Universidade do Minho
Departamento de Informática

Sónia Marlene Pereira de Sousa

Play Scrum – Um Jogo para a Aprendizagem do Método Ágil Scrum

Outubro de 2009

Universidade do Minho
Departamento de Informática

Sónia Marlene Pereira de Sousa

Play Scrum – Um Jogo para a Aprendizagem do Método Ágil Scrum

Tese de Mestrado em Informática

Trabalho efectuado sob a orientação do
Professor João Miguel Fernandes

Outubro de 2009

DECLARAÇÃO

Nome: Sónia Marlene Pereira de Sousa

Endereço Electrónico: sonyasousa1981@hotmail.com

Título da Dissertação: Play Scrum - Um Jogo para a Aprendizagem do Método Ágil Scrum

Orientador: Professor João Miguel Fernandes

Ano de Conclusão: 2009

Designação do Mestrado: Mestrado em Informática

Universidade do Minho, 30 de Outubro de 2009

Agradecimentos

Esta dissertação é a consequência de um percurso de aprendizagem e estudo que não seria possível sem o apoio de inúmeras pessoas que partilharam comigo esse percurso.

Assim sendo, queria começar por agradecer ao Professor João Miguel Fernandes pela orientação deste trabalho e pela disponibilidade demonstrada para que a sua elaboração fosse possível.

Aos meus colegas de mestrado e em particular à Sandrina Maciel que fizeram com que esta caminhada fosse mais fácil e agradável.

À turma da disciplina de Análise e Concepção de Software do 1º ano do Mestrado de Informática 2009/2010 pela sua cooperação na validação deste trabalho.

Finalmente, e não menos importante aos meus pais que me ajudaram sempre no que mais precisava e tiveram sempre ao meu lado nas horas de maior angústia. Obrigada por me terem dado a oportunidade de ter uma vida melhor e pelo vosso amor incondicional.

Resumo

Nos dias de hoje e num tempo em que cada vez mais o mercado laboral se torna competitivo, mais importante é a preparação com que os alunos devem sair do Ensino Superior. A forma mais eficaz de preparar um aluno para o futuro é a de conseguir mantê-lo motivado de modo a estar sempre empenhado na sua aprendizagem. Infelizmente, os conteúdos, por vezes demasiado teóricos, acabam por desmotivar o aluno que não consegue ter a percepção da aplicabilidade prática dos conceitos leccionados. Para contrariar esta tendência, certos investigadores sugerem novas formas de ensino capazes de motivar o aluno e, consequentemente, de melhorar a qualidade do ensino. O uso de jogos na educação vem ao encontro desta necessidade e tem vindo a alcançar sucesso no ensino de certas disciplinas, entre as quais a Engenharia de Software. Em alguns contextos de aprendizagem da engenharia de software, parece ser mais eficaz e motivador o uso de jogos, tais como o 'SESAM', 'Problems and Programmers', ou 'SIMSE', em especial quando conjugados com formas de ensino mais convencionais. A presente proposta de dissertação tem por objectivo principal o desenvolvimento do jogo Play Scrum, jogo de cartas para a aprendizagem do método ágil SCRUM de desenvolvimento de software.

Abstract

Nowadays and at a time when the labor market is becoming more and more competitive, most important is the preparation with which students finish higher education. The most effective way to prepare a student for the future is to keep him motivated while he is learning. Unfortunately, the content, sometimes too theoretical, end up discouraging the student and loses perception of the practical applicability of lectured concepts. To contract this tendency, some investigators suggest new ways of teaching that motivate students and, consequently, improve the quality of education. The use of games in teaching addresses this necessity and has obtained success in the teaching of certain subjects, including Software Engineering. In some learning contexts of software engineering, the use of games such as 'Sesame', 'Problems and Programmers' or 'Sims' seems to be more effective and motivating, especially when conjugated with more traditional teaching methods. The present proposal of dissertation has as its principal objective the development of the game Play Scrum, a card game for teaching the agile software development method SCRUM.

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1 Enquadramento

A Engenharia de Software é uma área de conhecimento voltada para a especificação, concepção, implementação, teste e manutenção de sistemas de software através da aplicação de tecnologias e práticas da gestão de projectos, do controle de qualidade, da elaboração de documentação entre outras disciplinas.

A engenharia de software tem por objectivo principal a construção de produtos de software com qualidade induzindo como tal uma melhoria na gestão da organização, na sua produtividade e na sua qualidade.

O processo de engenharia de software, representativo de uma série de práticas que visam o desenvolvimento de sistemas de software, apoia-se em metodologias de modo a obter aplicações de software robustas e desenvolvidas de forma sistemática e organizada [27].

Os Métodos Ágeis são parte destas metodologias e vieram propor abordagens e princípios distintos daqueles sugeridos pelos métodos mais tradicionais, tais como o "Modelo Cascata". Os métodos tradicionais baseiam-se num desenvolvimento por actividades sequenciais, por vezes considerados pesados devido à sua grande burocracia e à sua pouca adaptabilidade à realidade. O método SCRUM é um dos métodos ágeis mais populares e é visto como um método em que existe pouca burocracia envolvida no desenvolvimento. É vocacionado para pequenas equipas de desenvolvimento (cerca de 10 pessoas), em que o software é produzido incrementalmente e em que os requisitos vão sendo definidos ao longo do próprio processo de desenvolvimento [21, 20].

O estudo da Engenharia de Software requer uma aprendizagem muito vasta de técnicas e conceitos. Isto não se torna muito fácil nos casos em que os alunos somente têm acesso à sua abordagem teórica, sem ter a percepção de como os processos decorrem na prática. Esta falta de entendimento prático dos alunos, mesmo no momento

em que terminam a sua formação académica, não satisfaz os empregadores que acabam por se ver forçados a contratá-los, apesar de eles estarem pouco preparados para as realidades organizacionais. Os conceitos podem ter sido assimilados, porém o aluno não percebe como e quando os deve aplicar no mundo exterior.

Para contrariar esta insatisfação surgiram novas formas de aprendizagem que tendem a dar uma visão mais realista do processo de Engenharia de Software. O ensino da Engenharia de Software através de jogos surge como uma destas novas formas de aprendizagem. Estes jogos permitem a simulação do processo da Engenharia de Software de uma forma mais agradável e lúdica, em que o aluno acaba por ser personagem do processo. Os jogos têm de ser de entendimento relativamente fácil e interessantes, de modo a cativar os alunos e desta forma melhorar a qualidade do ensino da matéria.

O jogo de cartas ‘Problems and Programmers’ [1] é um dos diversos jogos criados neste sentido. É um jogo competitivo em que os alunos representam gestores de projecto da mesma empresa. Durante o jogo, os jogadores são confrontados com as diversas situações (relacionadas com os requisitos do cliente, com o orçamento disponível para o projecto, com a qualidade do software desenvolvido, entre muitas outras coisas) que tipicamente surgem num processo real de desenvolvimento de software, baseado no método em cascata. É o grande vencedor o jogador que conseguir terminar o jogo em primeiro, o que normalmente significa realizar, de acordo com as boas práticas de software, o projecto de que é responsável.

O modelo em cascata apoia-se num processo sequencial que começa com uma análise de requisitos seguida da concepção do sistema. Após esta segunda fase estar concluída surge a implementação do software seguindo-se a integração das várias unidades de implementação (módulos) e a realização de testes. Finalmente, a última tarefa consiste na manutenção deste mesmo sistema. Este método, na sua versão menos sofisticada, tem um ciclo inflexível pois não permite avançar de uma fase para outra sem a primeira estar concluída, da mesma forma que não permite voltar atrás sem sofrer grandes complicações.

Como tal, no caso do jogo ‘Problems and Programmers’ é de todo compreensível que o vencedor seja o jogador que consiga concluir em primeiro o seu projecto.

1.2 Motivação

A forte competitividade que se faz sentir actualmente no Mundo obriga a uma triagem cada vez maior por parte das empresas na hora de contratarem os seus colaboradores. Uma boa preparação académica é, de facto, um factor chave na hora de um profissional conseguir um emprego.

Porém e como já foi atrás transcrito, nem sempre o uso de métodos tradicionais de ensino se revela a forma mais eficaz para transmitir os conceitos de determinadas matérias. Os alunos não conseguem ter a real percepção do que lhes é transmitido pelo facto de não entenderem qual o cenário em que a disciplina se insere no mundo empresarial. O uso de jogos como métodos alternativos de ensino pode ser uma alternativa para aprendizagem destas matérias. Vários autores [1, 2, 5, 6, 10, 12, 16, 23] têm vindo a comprovar que desta forma um aluno consegue assimilar realmente o que lhe é ensinado pondo de lado os seus medos e ansiedades e dedicando-se completamente ao jogo. A concentração é intensa e com isso a matéria é absorvida [5].

O universo do desenvolvimento de software é cada vez maior num Mundo dominado pelas novas tecnologias. Um bom planeamento para desenvolvimento de software é fundamental para um bom produto final. O método Ágil SCRUM é usado por empresas com uma posição vincada no mercado tais como Google, Yahoo, Siemens, British Telecom. Este método é, como tal, visto como um modelo fiável para desenvolvimento de software e é, deste modo, escolhido como modelo a ser ensinado no jogo que é desenvolvido no contexto desta tese.

1.3 Objectivos

O objectivo principal desta dissertação consiste na definição de um jogo de cartas para ensino do Método Ágil SCRUM a alunos do ensino universitário. Os alunos deverão aprender os conceitos, as práticas e as consequências, associados à utilização do método SCRUM num processo de desenvolvimento de software. Este objectivo principal implica a prossecução dos seguintes passos:

- Elaborar um estudo aprofundado do método ágil SCRUM de modo a ser possível definir com objectividade a implicação que tem a utilização deste

método num processo de software. Serão definidas as boas práticas a serem seguidas e as que se devem evitar.

- Investigar técnicas e métodos para a criação de jogos de modo a que este siga uma série de regras que tornem o seu uso viável num ambiente académico. Isto implica igualmente, analisar os jogos para o ensino de métodos de desenvolvimento de software já existentes tais como o jogo ‘Problemas and Programmers’ e ‘SESAM’.

1.4 Estrutura do Documento

Para além do actual, a presente dissertação divide-se em 5 capítulos.

O Capítulo 2 apresenta um estudo aprofundado ao método ágil SCRUM determinando quais os seus intervenientes, o papel que cada qual interpreta, quais as suas regras etc.

O Capítulo 3 começa por apresentar vários jogos usados como métodos alternativos de ensino para da engenharia de software, acabando por se focar no jogo ‘Problems and Programmers’ sobre qual o Play Scrum se baseia.

O Capítulo 4 descreve os componentes e a dinâmica do jogo Play Scrum. No final deste capítulo é também apresentado uma tabela que evidencia as características do método Scrum que foram tidas em conta no jogo Play Scrum

O Capítulo 5 valida o Play Scrum. O jogo foi jogado e um inquérito foi feito para se conseguir entender se os objectivos do Play Scrum foram atingidos e o que deve ser melhorado no futuro.

Finalmente, o Capítulo 6 apresenta as conclusões deste trabalho definindo as melhorias e o trabalho futuro.

2. Método Ágil SCRUM

2.1 Introdução

As práticas habitualmente empregues no desenvolvimento de software mudaram ao longo das últimas décadas. As organizações descobriram que não basta produzirem produtos inovadores, de elevada qualidade a preço reduzido para serem líderes de um mercado cada vez mais competitivo. Essa liderança passa igualmente por disponibilizar esses produtos de forma rápida e providenciando flexibilidade no seu desenvolvimento [13].

Segundo [13], em 1986 Hirotaka Takeuchi e Ikujiro Nonaka apresentam uma nova abordagem de desenvolvimento que vem ao encontro destas novas necessidades do mercado. Enquanto as abordagens de desenvolvimento tradicionais assentam num desenvolvimento sequencial e segmentado, esta nova abordagem assenta no desenvolvimento do produto como um todo, feito por uma equipa multi-disciplinar que trabalha em conjunto durante todo o processo.

Em 1993, Jeff Sutherland usa um método de desenvolvimento no seio da organização 'Easel Corporation' baseado no estudo publicado por Hirotaka Takeuchi e Ikujiro Nonaka em 1986 e é o primeiro a dar-lhe o nome de Scrum [21]. Em 1995, Ken Schwaber formaliza o processo de desenvolvimento Scrum para a indústria de desenvolvimento de software com a publicação de um artigo apresentado no congresso OOPSLA (Object-Oriented Programming, Systems, Languages & Applications), em Austin nos Estados Unidos [17].

O método Scrum é utilizado para organizar equipas e fomentar um trabalho mais produtivo, com maior qualidade. A sua ideia principal é que o desenvolvimento de software envolve variáveis de ambiente que tornam o processo de desenvolvimento

imprevisível e complexo, requerendo flexibilidade para acompanhar as mudanças [21]. Estas variáveis de ambiente englobam factores tais como:

- A disponibilidade de profissionais especializados – quanto mais recente a tecnologia utilizada menor é o número de mão-de-obra devidamente qualificada;
- A estabilidade de implementação – quanto mais recente for a tecnologia a implementar menor é a estabilidade de implementação do produto e maior é a necessidade de recorrer a outras tecnologias;
- A estabilidade e poder das ferramentas de desenvolvimento – quanto mais recentes forem as ferramentas utilizadas para o desenvolvimento de software menor é o número de profissionais qualificados;
- A eficácia dos métodos utilizados no processo de desenvolvimento – como a modelação, a análise e o controlo de versão vão ser utilizados e de que forma vão ser realmente eficazes;
- O nível de conhecimento do negócio dos profissionais;
- As novas funcionalidades – que novas funcionalidades vão ser adicionadas e como se vão encaixar com as já existentes?
- A metodologia – será que a abordagem utilizada para o desenvolvimento do sistema promove a flexibilidade ou pelo contrário é uma abordagem demasiadamente detalhada que impede um desenvolvimento flexível?
- A concorrência – o que vai desenvolver a concorrência durante a duração do projecto, que novas funcionalidades serão apresentadas?
- O Tempo e o Orçamento – qual a duração prevista do projecto e qual o orçamento disponível para a sua execução?
- Outras variáveis que englobam qualquer outro factor que possa surgir durante o desenvolvimento do sistema e que deve ser tratado para garantir o sucesso do projecto;

Para conseguir desenvolver um sistema de qualidade, face a todas as variáveis descritas anteriormente, o método Scrum permite às equipas de desenvolvimento escolherem a quantidade de trabalho a desenvolver e decidirem qual a melhor forma de o fazer providenciando, deste modo, um ambiente de trabalho mais flexível e produtivo. O método Scrum dá prioridade ao trabalho com mais valor para o cliente, melhorando assim a utilidade do produto entregue, e aumentando as receitas geradas com o software disponibilizado. Concebido para se adaptar à evolução das necessidades do cliente

durante o processo de desenvolvimento, o método Scrum é dividido em iterações que têm a duração máxima de quatro semanas, permitindo a definição de novos requisitos e o ajuste dos requisitos a serem desenvolvidos em próximas iterações. Desta forma é possível entregar ao cliente o que ele realmente precisa aumentando, consequentemente, o seu grau de satisfação [16].

O método Scrum tem uma estrutura bem definida composta por três grupos de intervenientes, três reuniões e três documentos [4] de modo a poder conseguir adaptar-se aos novos requisitos que vão surgindo ao longo do desenvolvimento do software. Os seus intervenientes são divididos entre o Cliente que é o representante dos interesses do cliente no projecto, o Mestre Scrum encarregue de transmitir as boas práticas do Scrum a todos os restantes intervenientes no processo e finalmente a Equipa encarregue da implementação do software.

Cada uma das iterações do projecto é composta no seu início por uma reunião em que vão ser definidas as tarefas a serem desenvolvidas, uma pequena reunião diária em que a equipa se junta para fazer um breve ponto de situação do andamento do seu trabalho e uma reunião no fim da iteração para apresentar o trabalho desenvolvido e as possíveis dificuldades encontradas. Finalmente, todo o desenvolvimento do software está descrito em três tipos de documentos que registam todos os requisitos a serem tidos em conta. Existe, assim, uma listagem dos requisitos do sistema – '*Product Backlog*' desenvolvida pelo Cliente e que identifica quais os requisitos prioritários e quando devem ser entregues. Existe igualmente, o '*burndown chart*', um gráfico que apresenta a quantidade de trabalho que resta a ser desenvolvida em qualquer altura do projecto e, finalmente, aparece a lista de requisitos a serem desenvolvidos em cada uma das iterações – '*Sprint Backlog*' definida pela Equipa de desenvolvimento.

Os resultados obtidos com o uso de Scrum são muito positivos [17] e nos últimos anos este método tornou-se num fenómeno que envolve dezenas de milhares de projectos em centenas de empresas líderes em desenvolvimento de software. O desenvolvimento dividido em iterações que prioritizam o desenvolvimento das funcionalidades com maior retorno de investimento permite obter um produto comercializável logo nas primeiras iterações. Este factor tem sido decisivo na hora das empresas optarem por um método de desenvolvimento e, como tal, nos últimos anos milhares de novos ScrumMasters Certificados têm vindo a ser treinados, em especial, nos Estados Unidos e na Europa.

2.2 O Método Scrum

A metodologia é o um dos factores mais importantes para determinar a probabilidade de sucesso do desenvolvimento de um sistema. Métodos que suportam uma maior flexibilidade têm um grau muito elevado de tolerância a mudanças de variáveis. Com o uso destes tipos de métodos, o processo de desenvolvimento é visto como imprevisível numa fase inicial criando-se, como tal, mecanismos de controlo para gerir tal imprevisibilidade [21].

Os métodos de desenvolvimento de software em Cascata e em Espiral determinam o contexto e a definição do produto no início do projecto. Contrariamente, o método Scrum reconhece que o processo de desenvolvimento nunca está completamente definido de início e, como tal, utiliza mecanismos de controlo para melhorar a sua flexibilidade. Esses mecanismos passam por determinar que em todas as iterações devem existir as fases de análise, de modelação, de desenvolvimento e de fecho das funcionalidades.

O método Scrum determina que a fase de análise do projecto serve para definir os métodos que vão ser usados para desenvolver todos os elementos do sistema. Um projecto que usa este método inicia-se com uma visão bastante abstracta do sistema a ser desenvolvido. Essa visão é baseada nos resultados esperados a nível do negócio. Porém à medida que o projecto vai avançando os requisitos do sistema vão se tornando cada vez mais claros e objectivos [19]. Como tal, muitos dos processos da fase de desenvolvimento não são identificáveis ou controláveis, esta fase é tratada como uma incógnita que precisa de controlo externo apoiado pelo uso de ferramentas de gestão de risco para eliminar o caos e manter a máxima flexibilidade no desenvolvimento.

O processo de desenvolvimento Scrum determina que os requisitos do sistema podem ser alterados em qualquer altura das fases de planeamento e desenvolvimento, como tal o projecto está exposto às diversas variáveis de ambiente até o seu fecho e o aspecto final do sistema é definido durante a execução do projecto mediante estas mesmas variáveis de ambiente.

