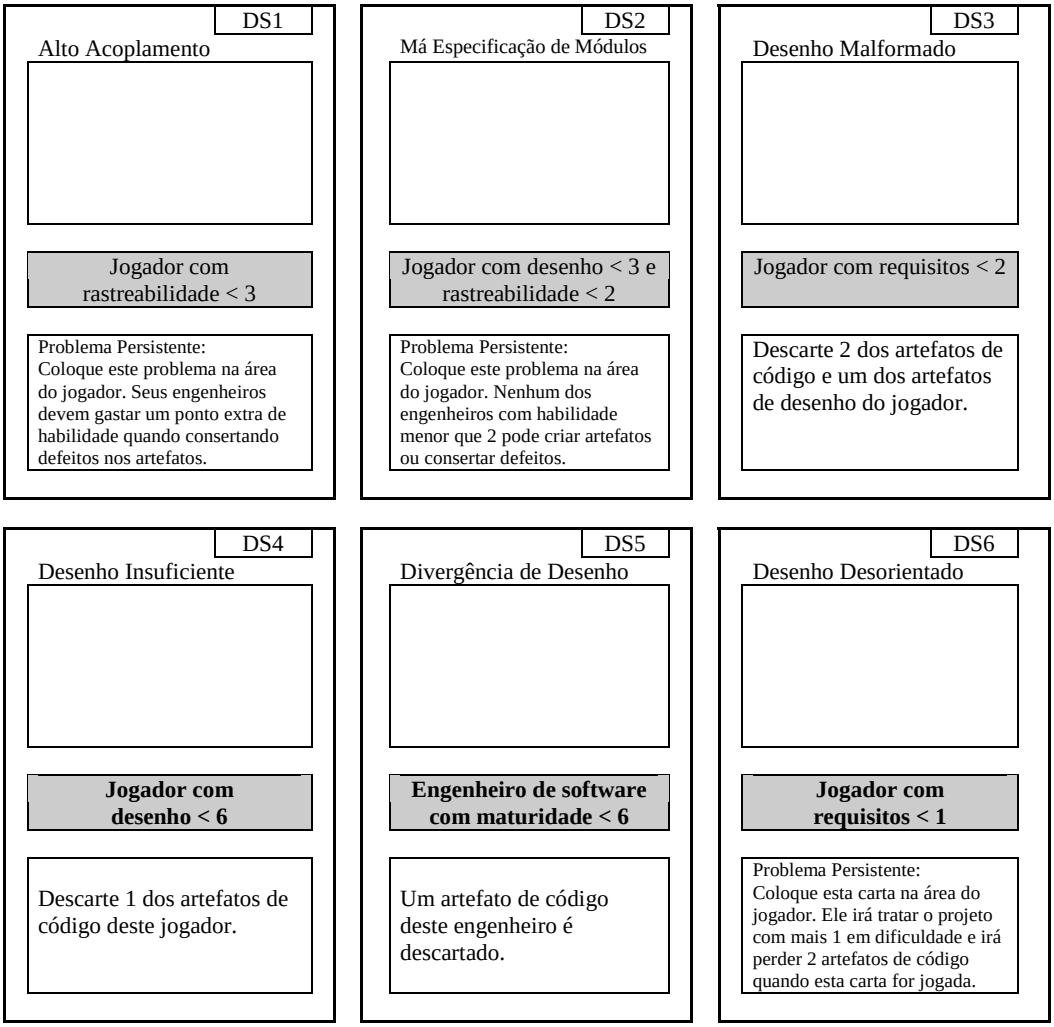
Replicação de Código

[Fowler 1999, cap. 3]
[Jacobson et al. 1997]

Jogador com rastro < 4 e código > 2

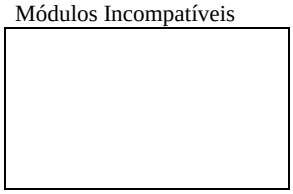
O jogador perde um artefato de código.

A2.2 Cartas de Problemas de Desenho



DS7

Módulos Incompatíveis

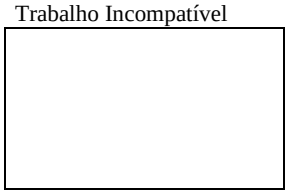


Jogador com desenho < 2

Escolha e descarte metade dos artefatos do jogador (arredonde a perda para baixo).

DS8

Trabalho Incompatível

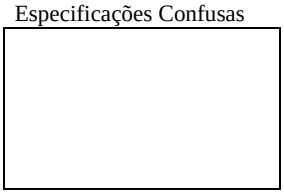


Jogador com desenho < 3

Descarte um artefato de código, um de desenho e um de rastro do jogador.

DS9

Especificações Confusas

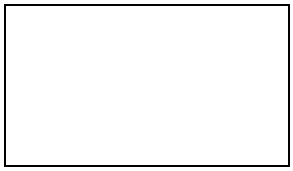


Qualquer jogador

Descarte todos os artefatos cinza de requisito e desenho dos engenheiros do jogador.

DS10

Baixa Coesão

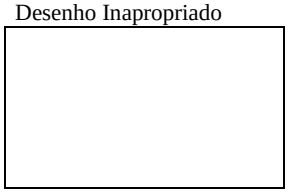


Jogador com desenho < 3

Problema Persistente:
Coloque esta carta ao lado do jogador. Seus engenheiros devem inspecionar o código gastando mais um ponto em habilidade.

DS11

Desenho Inapropriado

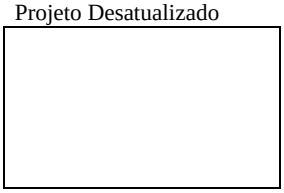


Jogador com desenho < 2

Problema Persistente:
Coloque este problema ao lado do jogador. Ele deve tratar o projeto como se tivesse um ponto a mais de dificuldade.

