UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ADRIAN MACIEL LAUBISCH MULLER

SCRUM 4 *GAMES* – UMA ADAPTAÇÃO DA METODOLOGIA *SCRUM* PARA O DESENVOLVIMENTO DE PROJETOS DE *GAMES*

NITERÓI

2009

_____	Laubisch, Adrian *Scrum4Games*: Uma adaptação da metodologia *Scrum* para o desenvolvimento de projeto de *Games* / Adrian Maciel Laubisch Müller – Niterói: [s.n.], 2009 59 fl: Il., 30 cm. Monografia (Bacharelado em Ciência da Computação) – Universidade Federal Fluminense, 2009. Orientador: Esteban Walter Gonzalez Clua 1. Engenharia de Software. 2. Games. 3. Scrum. 4. Ágil. I. Título

ADRIAN MACIEL LAUBISCH MULLER

SCRUM 4 *GAMES* – UMA ADAPTAÇÃO DA METODOLOGIA *SCRUM* PARA O DESENVOLVIMENTO DE PROJETOS DE *GAMES*

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel. Áreas de Concentração: Engenharia de *Software*, Desenvolvimento de *Games* / Jogos

Orientador: Prof. Dr. ESTEBAN WALTER GONZALEZ CLUA

NITERÓI
2009

ADRIAN MACIEL LAUBISCH MULLER

SCRUM 4 *GAMES* – UMA ADAPTAÇÃO DA METODOLOGIA *SCRUM* PARA O DESENVOLVIMENTO DE PROJETOS DE *GAMES*

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel. Áreas de Concentração: Engenharia de *Software*, Desenvolvimento de *Games* / Jogos.

Aprovada em dezembro de 2009,

BANCA EXAMINADORA

Prof. Dr. ESTEBAN WALTER GONZALEZ CLUA – Orientador
UFF

Prof ª. Dr ª. ROSANGELA GOMES LIMA
UFF

Prof. Msc. LUIZ WALTER BRAND GOMES
UFF

Niterói
2009

Aos meus pais Günther Sergio Müller e Maria Lúcia Maciel Müller, minha irmã Leila Muller, meu bisavô Karl Ersnt Laubisch, aos meus grandes amigos Carolina Caravana, Rafael Cordeiro, Sonia Trois e Willen Jorge e à minha querida empresa, Aiyra.

AGRADECIMENTOS

À Universidade Federal Fluminense e os Departamentos de Matemática Aplicada, Geometria, Análise, Estatística, Física e especialmente Ciência da Computação, bem como a todos os professores dos respectivos departamentos que estiveram verdadeiramente dedicados à arte de ensinar.

Ao meu orientador, Esteban Walter Gonzalez Clua, pela seriedade que ajudou e muito a este trabalho a chegar ao papel;

Ao professor e amigo, Leonardo Gresta Paulino Murta, que não foi menos que um co-orientador, criticando, sugerindo, acrescentando.

À Aiyra e Aiyra.Net, minhas empresas, e todos os seus profissionais que tanto ajudaram nesse trabalho, direta ou indiretamente.

Aos amigos, parte essencial da nossa vida, Rafael Cordeiro, Rafael Moulin, Carolina Caravana, Diego "Dukão" Guimarães, Renata Lopez, Willen Jorge, Fernanda Lopez, Sonia Trois, Pedro Ivo, Marcos Ramos, Márcio Velloso, Massao Iwanaga, Bruno Lorenzo, Pedro Nucci entre outros, por garantir essenciais momentos de alegria e companheirismo.

A todos aqueles que, de alguma forma, contribuíram para a conclusão do meu curso, que culminou neste trabalho. Agradeço inclusive àqueles que eventualmente tenham contribuíram negativamente, pois me fizeram crescer e ver que o mundo não é fácil.

À minha família (Günther, Lúcia, Leila, Christian, Klaus) que, afinal, são a base de tudo.

RESUMO

Laubisch, Adrian. *Scrum4Games*: Uma adaptação da metodologia *Scrum* para o desenvolvimento de projeto de *Games*. Niterói, 2009. 49 p. Trabalho de Conclusão de Curso – Instituto de Computação, Universidade Federal Fluminense.