De um modo geral, é possível afirmar que os projectos que seguem o método Scrum possuem as seguintes características:

- Entregas flexíveis em que o conteúdo do sistema final é determinado pelas diversas variáveis de ambiente;
- Prazos flexíveis que se ajustam às necessidades do cliente;

- Pequenas equipas de desenvolvimento compostas por cerca de dez elementos;
- Colaboração interdisciplinar;
- Arquitectura de desenvolvimento orientada ao objecto;

2.2.1 As Fases do método Scrum

O método Scrum é composto por três grandes fases durante a execução de um projecto:

1. Uma fase de **análise** composta por 2 actividades:
    - **Planeamento** que consiste na definição da nova versão do sistema baseada nos requisitos definidos até à data. São igualmente determinados os prazos e custos de execução. Se o sistema a ser desenvolvido for completamente novo, esta fase é de análise e conceptualização;
    - **Modelação** que define, ao pormenor, a forma de implementar os requisitos. Esta fase envolve a definição da nova arquitectura lógica e física;
2. Uma fase de **desenvolvimento** composta por várias iterações que consiste no desenvolvimento das funcionalidades do sistema, tendo sempre em atenção as diversas variáveis de ambiente que podem afectar o projecto tais como os prazos de entrega, o orçamento associado, a qualidade do sistema, a concorrência, entre outras.
3. O **fecho** que consiste na preparação do sistema para a entrega final ao cliente. Esta preparação envolve as tarefas de documentação do sistema, de testes e a própria entrega [21].

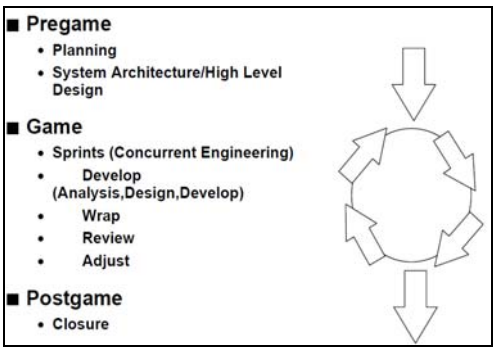


Figura 1 – Fases do Método Scrum (Fonte: http://www.controlchaos.com/old-site/scrumwp.htm)

2.2.2 Tarefas do Método Scrum

Cada uma das fases descritas anteriormente é composta por uma série de tarefas que precisam de ser executadas para garantir o sucesso do projecto.

1. O Planeamento pertence à fase de análise e é composto pelas seguintes tarefas:
    - Desenvolvimento de uma lista dos requisitos do sistema;
    - Definição dos prazos e funcionalidades a serem entregues;
    - Prioritização das funcionalidades;
    - Definição da equipa de desenvolvimento do sistema;
    - Avaliação dos riscos de desenvolvimento e das medidas a serem tomadas para evitar derrapagens;
    - Revisão e possíveis ajustem à lista de funcionalidades;
    - Validação das ferramentas de desenvolvimento e das infra-estruturas existentes;
    - Estimativa do custo de desenvolvimento incluindo o material a ser usado, o marketing, a formação aos novos utilizadores etc.
    - Verificação da aprovação da gestão e do financiamento;

2. A Modelação do sistema a desenvolver pertence à fase de análise e é composta pelas seguintes tarefas:
    - Revisão das funcionalidades definidas;
    - Refinamento da arquitectura do sistema para suportar o novo contexto e os novos requisitos;
    - Identificação de possíveis problemas inerentes ao desenvolvimento do sistema;
    - Reunião de revisão da modelação em que cada uma das equipas envolvidas apresenta a sua visão de implementação dos requisitos do sistema;

3. O Desenvolvimento é composto por uma série de iterações que envolvem por sua vez várias tarefas entre as quais:
    - Reunião com as equipas para rever o plano de requisitos do sistema;

- Distribuição, revisão e ajustes das normas a que o produto vai ter de obedecer;

Uma iteração pode ser definida como um período de no máximo quatro semanas em que é desenvolvido um conjunto de actividades. A duração da iteração é determinada pela complexidade das funcionalidades que a vão compor, os seus riscos são avaliados de forma contínua e as actividades de controlo necessárias são postas em prática. Em cada iteração uma ou várias equipas devem:

- Desenvolver as funcionalidades definidas para a iteração. É necessário igualmente testar e documentar o desenvolvimento feito;
- Preparar a entrega ao cliente que corresponde na criação de um executável do produto contendo as funcionalidades que foram implementadas na iteração;
- Rever a iteração em que a equipa de desenvolvimento e os representantes do produto se reúnem para apresentar o trabalho desenvolvido e resolver possíveis problemas. Os riscos associados ao projecto são revistos e são tomadas as medidas necessárias para os combater. É nesta reunião que são repensados os requisitos do sistema dependendo das ocorrências das iterações anteriores;

4. O Fecho em que se prepara o sistema desenvolvido para a sua entrega final. Esta fase tem as seguintes actividades:
    - Testar o sistema;
    - Documentar o sistema;
    - Preparar o material para dar formação ao cliente;
    - Preparar o material para marketing;

2.2.3 Formas de Controlo do Método Scrum

O método Scrum por ser tão flexível requer mecanismos de controlo para evitar que o desenvolvimento do sistema caia no caos. Estes mecanismos passam pelo uso de técnicas de apoio ao desenvolvimento orientadas ao objecto tais como:

- Elaboração de listas que descrevem os requisitos que não foram devidamente desenvolvidos. Todos os erros, defeitos ou requisitos em que o cliente solicitou melhorias estão aí apontados. Este documento regista igualmente quais os

resultados esperados do sistema quer a nível de vantagem competitiva que se pretende atingir como em relação aos seus aspectos tecnológicos;

- Definição dos componentes do produto que têm de ser alterados para se ajustarem aos requisitos do novo sistema;
- Definição das alterações que devem ocorrer na implementação de novos requisitos;
- Determinação dos problemas técnicos existentes e que devem ser resolvidos para implementar as alterações pretendidas;
- Definição dos riscos que podem afectar o sucesso do projecto e produzir de forma contínua respostas para os contradizer;

Estas técnicas de controlo são utilizadas nas várias fases do método Scrum já que os gestores do produto as usam para gerir os requisitos do sistema, enquanto a equipa de desenvolvimento as utilizam para gerir as mudanças e os problemas que possam surgir. Todos os elementos identificados são revistos no final de cada iteração.

2.2.4 Grupos de Intervenientes Scrum

Os projectos usando a metodologia Scrum são divididos em pequenas equipas de cerca de dez pessoas compostas por três grupos de intervenientes.

O Cliente, representante dos interesses dos que encomendam o sistema, apresenta, no início do projecto, uma lista dos requisitos que se pretendem atingir com a elaboração do sistema, o retorno do investimento esperado, assim como o plano de entregas dos requisitos. Este documento, conhecido como o *"Product Backlog"*, permite ao Cliente assegurar-se que as funcionalidades mais valiosas serão as primeiras a serem entregues pela Equipa responsável pelo desenvolvimento do sistema [19].

A Equipa é responsável pelo desenvolvimento técnico do projecto. As equipas têm o poder de se auto-gerir, de se auto-organizar e são responsáveis por transformar o *"Product Backlog"* fornecido pelo Cliente numa sequência de tarefas a serem desenvolvidas numa iteração [19]. Como tal, são as próprias equipas que definem qual dos seus membros vai ficar encarregue de desenvolver determinada tarefa baseando sua decisão nos conhecimentos técnicos de cada membro e na sua vontade pessoal.

O Mestre Scrum é responsável por todo o processo Scrum, por ensinar a todos os intervenientes do projecto as regras deste método, por implementá-lo para que ele se

encaixe na cultura da organização e forneça os resultados esperados e para se assegurar que todos seguem as regras e práticas do Scrum [19].

Os membros destes três grupos formam as equipas dos projectos Scrum que não devem ter um número superior a dez pessoas. Todos os restantes intervenientes podem ter o seu interesse nos resultados do sistema mas não podem intervir na sua concepção. O método Scrum faz uma distinção muito clara entre estes dois grupos e assegura-se que a equipa responsável pelo projecto tenha autoridade suficiente para fazer o que for necessário para o sucesso do projecto.

2.2.5 Documentos Scrum

Todo o processo Scrum, apresentado anteriormente, precisa registar todos os requisitos definidos para o sistema assim como as prioridades de sua execução.

O *"Product Backlog"* lista os requisitos do sistema pretendido. Este documento é elaborado pelo Cliente, e ele é responsável pelo seu conteúdo, pela atribuição da prioridade dos requisitos e pelos prazos de entrega do sistema. Este documento nunca está completo, e como tal, acaba por ser uma mera ferramenta que permite estimar os requisitos que vão ser necessários. O *"Product Backlog"* evolui à medida que o produto e o ambiente de desenvolvimento do produto evoluem. Ele é constantemente alterado de modo a identificar o que o produto deve ter para ser líder do mercado e será válido enquanto o produto existir.

Backlog Description	Initial Estimate	Adjustment Factor	Adjusted Estimate	work remaining until completion						
Title Import				1 256	2 209	3 193	4 140	5 140	6 140	7 140
Project selection or new	3	0.2	3.6	3.6	0	0	0	0	0	0
Template backlog for new projects	2	0.2	2.4	2.4	0	0	0	0	0	0
Create product backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Create sprint backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0.2	2.4	2.4	0	0	0	0	0	0
Sprint-1	13	0.2	15.6	16	0	0	0	0	0	0
Create a new window containing product backlog template	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Create a new window containing sprint backlog template	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Burndown window of product backlog	5	0.2	6	6	6	0	0	0	0	0
Burndown window of sprint backlog	1	0.2	1.2	1.2	1.2	0	0	0	0	0
Display tree view of product backlog, releases, prints	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Display burndown for selected sprint or release	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Sprint-2	16	0.2	19.2	19	19	1.2	0	0	0	0
Automatic recalculating of values and totals	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
As changes are made to backlog in secondary window, update burndown graph on main page	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Hide/automatic redisplay of burndown window	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
Insert Sprint capability ... adds summing Sprint row	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Insert Release capability ... adds summary row for backlog in Sprint	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Owner/assigned capability and columns optional	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Print burndown graphs	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Sprint-3	14	0.2	16.8	17	17	17	0	0	0	0
Duplicate incomplete backlog without affecting totals	5	0.2	6	6	6	6	6	6	6	6
Note capability	6	0.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2
What-if release capability on burndown graph	15	0.2	18	18	18	18	18	18	18	18
Trend capability on burndown server	2	0.2	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4
Publish facility for entire project, publishing it as HTML web pages	11	0.2	13.2	0	0	13	13	13	13	13
Future Sprints	39	0.2	46.8	34	34	47	47	47	47	47
Release-1				86	70	65	47	47	47	47

Figura 2 – "Product Backlog" (Fonte: http://www.tar.hu/agilescrum/agile_scrum_0011.html)

O *"Burndown Chart"* apresenta a quantidade de trabalho desenvolvida em qualquer altura de execução do projecto [19]. Este gráfico tem grande utilidade pois permite visualizar a correlação entre o trabalho realizado e a quantidade de trabalho que resta desenvolver pela equipa em qualquer altura do projecto. Este documento permite ao Cliente avaliar com maior exactidão a implicação de adicionar ou remover funcionalidades e permite-lhe ainda ter a noção de um possível desfasamento entre a realidade de execução e o que era inicialmente previsto.

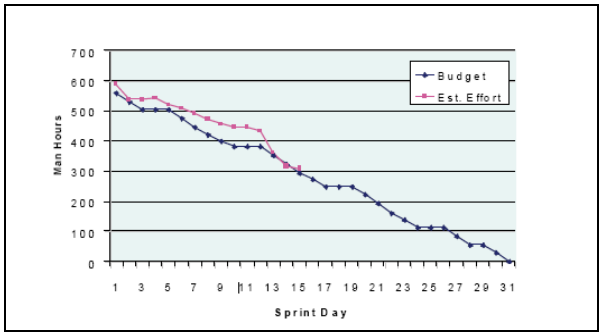


Figura 3 – "Burndown Chart" (Fonte: http:// www.softhouse.se)

O *"Sprint Backlog"* é um documento feito pela Equipa para registar quais as tarefas a serem desenvolvidas ao longo da iteração. A Equipa compila a lista inicial das tarefas a serem desenvolvidas na segunda parte da reunião. Esta parte da reunião dá início à iteração. As tarefas têm de ser divididas para que o seu tempo de execução seja compreendido entre quatro e dezasseis horas. As tarefas com tempos de execução superiores a dezasseis horas são vistas como tarefas que não foram apropriadamente definidas. Este documento só pode ser alterado pela Equipa já que ele representa com clareza o retrato do trabalho que a Equipa pretende desenvolver durante a iteração.

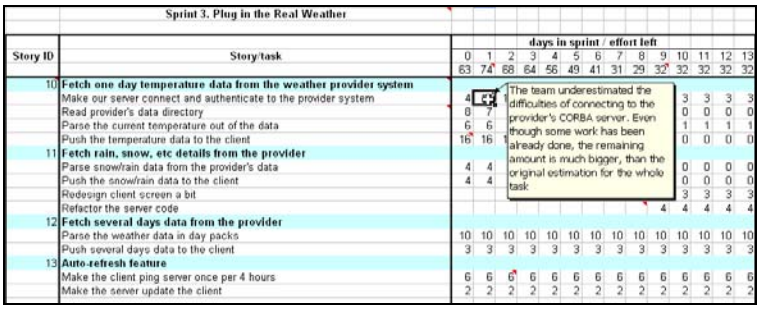


Figura 4 – "Sprint Backlog" (Fonte: http://agilesoftwaredevelopment.com/)

O método Scrum requer que as Equipas desenvolvam e entreguem novas funcionalidades a cada iteração. Estas funcionalidades terão de estar concluídas e prontas a serem apresentadas ao Cliente de modo a que este possa, se assim o pretender, utilizá-las de imediato. Isso implica que o código fonte dessa funcionalidade tenha sido revisto, testado e que esteja muito bem estruturado. A documentação da funcionalidade desenvolvida deve ter sido elaborada e entregue ao Cliente.

2.3 O Processo Scrum

O Cliente é responsável, perante os que encomendam o sistema, por se assegurar que o sistema obtido corresponde realmente ao esperado podendo, dessa forma, maximizar o retorno do investimento. O Cliente elabora um plano do projecto, o *"Product Backlog"*, que apresenta uma lista de requisitos funcionais e não funcionais do projecto. Estes requisitos, quando transformados em funcionalidades, apresentam a visão total do sistema final esperado. O *"Product Backlog"* tem uma hierarquia de prioridades e está dividido em várias entregas. A prioridade máxima dessa lista é o ponto de partida do desenvolvimento do projecto, no entanto, as restantes prioridades costumam mudar durante a execução do projecto, sendo muito frequente já não serem as mesmas a meio do processo de desenvolvimento. Essas alterações no *"Product Backlog"* reflectem mudanças nos requisitos do negócio e a velocidade com que a Equipa consegue implementar os requisitos definidos.

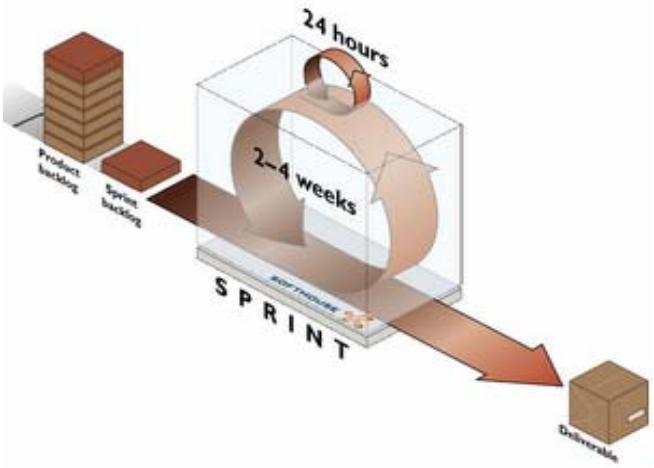


Figura 5 – Ciclo de vida do Processo SCRUM (Fonte: http:// www.softhouse.se)

Todo o trabalho desenvolvido está dividido em iterações com duração máxima de trinta dias consecutivos. Cada iteração começa com uma reunião em que o Cliente e a Equipa definem em conjunto quais as tarefas a serem desenvolvidas nessa iteração. Seleccionando os requisitos com maior prioridade no *"Product Backlog"*, o Cliente apresenta à Equipa os requisitos que ele pretende ver desenvolvidos nessa iteração enquanto a Equipa informa o Cliente qual a percentagem desses requisitos que poderá ser efectivamente implementada na iteração. Cada uma dessas reuniões não pode exceder as oito horas e é dividida em duas partes. As primeiras quatro horas permitem ao Cliente apresentar as tarefas prioritárias à Equipa que, por sua vez, pode questioná-lo sobre o conteúdo e os objectivos a serem atingidos com a implementação das tarefas. Após a Equipa fazer a análise a todos os requisitos definidos pelo Cliente, ela selecciona os requisitos que podem ser desenvolvidos e testados durante a iteração. As restantes quatro horas vão servir para a Equipa, responsável por monitorizar o seu trabalho, planear o seu trabalho durante a iteração. As tarefas que compõem esse plano ficam registadas no *"Sprint Backlog"* e vão surgindo à medida que a iteração decorre. No começo da segunda metade da reunião a iteração já se iniciou e terá, a partir desse momento, a duração máxima de trinta dias.

Diariamente, a Equipa reúne-se cerca de quinze minutos para fazer o ponto de situação do projecto. Cada membro da Equipa tem de responder a três perguntas [19]:

- O que fez desde a última reunião?
- O que pensa fazer hoje?
- Quais são os factores que o podem impedir de cumprir os seus objectivos para esta iteração?

O objectivo desta reunião diária é o de sincronizar o trabalho de todos os membros da Equipa e o de agendar quaisquer reuniões para a Equipa apresentar os seus progressos.

No final de cada iteração, é realizada uma reunião de, no máximo quatro horas, em que a Equipa apresenta ao Cliente o que foi desenvolvido durante a iteração. Esta reunião de carácter informal destina-se a reunir os intervenientes no processo de desenvolvimento do software e permitir-lhes determinar quais as opções a serem seguidas pela Equipa. Após esta reunião e antes da reunião que inicia a próxima iteração, o Mestre Scrum tem uma reunião com a Equipa para fazer uma retrospectiva do que aconteceu na última iteração. Durante três horas, o Mestre Scrum incentiva a Equipa a rever, no âmbito da metodologia Scrum, o seu processo de desenvolvimento de modo a torná-lo mais eficaz na próxima iteração.

Todas estas reuniões periódicas são o meio de verificação da correcta implementação das práticas inerentes ao método Scrum. Este processo repete-se durante toda a duração do projecto, sabendo que à medida que o projecto for avançando novos requisitos e prioridades poderão ser definidos mediante as necessidades do Cliente. No entanto, nunca novos requisitos que possam surgir numa iteração podem ser desenvolvidos de imediato, o plano traçado para cada iteração não pode ser quebrado e, como tal, se surgirem novas necessidades elas terão de ser registadas e serão desenvolvidas em futuras iterações mediante o seu nível de prioridade.

2.4 Vantagens do Método Scrum

Os métodos de desenvolvimento tradicionais são concebidos para responder à imprevisibilidade dos ambientes externos e do desenvolvimento somente na fase inicial dos projectos. Abordagens como o método em Espiral são ainda mais limitadas na sua capacidade de responder a novas exigências uma vez que o projecto tenha começado [21].

Ao invés, o método Scrum é projectado para ser bastante flexível durante todo o processo de desenvolvimento fornecendo mecanismos de controlo para o planeamento do produto e gerindo as variáveis de ambiente à medida que o projecto avança. Estes factores permitem às organizações alterarem a sua definição do projecto a qualquer momento e obterem dessa forma o sistema que mais lhes convier.

O método Scrum permite à equipa de desenvolvimento tomar as decisões mais adequadas ao desenvolvimento do sistema mediante as alterações de requisitos que podem surgir. Pequenas equipas de programadores partilham de forma rápida e eficaz os seus conhecimentos sobre os processos de desenvolvimento e, consegue-se, dessa forma obter um óptimo ambiente de formação.