DS12

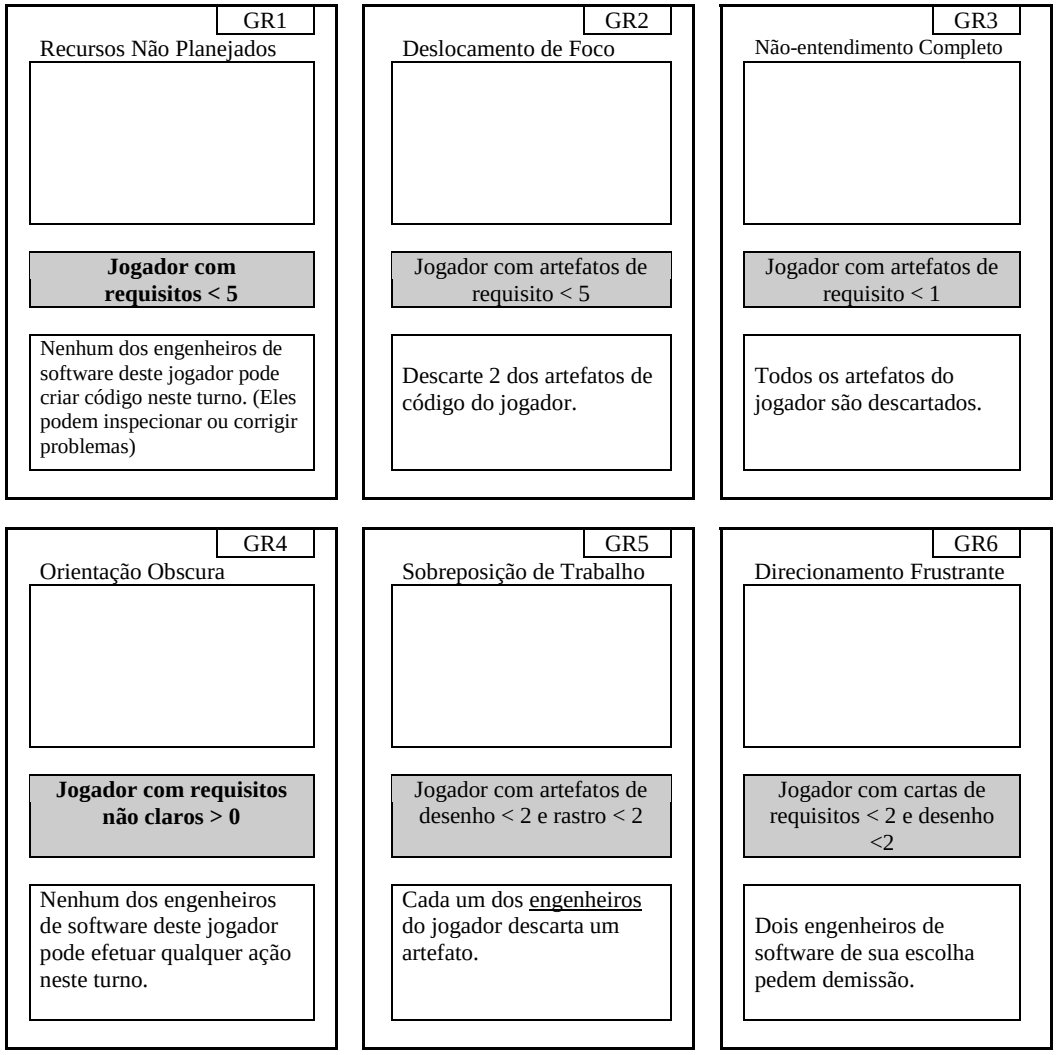
Projeto Desatualizado



Engenheiro de software com maturidade < 2

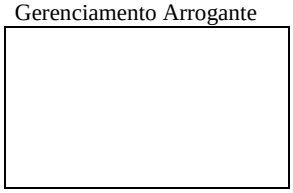
Descarte um artefato de desenho para cada engenheiro de software que possui maturidade < 2.

A2.3 Cartas de Problemas de Gerência



GR7

Gerenciamento Arrogante

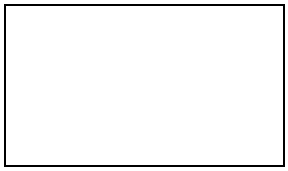


Afeta todos os jogadores

Cada jogador (inclusive você) conta seus cartões de conceito. Cada jogador com mais de 3 destes cartões deve escolher e descartar alguns até que tenha uma quantidade igual a 3.

GR8

Plataforma Errada

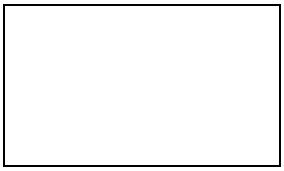


Jogador com requisitos < 2

Cada um dos engenheiros de software do jogador em questão descarta seu cartão de código posicionado mais inferiormente.

GR9

Idéia Geral Errada

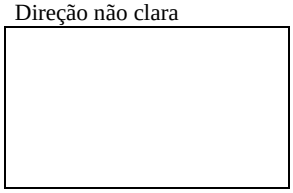


Jogador com requisitos não claro > 2

Se o jogador em questão possui mais que 3 cartões de código, escolha apenas 3 destes cartões para o jogador. O resto é descartado.

GR10

Direção não clara

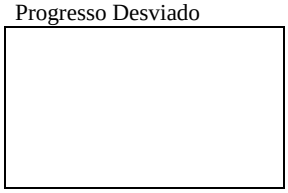


Jogador com requisitos < 4

Nenhum dos programadores do jogador pode realizar qualquer ação nesta rodada.