A industria de *games* supera hoje em faturamento a de cinema, o que demonstra a grande explosão que esta mídia teve em pouco tempo. O chamado entretenimento digital evoluiu de meros bits a complexas realidades em pouco mais de duas décadas, expandindo seu mercado e público alvo. Tal crescimento foi acompanhado invariavelmente pela evolução tecnológica, muitas vezes sendo o causador da mesma. Com o aumento da complexidade dos recursos à disposição, cresceu também a complexidade da execução dos projetos de *games*. Neste aspecto, dois pontos devem ser considerados: Um *game* é um projeto multidisciplinar e multimídia, envolvendo diversos tipos de profissionais e métodos; Em adição, um game também é um *software*, e segue os preceitos desta indústria. Uma vez que o sucesso do desenvolvimento de *software* está intimamente ligado à metodologia utilizada no mesmo, com o passar dos anos esta industria teve de se adaptar às crescente demandas de alterações de requisitos durante um projeto. Em resposta a isso, surgiram propostas como as metodologias ágeis. Por perceber que a industria de *games* também precisa de uma reformulação em suas metodologias, este trabalho partiu de estudos de caso e bibliográfico para criar uma adaptação de uma famosa metodologia ágil de software, o *Scrum*, para que pudesse ser aplicada em projetos de *games*.

Palavras-chave

Games, Jogos, Game Design, Indústria de Games, Engenharia de Software, Metodologia, Processo, SCRUM, Scrum *Master*, Product Owner, Daily Meeting, Scrum Planning Meeting, Scrum Review Meeting, Product Backlog, Sprint, Sprint Backlog, Burndown Chart, Scrum4Games.

ABSTRACT

Laubisch, Adrian. *Scrum4Games*: An adaptation of the Scrum methodology for Game Development projects. Niterói, 2009. 49 p. Term Paper – Instituto de Computação, Universidade Federal Fluminense.

The Game's Industry currently surpasses the sales of the Movies, which shows the great growth that this media had in a short period of time. The so-called digital entertainment has evolved from mere bits to complex realities in a practically two decades, expanding its market and target audience. This growth was invariably followed by technological evolution, often being the very cause of it. With the advancing complexity of available resources, it has also increased the complexity of game projects implementation. In that matter, two topics should be considered: A game is a multimedia and multidisciplinary project involving several kinds of professionals and methods; In addition, a game is also software, and follows the precepts of its industry. Since the success of software development is closely linked to the methodology used in it, over the years this industry has had to adapt to increasing demands for requirements changes in projects. In response, proposals have emerged, such as the agile methodologies. Realizing that the game industry also needs a quality rupture in their methodologies, this work make use from case studies and literature research to create an adaptation of a well-known agile software development methodology, Scrum, so it could be applied in projects of games.

Keywords

Games, Game Industry, Game Design, Software Engineering, Methodology, Process, Scrum, Scrum *Master*, Product Owner, Daily Meeting, Scrum Planning Meeting, Scrum Review Meeting, Product Backlog, Sprint, Sprint Backlog, Burndown Chart.

SUMÁRIO

LISTA DE FIGURAS

1. Introdução

Atualmente, a indústria de *videogames* supera em faturamento, por muito, aquela que é considerada a sétima arte: o cinema. Como é possível para uma forma de entretenimento tão nova alcançar tamanha expressividade em tão pouco tempo? Como pode uma invenção iniciada em pesquisas militares ter alcançado tal status sócio-econômico a ponto de despertar o interesse de grandes empresas e seus investimentos? Em sua curta, porém diversificada história, a indústria do desenvolvimento de *Games* passou por inúmeras fases, evoluções, revoluções e crises até que alcançasse a forma que possui hoje. Foram justamente tais mudanças que a tornaram tão madura, convertendo-se em uma mídia altamente expressiva (artisticamente considerando), assim como um lucrativo negócio (do ponto de vista do mercado).

O processo de desenvolvimento de *Games* pode ser visto como variada atividade multimídia. Da produção de um título participam profissionais das mais diversas áreas, agregando valores e conhecimentos de forma a contribuir para um produto que possua qualidade e com isso possa atingir seu principal intuito: entreter seu público.

Boa parte da percepção comum dos jogadores a respeito dos *videogames* está relacionada com o impacto artístico que os mesmos causam. O público olha com considerável valor um título que possua uma linguagem visual e/ou sonora que o atraia, prenda, o faça sentir imerso no universo do mesmo. O avanço gráfico dos *Games* é um dos pontos que força a evolução da tecnologia de hardwares, precisando de cada vez mais e mais recursos para processar toda a informação necessária. O perfil dos profissionais que atuam nesta área está cada vez mais complexo, agregando Modeladores 3D, Artistas Conceituais, Desenhistas e até mesmo outros profissionais mais especializados como Arquitetos, Decoradores, Maquiadores, Escultores entre outros. Na área de composição musical e efeitos sonoros, o produto gerado evoluiu de meros ruídos produzidos pelo computador para verdadeiras obras musicais que requerem a participação de profissionais de qualidade, como Compositores, Cantores, Engenheiros de Efeitos Sonoros, etc. Com tal equipe de produção audiovisual, o desenvolvimento de um *Game* hoje em dia muito se assemelha, de fato, com a produção de um grande filme.