O método Scrum apresenta vantagens para quem encomenda o produto já que os sistemas disponibilizados, devido à sua grande flexibilidade, se adequam realmente às suas necessidades. Os clientes acabam por conseguir tirar a máxima vantagem competitiva oferecida pelo produto. E por outro lado, o método proporciona um ambiente de desenvolvimento realmente agradável para as suas equipas de desenvolvimento que acabam por poder gerir o seu trabalho da forma que acharem mais eficiente e vantajosa. Por estas razões, esta alternativa de desenvolvimento tem obtido

resultados significativos nas últimas décadas sendo o modelo adoptado pelas grandes empresas de desenvolvimento de software.

3. Métodos Alternativos para a Aprendizagem da Engenharia de Software

3.1 Introdução

O uso de jogos como método de aprendizagem da engenharia de software tem vindo a ser discutido por vários autores ao longo destes últimos anos [2, 6, 7, 10, 26]. Uma forte aposta foi feita na criação destes jogos com o objectivo de melhorar a qualidade da gestão de projecto, a aprendizagem dos processos da engenharia de software e, por consequente, melhorar de forma significativa o processo de tomada de decisão. Os resultados desta aposta estão visíveis em jogos tais como SESAM [26], Problems and Programmers [6], SIMSE [7] entre outros. Os seus criadores descobriram que o uso de jogos como complemento do ensino tradicional é bem mais eficiente do que como método único de ensino [6,26].

Porém e apesar da aposta na criação de experiências mais envolventes para os alunos de engenharia de software, a maioria dos estabelecimentos de ensino ainda recorre aos métodos de ensino tradicional como principal fonte de transmissão de conhecimento. Isso faz sentido no ensino de certas disciplinas, porém em áreas específicas que tratam de pessoas, processos e da aquisição de competências e habilidades na tomada de decisão recorrer a métodos alternativos como forma de aprendizagem torna-se muito mais eficiente. Os diversos elementos dos jogos [24] tais como as suas regras, objectivos, interacção, competitividade ou ênfase na resolução de problemas e tomada de decisão acabam por providenciar todos os ingredientes para uma fácil compreensão e aprendizagem da engenharia de software oferecendo diversão e motivação aos alunos.

Este capítulo serve de apresentação de alguns destes jogos com grande foco no jogo Problems and Programmers que serviu de base para o desenvolvimento deste trabalho.

3.2 SIMSE

Segundo [11], SimSE é um jogo de simulação jogado por um único jogador que deve desempenhar o papel de gestor de projecto liderando, para o efeito, uma equipa de programadores. Ao gerir o projecto de desenvolvimento de software, o jogador tem como funções contratar e despedir funcionários, atribuir tarefas à sua equipa, monitorizar o seu progresso, adquirir ferramentas que levem à melhoria do projecto, entre outras coisas. A interface gráfica é muito valorizada no jogo SimSE exibindo um escritório virtual em que o processo de engenharia de software decorre. O escritório é apresentado da forma mais real possível incluindo mesas, cadeiras, computadores e salas de reunião. Por outro lado, o jogador pode visualizar toda a informação de que precisa para poder controlar e gerir o projecto. Ele tem acesso à informação dos seus funcionários (a sua produtividade, o seu nível de energia bem como a tarefa que está a ser desempenhada), ao andamento das tarefas (tamanho, integridade da tarefa e nível de correcção), ao nível de satisfação dos seus clientes, ao orçamento e prazo de conclusão do projecto bem como aos ganhos obtidos com as ferramentas usadas. Os jogadores usam essa informação para poder tomar as decisões que melhor se apliquem ao andamento do seu projecto.

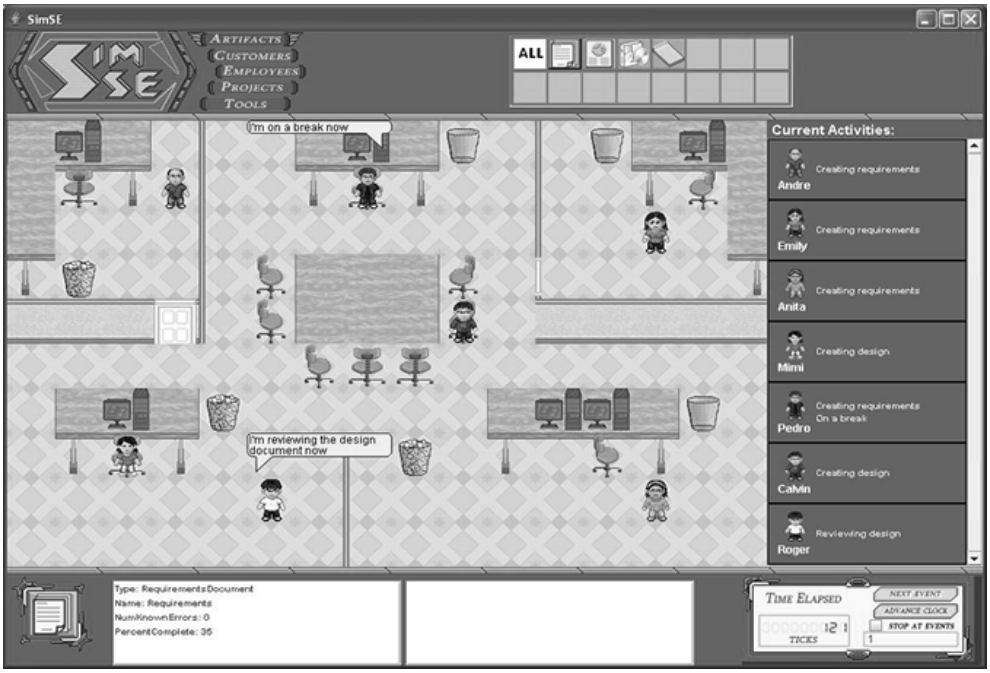


Figura 6 – Interface jogo SIMSE

Um dos principais objectivos do SimSE é permitir a personalização dos modelos de processo de software que ele simula. No mundo real, os processos de software variam de acordo com a cultura da organização e o seu domínio de aplicação e, como tal, SimSE deve ser capaz de retratar as diferenças que podem existir nestes processos. Além do mais, os educadores que recorrem ao jogo podem ter pensamentos e ideias diferentes quanto às melhores práticas de engenharia de software assim como aos resultados esperados com o ensino da disciplina. No sentido de ser vista como uma ferramenta de suporte à aprendizagem de diversos processos da engenharia de software, uma parte integrante do jogo é uma linguagem de modelação do processo de software com ferramentas associadas.

3.3 SESAM

Segundo [2], o projecto SESAM levou ao desenvolvimento de um simulador no qual um aluno pode desempenhar a tarefa de gestor de projecto. A simulação do projecto de software pode ser concluída em poucas horas pois ela é jogada num modo rápido. O jogador controla o simulador usando apensa uma interface textual, ou seja pela leitura e introdução de texto. Ele pode contratar ou demitir empregados e pedir-lhes para desempenhar qualquer uma das tarefas que são úteis ao desenvolvimento do software tais como preparar a especificação ou revisão do documento de concepção, ou ainda testar o código. Grande parte das mensagens que o jogador recebe são declarações dos empregados como por exemplo "Eu terminei a especificação", ou "Durante testes, eu detectei x bugs". O jogador é aconselhado a avaliar cuidadosamente essas declarações e reagir da forma mais apropriada já que é a única fonte de informação que ele tem para poder tomar as suas decisões. O simulador tem inúmeras variáveis internas mas estas não são visíveis ao jogador. Ele acede apenas à informação que qualquer gestor de projecto acede na realidade.

Quando o jogo termina, o jogador vê a pontuação que fez, ele pode analisar o seu desempenho recorrendo à ferramenta de análise do SESAM que lhe apresenta graficamente as variáveis internas do jogo. Nesta fase e para melhor analisar os seus erros, o jogador tem acesso a toda a informação do projecto a que ele não teve acesso antes.

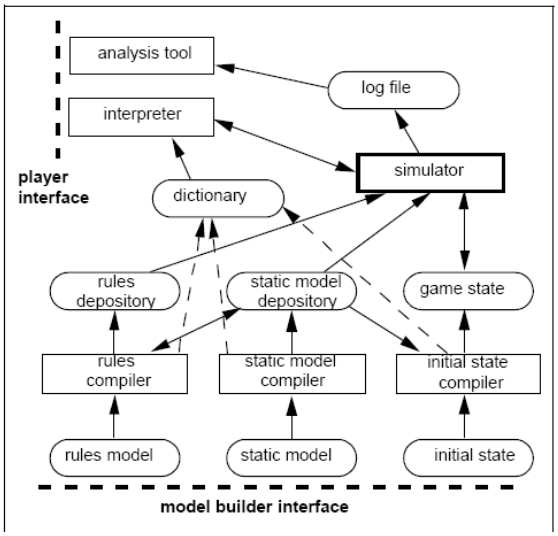


Figura 7 – Arquitectura do jogo SESAM

Em suma, SESAM pode ser descrito como um pequeno jogo de aventura que envolve muitas variantes internas enquanto o jogo está a decorrer, mas que depois se torna uma rica fonte estatística apresentando todos os dados até ao momento escondidos.

3.4 Planager

Segundo [9], o jogo Planager foi desenvolvido para apoiar o ensino de conceitos ligados à gestão de projectos. O jogo proposto não possui o objectivo de simular todos os processos da gestão de projectos, ele foca-se nos processos ligados ao planeamento, em particular em cinco processos de planeamento de duas das áreas de conhecimento do PMBOK 2004 (corpo de conhecimento em gestão de projecto do PMI – Project Management Institute). As áreas escolhidas são a gestão de objectivos e a gestão do tempo, pois elas possuem processos que servem de base para uma grande quantidade de outros processos que poderão ser utilizados em futuros módulos do jogo. Na gestão de objectivos foram escolhidos os processos da definição dos objectivos e da criação da Estrutura Analítica do Projecto. Na gestão de tempo foram escolhidos os processos de definição da actividade, da sequência de actividades e do desenvolvimento do cronograma (com foco no cálculo do caminho crítico).

O desenvolvimento da ferramenta apoiou-se na criação de um método de aprendizagem diferente dos métodos tradicionais, focando-se na didáctica e não tanto na

necessidade de decorar conceitos. Dois módulos foram criados: o módulo de tutorial, onde o aluno pode rever os conceitos de gestão de projectos aprendidos nas aulas e ter uma visão de como é o jogo; e o módulo jogo, onde o aluno pode praticar seus conhecimentos de uma forma interactiva. O objectivo do jogo é fazer com que o jogador passe por várias fases, sendo avaliado no final de cada uma delas.

O jogo possui dois tipos de utilizadores: o administrador e o jogador. O jogador pode utilizar o módulo tutorial para aprender sobre gestão de projectos e pode jogar em diversos cenários guardados na base de dados do jogo. Somente o administrador é capaz de adicionar, modificar e remover cenários que não são mais do que representações de projectos e são compostos por uma descrição e por cinco fases: objectivos, Estrutura Analítica do Projecto, definição das actividades, sequência de actividades e caminho crítico. Esta sequência de fases é geralmente a mesma sequência que os gestores utilizam num projecto real. O gestor de projectos utiliza as informações geradas pelos primeiros processos como entrada para vários outros (por exemplo, a definição de actividades gera informações que são utilizadas na definição da sequência de actividades). O jogador deverá utilizar as informações contidas nas fases anteriores para conseguir resolver correctamente as próximas fases do jogo.

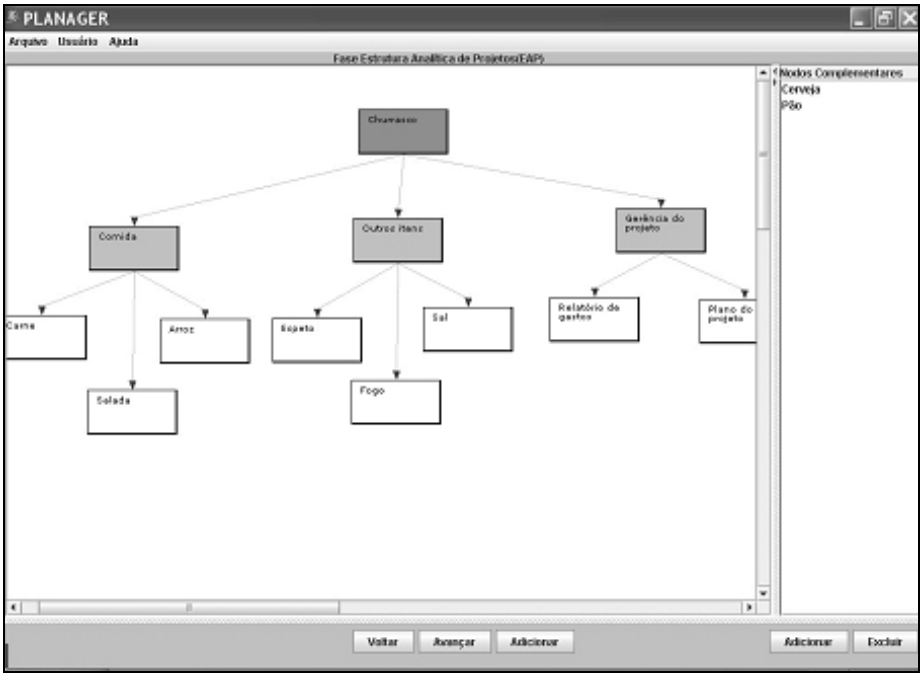


Figura 8 – Criação da Estrutura Analítica do Projecto

Como a ferramenta possui um módulo de criação de cenários, é possível optar por desempenhar o papel de administrador criando novos cenários, aprimorando os conceitos utilizados durante o jogo de forma mais aprofundada. Do ponto de vista de aplicabilidade dos resultados, o jogo está pronto para ser utilizado em disciplinas de graduação que ensinam conceitos básicos de gestão de projectos de software.

3.5 Scrumming

Segundo [8], o jogo Scrumming é um jogo educacional que simula o uso de algumas práticas do método Scrum, e busca suprir as necessidades encontradas no ensino dos métodos ágeis para gestão de projectos. O jogo não tem por objectivo simular todos os processos utilizados pelo Scrum. Ele foca-se na definição e simulação de iterações ou seja, na definição de um conjunto de tarefas realizadas durante um período pré-definido.

O jogo possui dois tipos de utilizadores, o administrador e o Scrum Master, membro da equipa de projecto que tem como responsabilidade aplicar os valores e práticas do Scrum, remover obstáculos, garantir a plena funcionalidade e produtividade da equipa e garantir a colaboração entre os diversos intervenientes. O jogo possui ainda dois módulos: o módulo administrativo, utilizado pelo administrador para levar a cabo as actividades que precedem a simulação (adicionar funcionários, adicionar ou remover tarefas associadas ao projecto) e o módulo de simulação que permite simular as iterações de um projecto. Para a simulação, cinco tipos de funcionários podem ser criados, a saber o gestor de projectos, o líder técnico, o programador, engenheiro de teste e elemento que testa o software. Além disso, para cada tipo de funcionário existem três níveis de experiência: sénior, intermédio e iniciante, variando na competência e produtividade.

Para simular uma iteração, é necessário registar um conjunto de actividades no Product Backlog que não é mais do que a lista das tarefas que devem ser realizadas durante o todo projecto ordenadas por nível de prioridade. De seguida, selecciona-se o subconjunto de tarefas com prioridade mais elevada para formar o Sprint Backlog (lista de tarefas que define o trabalho da equipe durante a iteração). Na simulação, o utilizador age como se fosse um Scrum Master, realizando tarefas tais como definir uma iteração, monitorizar o andamento da iteração através da taskboard (painel onde são colocadas as diversas informações relevantes para o acompanhamento da iteração), visualizar o

gráfico de Burndown (principal gráfico de controle do Scrum), e ainda adicionar ou remover tarefas do Product Backlog.

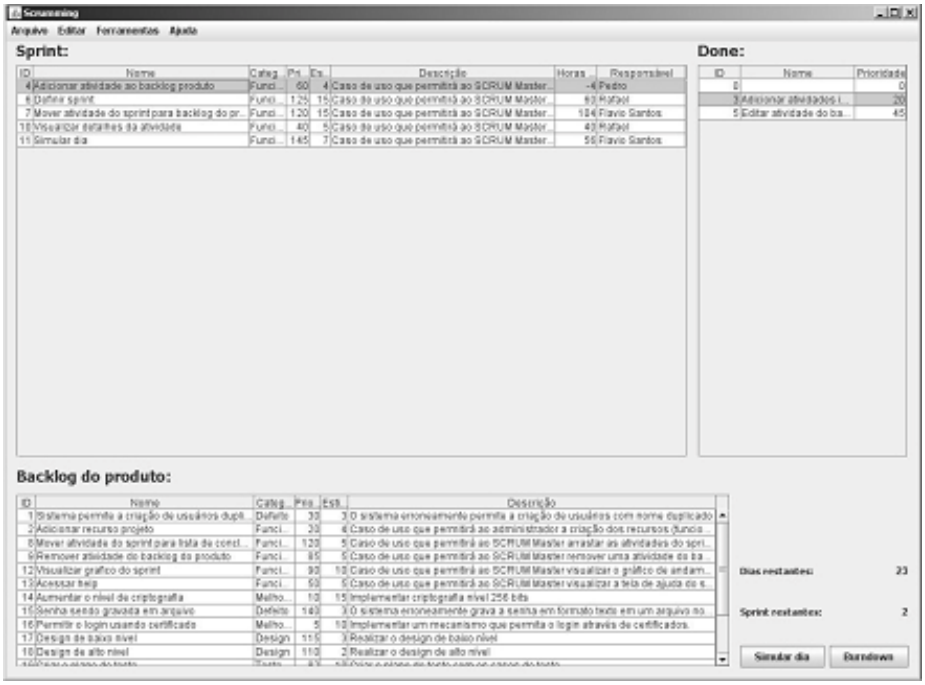


Figura 9 – Definição de uma iteração

Em relação ao seu público-alvo, inicialmente o jogo foi concebido para ser utilizado por profissionais da indústria ou alunos de graduação. Entende-se que, por apresentar os conceitos de forma genérica, é possível utilizá-la com foco na gestão de projectos como um todo e não apenas em projectos de software. Do ponto de vista da aplicabilidade dos resultados, o jogo está pronto para ser utilizado em disciplinas de graduação que ensinam conceitos básicos de gestão de projectos de software.

3.6 The Incredible Manager

Segundo [3], o jogo de simulação *The Incredible Manager* foi desenvolvido de forma a avaliar a aprendizagem baseada em jogos. Ao utilizar o jogo, o estudante é levado a agir como um gestor de projectos, planeando e controlando projectos de software para que estes sejam executados dentro do tempo e orçamento previstos. A construção do jogo é baseada em três elementos principais: um modelo de simulação, um simulador de modelos e uma máquina de jogo.

O jogo é dividido em cinco fases que apresentam as diversas etapas presentes no desenvolvimento de um projecto de software. A primeira fase, Início do Projecto, apresenta o projecto que vai ser gerido pelo jogador. O documento de descrição do projecto inclui a descrição do produto a ser entregue, os cenários que podem ter impacto no desenvolvimento do projecto e algumas das suas características tais como o nome, a organização, as tarefas a serem desenvolvidas, as normas de qualidade a serem seguidas, o cronograma e finalmente orçamento disponível para o projecto.