GR11

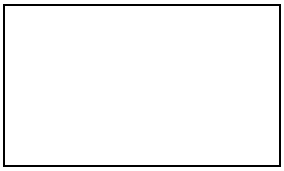
Progresso Desviado



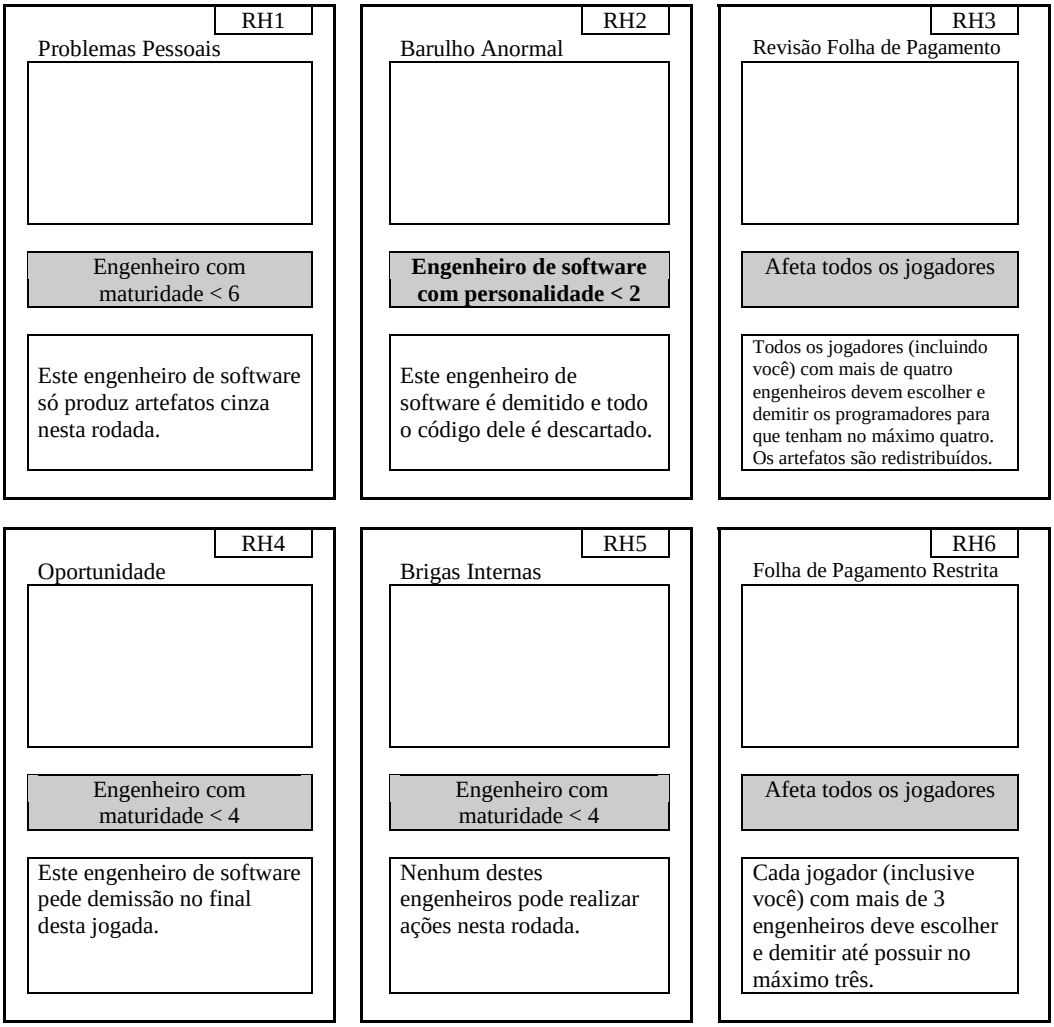
Jogador com requisitos < 2

Cada um dos programadores do jogador deve descartar duas de suas cartas de código superiores.

GR12



A2.4 Cartas de Problemas de Recursos Humanos



RH7

Colega de Trabalho Ofendido

Engenheiro com maturidade < 2

Escolha outro engenheiro controlado pelo mesmo jogador. Este engenheiro escolhido pedirá demissão.

RH8

Trabalho Isolado

Engenheiro de software com personalidade < 5

Problema Persistente:
Coloque este problema debaixo do engenheiro de software em questão. Quando este engenheiro de software der ou receber ajuda, incorre em 2 pontos adicionais de penalidade.

RH9

Política Ignorada

Engenheiro de software com personalidade < 3

Se pelo menos dois dos engenheiros de software do jogador em questão possuem 2 ou menos em personalidade, escolha e descarte um de seus conceitos.

RH10

Vírus

Engenheiro de software com maturidade < 3

Este engenheiro de software perde 2 de seus artefatos. O mesmo ocorre com os outros engenheiros deste jogador que tenham maturidade < 3.

RH11

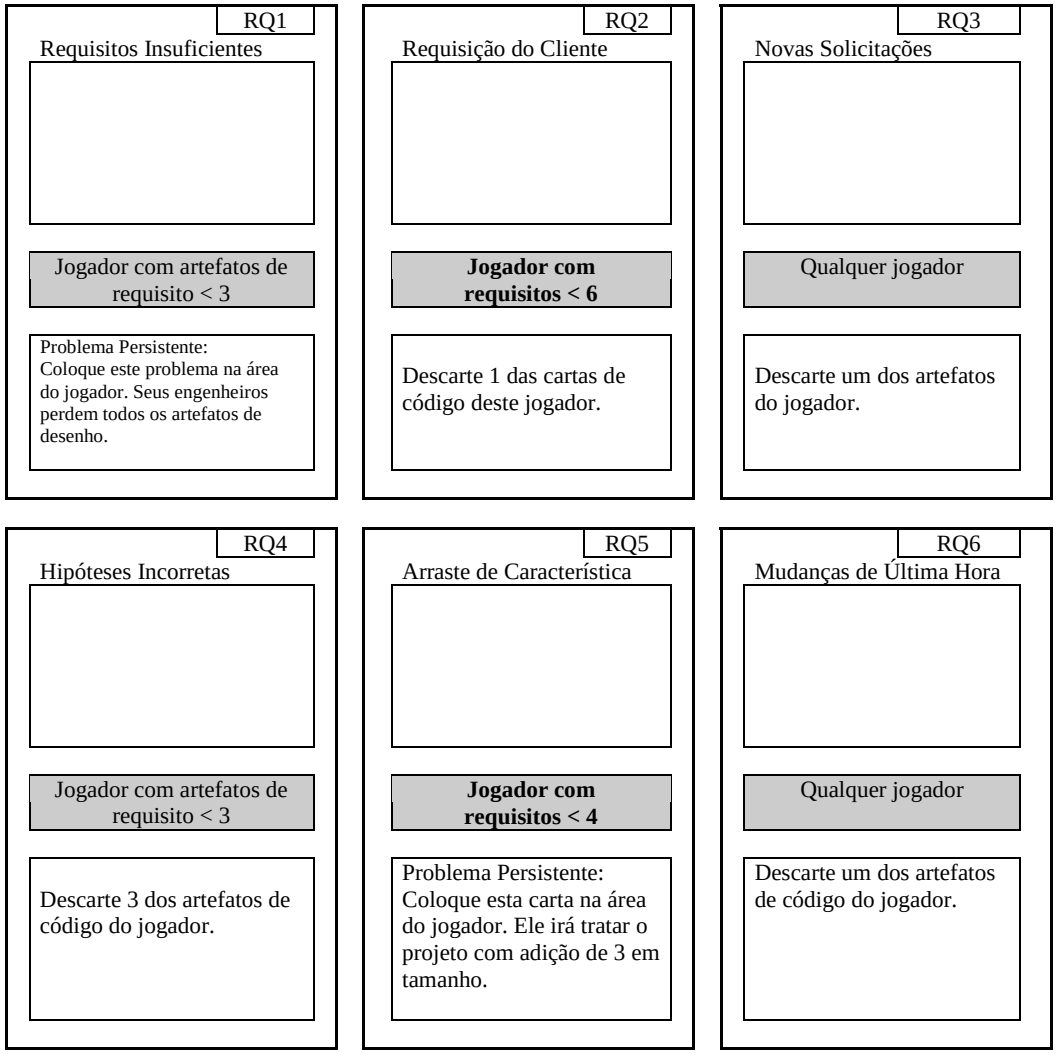
Doença

Engenheiro de software com maturidade < 5

Este engenheiro não pode realizar qualquer ação nesta rodada.