Participando do projeto, há também os *Game Design*ers. Assim são chamados os profissionais responsáveis pela criação do conceito e da mecânica do jogo em si. São esses os responsáveis diretos pela diversão que o jogo irá proporcionar, através do desenvolvimento de uma interação que cause satisfação no jogador. Seu trabalho está diretamente ligado à inovação do referido *Game* em relação aos outros que já existem, e é tão admirado quando o resultado dos artistas, caso obtenha um bom êxito.

Também, há outra categoria de profissionais que justamente torna possível o básico de um jogo, ou seja, a existência física do *Game* como material jogável e interativo. Estes profissionais são os programadores e outros partícipes do desenvolvimento de *software* (Engenheiros de *Software*, Analistas, etc.). Esses profissionais constroem o *software* que é a base de todo o processo de desenvolvimento. Muitos desafios são inerentes do processo de programação de um título. Algo bastante comum diz respeito à superação de restrições impostas pelos recursos de hardware: usualmente, a equipe artística requer mais do que é possível processar, e cabe aos programadores contornar esta situação criativamente para que o resultado seja o mesmo que imaginado, aliado a uma performance aceitável. Muitas vezes, utilizam ferramentas que aceleram o processo, para que não seja necessário reprogramar algo que já tenha sido desenvolvido, mas sempre estão prontos para desenvolver algo do zero caso seja necessário.

É justamente nesse ponto que percebemos que o desenvolvimento de um *Game* é mais complexo do que possa parecer em uma primeira análise: há não somente a necessidade de um gerenciamento cuidadoso das tarefas artísticas, como também do processo de desenvolvimento de *software*. E como todo processo deste gênero, é altamente recomendável que este siga uma metodologia de alta taxa de sucesso para o caso em questão. Dessa forma, não somente o processo fica mais definido, como também favorece-se um cenário de aprendizado e evolução durante a execução do projeto.

1.1 Motivações

O autor do trabalho é sócio de uma empresa de desenvolvimento de *Games*, que foi contemplada, no fim do ano de 2008, com um projeto aceito em um edital promovido pela Secretaria de Cultura do Estado do Rio de Janeiro. O objetivo do edital vislumbrava oferecer recursos financeiros para que empresas do estado do Rio de Janeiro pudessem desenvolver uma versão de demonstração (“demo”) de um *Game*. Esta empresa, chamada Aiyra, encontra-

se atualmente no fim de seu processo de Incubação, após de um período de dois anos acumulando experiências sobre o mercado e trabalhos em geral.

O envolvimento com diversas formas de projeto e também de métodos para gerenciá-los foi decisivo para que ao longo do tempo fosse buscado, constantemente, um aprimoramento nestas técnicas. Com o crescimento do cenário de desenvolvimento ágil de projetos, em especial para o desenvolvimento de *softwares*, foi natural um eventual interesse a respeito deste tipo de metodologia.

Uma vez que o projeto ganho com o edital possuía recursos finitos e tempo curto para ser executado, mostrou-se necessário uma forma de realizá-lo com o máximo de aproveitamento possível. Com isso, poder-se-ia não somente realizar o objetivo de apresentar o projeto para a Secretaria de Cultura, como também possuir um demo de qualidade para integrar seu portfólio. Aliando então conhecimento acadêmico à prática de mercado, iniciou-se uma pesquisa de metodologias para garantir este bom aproveitamento, chegando-se às metodologias ágeis e finalmente, ao *Scrum*, que serão estudadas em detalhes no capítulo do trabalho.

1.2 Objetivo

Através das pesquisas realizadas anteriormente citadas, um ponto comum defendido por todos os autores diz respeito ao fato que o uso da metodologia *Scrum* para um projeto de *Games* não pode ser feita sem adaptações. Ao mesmo tempo em que são *softwares* e possuem características marcantes dos mesmos, os *Games* são obras diferentes e que possuem nuances relativamente opostas, em alguns momentos. Tais nuances, quando confrontadas com a metodologia *Scrum* clássica, muitas vezes resultam em conflitos que prejudicam o bom andamento dos projetos.

É imperativo, portanto, uma adaptação da metodologia para projetos de *Game*. A grande dificuldade reside no ponto que podem existir títulos muito diferentes entre si, e essa diversidade impacta na forma correta de se gerenciar cada projeto. Por isso, existem relatos sobre experiências que foram bem-sucedidas em um tipo de equipe/projeto, enquanto para outro tipo desencadearam um grande caos no ambiente de desenvolvimento.

Este trabalho analisa todo o processo de aplicação da metodologia *Scrum* ao projeto do demo de *Game* desenvolvido pela empresa Aiyra, através do estudo de caso, avaliando prós, contras, acertos e erros. Pesquisas bibliográficas tradicionais, aliadas à analise de outros estudos de caso, completam o estudo. O objetivo final é contribuir para a comunidade, criando um arcabouço de projeto de *Games* utilizando *Scrum*, para que possa ser utilizado, reutilizado, criticado, adaptado e ao longo do tempo, refinado.