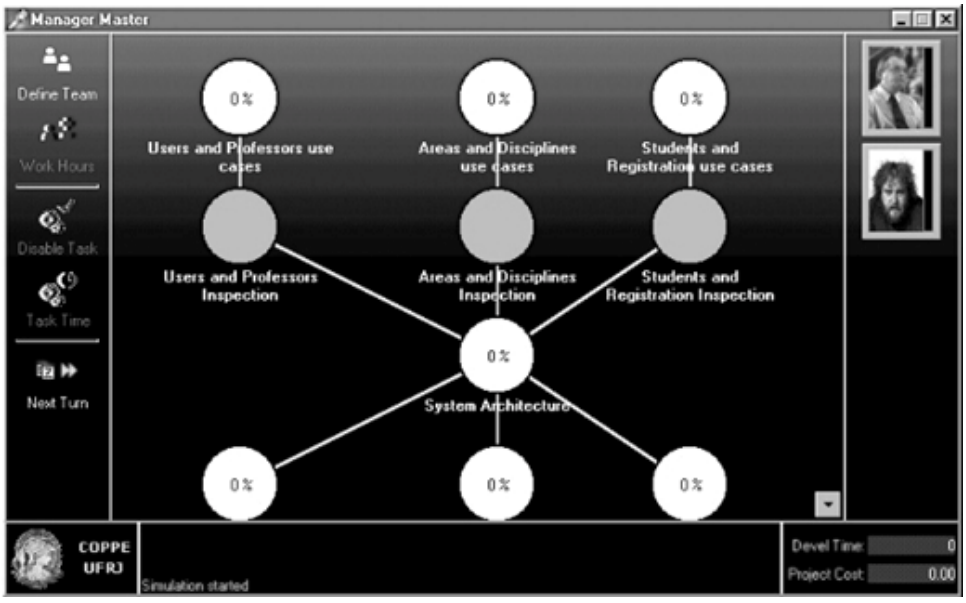


Figura 10 – Painel principal do jogo 'The Incredible Manager'

A fase seguinte é a de Planeamento do Projecto. Nesta o jogador deve elaborar um plano do projecto que contenha a sua equipa de programadores, o cronograma do projecto e a organização das tarefas a serem desenvolvidas no projecto. O jogador deve pesquisar e contratar programadores disponíveis no banco de currículos. Cada programador, conforme mencionado anteriormente, possui características individuais. Uma vez definida a equipa, é preciso alocar os programadores às diversas tarefas do projecto. Cada tarefa só pode ser executada por um único programador ao mesmo tempo e o jogador também deve definir o esforço (em número de dias) que ele deseja para cada uma. O esforço em tarefas de qualidade (inspecções) é opcional: o jogador pode querer ou não realizar este tipo de tarefa. O plano do projecto pode ser modificado pelo jogador a qualquer instante durante a execução do jogo, demitindo e contratando programadores, modificando a carga horária de trabalho diária de cada um (de 8 a 12

horas de trabalho por dia), a duração das tarefas e os programadores alocados a cada uma delas. No final do planeamento, o plano de projecto exibe o total do esforço e custo estimados pelo jogador para desenvolver o projecto.

A terceira fase presente no jogo é a da Aceitação do Planeamento. Uma vez feito o plano de projecto, este deve ser enviado pelo gestor para avaliação do chefe. O plano pode ser aprovado ou não. Um plano é recusado se as suas estimativas estiverem muito acima das restrições descritas no documento de apresentação do projecto. Se o plano for recusado, o jogador deve refazer o planeamento, ajustando as suas estimativas de dias e orçamento até que o plano seja aprovado com sucesso. O jogador, desta forma, deve tomar importantes decisões de custo/benefício na escolha da equipa, alocação desta e duração de tarefas para conseguir atender aos requisitos propostos pelo jogo.

Na fase de Execução do Projecto, as tarefas do projecto são executadas segundo o plano de projecto aceite anteriormente. As mudanças gráficas das personagens são variações nos desenhos dos programadores e balões de pensamento. Cada programador transita pelo escritório, indo, vindo e sentando na sua mesa de trabalho durante o decorrer de um dia. Os balões indicam se ele está motivado, cansado ou em estado de pânico. Um programador entra em pânico no caso de o projecto estar atrasado, sem fundos ou da tarefa que ele está executando estar atrasada. Uma personagem especial, o chefe, aparece para fazer reclamações quando o projecto está atrasado, sem fundos ou sem tempo. Uma outra forma de indicar as alterações de estados do modelo é pela emissão de relatórios. Um relatório é emitido quando um programador começa ou termina uma tarefa, quando ele percebe que a tarefa que ele executa está atrasada e quando ele se sente cansado por trabalho excessivo. Caso o chefe perceba que o projecto está atrasado, sem fundos ou sem tempo hábil para a sua conclusão, também são emitidos relatórios para alertar o gestor.

Na parte inferior da tela de execução, estão os recursos associados ao projecto (dias e fundos). A cor vermelha indica se há deficiência em algum deles. Os recursos decrescem a cada instante de tempo, uma vez que a execução do projecto acontece em turnos contínuos (porém ajustáveis pelo jogador). Com a passagem ininterrupta de tempo, o jogador deve ficar atento ao comportamento do projecto e da equipa, de forma a tomar as medidas necessárias para controlar o projecto e garantir que ele seja concluído com sucesso. Para evitar o fracasso, o jogador pode modificar o plano de projecto original durante o desenvolvimento. Desta forma, diferentes decisões tomadas por diferentes jogadores em diferentes circunstâncias fazem do jogo uma experiência

não linear, permitindo que os alunos vivam a experiência de gerir o mesmo projecto de forma diferente.

Finalmente a última fase, a fase de Fim do Projecto ocorre de duas formas. A primeira acontece quando os recursos disponíveis para o projecto (dias e fundos) acabam sem que todas as tarefas previstas para o projecto tenham sido concluídas. Neste caso, o projecto foi finalizado com fracasso. Por outro lado, quando todas as tarefas são concluídas dentro das estimativas e recursos previstos pelo planeamento, o projecto é finalizado com sucesso.

3.7 O Jogo 'Problems and Programmers'

O jogo de cartas 'Problems and Programmers' desenvolvido no ano de 2003, na Universidade da Califórnia, é o resultado de um trabalho de investigação realizado por Alex Baker sob orientação do Professor André van der Hoek. O seu principal objectivo é ensinar as boas práticas do Modelo Cascata da Engenharia de Software permitindo aos alunos terem a percepção do processo de software como um todo [1]. Segundo [1], cada evento no jogo deve ser associado a uma acção real permitindo desta forma que as regras do jogo sejam assimiladas de uma forma fácil e permitindo, por outro lado, que o jogo se torne realmente relevante no ensinamento das boas práticas da engenharia de software.

Aquando a concepção do jogo, os seus criadores tiveram o cuidado de o desenvolver apoiados em vários princípios para torná-lo num método de aprendizagem realmente eficaz e de fácil uso. As orientações seguidas por eles foram:

- **O jogo deve ensinar as regras gerais e específicas dos processos da engenharia de software**. Ambas as partes são essenciais para um real entendimento da execução dos processos de engenharia de software. As regras gerais incluem ideias tais como as que um processo de engenharia de software é orientado por diversos intervenientes ou ainda que a engenharia de software é um processo não linear. Estas ideias devem ser complementadas pelo ensino de regras mais específicas que permitam apresentar riscos associados ao processo que podem comprometer a sua boa execução [14] [15].
- **O jogo deve promover as boas regras da engenharia de software**. É importante que o jogo recompense os concorrentes que pratiquem as boas regras da engenharia de software e, pelo contrário, penalize quem não as cumpra. A má utilização

dos recursos associados ao projecto ou qualquer outro procedimento que se desvie dos procedimentos da engenharia de software podem originar resultados negativos que devem ser evidenciados no jogo [22].

- **O jogo deve ser de fácil aprendizagem**. Um doa principais atributos do jogo deve ser a sua capacidade de dar uma visão alargada dos processos da engenharia de software num período bastante reduzido. A simulação será bastante desvalorizada se os tempos de aprendizagem e do próprio jogo forem muito lentos [23] [16].
- **O jogo deve ser divertido.** A diversão oferecida pelo jogo vai melhorar a memorização das lições ensinadas [23]. O jogador tem de sentir que está realmente a liderar um projecto de modo a conseguir perceber as situações por que passa.

Em suma, o jogo deve ser prático, divertido e ensinar as boas práticas dos processos de engenharia de software. Cumprindo estes objectivos será criada uma ferramenta de ensino inovadora e eficaz [1].

3.7.1 Visão global do Jogo

O jogo Problems and Programmers é um jogo de competição em que cada aluno desempenha o papel de um gestor de projecto. Eles têm de liderar o mesmo projecto e ganha o jogador que conseguir concluir em primeiro o projecto. No entanto, para conseguirem terminar o projecto, os concorrentes devem conseguir gerir o seu orçamento, satisfazer as exigências do cliente e produzir um software de elevada qualidade, entre outras coisas. No fundo, eles devem seguir as boas práticas da engenharia de software de modo a evitar todas as contrariedades inerentes a um projecto e que os podem penalizar face aos seus concorrentes. As boas práticas e as condutas que devem ser evitadas no desenvolvimento de um projecto transmitidas pelo jogo encontram-se sumarizadas numa lista de 85 regras colectadas pelos criadores do jogo através de leitura científica sobre engenharia de software. Esta lista reune as melhores práticas da engenharia de software quer a nível académico quer a nível empresarial e podem ser encontradas em http://www.ics.uci.edu/~emilyo/SimSE/se_rules.html. O método de desenvolvimento de software a ser seguido no jogo é o Modelo Cascata e, como tal, o software desenvolvido pelos jogadores terá de passar pelas diversas fases do modelo (análise, modelação, desenvolvimento, integração e testes) de forma sequencial.

No entanto, o jogo transmite uma abordagem mais flexível do modelo permitindo aos jogadores voltar a uma fase anterior de desenvolvimento ou avançar uma próxima se necessário mediante uma penalização.

Independentemente do jogador poder ter qualquer tipo de acção em qualquer momento do jogo, este é fortemente levado a seguir as regras do modelo Cascata. Por exemplo e de acordo com [1], no início de um jogo, o jogador poder obter cartas de requisitos, cartas de modelação ou cartas de codificação. A criação de cartas de requisitos torna obsoleto qualquer tipo de documento de modelação existente e, como tal algumas cartas de modelação são perdidas. Desta forma, os jogadores podem entender que eles devem definir grande parte dos requisitos do sistema antes de iniciar qualquer tipo de modelação, pois de outra forma os requisitos podem ser alterados e a modelação do sistema terá de ser feita de novo de acordo com os novos requisitos.

À medida que vão passando pelas diversas fases do modelo Cascata, os jogadores colocam cartas da esquerda para a direito como demonstrado na Figura 6.

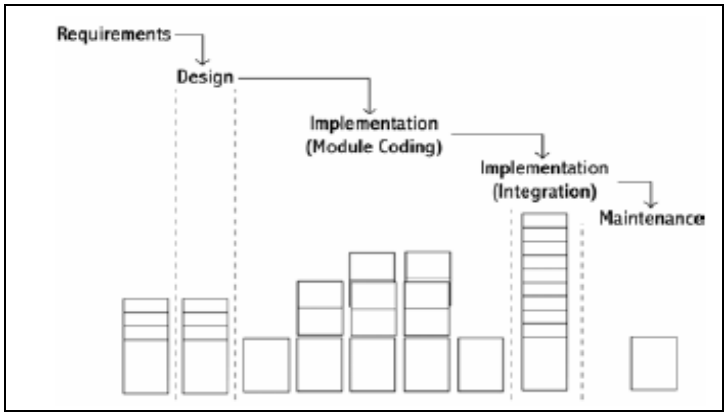


Figura 11 – Fases do Modelo Cascata e do jogo Problems and Programmers

Os jogadores começam, como tal, por criar uma coluna com cartas de análise construindo de seguida uma coluna de cartas de modelação à direita da primeira. Eles continuam o jogo adquirindo e jogando cartas de codificação que serão, no final, agrupadas numa coluna de integração de código mais à direita. Desta forma, o progresso pelas diversas fases do modelo é apresentado de forma simples e os jogadores podem facilmente monitorizar o seu progresso.

De acordo com [1], mesmo seguindo o ciclo de vida tradicional de desenvolvimento de software, o jogo dá liberdade aos jogadores de definirem qual a melhor forma de atingirem o seu objectivo. Enquanto certos jogadores vão dar grande ênfase à fase de análise e modelação e inspeccionar cuidadosamente o código desenvolvido antes de integrá-lo, outros vão apressar todas estas fases de modo a entregar o projecto o mais rápido possível. O jogo permite observar a diferenças estratégicas dos vários concorrentes e apresentar um exemplo poderoso das consequências inerentes a cada uma das estratégias utilizadas na engenharia de software.

3.7.2 Preparação para o Jogo

No início do jogo, os jogadores devem baralhar os 4 montes de cartas existentes que se dividem entre:

- O monte das cartas de codificação que contém todas as cartas de codificação usadas pelos jogadores para terminar o seu projecto;
- O monte das cartas de documentação que serão usadas na fase da análise de requisitos;
- O monte das cartas de projecto que contém os 9 projectos que compõem este jogo;
- O monte principal que contém todas as cartas de conceitos, os programadores e as cartas de problemas;

Após terem dividido as cartas nos seus respectivos montes, um jogador deve escolher uma carta de projecto que determinará o projecto a ser desenvolvido. Esta define as características do projecto a ser entregue tais como o seu tamanho, complexidade, qualidade e orçamento como apresentado na Figura 7.

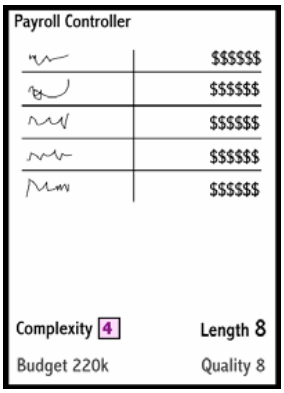


Figura 12 – Carta de Projecto

A complexidade do projecto determina o grau de dificuldade da escrita do código e acaba por determinar qual o nível de conhecimento que devem possuir os programadores para obter cartas de codificação. Por sua vez, o tamanho do projecto determina a quantidade de código necessária à execução do projecto, enquanto a qualidade determina qual a quantidade de código que deve ser revisto para detecção de erros no final do projecto. Finalmente, o orçamento do projecto limita o número de programadores e conceitos que um jogador pode ter, forçando-os a tomar decisões sobre o que é realmente fundamental para completar a execução do projecto. Cada conjunto de atributos associados a uma carta de projecto requer uma abordagem diferente que permite recriar vários cenários possíveis.

Após ter sido escolhida a carta de projecto, os jogadores têm um tempo para reflectir na melhor forma de levar a cabo o projecto, cada um deles recolhe cinco cartas do monte principal de cartas. Ser-lhe-ão entregue três tipos de cartas: conceitos, programadores e problemas.

As cartas de conceitos, ilustradas pela Figura 8, apresentam as decisões que um jogador pode tomar mediante a sua abordagem ao projecto.

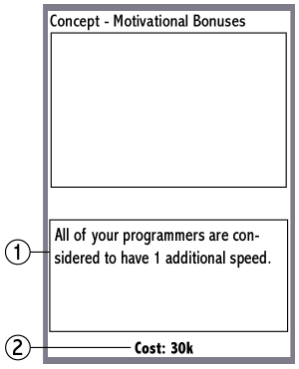


Figura 13 – Carta de Conceitos

Estas cartas são compostas por duas partes essenciais:

1) Efeito: o que a carta permite fazer;
2) Custo: certas cartas de conceito têm um custo associado e se este for muito elevado, a carta de conceito não pode ser usadas;

As cartas de programadores são os trunfos dos jogadores e são necessárias para escrever, inspeccionar e alterar código. Elas são compostas por 3 atributos: salário, nível de conhecimento e personalidade que o jogador terá de analisar antes de ficar com a carta.

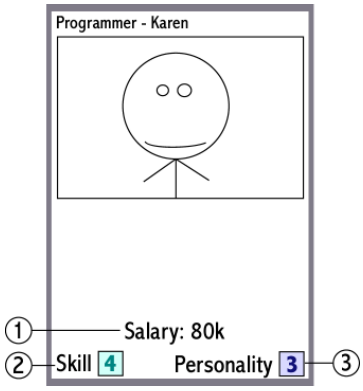


Figura 14 – Carta de Programador

O atributo salário dá-nos indicação de qual o custo inerente à contratação do programador e o jogador deve ver se o seu orçamento lhe permite tal aquisição antes de tomar qualquer decisão. O nível de conhecimento do programador dita a sua capacidade em desenvolver o código inerente ao projecto seleccionado. O valor deste atributo indica o número de pontos que o programador possui a cada turno. Cada acção tomada por um programador requer um certo número de pontos e como tal, quanto maior foi o valor deste atributo mais

acções pode tomar um programador. O nível de conhecimento é avaliado numa escala de 1 a 5. A personalidade do programador é igualmente evidenciada demonstrando, desta forma, que não basta ter grandes conhecimentos técnicos para se ser um bom profissional. Esta característica determina a sua capacidade em seguir as boas práticas de engenharia de software, a sua capacidade em trabalhar em equipa, assim como a sua amabilidade para com os outros. Quanto menor for o valor do atributo personalidade mais será o programador vulnerável às cartas de problemas.

As cartas de problemas são o ponto crucial do jogo 'Problems and Programmers' e são jogadas em confrontos directos de jogadores. Se o jogador 'atacado' possui as características contempladas pela carta, ele terá de sofrer as consequências definidas na carta.

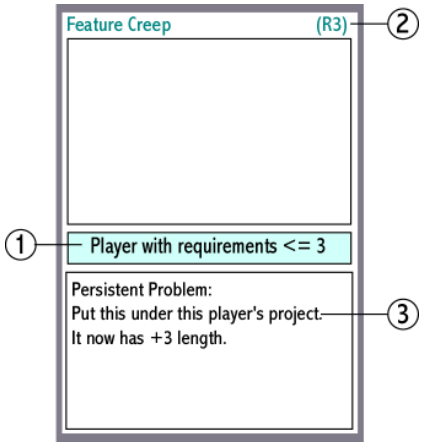


Figura 15 – Carta de Problemas

Esta carta é composta por 3 partas:

1) Critério: condição que deve existir para que a carta tenha algum efeito num adversário. Por exemplo, na carta de problemas ilustrada pela Figura 10, somente um jogador com 3 ou menos cartas de análise poderá ser contemplado;
2) Abreviatura do Critério: situada no canto superior direito da carta de problemas, apresenta de forma rápida qual o critério da carta;
3) Efeito: qual a consequência da carta de problemas sobre o jogador que a vai receber. Uma grande parte das cartas de problemas é usada e imediatamente retirada da mesa de jogo. No entanto, as cartas de problemas persistentes como a carta representada pela Figura 10 são jogadas e perseguem o jogador que as recebeu;

3.7.3 As Rondas

Após cada jogador ter as suas cinco cartas, o jogo inicia-se. Em cada ronda os jogadores devem passar pelas seguintes etapas:

- Decidir se passam para a próxima fase do modelo;
- Ir buscar cartas;
- Agir mediante a fase em que está;
- Jogar uma carta de programadores ou de conceitos;
- Descartar as cartas indesejadas;

Esta estrutura faz com que as cartas se desloquem dos seus montes para as mãos dos jogadores e, por sua vez, para a área de jogo. De acordo com [1], o jogo dificulta a compra de programadores e conceitos. Se os jogadores gastarem todo o seu orçamento, eles não poderão contratar mais programadores até à próxima ronda que não poderão adquirir cartas de codificação na ronda em que forem adquiridos. O jogo permite, deste modo, evidenciar o tempo de adaptação de um programador a um novo ambiente. O passo mais importante de cada ronda é a acção que o jogador vai tomar mediante a fase em que se encontra. É neste passo que são postas em práticas as regras inerentes ao desenvolvimento de software seguindo o modelo Cascata.