RH

A2.5 Cartas de Problemas de Requisitos



RQ7

Características Insuficientes

Jogador com requisitos não claro > 1

Problema Persistente:
Coloque este problema ao lado do jogador em questão. Ele deve tratar o projeto como se este tivesse 2 a mais de tamanho.

RQ8

Falha de Hardware

Qualquer jogador

Descarte 2 carta de código deste jogador.

RQ9

Falta de Requisitos

Engenheiro de software com maturidade < 4

Este engenheiro descarta dois de seus artefatos.

RQ10

Falta de Requisito

Jogador com desenho < 4

Problema Persistente:
Coloque esta carta ao lado do jogador. Ele deve tratar o tamanho do projeto como aumentado de 2.

RQ11

Requisitos não atualizáveis

Programador com personalidade < 2

Descarte uma carta de requisitos para cada programador com personalidade < 2.

RQ12

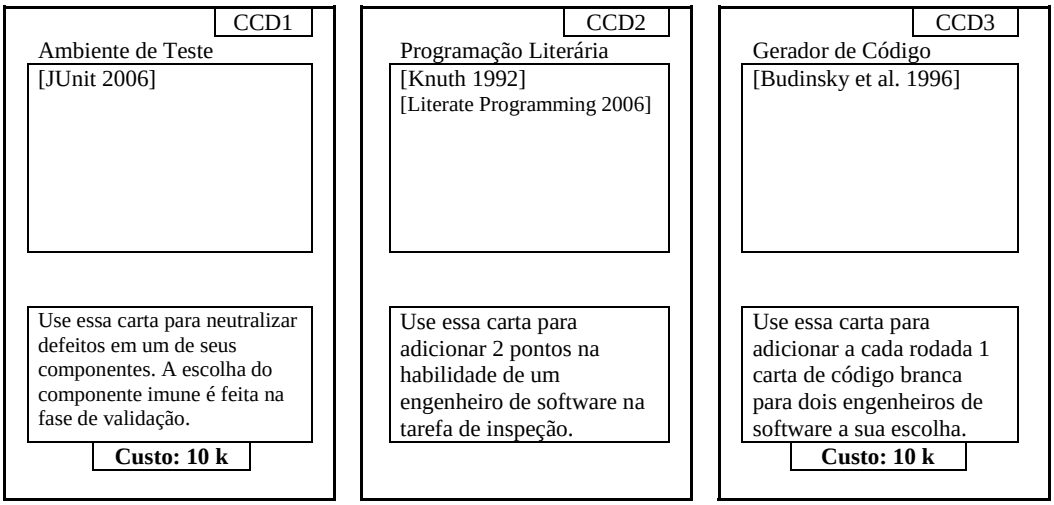
Faltando Características

Jogador com requisitos < 3

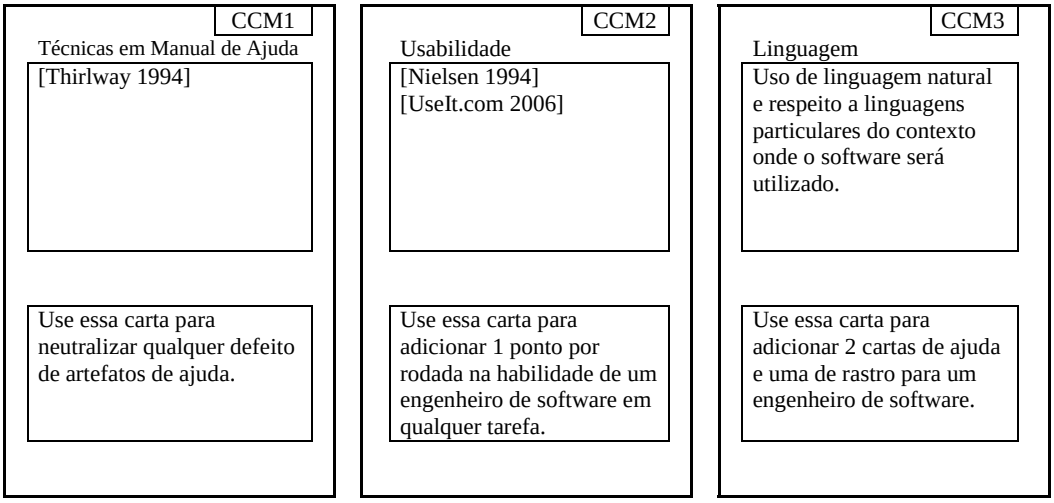
Problema Persistente:
Coloque esta carta ao lado do jogador. Ele deverá tratar o projeto como se fosse aumentado de 4 no tamanho.

Apêndice 3 Cartas de Conceitos

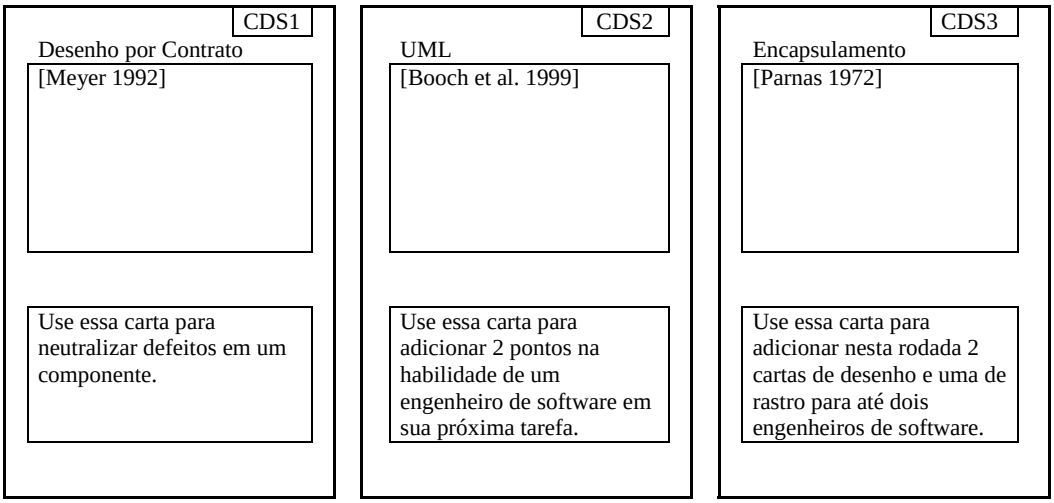
A3.1 Cartas de Conceitos de Código



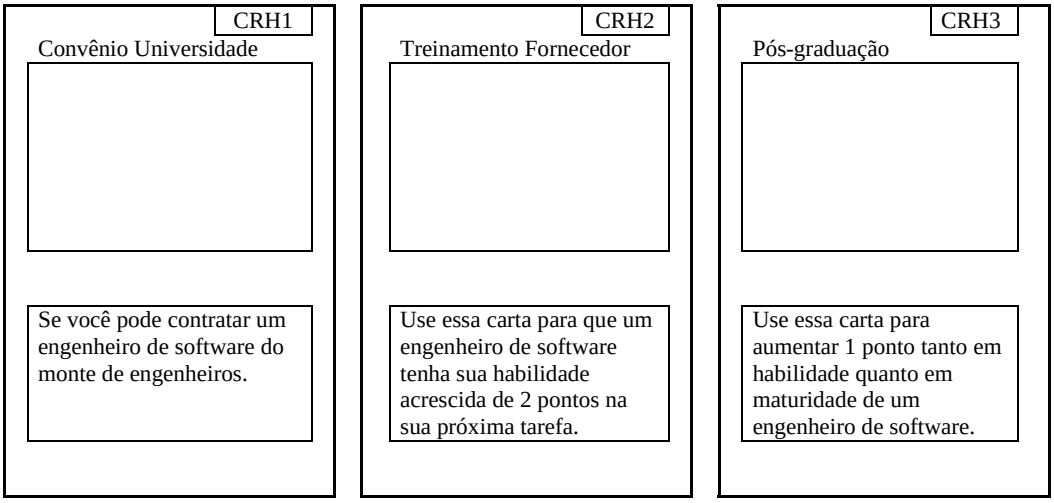
A3.2 Cartas de Conceitos de Comunicação



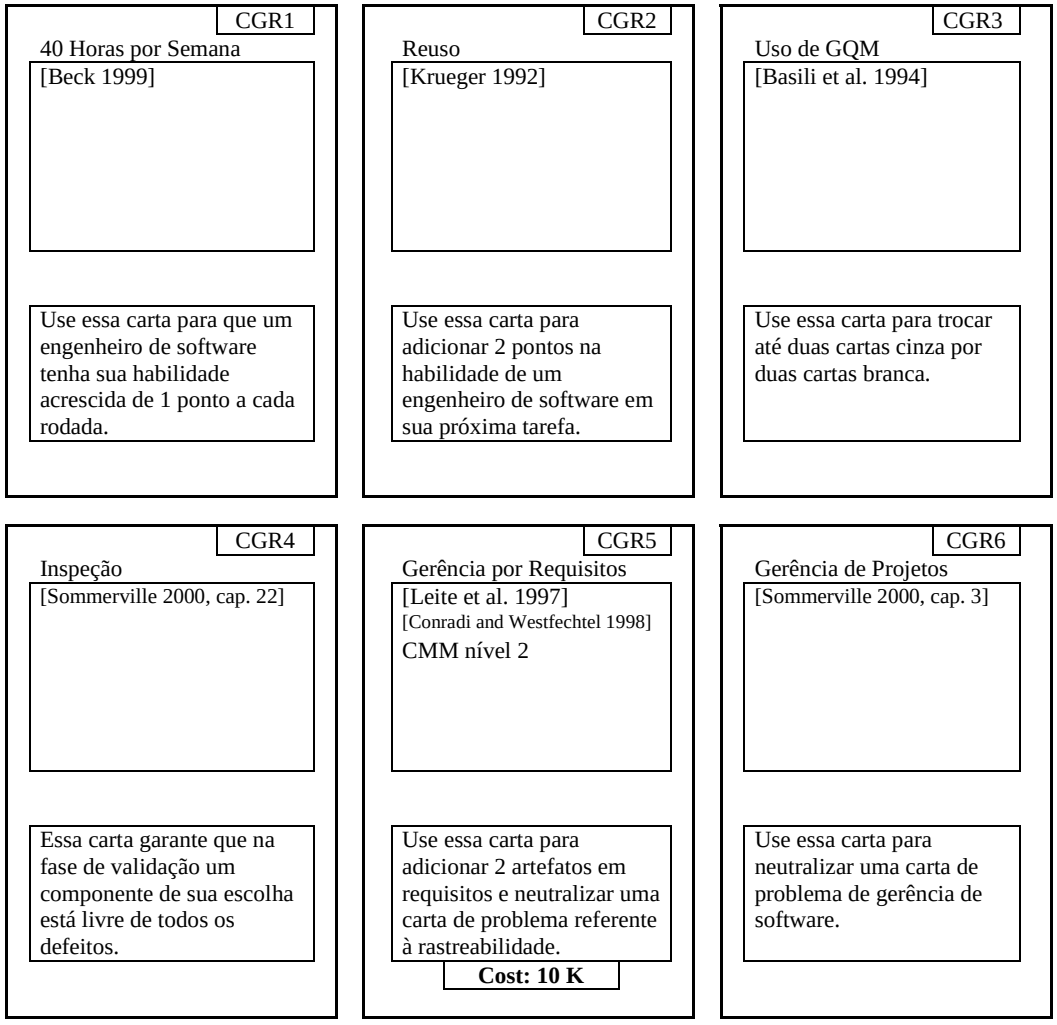
A3.3 Cartas de Conceitos de Desenho



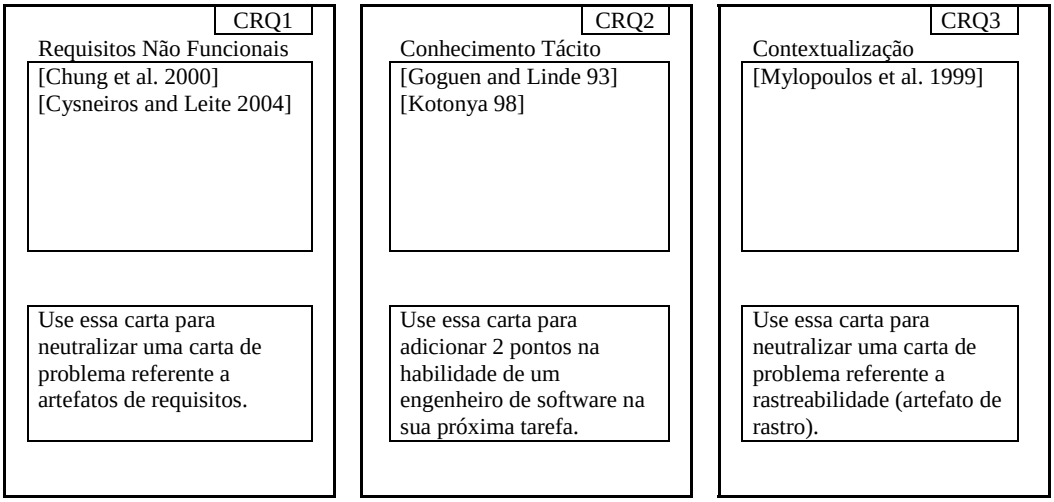
A3.4 Cartas de Conceitos de Recursos Humanos



A3.5 Cartas de Conceitos de Gerência



A3.6 Cartas de Conceitos de Requisitos



Apêndice 4 Cartas de Engenheiros de Software

Eng. de Software	ES1
Janaína	

Profissional veterana, mas com pouca habilidade no desenvolvimento.

Salário: 40 K

Habilidade	1
Maturidade	4

Eng. de Software	ES2
Roberto	

Alguma boa experiência, mas frequentemente é excessivamente cauteloso.

Salário: 40 K

Habilidade	1
Maturidade	5

Eng. de Software	ES3
Manuel	

Trabalha há dois anos na indústria. Perdeu seu último trabalho após uma briga na equipe.

Salário: 40 K

Habilidade	2
Maturidade	1

Eng. de Software	ES4
Sidney	

Jovem promessa com boa mentalidade para negócios.

Salário: 60 K

Habilidade	2
Maturidade	4

Eng. de Software	ES5
Monalisa	

Tem trabalhado em muitos empregos durante anos, mas encontra dificuldade em se posicionar na equipe.

Salário: 50 K

Habilidade	2
Maturidade	2

Eng. de Software	ES6
Karen	

Ainda em início de carreira, mas é entusiasmada com novos aprendizados.

Salário: 50 K

Habilidade	2
Maturidade	3

Eng. de Software	ES7

Augusto

Não tem programado há 20 anos, mas conhece um pouco de negócios.

Salário: 30 K

Habilidade	1
Maturidade	3

Eng. de Software	ES8

Michael

Recém formado. Tem muito a aprender sobre práticas de negócios.

Salário: 20 K

Habilidade	1
Maturidade	2

Eng. de Software	ES9

Estevão

O filho do chefe. Procure deixá-lo ajudar.

Salário: 0 K

Habilidade	1
Maturidade	1

Eng. de Software	ES10

Bob

Cinco anos de experiência em desenvolvimento e gerência de software.

Salário: 70 K

Habilidade	3
Maturidade	3

Eng. de Software	ES11

Milena

Engenheira de software experiente, mas se sente um pouco inferiorizada.