3.7.4 A Fase da Análise de Requisitos

Os jogadores são incentivados a manterem-se nesta fase e jogarem cartas de análise. As regras do jogo não obrigam os jogadores a permanecerem uma ronda sequer nesta fase, no entanto, os jogadores rapidamente descobrem que é necessário despender tempo na elaboração de um documento de análise para conseguir concluir o projecto. Em cada turno, quando estão nesta fase, os jogadores retiram duas cartas de requisitos e colocam-nas à sua frente para evidenciar o trabalho que foi gasto por eles na elaboração do seu plano de requisitos. Quantas mais cartas de requisitos o jogador tiver, menos vulnerável ele será às cartas de problemas.

Grande parte das cartas de requisitos estão vazias no entanto, algumas delas estão marcadas como incertas. As cartas de problemas podem causar problemas aos jogadores com cartas de requisitos incertos e, como tal, estes podem substitui-las (até duas por turno) por outras. Estas cartas de requisitos incertos permitem demonstrar ao

jogador que não basta elaborar um documento de análise dos requisitos do sistema, este deve ter qualidade e ser rigoroso para que o projecto possa ser executado com sucesso. No entanto, um jogador não pode passar demasiado tempo na fase de análise pois, como já foi dito, o objectivo do jogo é terminar o projecto em primeiro. Como tal, o jogador vê se obrigado a ponderar a sua estadia nesta fase, analisando o seu jogo e as possíveis situações e constrangimentos que possam surgir.

3.7.5 A Fase de Modelação

A fase da modelação desenrola-se de forma semelhante à fase de análise, excepto que nesta fase os jogadores produzem cartas que representam a modelação do sistema. Tal como para a fase de análise, os jogadores não são obrigados a ficar nesta fase uma ronda sequer. No entanto, e como na fase anterior, existe um grande número de cartas de problemas que podem inviabilizar jogadores com uma fraca modelação do sistema. Por isso, e como para a fase de análise, esta fase é apresentada como crucial para a boa execução de um projecto de desenvolvimento de software.

3.7.6 A Fase de Implementação

A fase de implementação surge quando os jogadores determinarem sua modelação suficiente e se sentirem preparados para reagir a qualquer adversidade de uma carta de problemas sem comprometer a boa execução do desenvolvimento do projecto. Na fase de implementação, as cartas de programadores podem agir mediante o nível de conhecimento que têm. As opções que podem tomar incluem:

- Produzir código de elevada qualidade que demora o tempo referenciado pela complexidade do projecto na carta de projecto;
- Produzir código apressadamente que demora metade do tempo referenciado pela complexidade do projecto na carta de projecto;
- Inspeccionar código. Por um ponto, um pedaço de código pode ser inspeccionado e desta forma a sua carta fica virada para cima;
- Resolver erros. Por um ponto, um programador pode trabalhar na resolução de erros do seu código.

O uso destas acções permite ilustrar diversas formas de desenvolver código, enquanto certos programadores vão produzir código de elevada qualidade, inspeccionando-o e

resolvendo seus erros, outros vão criar código de forma apressada e inspeccioná-lo somente no final do projecto.

Cada carta de codificação terminada é alocada em frente ao programador que a desenvolveu com a cor vermelha que define que o código foi desenvolvido de forma apressada ou pela cor azul que indica o desenvolvimento de código de qualidade seleccionada. Não se sabe a carta possui ou não bugs até ela ser inspeccionada, e quando isto acontece a carta é virada para cima com a cor escolhida como demonstra a Figura 11.

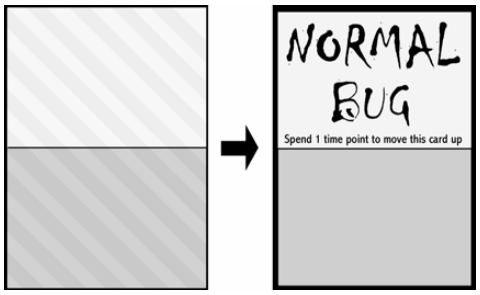


Figura 16 – Carta de código

Como se pode verificar na figura acima, esta carta de código desenvolvido apressadamente possui um erro. Certas cartas possuem erros em ambos os lados, outras por sua vez não guardam qualquer tipo de erro, e outras cartas, como a representada pela Figura 11, só possuem erros do lado do código desenvolvido apressadamente. Existem igualmente três tipos de erros no jogo. Os erros simples podem ser substituídos por novas cartas por um ponto somente. Os erros normais não podem ser eliminados directamente. Cada ponto gasto com um erro normal faz com que este suba de um lugar na sua coluna e somente quando chega ao topo ele pode ser removido definitivamente. Finalmente, surgem os erros graves que, quando detectados, descartam a carta acima deles. Desta forma, o jogo demonstra com clareza as implicações que os erros no código podem trazer.

3.7.7 A Fase da Integração

Após a fase da implementação surge a fase da integração do código. Cada ronda desta fase permite a integração do código de um programador. O jogo só termina quando o código requisitado pelo projecto está concluído e integrado. Isto significa que

os projectos com um grande número de programadores desenvolvendo código vão demorar mais tempo na fase da integração. De acordo com [1], desta forma é possível ilustrar as dificuldades existentes na integração de código de um grande número de programadores, demonstrando as vantagens em usar equipas mais pequenas.

3.7.8 A Entrega do Produto

A última fase do jogo consiste na entrega do produto. Durante a fase de implementação o código pode não ser inspeccionado e como tal os erros podem não ser detectados. No entanto, se erros são descobertos na entrega do projecto o jogador pode perder o seu cliente. Nesta fase, o jogador baralha todas as suas cartas de codificação e vira o número de cartas igual ao número da qualidade na carta de projecto. Se forem detectados erros simples ou normais o código terá de ser novamente implementado e integrado, o que atrasará bastante o jogador. Se por sua vez forem detectados erros graves o jogador perde o jogo. Desta forma os jogadores entendem com clareza que podem ser corridos certos riscos, no entanto um produto de engenharia de software tem de ter elevada qualidade. Pelo contrário, se não for detectado nenhum erro, o jogador ganha o jogo.

Em suma, a última fase do jogo envolve bastante o factor sorte e é possível que projectos saem com algum erro. No entanto, raramente isso acontece e como tal estes riscos acabam por não serem corridos. De um modo geral, o jogador que toma todas as precauções e providências costuma ser o grande vencedor do jogo. Os jogadores precisam de jogar várias partidas para entenderem realmente qual o nível de prudência que precisam de ter para levar a cabo o projecto e vencerem o jogo. Desta forma eles vão jogando e assimilando as práticas da engenharia de software.

4. Play Scrum

4.1 Introdução

O jogo de cartas ‘Play Scrum’, apoiado no seu antecessor ‘Problems and Programmers’ é um jogo de competição em que cada aluno desempenha o papel de um Scrum Master, sendo que neste o desenvolvimento de software segue as práticas definidas pelo método ágil SCRUM.

Play Scrum deve ser jogado por 2 a 5 jogadores. O jogo é dividido em iterações que diferem de projecto para projecto e durante as quais cada jogador deverá desenvolver um número de tarefas definido no início do jogo.

Os recursos do jogo são:

- As cartas de Product Backlog que determinam as características do projecto;
- Um tabuleiro por jogador que permite a cada jogador dispor as suas cartas assim como garantir de forma fácil que um jogador não ultrapassa o orçamento definido para o projecto;
- As cartas de problemas que são lançadas pelos adversários de um jogador;
- As cartas de conceitos que são usadas, pelo jogador, como defesa às cartas de problemas lançadas pelos seus adversários;
- As cartas de programadores que correspondem à equipa de desenvolvimento de cada jogador;
- Os artefactos que representam as tarefas realizadas pelo jogador durante o projecto;
- Um dado;

Vence o jogador que conseguir realizar o número de tarefas sem erros definido para o projecto antes do fecho da última iteração ou o jogador que, após o fecho da última iteração, tiver a maior percentagem de tarefas sem erros.

4.2 O Product Backlog

Os cartões Product Backlog providenciam toda a informação relativa ao projecto. No método Scrum, o Product Backlog é o guião do projecto onde são definidas as iterações que o vão compor e distribuídas as tarefas. As tarefas são alocadas nas iterações conforme a sua prioridade no projecto. As tarefas com prioridade mais alta serão desenvolvidas em primeiro e isso para o cliente conseguir ter o retorno do seu investimento o mais rápido possível. A Figura 17 apresenta um exemplo de uma carta Product Backlog do Play Scrum. As informações que podemos encontrar no Product Backlog são:

- **Descrição**: texto em linguagem natural que descreve as principais características do projecto. Esta descrição torna o jogo mais realista e ajuda o jogador a compreender os principais requisitos do sistema.
- **Sprints**: indica o número de iterações que vai ter o projecto. Cada iteração equivale e 4 rodadas. Como tal, se como na figura 17 um projecto tiver 7 iterações, então o jogo será disputado em 28 rodadas.
- **Complexity**: este valor indica o número de pontos que um programador tem de gastar para adquirir um bom artefacto. Artefactos de má qualidade custam metade deste valor. A complexidade do projecto pode ser igual a 2 ou a 4.

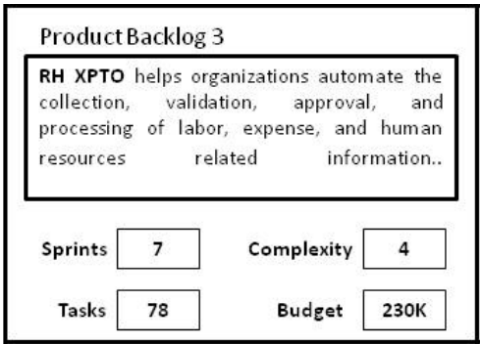


Figura 17 – "Product Backlog"

- **Tasks**: Este atributo determina o número de artefactos sem erros que o utilizador deverá completar para ganhar o jogo. Este número poderá não ser atingindo nas rodadas definidas para o projecto, e nesse caso vencerá o jogador que mais se aproximar desse valor.
- **Budget**: Dinheiro disponível para cada jogador gastar durante o projecto. Este atributo é a restrição para a contratação de programadores bem como para o uso de certas cartas de conceitos. Durante todo o jogo o valor dos salários de todos os programadores somado com o custo dos conceitos não deve ultrapassar o orçamento do projecto.

4.3 O Tabuleiro

A análise ao jogo Problems and Programmers permitiu identificar certas lacunas presentes no jogo. Uma delas provinha da inexistência de um tabuleiro para os jogadores disporem as suas cartas. À medida que os participantes iam adquirido cartas, tornava-se muito complicado identificar com clareza a que fase uma determinada carta pertencia. Como tal, no Play Scrum foi introduzido o elemento tabuleiro para permitir ao jogador dispor melhor o seu jogo.

O tabuleiro é uma área na qual cada jogador dispõe os seus programadores em colunas e os artefactos adquiridos em linhas. As cartas de artefactos são colocadas nas células do tabuleiro, abaixo do programador que as adquiriu. Os artefactos devem ser igualmente separados em dois. Por um lado, temos os artefactos adquiridos na rodada em questão que ficam na primeira linha e por outro, os artefactos adquiridos nas rodadas anteriores da iteração. Esta separação existe devido a certas cartas de problemas que diferenciam este aspecto.

Finalmente, o tabuleiro permite controlar de forma mais fácil que um jogador não ultrapasse o orçamento definido para o jogo. E isto porque o tabuleiro permite ao aluno dispor todos os seus programadores e conceitos de forma organizada e visível para todos os participantes do jogo, sendo desta forma muito mais fácil detectar qualquer fraude.

	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5
Tasks produced in this round					
Tasks produced in previous rounds					

Figura 18 – Tabuleiro

4.4 As cartas

Os principais recursos do jogo Play Scrum são as cartas que compõem o jogo. Estas dividem-se em quatro tipos: problemas, conceitos, programadores e artefactos.

- **Problemas**: as cartas de problemas do Play Scrum tanto podem descrever os problemas clássicos que decorrem no desenvolvimento de projectos de engenharia de software assim como os problemas inerentes ao não respeito das normas impostas pelo método Scrum. Essas cartas são utilizadas como obstáculos ao progresso dos jogadores adversários. Elas possuem os seguintes atributos como demonstrado na Figura 19, a):
    a) Definição do Problema – pequena frase que apresenta ao utilizador o problema a ser tratado. Desta forma o aluno consegue perceber quais os problemas que podem afectar um projecto sob o método Scrum.
    b) Condição para que se aplique o problema – o problema pode afectar qualquer jogador, ou afectar somente um jogador que reúna determinadas características. Esta diferença depende do problema e realça o facto de existirem variáveis externas ao projecto que o podem levar ao fracasso.
    c) O efeito resultante da carta – este atributo define qual a sanção a ser aplicada ao jogador por ter recebido essa carta.

  As cartas de problemas são as principais fontes de retorno negativo para o jogador e ocorrem, em grande parte, quando este toma más decisões do ponto de vista da engenharia de software. Desta forma, o jogador pode reconhecer os erros que cometeu e associá-los a eventos que ocorrem no mundo real.

- **Conceitos**: as cartas de conceitos descrevem as boas práticas da engenharia de software. Essas cartas podem ser utilizadas pelos jogadores para avançarem face ao seu objectivo e contraporem as cartas de problemas lançadas pelos seus adversários. Como demonstrado na Figura 19, b) os principais atributos das cartas de conceitos são:

a) Definição do Conceito – pequena frase que apresenta ao utilizador o conceito. Desta forma o aluno consegue perceber quais as boas condutas que levam a uma mais-valia no desenvolvimento de um projecto sob o método Scrum.
b) O efeito resultante da carta – este atributo define o efeito positivo que tem o conceito.
c) O custo – certas cartas de conceito têm um custo associado.

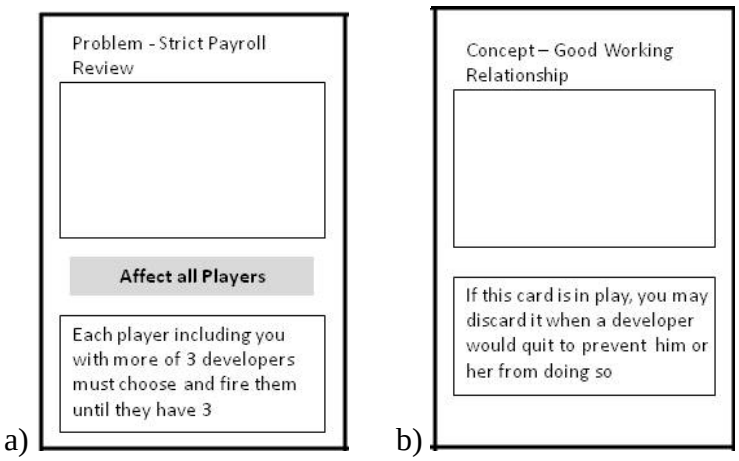


Figura 19 – Cartas de Problemas e Conceitos

- **Programadores**: este tipo de carta é o principal recurso do jogador para progredir no jogo. Os programadores produzem artefactos (tarefas) que são necessários para terminar o projecto. As cartas de programadores apresentam as seguintes informações (ver figura 20):
    a) O nome do Programador.
    b) Uma pequena descrição pessoal do programador.
    c) O seu salário – este atributo é importante pois a soma dos salários dos programadores e das cartas de conceito não deve ultrapassar o orçamento previsto para o projecto.
    d) Personalidade – este atributo (*"personality"*) reflecte a tendência do programador em ser um bom empregado. Este aspecto é bastante relevante na hora de receber cartas de problemas. A personalidade é medida numa escala de 1 a 5.

e) Conhecimento – este atributo (*"skill"*) reflecte o nível de conhecimento do programador. Este atributo é de grande relevo pois determina os pontos que possui um programador e que lhe permite desempenhar acções durante o jogo (adquirir, inspeccionar e corrigir artefactos). O conhecimento é medido numa escala de 1 a 5.

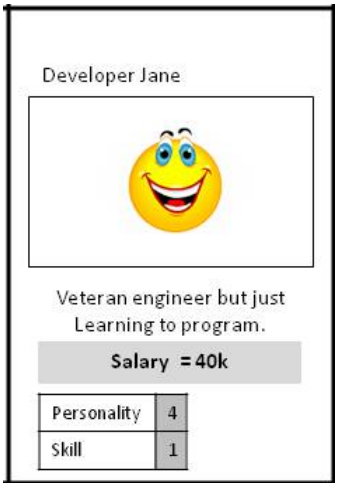


Figura 20 – Carta de Programador

- **Artefactos**: correspondem às tarefas desenvolvidas pelos programadores que podem ou não conter erros. Existem dois tipos de artefactos, os artefactos azuis e os artefactos vermelhos.
    a) Os artefactos azuis são considerados bons pois, independentemente de poderem conter erros, a probabilidade disto acontecer é de 1/5. Eles podem ser adquiridos pelo valor do atributo "complexidade" do Product Backlog.

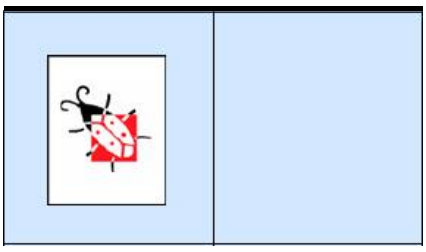


Figura 21 – Artefacto Bom

b) Os vermelhos, por sua vez, são vistos como artefactos maus pois a probabilidade de conterem erros é de 3/5. Como tal, estes podem ser adquiridos por metade do valor definido no atributo “complexidade” do Product Backlog.

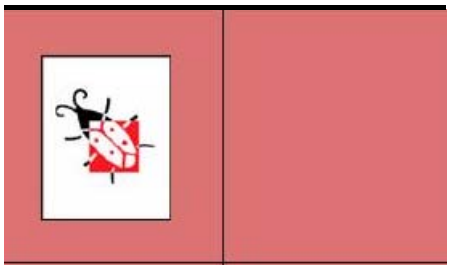


Figura 22 – Artefacto Mau

4.5 Intervenientes no Play Scrum

Existem dois tipos de intervenientes no jogo Play Scrum: o jogador e o moderador.

O jogador é todo aquele que joga o jogo simulando ser um Scrum Master e que tenta desenvolver todas as tarefas definidas pelo Product Backlog antes do final da última iteração.

O moderador, por sua vez, pode ser jogador. O seu papel no jogo é o de apontar num cartão o número de tarefas com erros e sem erros desenvolvidas por cada jogador no final de cada iteração. Ele deve, igualmente, colocar todas as tarefas no seu monte após as ter contado.

4.6 A dinâmica do jogo

Antes do início do jogo, um cartão ‘Product Backlog’ é escolhido aleatoriamente de uma série de cartões de ‘Product Backlog’ disponíveis.

Esta carta determina qual o número de iterações (“*Sprints*”) que compõe o projecto e qual o número de tarefas (“*Tasks*”) que devem ser realizadas para terminar o jogo. O cartão define ainda o nível de complexidade (“*Complexity*”) do projecto que determina

o custo dos artefactos, assim como o orçamento (“*Budget”*) disponível para cada jogador para o projecto.

As iterações, definidas no Product Backlog, têm uma duração equivalente a 4 rodadas, como tal se um projecto tiver a duração de 8 iterações, este será disputado em 32 rodadas.

Cada jogador recebe 2 cartas de programadores de modo a ter uma equipa já definida antes do início do jogo e dar a possibilidade de vitória a todos os jogadores. A soma de todos os programadores e cartas de conceito de um jogador não podem ultrapassar o valor do orçamento definido para o projecto.