Salário: 60 K

Habilidade	3
Maturidade	2

Eng. de Software	ES12

Ricardo

Experiente engenheiro de software, mas um pouco lento em programação.

Salário: 60 K

Habilidade	2
Maturidade	4

Eng. de Software	ES13

Bartolomeu

Funcionário confiável com bom relacionamento com a equipe.

Salário: 60 K

Habilidade	3
Maturidade	3

Eng. de Software	ES14

Jennifer

Muito experiente, habilidosa e com excelentes referências.

Salário: 100 K

Habilidade	5
Maturidade	4

Eng. de Software	ES15

Donna

Doutora em Ciência da Computação, boa habilidade em programação, mas sem muita experiência em sistemas reais.

Salário: 80 K

Habilidade	4
Maturidade	2

Eng. de Software	ES16

Samuel

Tem problemas pessoais, mas sua habilidade em desenvolvimento o faz uma boa opção.

Salário: 50 K

Habilidade	3
Maturidade	1

Eng. de Software	ES17

Felipe

Pouca educação formal, mas formado pela experiência prática.

Salário: 70 K

Habilidade	3
Maturidade	3

Eng. de Software	ES18

Hernani

Muita experiência e maturidade. Apaixonado pela Paula.

Salário: 70 K

Habilidade	3
Maturidade	4

Eng. de Software	ES19
Paula	

Boa experiência e maturidade.

Salário: 70 K

Habilidade	3
Maturidade	4

Eng. de Software	ES20
Maria	

Tem trabalhado para esta empresa há alguns anos. Leal e pronto para aprender novas coisas.

Salário: 70 K

Habilidade	2
Maturidade	5

Eng. de Software	ES21
Carlos	

Experiência construída em dez anos de engenharia de software, mas não é amigável à equipe.

Salário: 70 K

Habilidade	5
Maturidade	1

Eng. de Software	ES22
Silas	

Trabalha a sete anos nesta empresa e se interessa por novos projetos.

Salário: 90 K

Habilidade	4
Maturidade	4

Eng. de Software	ES23
Aline	

Tem trabalhado em muitas empresas. Possui um sólido conhecimento no desenvolvimento de software.

Salário: 90 K

Habilidade	4
Maturidade	5

Eng. de Software	ES24
Natália	

Engenheira de software com sete anos de experiência.

Salário: 80 K

Habilidade	4
Maturidade	3

Eng. de Software	ES25
Marcos	

Bastante dedicado ao seu trabalho e de bom conhecimento.

Salário: 80 K

Habilidade	3
Maturidade	5

Eng. de Software	ES26
Nélio	

Oito anos de experiência, mas pouca habilidade em inglês e pouco entusiasmo para novos aprendizados.

Salário: 70 K

Habilidade	4
Maturidade	1

Eng. de Software	ES27
Horácio	

Mestre em Informática e sólido background.

Salário: 80 K

Habilidade	4
Maturidade	3

Eng. de Software	ES28
Nancy	

Experiência em alguns projetos de software complexos.

Salário: 90 K

Habilidade	5
Maturidade	3

Eng. de Software	ES29
Brian	

Consultor: perde um ponto de habilidade a menos quando ajuda outros da equipe.

Salário: 110 K

Habilidade	5
Maturidade	2

Eng. de Software	ES30
Jader	

Doutor em Ciência da Computação e 10 anos de experiência. Excelente relacionamento.

Salário: 110 K

Habilidade	5
Maturidade	5

Eng. de Software	ES31

Arnaldo

Grande experiência, mas seu individualismo pode dificultar o andamento do grupo.

Salário: 90 K

Habilidade	5
Maturidade	2

Eng. de Software	ES

Salário: K

Habilidade	
Maturidade	

Eng. de Software	ES

Salário: K

Habilidade	
Maturidade	

Eng. de Software	ES

Salário: K

Habilidade	
Maturidade	

Eng. de Software	ES

Salário: K

Habilidade	
Maturidade	

Eng. de Software	ES

Salário: K

Habilidade	
Maturidade	

Apêndice 5 Cartões de Projetos

Projeto	PR 1

Expert Committee

Expert Committee é um sistema multi-agente aberto para suporte ao gerenciamento de submissões e revisões de artigos submetidos a uma conferência ou workshop. O sistema oferece suporte a diferentes atividades, tais como, envio de trabalhos, atribuição de um artigo a um revisor, seleção de revisores, notificação da aceitação e recusa de artigos.

[Garcia et al. 2004]

Complexidade	**2**
Tamanho	**2**
Qualidade	**3**
Orçamento	**180 K**

	Módulos
1	2 RQ + 1 DS + 1 CD
2	1 DS + 2 RT + 1 AJ +1 CD
3	
4	
5	
6	

Projeto	PR2

Telestrada

Telestrada é um sistema de informação para viajantes em auto-estradas brasileiras. Ele envolve cinco subsistemas: Central de Banco de Dados, Sistema de Informação Geográfica, Centro de Operação de Chamadas (call-center), Sistema de Operações em Estradas e Sistema de Gerência de Reclamações..

[Filho et al. 2005] [Filho et al. 2006]

Complexidade	**2**
Tamanho	**4**
Qualidade	**2**
Orçamento	**200K**

	Módulos
1	2 RQ + 1 DS + 1 CD
2	1 DS + 2 RT + 1 CD
3	2 DS + 1 AJ + 1 CD
4	2 AJ + 2 CD
5	
6	

Projeto	PR3

Eclipse

Eclipse é uma plataforma aberta para a criação de ambientes integrados de desenvolvimento (IDEs). Ela possibilita desenvolver diversos programas, aplicativos e ferramentas, de forma otimizada e padronizada, baseando-se nas iniciativas de software livre.

[Eclipse 2006] [Budinsky et al. 2003]

Complexidade	**4**
Tamanho	**6**
Qualidade	**4**
Orçamento	**250K**

	Módulos
1	2 RQ + 1 DS + 1 CD
2	2 RQ + 1 RT + 1 CD
3	1 DS + 2 RT + 1 CD
4	2 DS + 2 RT + 1 CD
5	1 DS + 2 AJ + 1 CD
6	3 AJ + 2 CD

Projeto	PR4

Health Watcher

A principal funcionalidade do sistema Health Watcher é registrar queixas em relação à qualidade dos serviços de saúde. Sua implementação deve envolver várias tecnologias Java como Invocação Remota de Método (RMI), Servlets e Conexão com Banco de Dados (JDBC).

[Kulesza et al. 2005] [Filho et al. 2006]

Complexidade	**2**
Tamanho	**2**
Qualidade	**2**
Orçamento	**160K**

	Módulos
1	2 RQ + 2 DS + 1 CD
2	1 RT + 3 AJ + 2 CD
3	
4	
5	
6	

Projeto	PR5

JUnit

JUnit é um Framework, que oferece a possibilidade de teste do código (escrito na linguagem de programação Java) antes de utilizá-lo efetivamente. Permite a realização de testes de unidades, conhecidos como "caixa branca", facilitando assim a correção de métodos e objetos.

[JUnit 2006] [Gamma and Beck 1999]

Complexidade	**4**
Tamanho	**3**
Qualidade	**3**
Orçamento	**200K**

	Módulos
1	2 RQ + 2 RT + 2 CD
2	1 DS + 1 RT + 1 CD
3	1 DS + 2 AJ + 2 CD
4	
5	
6	

Projeto	PR6

Tomcat

O Tomcat é um servidor de aplicações Java para web. É distribuído como software livre e desenvolvido como código aberto dentro do projeto Apache Jakarta e oficialmente endossado pela Sun como a Implementação de Referência (RI) para as tecnologias Java Servlet e JavaServer Pages (JSP).

[Apache Tomcat 2006]

Complexidade	**4**
Tamanho	**4**
Qualidade	**5**
Orçamento	**210K**

	Módulos
1	3 RQ + 2 DS + 1 CD
2	2 RQ + 2 RT + 1 CD
3	2 DS + 1 RT + 2 CD
4	1 DS + 3 AJ + 2 CD
5	
6	

Apêndice 6 Lista de Referências do Jogo

"**Apache Tomcat**" (2006). Disponível on-line em http://tomcat.apache.org/. Acessado em Agosto, 2006.

BAKER, A.; NAVARRO, E.; HOEK A. (2005) "**An Experimental Card Game for Teaching Software Engineering Processes**". In: Journal of Systems and Software, pages 3-16, vol. 75, issue 1-2.

BASILI, V. CALDIEIRA, G. ROMBACH, H. (1994) "**The Goal Question Metric Approach**". Encyclopedia of Software Engineering (J.Willey & Sons).

BECK, K. (1999) "**Extreme Programming Explained: Embrace Change**". Addison-Wesley Longman Publishing.