Tal como no jogo “Problems and Programmers”, o cartão do programador determina as características do programador como o seu custo (*“Salary”*), a sua personalidade (“*Personality”*) e o seu nível de conhecimento (“*Skill*”). Este último atributo tem grande relevância pois define se um programador pode adquirir artefactos. E isto porque, como mencionado anteriormente, o custo de um artefacto é definido pela complexidade do projecto. Se tal como na Figura 17 os artefactos tiverem um valor de 4, o programador presente na Figura 20 não os poderá adquirir já que o seu nível de conhecimento é de 1.

De seguida, cada jogador monta seu tabuleiro e as cartas são separadas em quatro montes: um para programadores, outro para problemas e conceitos, um para artefactos azuis e outro para artefactos vermelhos.

Com o dado determina-se quem começa o jogo e o jogo deve prosseguir no sentido dos ponteiros do relógio.

A cada jogada, um jogador lança o dado e de acordo com o número tirado, retira cartas dos montes dependendo do valor obtido:

- Se o jogador tirar entre 1 e 3, ele poderá apenas tirar do monte de problemas e conceitos o número de cartas indicadas pelo dado, sem poder retirar nenhuma carta de programador.
- Se o jogador tirar entre 4 e 6 no dado, ele escolherá 3 cartas de problemas e conceitos e a diferença (valor tirado no dado – 3) no monte de programadores.

As cartas de problemas e conceitos são guardadas na mão do jogador até o momento que ele achar oportuno. As cartas de programadores também podem ser guardadas na mão ou podem ser colocadas imediatamente no tabuleiro, indicando, dessa forma, a contratação do programador.

O jogador pode ficar no máximo com seis cartas na mão, incluindo conceitos, problemas e programadores. Se o jogador tiver mais de 6 cartas, ele deverá descartar as cartas excessivas antes do final da sua jogada.

As cartas de artefactos são retiradas dos montes de acordo com os programadores presentes no tabuleiro do jogador. O jogador pode retirar tantos artefactos quanto lhe convier, desde que estes estejam dentro da produtividade (*"Skill"*) da sua equipa de programadores. Um artefacto corresponde a uma tarefa definida pelo Product Backlog e, quando adquiridos, devem ser colocados no tabuleiro abaixo do programador que os adquiriu. É necessário, também, separar os artefactos adquiridos na rodada em questão ou em rodadas anteriores e isso porque existem cartas de problemas e conceitos que diferenciam este aspecto. Para tal, os artefactos adquiridos na rodada em questão devem ser colocados na primeira linha do tabuleiro abaixo do programador que os adquiriu, enquanto os artefactos obtidos em rodadas anteriores devem ser posicionados na última linha.

Existem 2 tipos de artefactos no Play Scrum. Os artefactos bons, de cor azul e os artefactos maus de cor vermelha.

Os artefactos azuis são considerados bons pois, independentemente de poderem conter bugs, a probabilidade disto acontecer é de 1/5. Eles podem ser adquiridos pelo valor definido pelo atributo "complexidade" do Product Backlog.

Os vermelhos, por sua vez, são vistos como artefactos maus pois a probabilidade de conterem bugs é de 3/5. Como tal, estes podem ser adquiridos por metade do valor definido no atributo "complexidade" do Product Backlog.

Resumindo tudo o que foi transcrito acima, numa jogada o jogador começa por lançar o dado, mediante o resultado obtido com o lançamento do dado recolhe as cartas a que tem direito, contrata programadores se assim o entender e adquire os artefactos que puder mediante a sua equipa de programadores.

Para além disso, durante a jogada, cada programador pode exercer outras tarefas tais como:

- Inspeccionar um dos seus artefactos: o jogador vira um dos artefactos para verificar se contem um erro. Esta acção tem o custo de 1 ponto.
- Corrigir erros dos seus artefactos: após inspeccionar o artefacto e descobrir um erro, o programador pode corrigi-lo, substituindo o artefacto com erro por um novo artefacto do mesmo tipo (azul ou vermelho). Esta acção tem o custo de 1 ponto.

Os jogadores podem inspeccionar e corrigir os seus artefactos durante toda a iteração, independentemente de estes terem sido adquiridos na rodada disputada. Da mesma forma, os problemas e conceitos aplicam-se aos artefactos adquiridos durante toda a iteração. Isto só não se verifica se for a própria carta a invalidar este facto.

Por tudo o que já foi enunciado é fácil perceber-se a importância do nível de conhecimento de um programador. E isto porque se um programador tiver um nível de conhecimento muito elevado (o nível de conhecimento vai de 1 a 5), ele poderá numa mesma rodada adquirir artefactos, inspeccioná-los e ainda corrigi-los. Ao invés, um programador com um nível de conhecimento muito baixo não poderá agir.

O jogador pode igualmente despedir e contratar programadores durante o jogo. No entanto, e para demonstrar o tempo de aprendizagem que a contratação de novo um programador implica, este processo terá de seguir uma sequência cronológica rígida. O jogador só pode despedir e contratar programadores no final da sua jogada. Como tal, ele despede o programador numa ronda e só poderá contratar um novo na rodada seguinte. Visto a contratação ser feita no final da sua jogada, o novo programador só começará a produzir tarefas na rodada seguinte. Como tal, o novo programador ficará uma rodada sem poder produzir.

No final da sua jogada, o jogador está apto a receber as cartas de problemas de seus adversários. Ele pode receber cartas dos três jogadores que jogaram imediatamente antes dele. As cartas de problemas podem ter efeito na rodada que sucedeu ou na próxima e neste último caso, no início da jogada na rodada seguinte, se o jogador tiver 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.

O Product Backlog, como demonstrado acima, é dividido em iterações que têm a duração de 4 rodadas. Cada jogador vai tentar, durante essas 4 rodadas adquirir o máximo de artefactos sem erros dependendo dos seus programadores e das cartas de problemas que forem usadas contra ele pelos seus adversários.

No final das 4 rodadas, o moderador vai inspeccionar os artefactos de cada um dos jogadores e vai apontar quantos artefactos sem erros e quantos artefactos com erros (serve para diferenciar jogadores em caso de empates) possui cada jogador

Após essa verificação, o moderador descarta os artefactos de cada jogador e o processo repete-se pelas iterações seguintes. Os artefactos voltam a ser colocados no monte a que pertencem.

Se no final de uma iteração um jogador conseguir atingir o número de tarefas sem erros definidas pelo Product Backlog, ele é o grande vencedor. Em caso de empate verifica-se qual dos jogadores tem menos artefactos com erros. Se persistir mesmo assim o empate, vence o jogador que teve menos artefactos maus durante o projecto.

Se nenhum jogador conseguir atingir o número de tarefas sem erros imposto pelo Product Backlog até ao final da última iteração, ganha o jogador que mais se aproximou desse número. No caso de empate, ganha o jogador que mais tem artefactos sem erros e, mantendo-se o empate, que menos teve artefactos maus durante o jogo.

4.7 'Play Scrum' vs SCRUM

Nesta secção vai ser apresentado um quadro comparativo entre o método Scrum e o jogo Play Scrum. Esta tabela vai permitir evidenciar a preocupação em criar um jogo que transmitisse os princípios gerais do método Scrum, assim como expor os aspectos que não foram contemplados e que deverão ser melhorados numa próxima versão do jogo.

Tabela 1 – Tabela Comparativa método Scrum e Play Scrum

Requisitos Scrum	**Play Scrum**
Product Backlog	Sim, a existência do Product Backlog tem grande destaque já que é esta carta que apresenta todas as características do projecto no Play Scrum.
Sprint Backlog	Sim, o Sprint Backlog que não é mais do que a lista de tarefas a serem desenvolvidas numa iteração é contemplado pelo Play Scrum nas cartas de problemas.
Burndown Chart	Não, esta ferramenta que apresenta a quantidade de trabalho desenvolvida em qualquer altura de execução do projecto e que permite visualizar a correlação entre o

	trabalho realizado e a quantidade de trabalho que resta desenvolver pela equipa em qualquer altura do projecto não é contemplado pelo Play Scrum
Tarefas alocadas por grau de prioridade	Não, o jogo não evidencia este conceito. Existe um número de tarefas que devem ser realizadas para vencer o projecto, porém nada é dito quanto à sua prioridade.
Iterações	Sim, este é uma das grandes características do Scrum. As iterações têm todas a mesma duração. Isso é evidenciado pelo facto das iterações serem todas equivalentes a quatro rodadas.
Daily Meeting	Sim, o daily meeting é uma reunião diária em que o scrum master se reúne com a equipa de modo a conseguir perceber como está a correr o projecto. Este requisito aparece no Play Scrum e a sua importância são realçados nas cartas de problemas e conceitos.
Entrega das funcionalidades 100% desenvolvidas no fim da iteração	Sim, esta característica é evidenciada no Play Scrum, já que no final de cada iteração o moderador do jogo recolhe todos os artefactos desenvolvidos pelos programadores, verifica e aponta os que contêm erros e os que não. Para efeitos de vitória só interessam os que não contêm erros.
Pequenas Equipas	Sim, esse facto é demonstrado já que nunca um jogador

	poderá ter uma equipa de programadores muito grande já que o orçamento nunca o permitirá.
Flexibilidade e Adaptabilidade	Sim, estes critérios são evidenciados nas cartas de problemas e conceitos do Scrum Play

A tabela acima apresentada demonstra que o jogo Play Scrum contempla quase todos os requisitos do método Scrum.

Os pontos que não foram abordados referem-se à prioridade da distribuição das tarefas e ao uso da ferramenta burndown chart para medir a quantidade de trabalho já feita a qualquer momento do projecto. As reuniões de início e final de cada iteração que permitem entregar as funcionalidades desenvolvidas ao Cliente e definir tarefas para a próxima iteração também não foram contempladas.

No entanto, os pontos principais do método Scrum tais como o conceito de iterações, a existência do Product Backlog e do Sprint Backlog, o desenvolvimento do projecto por pequenas equipas assim como a entrega das funcionalidades no final de cada iteração completamente desenvolvidas foram contemplados no Play Scrum.

5. Validação do Play Scrum

De forma a proceder a uma avaliação inicial do jogo projectou-se uma simulação do Play Scrum em que treze alunos do primeiro ano do Mestrado de Informática da Universidade do Minho foram ensinados a jogar o jogo. Os alunos frequentam a disciplina de Análise e Concepção de Software e, desta forma, satisfazem o perfil dos utilizadores para os quais se destina o jogo. Após terem jogado o jogo, foi-lhes distribuído um questionário em que cada um teve de responder de forma livre a um conjunto de questões.

Embora esta seja uma forma mais subjectiva de avaliação quando comparada a outros métodos, neste caso, esta é a forma mais adequada de obter as respostas necessárias para uma correcta avaliação do jogo.

No futuro, poderão ser utilizadas abordagens mais formais tais como a realização de estudos comparativos, nas disciplinas em que se lecciona o método Scrum da engenharia de software, entre as aptidões dos alunos que jogaram o jogo e os que não o fizeram.

Numa fase inicial os alunos reuniram-se em grupos e o jogo foi disputado por 8 jogadores. No entanto, este número de participantes revelou-se demasiado grande já que a primeira iteração da partida demorou cerca de uma hora. Como tal, no final da primeira iteração foram eliminados os 4 jogadores com pior pontuação. A partir desse momento, o jogo ganhou outro dinamismo e processou-se de forma bem mais rápida. Logo de seguida, foi-lhes entregue o questionário para avaliar a sua opinião geral do Play Scrum, a sua opinião sobre a eficácia pedagógica do jogo no ensino do método Scrum e no ensino da própria gestão de projectos na engenharia de software.

As perguntas do questionário obrigavam a uma resposta numérica numa escala de 1 a 5 e pediam para completar esse resultado com uma resposta textual de modo a ser mais fácil entender a pontuação atribuída e serem tiradas as reais conclusões da avaliação do jogo.

5.1 Resultados da Avaliação do Play Scrum

De um modo geral, a opinião dos treze alunos quanto ao Play Scrum foi bastante favorável como se pode verificar pelos resultados do questionário apresentados na Tabela 2.

A primeira pergunta "*Considera que o Play Scrum é agradável de se jogar? Avalie a sua resposta numa escala de 1 a 5 em que 1 é o resultado menor e 5, o melhor resultado. Pode me indicar o que mais gostou e o que menos gostou ao jogar o jogo.*" teve uma pontuação de 4.4 em 5. Este resultado demonstra com toda a clareza que os 13 participantes gostaram da sua experiência com o Play Scrum. Para reforçar essa ideia alguns alunos escreveram:

- *"O jogo é bastante interactivo e permite que cada jogador pense nas próximas jogadas antecipadamente";*
- *"O jogo é interessante e bastante agradável de jogar. É uma maneira diferente de ver os problemas existentes nestas áreas";*
- *"Gostei do jogo pois ensina alguns dos conceitos usados pelo Scrum e problemas que podem aparecer em projectos reais";*

Foi também realçado um ponto negativo que levou a uma mudança no número de jogadores máximos para jogar o Play Scrum:

- *"O ponto mais negativo que encontrei é o tempo do jogo que pode ser bastante longo";*

Inicialmente, o número máximo de jogadores era 8. No entanto após esta validação este número foi reduzido para 5.

A segunda questão *"Qual o nível de dificuldade do Jogo? Avalie a sua resposta numa escala de 1 a 5 em que 1 é o nível de dificuldade menor e 5, o nível de dificuldade máximo. Pode me indicar qual a parte que considera mais difícil no jogo?"* teve uma pontuação de 2.5 em 5. Este resultado demonstra que o nível de dificuldade do jogo é o adequado.

No que diz respeito à parte mais complexa do jogo, certos alunos ainda responderam:

- *"A gestão do orçamento e quais as tarefas escolher";*
- *"O jogo é de uma dificuldade razoável, sendo a parte mais difícil 'defender-se' dos ataques, e trabalhar com um orçamento limitado";*

- *"Por vezes temos que considerar se a nossa jogada valerá a pena, porque apesar de ser boa no imediato podemos receber um problema que prejudique a equipa. Por exemplo um programador com baixa personalidade pode ser a causa para um bom programador ser despedido";*

Tabela 2 – Resultados do Questionário de validação do jogo Play Scrum

Perguntas	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***Média***
Pergunta 1	4	4	4	-	4	5	4	5	5	4	5	5	4	4.4
Pergunta 2	3	1	1	4	2	3	3	3	3	2	3	2	3	2.5
Pergunta 3	3	4	4	4	3	4	3	4	4	5	5	3	3	3.8
Pergunta 4	3	3	4	-	3	5	2	3	3	1	4	4	4	3.3
Pergunta 5	3	3	-	4	4	5	3	3	3	4	4	-	-	3.6
Pergunta 6	5	3	-	-	3	5	4	3	4	5	5	-	5	3.8
Pergunta 7	4	5	-	2	3	5	4	4	3	2	5	-	4	3.7
Pergunta 8	3	2	-	4	1	4	3	2	3	5	5	-	2	3.1

A terceira pergunta "*Concorda com o facto do jogo Play Scrum ser um bom complementa à matéria de Engenharia de Software dada nas aulas? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa discordo completamente e 5 concordo plenamente. Indique por favor quais as vantagens que vê em usar este jogo como complemento às aulas teóricas"* teve um resultado de 3.8 em 5 que permite perceber que os alunos realmente entendem que o Play Scrum é um bom complemento às aulas teóricas.

Certos alunos ainda responderam:

- *"Permite-nos entender facilmente o conceito dos métodos ágeis";*

- *"Podemos perceber facilmente os problemas encontrados no desenvolvimento de software";*
- *"É uma forma didáctica muito agradável para melhor se compreender o método Scrum";*
- *"O Jogo pode ser um bom complemento às aulas porque é uma forma divertida de encarar a gestão de projectos";*

A pergunta 4 *"O Play Scrum permite obter novos conhecimentos sobre a Engenharia de Software dada nas aulas? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa discordo completamente e 5 concordo plenamente"* obteve uma pontuação de 3.3 em 5. Esta avaliação demonstra que os alunos encontram no Play Scrum uma nova forma de aprendizagem da Engenharia de Software mas que este deve ser visto como complemento das aulas teóricas.

A quinta pergunta *"Este jogo ensina, em geral, o processo de engenharia de software? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa discordo completamente e 5 concordo plenamente. Porquê? Em que medidas?"* obteve uma pontuação de 3.6 em 5. Analisando este resultado consegue-se perceber que os alunos conseguiram entender melhor o processo de engenharia de software através do uso do Play Scrum.

Para reforçar esta ideia certos alunos ainda escreveram:

- *"Ajuda a perceber o conceito de iterações e desenvolvimento incremental no processo de engenharia de software ";*
- *"Mostra algumas estratégias que podem ser aplicadas no dia-a-dia";*
- *"Permite ter noção das dificuldades existentes no desenvolvimento de software";*
- *"Indica as várias fases de um projecto de software e os seus problemas. Pois a meio de um projecto acontecem sempre coisas que na análise de requisitos não são detectadas";*
- *"Ajuda a perceber a dificuldade de se produzir software de forma correcta, rápida e dentro do orçamento";*

A sexta questão *"Incluiria o jogo Play Scrum nas aulas de engenharia de software? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa não, nem pensar e 5 sim, totalmente"* teve um resultado de 3.8 em 5. Avaliando este resultado, é possível dizer-se que sim os alunos entendem que o Play Scrum deveria ser parte integrante do programa da disciplina de Engenharia de Software.

Apesar de esta pergunta não requerer uma resposta textual alguns dos alunos optaram por explicar a pontuação atribuída.

- *"Sim, porque acabou por ser como uma aplicação prática daquilo que demos nas aulas";*

As duas perguntas que se seguem vêm de encontro à pergunta anterior. E isto porque elas permitem identificar se os alunos vêem o Play Scrum como uma vertente opcional do programa da disciplina de Engenharia de Software.

A pergunta 7 *"Incluiria o jogo Play Scrum como parte opcional do programa? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa não, nem pensar e 5 sim, totalmente"* obteve uma pontuação de 3.7 enquanto a oitava pergunta *"Incluiria o jogo Play Scrum como parte obrigatória do programa? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa não, nem pensar e 5 sim, totalmente"* teve um resultado de 3.1 em 5. Apesar destes dois resultados serem positivos, consegue-se perceber que os alunos assumem que o jogo deve ser parte opcional do programa.

Finalmente, a nona questão *"Diga-me por favor qual a sua visão global do jogos Quais os pontos que devem melhorados? Quais os pontos fortes do jogo? O que mais gostou? O que menos gostou?"* tem como finalidade conseguir entender de uma forma geral o que deve ser alterado no Play Scrum. Esta questão permite igualmente definir várias direcções possíveis para o trabalho futuro sobre o jogo.

Os alunos foram bastante cooperativos nas suas respostas e deram certas sugestões:

- *"O jogo está muito bem concebido apesar de não cobrir todas as variáveis da vida real (o que até nem é possível). Podíamos ter a hipótese de despedir programadores, aumentar o orçamento tendo desvantagens noutros pontos...";*
- *"De um modo geral está bom. Melhoraria as regras, pois há regras que não são totalmente esclarecedoras. Revia também os problemas e os conceitos já que alguns estão confusos";*
- *"Os pontos fortes do jogo são o trabalho com o orçamento, programadores, tarefas e ataques. E o ponto fraco a parte de poder 'virar' tarefas e trocá-las quase nunca ser usado";*
- *"Relacionar o número de tarefas e sprints com o número de jogadores para evitar que o jogo se torne muito moroso ou até muito rápido";*

O despedimento de programadores também foi incluído nas regras do jogo após esta validação pois realmente faz todo o sentido se poder dispensar um elemento que não cumpre os objectivos esperados.

Durante a validação do jogo consegue-se igualmente perceber que os alunos também apreciaram o facto de o jogo lhes permitir entender de uma forma muito clara o papel de um Scrum Master e quais os pontos que se devem ter em conta quando se gere um projecto. Questões como "*Mais vale adquirir um bom programador ou dois médios?*" foram discutidas pelos alunos no final do jogo. O orçamento dos projectos obriga os alunos a entenderem as decisões que devem ser tomadas e pensadas na hora de desenvolver um projecto.

O jogo demonstra, igualmente, a importância em desenvolver software de boa qualidade. No início do jogo, os alunos arriscavam bastante adquirindo artefactos de má qualidade, pois desta forma podiam obter o dobro dos artefactos no mesmo número de rodadas. No entanto, após verem a quantidade de artefactos com erros que eles tinham, eles rapidamente optaram por escolher artefactos de boa qualidade, escolhendo os de má qualidade apenas quando não lhes era possível escolher os de boa qualidade. Este aspecto do jogo é muito importante pois desta forma os alunos entendem de uma forma clara que a qualidade de uma funcionalidade é primordial e todo o produto entregue a um cliente não pode ter falhas. Como tal, é menos custoso produzir software de qualidade desde o início do que produzir software de má qualidade que deverá ser desenvolvido de novo e que atrasará a entrega do produto ao cliente.

Independentemente de grande parte das respostas terem sido bastante positivas, foi bem visível que os alunos que menos sabiam do método Scrum tiveram bem mais dificuldade a entender o jogo e a sua dinâmica. Este ponto demonstra com clareza que o jogo não deve ser utilizado de forma isolada. Ao invés, ele deve complementar os conceitos teóricos leccionados nas aulas para ter um total aproveitamento do jogo.

Em suma, a avaliação do jogo foi muito positiva e ficou demonstrado que o jogo consegue transmitir as práticas e procedimentos do método Scrum e da Engenharia de Software de um modo mais abrangente.

6. Conclusões e Trabalho Futuro

6.1 Trabalho Realizado

Este trabalho foi elaborado com a finalidade de criar o jogo Play Scrum. Este jogo de cartas deve permitir a aprendizagem do método ágil Scrum a estudantes universitários. Para atingir este resultado foi necessário passar por um conjunto de etapas.

No capítulo 2 apresenta-se o resultado do estudo ao método ágil Scrum. Aí apresentam-se as fases que compõem o método Scrum, as tarefas inerentes a cada uma destas fases. São igualmente descritas as formas de controlo que existem para controlar este método tão flexível e sensível às variáveis externas. O capítulo prossegue com uma apresentação das características dos projectos Scrum, dos seus intervenientes e dos documentos que são produzidos por quem segue este método. De seguida, é definido todo o desenrolar do processo Scrum e o capítulo termina com uma apresentação das vantagens do uso do método Scrum.

O capítulo 3 descreve o uso de jogos como método alternativo para a aprendizagem da engenharia de software. O capítulo divide-se em duas partes. A primeira parte apresenta 5 jogos que simulam o processo de engenharia de software. Os jogos são: o SIMSE, o SESAM, o Planager, o Scrumming e, finalmente, o jogo The Incredible Manager. Para cada um deles é feita uma breve apresentação das características e dinâmica de jogo de cada um.

A segunda parte do capítulo 3 apresenta o jogo Problems and Programmers. É dado especial destaque a este jogo porque foi neste que se baseou o desenvolvimento do Play Scrum. Todos os intervenientes, características e regras do jogo são descritos no capítulo 3.

No capítulo 4 é apresentado o jogo Play Scrum. O jogo concebido e grande propósito deste trabalho baseou-se no material obtido da pesquisa feita nos capítulos

anteriores. Este jogo é o resultado de toda a pesquisa feita nos capítulos anteriores e o grande propósito deste trabalho. O capítulo começa por apresentar todos os componentes do jogo entre os quais o product backlog que define as características do projecto, o tabuleiro do jogo distribuído a cada jogador e as cartas que compõem o jogo (cartas de problemas, cartas de conceitos, cartas de programadores e os artefactos). De seguida é feita uma descrição dos intervenientes do Play Scrum apresentando o papel de todos os que intervêm no jogo. Uma vez que todas as peças do jogo foram descritas, é possível apresentar toda a sua dinâmica explicando de que forma se desenrolam todas as rodadas do Play Scrum. Finalmente, o capítulo 4 apresenta uma tabela que apresenta as características principais do método Scrum e determina se o jogo Play Scrum as incorpora nas suas regras.

O capítulo 5 permite apresentar a validação ao Play Scrum. Esta validação foi feita através de um questionário dado a três alunos universitários que jogaram Play Scrum. O capítulo apresenta as perguntas e analisa as respostas dadas pelos três alunos permitindo perceber os pontos fortes do jogo e os pontos a melhorar no futuro.

6.2 Trabalho Futuro

Dada a inerente exiguidade temporal associada a uma dissertação deste género, o seu desenvolvimento não permite a exploração de várias vertentes que, apesar de complementares ao tema a tratar, tornariam o âmbito demasiadamente abrangente para possibilitar uma obtenção atempada de resultados. Assim a continuidade a dar a esta dissertação poderá trilhar diversos caminhos: por um lado melhorando o jogo de cartas Play Scrum. Como foi demonstrado ao longo desta dissertação, certas características do método Scrum não foram abrangidas pelo jogo. Em futuras melhorias do jogo, seria interessante incluir estas características no jogo.

As cartas do jogo poderiam ser aperfeiçoadas. As cartas de problemas e conceitos poderiam explicar de forma mais explícita ao que se referem de modo a transmitir de forma mais concreta ao aluno o seu propósito. Finalmente deveria ser feita uma validação mais aprofundada ao jogo de modo a avaliar com maior exactidão os pontos a serem melhorados.

Por outro lado, desenvolvendo uma versão electrónica do jogo Play Scrum. Essa versão poderia ser ainda mais agradável de jogar e poderia ter uma área que permitisse ao aluno aceder a todo o conteúdo teórico do método Scrum, complementando desta

forma o jogo. Esta vertente implicaria um grande investimento técnico mas usaria a investigação já feita para esta dissertação.

Finalmente, optando por desenvolver uma outra versão do jogo, mais genérica, que permitiria a aprendizagem dos vários métodos ágeis num só jogo. Este caminho implicaria uma investigação aos restantes métodos ágeis comparando-os de modo a conseguir definir as regras do jogo.

Bibliografia

[1] A. Baker, E. Oh Navarro, A. van der Hoek (2003), *An Experimental Card Game for Teaching Software Engineering*, Journal of Systems of Software 75(1-2):3-16, Fev/2005. DOI 10.1016/j.jss.2004.02.033.

[2] A. Drappa, L. Jochen (2000), *Simulation in software Engineering Training*, ICSE.

[3] A. Ribeiro Dantas, M. de Oliveira Barros, C.M. Lima Werner (2008), *Treinamento Experimental com Jogos de Simulação para Gerentes de Projeto de Software*, 18º Simpósio Brasileiro de Engenharia de Software, Brasilia

[4] A. Scotland (2005), *Scrum @ the BBC*, in JAOO, Aarhus, Denmark: EOS

[5] B. Shneiderman (2004), *Designing for Fun: Can we design user interfaces to be more fun?*, University of Maryland

[6] D. Carrington, A. Baker, A. van der Hoek (2004), *It's All in the Game: Teaching Software Engineering Concepts*, 34th ASEE/IEEE Frontiers in Education Conference.

[7] D. Thatcher, C. Donald (1990), *Promoting learning through games and simulations*, Simulation & Gaming. 21 (3), 262-273.

[8] E. Isotton (2008), *Scrumming – Ferramenta Educacional para Apoio ao Ensino de Práticas de SCRUM*, Monografia de Trabalho de Conclusão de Curso de Graduação, Sistemas de Informação, FACIN, PUCRS, Porto Alegre.

[9] E. Kieling, R. Rosa (2006), *Planager - Um Jogo para Apoio ao Ensino de Conceitos de Gerência de Projetos de Software*, Monografia Trabalho de Conclusão de Curso de Graduação, Ciência da Computação, FACIN, PUCRS, Porto Alegre,

[10] E. Navarro, A. Baker, A. van der Hoek (2004), *Teaching Software Engineering Using Simulation Games.*

[11] E. Oh Navarro, A. van der Hoek (2004), *Software Process Modeling for an Educational Software Engineering Simulation Game*, In Software Process Improvement and Practice: 10 (3), 311-325.

[12] G. Taran (2007), *Using Games in Software Engineering Education to Teach Risk Management*, Proceedings of the 20th Conference on Software Engineering Education & Training, Pages 211-220, ISBN ~ ISSN:1093-0175 , 0-7695-2893-7

[13] H. Takeuchi, I. Nonaka (1986), *The New New Product Development Game*. Harvard Business Review

[14] J.R. Anderson, et al. (1995), *Cognitive Tutors: Lessons Learned, The Journal of the Learning Sciences*, 4 (2), 167-207

[15] J. McKendree (1990), *Effective Feedback Content for Tutoring Complex Skills*, Human-Computer Interaction. 5, 381-413.

[16] J.M. Randel, et al. (1992), *The Effectiveness of Games for Educational Purposes: A Review of Recent Research*, Simulation and Gaming. 23 (3), 261-276.

[17] J. Sutherland (2004), *Agile Development: Lessons learned from the first Scrum*, jeffsutherland.com, Cutter Agile Project Management, Executive Update, Vol. 5, No. 20

[18] J. Sutherland, K. Schwaber (2007), *The Scrum Papers: Nuts, Bolts and Origins of an Agile Process*

[19] K. Schwaber (2004), *Agile Project Management with Scrum*, Microsoft Press, 163pp, ISBN 0-7356-1993-X

[20] K. Schwaber, M. Beedle (2002), *Agile Software Development with Scrum*, Prentice-Hall.

[21] K. Schwaber (1995), *Scrum Development Process*, OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag

[22][M.T.H. Chi, et al. (1994), *Eliciting Self-Explanations Improves Understanding*, Cognitive Science. 18, 439-477.

[23] M. Ferrari, R. Taylor and K. VanLehn (1999), *Adapting Work Simulations for Schools*, The Journal of Educational Computing Research. 21 (1), 25-53.

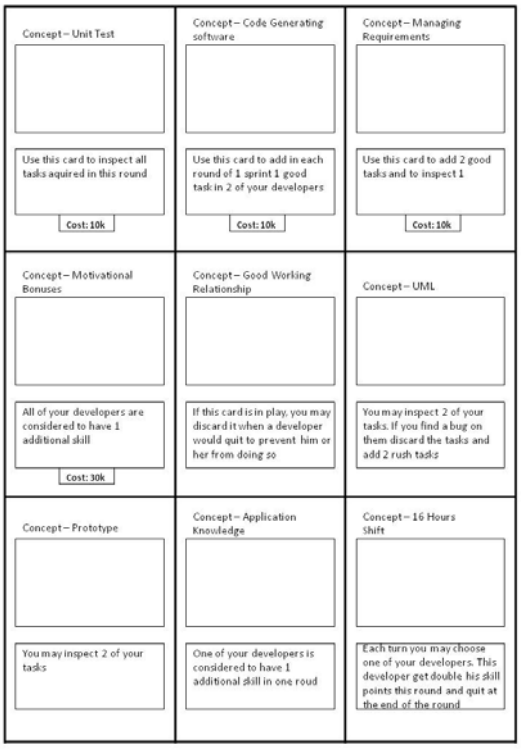
[24] M. Prensky (2001), *Why Games Engage Us" in Digital Games Based Learning*, Mc Graw Hill.

[25] P. Kruchten (2001), *Agility with the RUP*. Cutter IT Journal, Arlington, v.14, n.12, p. 27-33

[26] P. Mandell-Striegnitz (2001), *How to Successfully Use Software Project Simulation For Educating Software Project Managers*, 31st ASEE/IEEE Frontiers in Education Conference.

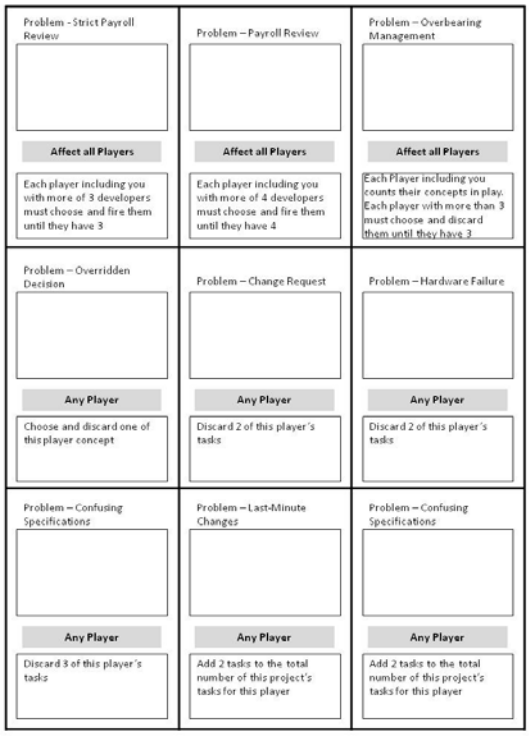
[27] R. S. Pressman (1997), *Software engineering: A practitioner's approach*, 4th. Ed. McGraw-Hill, 22–53

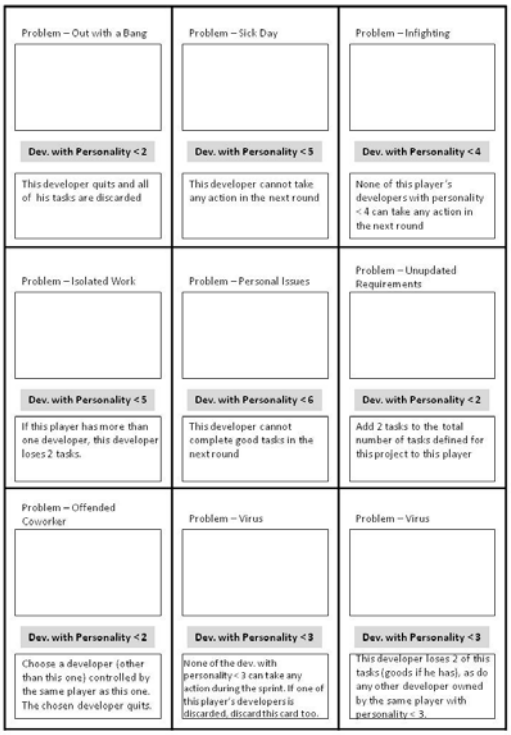
ANEXO 1 – Cartas do Jogo Play Scrum


Concept – Unit Test
Use this card to inspect all tasks aquired in this round
Cost: 10k
Concept – Code Generating software
Use this card to add in each round of 1 sprint 1 good task in 2 of your developers
Cost: 10k
Concept – Managing Requirements
Use this card to add 2 good tasks and to inspect 1
Cost: 10k
Concept – Motivational Bonuses
All of your developers are considered to have 1 additional skill
Cost: 30k
Concept – Good Working Relationship
If this card is in play, you may discard it when a developer would quit to prevent him or her from doing so
Concept – UML
You may inspect 2 of your tasks. If you find a bug on them discard the tasks and add 2 rush tasks
Concept – Prototype
You may inspect 2 of your tasks
Concept – Application Knowledge
One of your developers is considered to have 1 additional skill in one roud
Concept – 16 Hours Shift
Each turn you may choose one of your developers. This developer get double his skill points this round and quit at the end of the round


Concept – Meeting
You may discard this card to allow your developers to have +1 personality in this round
Concept – Contract
You may discard this card to inspect one of your tasks
Concept – Social Environment
All of your developers get +1 personality in this sprint
Concept – Executive Favor
If this card is in play, you may discard it when a developer would be fired to prevent him from happening
Concept – Display of Personnal Effort
You may discard this card to allow each of your developers to have 2 additional skill points to use in this round
Concept – Collaborative Software
If all of your developers have at least 3 personality you can inspect 1 task in each round of this sprint
Cost: 10k
Concept – CM Software
You may discard this card to inspect 1 of your tasks
Concept – Reusable Code
You may discard this card to add one good task for each of your developers
Concept – Literate Programming
Use this card to add 2 skill points for one developer to be used to inspect his tasks in one sprint


Concept – Standards
Use this card to add 1 point of skill of one developer in each roud of this sprint
Concept – University Consortium
You can take one developer card of the developer's deck
Concept –Master Degree
Use this card to add 1 skill point to one of your developers
Concept – Inspection
You may discard this card to inspect 3 of your tasks
Concept – Scrum Training
You may discard this card to exchange 2 of your rush tasks for 2 good tasks


Problem - Strict Payroll Review
Affect all Players
Each player including you with more of 3 developers must choose and fire them until they have 3
Problem – Payroll Review
Affect all Players
Each player including you with more of 4 developers must choose and fire them until they have 4
Problem – Overbearing Management
Affect all Players
Each Player including you counts their concepts in play. Each player with more than 3 must choose and discard them until they have 3
Problem – Overridden Decision
Any Player
Choose and discard one of this player concept
Problem – Change Request
Any Player
Discard 2 of this player's tasks
Problem – Hardware Failure
Any Player
Discard 2 of this player's tasks
Problem – Confusing Specifications
Any Player
Discard 3 of this player's tasks
Problem – Last-Minute Changes
Any Player
Add 2 tasks to the total number of this project's tasks for this player
Problem – Confusing Specifications
Any Player
Add 2 tasks to the total number of this project's tasks for this player

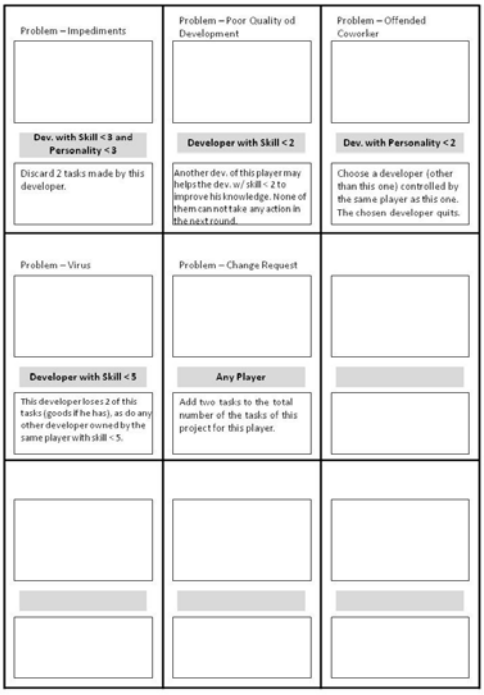


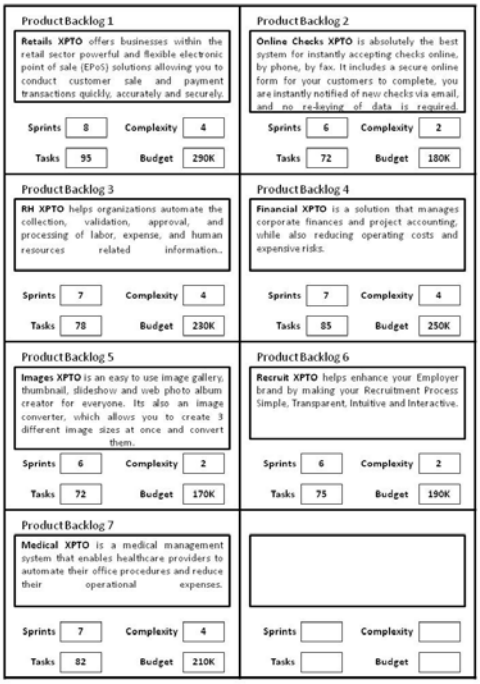
Problem – Out with a Bang	Problem – Sick Day	Problem – Infighting
Dev. with Personality < 2	**Dev. with Personality < 5**	**Dev. with Personality < 4**
This developer quits and all of his tasks are discarded	This developer cannot take any action in the next round	None of this player's developers with personality < 4 can take any action in the next round
Problem – Isolated Work	**Problem – Personal Issues**	**Problem – Unupdated Requirements**
Dev. with Personality < 5	**Dev. with Personality < 6**	**Dev. with Personality < 2**
If this player has more than one developer, this developer loses 2 tasks.	This developer cannot complete good tasks in the next round	Add 2 tasks to the total number of tasks defined for this project to this player
Problem – Offended Coworker	**Problem – Virus**	**Problem – Virus**
Dev. with Personality < 2	**Dev. with Personality < 3**	**Dev. with Personality < 3**
Choose a developer (other than this one) controlled by the same player as this one. The chosen developer quits.	None of the dev. with personality < 3 can take any action during the sprint. If one of this player's developers is discarded, discard this card too.	This developer loses 2 of this tasks (goods if he has), as do any other developer owned by the same player with personality < 3.