BINDER, R. (1999) "**Testing Object-Oriented Systems: Objects, Patterns, and Tools**". Addison-Wesley Professional, 1st edition.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. (1999) "**The Unified Modeling Language User Guide**". Addison-Wesley, 1st edition.

BUDINSKY, F.; STEINBERG, D.; MERKS, E.; ELLERSICK, R.; GROSE, T. (2003) "**Eclipse Modeling Framework**". Addison-Wesley Professional, 1st edition.

BUDINSKY, F.; FINNIE, M.; YU, P.; VLISSIDES, J. (1996) "**Automatic Code Generation from Design Patterns**". In IBM Systems Journal, 35(2), pages 151-171.

CHUNG, L.; NIXON, B.; YU, E.; MYLOPOULOS, J. (2000) "**Non-Functional Requirements in Software Engineering**". Kluwer Academic.

CONRADI, R.; WESTFECHTEL, B. (1998) "**Version Models for Software Conguration Management**". In: ACM Computing Surveys, 30(2), pages 232-282.

CYSNEIROS, L.; LEITE, J. (2004) "**Non-Functional Requirements: From Elicitation to Conceptual Model**" In: IEEE Transactions on Software Engineering, vol. 30, n. 5, pages 328-350.

"**Eclipse Project**" (2006). Disponível on-line em http://www.eclipse.org/. Acessado em Agosto, 2006.

FILHO, F.; CACHO, N.; MARANHÃO, R.; FIGUEIREDO, E.; GARCIA, A.; RUBIRA, C. (2006) "**Exceptions and Aspects: The Devil is in the Details**". In: ACM Symposium On Foundations of Software Engineering (FSE). Portland, USA.

FILHO, F.; RUBIRA, C.; GARCIA, A. (2005) "**A Quantitative Study on the Aspectization of Exception Handling**". In: ECOOP'2005 Workshop on Exception Handling in Object-Oriented Systems. Glasgow, UK.

FOWLER, M. (1999) "**Refactoring: Improving the Design of Existing Code**". Addison-Wesley Professional, 1st edition.

GAMMA, E.; BECK., K. (1999) "**JUnit: A Cook's Tour**". In: Java Report, 4(5), pages 27-38.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. (1995) "**Design Patterns: Elements of Reusable Object-Oriented Software**". Addison-Wesley Longman Publishing.

GARCIA, A.; SANT'ANNA, C.; CHAVEZ, C.; SILVA, V.; LUCENA, C.; STAA, A. (2004) "**Separation of Concerns in Multi-Agent Systems: An Empirical Study**". In: Software Engineering for Multi-Agent Systems II, Springer, LNCS 2940.

GOGUEN, J.; LINDE, C. (1993) "**Techniques for Requirements Elicitation**". In: International Symposium on Requirements Engineering, pages 152-164.

GOTEL, O.; FINKELSTEIN, A. (1994) "**An Analysis of the Requirements Traceability Problem**". In: IEEE International Conference on Requirements Engineering, pages 94-101.

JACOBSON, I.; GRISS, M.; JONSSON, P. (1997) "**Software Reuse: Architecture, Process and Organization for Business Success**". Addison-Wesley Professional, 1st edition.

"**JUnit.org**" (2006). Disponível on-line em http://www.junit.org/. Acessado em Agosto, 2006.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J.-M.; IRWIN, J. (1997) "**Aspect-Oriented Programming**". In: European Conference on Object-Oriented Programming (ECOOP), pages 220-242. Finland.

KOREL, B. (1990) "**Automated Software Test Data Generation**". In: IEEE Transactions on Software Engineering (TOSEN), 16(8), pages 870-879.

KOTONYA, G. (1998) "**Requirements Engineering: Processes and Techniques**". John Wiley and Sons Ltd.

KNUTH, D. (1992) "**Literate Programming**". Center for the Study of Language and Information.

KRUEGER, C. (1992) "**Software Reuse**". In: ACM Computing Surveys, vol. 24, n 2, p. 131-183.

KULESZA, U.; SANT'ANNA, C.; GARCIA, A.; LUCENA, C.; STAA, A. (2005) "**Aspectization of Distribution and Persistence: Quantifying the Effects of AOP**". In: IEEE Software, Special Issue on AOP.

LARMAN, C. (2004) "**Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development**". 3rd edition, Prentice Hall PTR.

LEITE, J.; ROSSI, G.; MAIORANA, V.; BALAGUER, F.; KAPLAN, G.; HADAD, G.; OLIVEROS, A. (1997) "**Enhancing a Requirements Baseline with Scenarios**". In: Requirements Engineering Journal, 2(4), pages 184-198.

"**Literate Programming**" (2006). Disponível on-line em http://www.literateprogramming.com/. Acessado em Agosto, 2006.

MEYER, B. (1992) "**Applying Design by Contract**". In: IEEE Computer, vol. 25, issue 10, pages 40-51.

MYLOPOULOS, J.; CHUNG, L.; YU, E. (1999) "**From Object-Oriented to Goal-Oriented Requirements Analysis**". In: Communications of the ACM, vol. 42, no. 1.

NAVARRO, E.; BAKER, A.; HOEK, A. (2004) "**Teaching Software Engineering Using Simulation Games**". In: International Conference on Simulation in Education (ICSIE). California.

NIELSEN, J. (1994) "**Usability Engineering**". Morgan Kaufmann, New Edition.

PARNAS, D. L. (1972) "**On the Criteria to Be Used in Decomposing Systems into Modules**". In: Communications of the ACM, vol. 15, issue 12, pages 1053-1058.

SOMMERVILLE, I. (2000) "**Software Engineering**". Addison Wesley, 6 edition.

SETHI, R. (1996) "**Programming Languages: Concepts and Constructs**". Addison-Wesley, 2nd edition.

STAA, A. (2000) "**Programação Modular**". Editora Campus.

THIRLWAY, M. (1994) "**Writing Software Manuals: a Practical Guide**". New York: Prentice Hall.

"**UseIt.com**" (2006). Disponível on-line em http://www.useit.com/. Acessado em Agosto, 2006.

Apêndice 7 Tabuleiro do Jogo

Tabuleiro do Jogador	**Engenheiros de Software**			
	Engenheiro 1	Engenheiro 2	Engenheiro 3	Engenheiro 4
Requisitos				
Desenhos				
Códigos				
Rastros				
Ajudas				