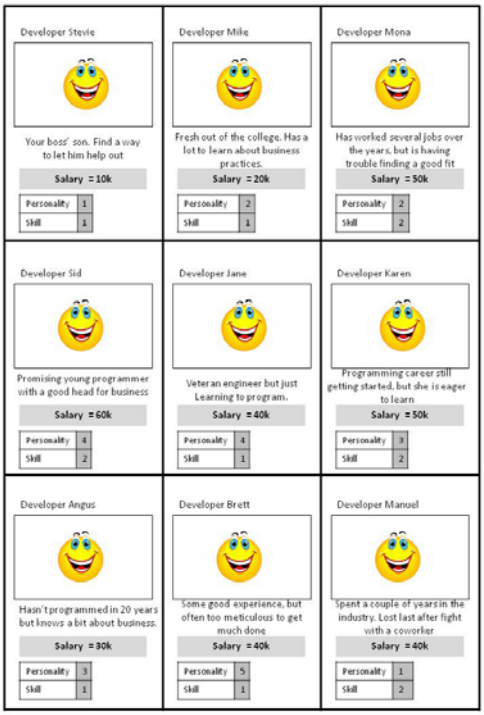
Problem – Better Job	Problem – Missed Requirements	Problem – Poor Version Control
Dev. with Personality < 4	**Dev. with Personality < 4**	**Dev. with Personality < 4 and Skill < 3**
This developer quits at the end of this sprint.	Discard 2 tasks of this developer	Discard two tasks of this player
Problem – Misleading Tests	**Problem – Bad Unit Tests**	**Problem – Non-Automated Tests**
Dev. with Personality < 5	**Dev. with Personality < 3**	**Dev. with Personality < 2 and Skill < 3**
Add 1 point to the cost of inspecting and fixing bugs for this developer during the game	This developer only can inspect tasks in the next round	This developer loses all tasks made in this sprint
Problem – Missing Tests	**Problem – Daily Meeting Unmarked**	**Problem – Inconcistency between Tasks**
Dev. with Personality < 4	**Dev. with Personality < 4 and Skill < 3**	**Developer with Skill < 3**
All developers with personality < 4 lose one task.	All developers with personality < 4 and skills < 3 lose all tasks made in this round	Put this card above the developer. Add one point for inspecting and fixing bugs

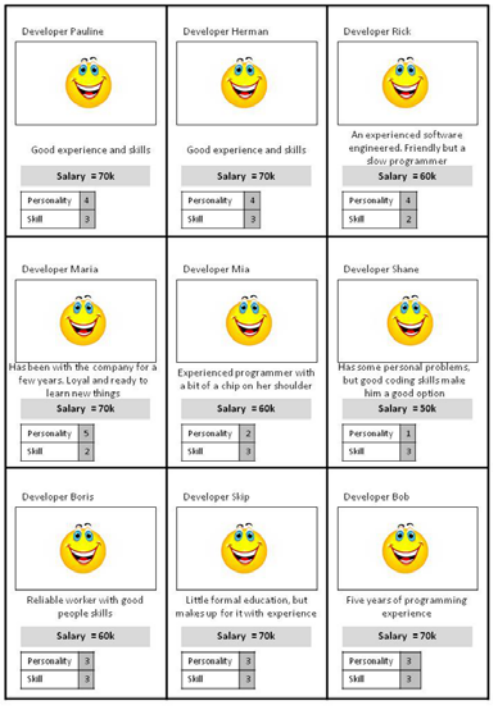
Problem – Bad Language Programming	Problem – Code Duplication	Problem – Code Redundancy
Developer with Skill < 2	**Developer with Skill < 4**	**Developer with Skill < 3**
Discard all tasks made by this developer on this sprint.	Discard 1 task made by this developer	Discard 1 task made by this developer
Problem – Missing Requirements	**Problem – Disease**	**Problem – Job Opportunity**
Developer with Skill < 4	**Dev. with Personality < 5**	**Dev. with Personality < 4**
Discard 2 tasks made by this developer	This developer cannot take any action in the next round	This developer get out at the end of the sprint
Problem – Deviation of the Scrum Practices	**Problem – Absents from Work**	**Problem – Poor Version Control**
Developer with Skill < 4	**Dev. with Personality < 3**	**Dev with Personality < 4 and Skill < 3**
Discard two tasks made by this developer	This developer cannot take any action in the next round	Discard one task of this developer

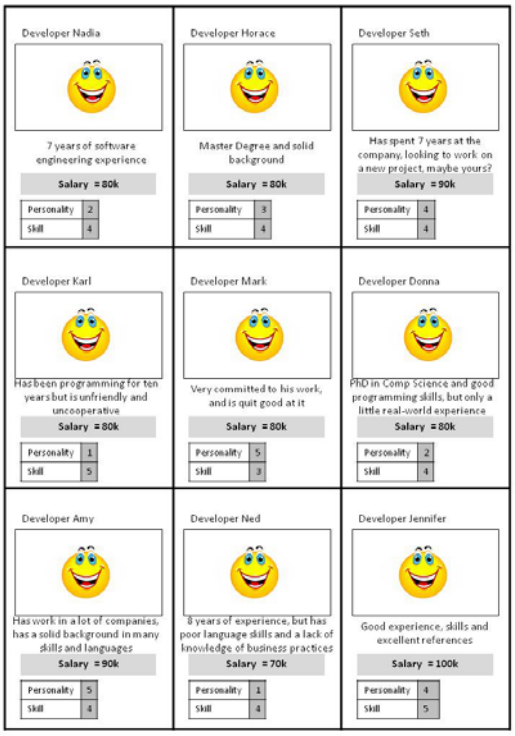
Problem – Bad Language Programming	Problem – Code Duplication	Problem – Persistent Bug
Developer with Skill < 2	**Developer with Skill < 4**	**Developer with Skill < 4**
Discard all tasks made by this developer on this sprint.	Discard 2 tasks made by this developer.	This developer can not take any action in the next round.
Problem – Employee Fired	**Problem – Change Request**	**Problem – Lack of Strategic Alignment**
Dev. with Personality < 3	**Any Player**	**Any Player**
This developer get out now and all tasks he made in this sprint must be discarded.	Add two tasks to the total number of the tasks of this project for this player.	Discard all tasks made by this player in this sprint.
Problem – Skip Test Phase	**Problem – Missing Daily Scrum**	**Problem – Bad Sprint Backlog**
Developer with Skill < 4	**Dev. with Personality < 4**	**Dev with Personality < 3 and Skill < 3**
In the next round this developer only can inspect and fix bugs.	In the next round this developer only can inspect and fix bugs.	Discard all tasks made in this sprint by all developers of this player with skill < 3 and personality < 3

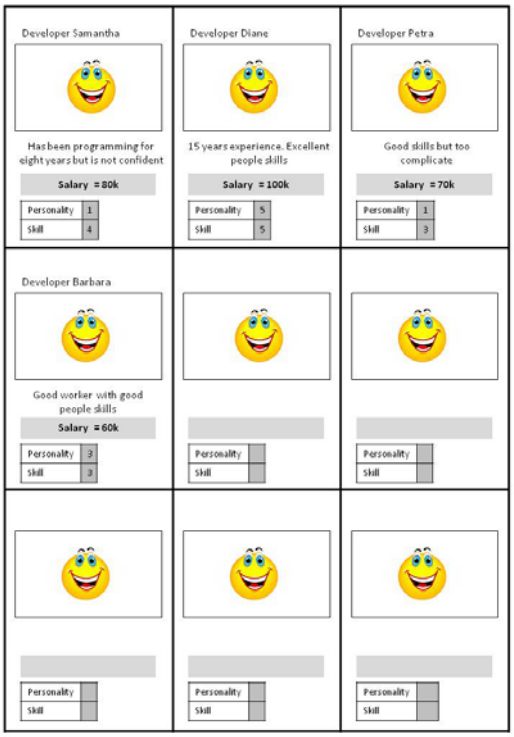

Problem – Impediments
Dev. with Skill < 3 and Personality < 3
Discard 2 tasks made by this developer.
Problem – Poor Quality od Development
Developer with Skill < 2
Another dev. of this player may helps the dev. w/ skill < 2 to improve his knowledge. None of them can not take any action in the next round.
Problem – Offended Coworker
Dev. with Personality < 2
Choose a developer (other than this one) controlled by the same player as this one. The chosen developer quits.
Problem – Virus
Developer with Skill < 5
This developer loses 2 of this tasks (goods if he has), as do any other developer owned by the same player with skill < 5.
Problem – Change Request
Any Player
Add two tasks to the total number of the tasks of this project for this player.

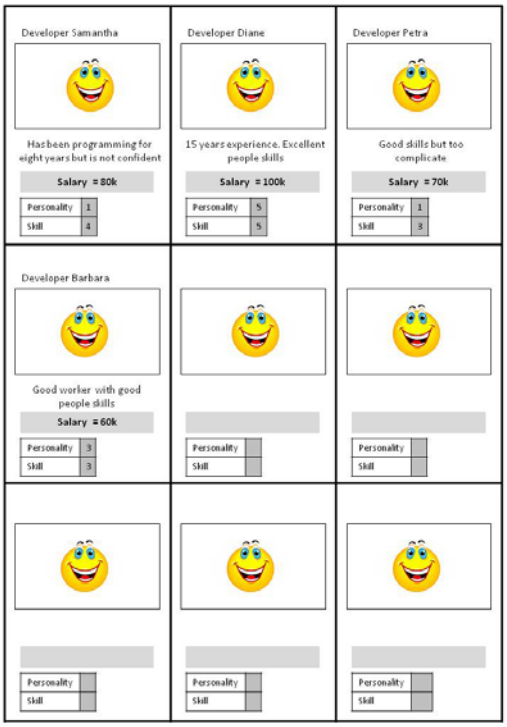

Product Backlog 1
Retails XPTO offers businesses within the retail sector powerful and flexible electronic point of sale (EPoS) solutions allowing you to conduct customer sale and payment transactions quickly, accurately and securely.
Sprints 8 Complexity 4
Tasks 95 Budget 290K
Product Backlog 2
Online Checks XPTO is absolutely the best system for instantly accepting checks online, by phone, by fax. It includes a secure online form for your customers to complete, you are instantly notified of new checks via email, and no re-keying of data is required.
Sprints 6 Complexity 2
Tasks 72 Budget 180K
Product Backlog 3
RH XPTO helps organizations automate the collection, validation, approval, and processing of labor, expense, and human resources related information..
Sprints 7 Complexity 4
Tasks 78 Budget 230K
Product Backlog 4
Financial XPTO is a solution that manages corporate finances and project accounting, while also reducing operating costs and expensive risks.
Sprints 7 Complexity 4
Tasks 85 Budget 250K
Product Backlog 5
Images XPTO is an easy to use image gallery, thumbnail, slideshow and web photo album creator for everyone. Its also an image converter, which allows you to create 3 different image sizes at once and convert them.
Sprints 6 Complexity 2
Tasks 72 Budget 170K
Product Backlog 6
Recruit XPTO helps enhance your Employer brand by making your Recruitment Process Simple, Transparent, Intuitive and Interactive.
Sprints 6 Complexity 2
Tasks 75 Budget 190K
Product Backlog 7
Medical XPTO is a medical management system that enables healthcare providers to automate their office procedures and reduce their operational expenses.
Sprints 7 Complexity 4
Tasks 82 Budget 210K
Sprints Complexity
Tasks Budget


Developer Stevie
Your boss' son. Find a way to let him help out
Salary = 10k
Personality 1
Skill 1
Developer Mike
Fresh out of the college. Has a lot to learn about business practices.
Salary = 20k
Personality 2
Skill 1
Developer Mona
Has worked several jobs over the years, but is having trouble finding a good fit
Salary = 50k
Personality 2
Skill 2
Developer Sid
Promising young programmer with a good head for business
Salary = 60k
Personality 4
Skill 2
Developer Jane
Veteran engineer but just Learning to program.
Salary = 40k
Personality 4
Skill 1
Developer Karen
Programming career still getting started, but she is eager to learn
Salary = 50k
Personality 3
Skill 2
Developer Angus
Hasn't programmed in 20 years but knows a bit about business.
Salary = 30k
Personality 3
Skill 1
Developer Brett
Some good experience, but often too meticulous to get much done
Salary = 40k
Personality 5
Skill 1
Developer Manuel
Spent a couple of years in the industry. Lost last after fight with a coworker
Salary = 40k
Personality 1
Skill 2

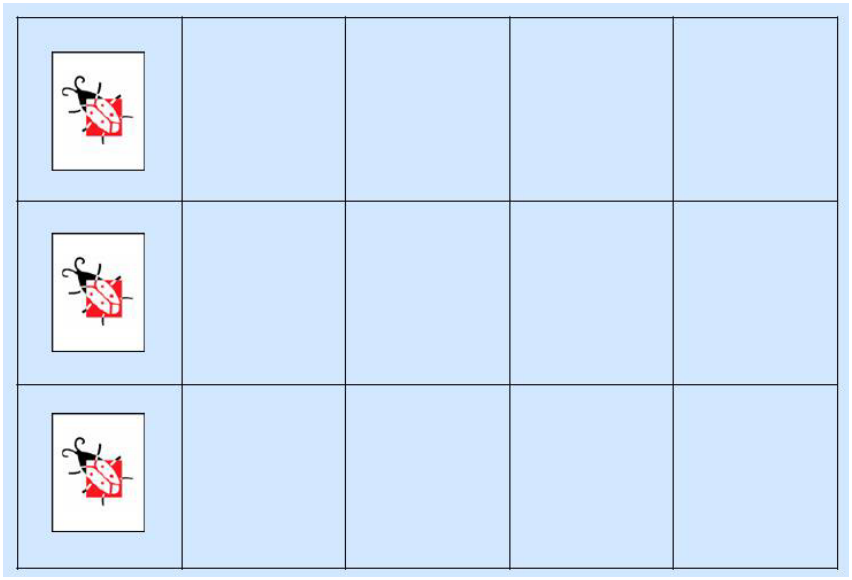

Developer Pauline
Good experience and skills
Salary = 70k
Personality 4
Skill 3
Developer Herman
Good experience and skills
Salary = 70k
Personality 4
Skill 3
Developer Rick
An experienced software engineered. Friendly but a slow programmer
Salary = 60k
Personality 4
Skill 2
Developer Maria
Has been with the company for a few years. Loyal and ready to learn new things
Salary = 70k
Personality 5
Skill 2
Developer Mia
Experienced programmer with a bit of a chip on her shoulder
Salary = 60k
Personality 2
Skill 3
Developer Shane
Has some personal problems, but good coding skills make him a good option
Salary = 50k
Personality 1
Skill 3
Developer Boris
Reliable worker with good people skills
Salary = 60k
Personality 3
Skill 3
Developer Skip
Little formal education, but makes up for it with experience
Salary = 70k
Personality 3
Skill 3
Developer Bob
Five years of programming experience
Salary = 70k
Personality 3
Skill 3


Developer Nadia
7 years of software engineering experience
Salary = 80k
Personality 2
Skill 4
Developer Horace
Master Degree and solid background
Salary = 80k
Personality 3
Skill 4
Developer Seth
Has spent 7 years at the company, looking to work on a new project, maybe yours?
Salary = 90k
Personality 4
Skill 4
Developer Karl
Has been programming for ten years but is unfriendly and uncooperative
Salary = 80k
Personality 1
Skill 5
Developer Mark
Very committed to his work, and is quit good at it
Salary = 80k
Personality 5
Skill 3
Developer Donna
PhD in Comp Science and good programming skills, but only a little real-world experience
Salary = 80k
Personality 2
Skill 4
Developer Amy
Has work in a lot of companies, has a solid background in many skills and languages
Salary = 90k
Personality 5
Skill 4
Developer Ned
8 years of experience, but has poor language skills and a lack of knowledge of business practices
Salary = 70k
Personality 1
Skill 4
Developer Jennifer
Good experience, skills and excellent references
Salary = 100k
Personality 4
Skill 5


Developer Samantha
Has been programming for eight years but is not confident
Salary = 80k
Personality 1
Skill 4
Developer Diane
15 years experience. Excellent people skills
Salary = 100k
Personality 5
Skill 5
Developer Petra
Good skills but too complicate
Salary = 70k
Personality 1
Skill 3
Developer Barbara
Good worker with good people skills
Salary = 60k
Personality 3
Skill 3
Personality
Skill


Developer Samantha
Has been programming for eight years but is not confident
Salary = 80k
Personality 1
Skill 4
Developer Diane
15 years experience. Excellent people skills
Salary = 100k
Personality 5
Skill 5
Developer Petra
Good skills but too complicate
Salary = 70k
Personality 1
Skill 3
Developer Barbara
Good worker with good people skills
Salary = 60k
Personality 3
Skill 3
Personality
Skill

	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5
Tasks produced in this round					
Tasks produced in previous rounds					



Anexo 2 – Questionário de Validação do Jogo Play Scrum

Play SCRUM

Por favor, responda as seguintes questões:

1º Considera que o Play Scrum é agradável de se jogar? Avalie a sua resposta numa escala de 1 a 5 em que 1 é o resultado menor e 5, o melhor resultado. Pode me indicar o que mais gostou e menos gostou ao jogar o jogo?

2º Qual o nível de dificuldade do Jogo? Avalie a sua resposta numa escala de 1 a 5 em que 1 é o nível de dificuldade menor e 5, o nível de dificuldade máximo. Pode me indicar qual a parte que considera mais difícil no jogo?

3º Concorda com o facto do Jogo Play Scrum ser um bom complemento à matéria de Engenharia de Software dada nas aulas? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa discordo completamente e 5 concordo plenamente. Indique por favor quais as vantagens que vê em usar este jogo como complemento às aulas teóricas.

4º O Play Scrum permite obter novos conhecimentos sobre a Engenharia de Software? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa discordo completamente e 5 concordo plenamente.

5º Este jogo ensina, em geral, o processo de engenharia de software? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa discordo completamente e 5 concordo plenamente. Porquê? Em que medidas?

6º Incluiria o jogo Play Scrum nas aulas de engenharia de software? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa não, nem pensar e 5 sim, totalmente.

7º Incluiria o jogo Play Scrum como parte opcional do programa? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa não, nem pensar e 5 sim, totalmente.

8º Incluiria o jogo Play Scrum como parte obrigatória do programa? Avalie a sua resposta numa escala de 1 a 5 em que 1 significa não, nem pensar e 5 sim, totalmente.

9º Diga-me, por favor, qual a sua visão global do jogo. Quais os pontos que devem ser melhorados? Quais os pontos fortes do jogo? O que mais gostou? O que menos gostou?