1.3 Estrutura do trabalho

A organização do conteúdo do trabalho foi proposta de modo a apresentar gradativamente ao leitor os conceitos necessários para a compreensão final, relativa ao arcabouço proposto.

Nesta "Introdução", como foi visto, apresentou-se o cenário da indústria, os profissionais envolvidos nesse tipo de projeto. Foi dado foco em demonstrar a versatilidade de profissionais que participam, perfil de público alvo, e foi também relatada a congruência não-completa dos projetos de *Games* com os projetos de *software*, assunto que será tratado nos capítulos posteriores.

O próximo capítulo é chamado "*Scrum* e Desenvolvimento Ágil". Nesse capítulo é apresentado de forma resumida todo o contexto do *Scrum*, mostrando os itens presentes, atividades realizadas, objetivos, metas e afins.

No capítulo 3, "Aplicando *Scrum* em Projetos de *Games*", é visto todo o resultado da pesquisa bibliográfica e do estudo de caso do projeto da empresa Aiyra. Foram selecionadas as peculiaridades mais relevantes, positivas ou negativas, destacando-se as formas que foram usadas para contornar os problemas gerados. Este capítulo apresenta um resumo relevante sobre os impactos da aplicação de *Scrum* a um projeto de *Game*.

O capítulo 4 é o ápice do estudo, apresentando o "*Scrum4Games*". Intitulado "Adaptando o *Scrum* para *Games* – *Scrum4Games*", este capitulo propõe alterações na metodologia padrão de modo a responder positivamente ao que foi exposto no capítulo 3. Como resultado, é gerado o arcabouço final.

O capítulo 5, “Considerações Finais”, apresenta as últimas acerca do trabalho, como um resumo do que foi tratado e também as dificuldades encontradas. Também são expostos os trabalhos futuros pretendidos e as limitações deste.

2. *SCRUM* e Desenvolvimento Ágil

A construção de *Games*, sobre vários aspectos, pode (e deve) ser considerada semelhante ao desenvolvimento de um *software* padrão. Portanto, da mesma forma que em aplicações não voltadas para o entretenimento, faz-se necessária a reavaliação dos métodos de projeto de desenvolvimento de *Games*, de forma a obter melhores resultados, em menor tempo, com menor custo.

Para a boa compreensão deste capítulo, é importante que um *Game* seja considerado como um *software*, pura e simplesmente. Dessa forma, poderá ser compreendido de maneira mais simples o conceito de desenvolvimento ágil e *Scrum*. No capítulo 3 será dada a atenção devida às diferenças entre projetos de *Games* e *softwares* comuns.

2.1 Cenário de criação do Desenvolvimento Ágil

Desenvolvimento de *software* se tornou, com o passar do tempo, uma tarefa bastante complexa. O avanço da tecnologia e das necessidades dos usuários tornou intrincado o ambiente onde esse desenvolvimento se dá, gerando muitas vezes uma quantidade de imprevistos que inviabiliza o bom andamento do projeto. Por isso é preciso uma análise profunda destes ambientes de forma a gerar métodos que possam acompanhá-lo de maneira adequada. SILVA & LEMOS (2008).

Há uma série de peculiaridades presentes, dentre as quais podemos destacar:

- Dificuldade de encontrar profissionais com formação adequada;
- Tecnologias confiáveis à disposição;
- Métodos efetivos e adaptáveis;
- Familiaridade da equipe com as regras de negócio;
- Estimativa real do risco;
- Prazo e Orçamento do projeto.

Todas essas variáveis, em conjunto, contribuem para a criação da complexidade citada.

Projetos de *software* desorganizados e sem controle são bastante comuns, nos quais não é dada a devida atenção aos requisitos ou processos de desenvolvimento, muitas vezes ignorando ate mesmo a existência de testes. Tal abordagem é extremamente arriscada, uma vez que determinadas informações possuem momentos corretos para serem obtidas, e caso esse momento seja perdido, dados valiosos podem se perder. Bons exemplos disso são as avaliações do ambiente ou controle de mudanças de requisitos ao longo do projeto.

Em resposta ao caos constantemente gerado, o processo é geralmente atacado com uma precisão e detalhamentos extremos, de forma a reduzir a desordem. Tais metodologias estão quase sempre destinadas ao fracasso, uma vez que o processo de desenvolvimento de *software* não é algo completamente definido, a prova de falhas.

Porém, algumas metodologias obtiveram sucesso no controle, mesmo que parcial, destes processos. O Modelo *Waterfall* (Cascata) foi a primeira tentativa de mapear cada etapa do desenvolvimento, identificando-as e elaborando boas práticas. Mas, por ser linear, este modelo não lidava bem com o surgimento de imprevistos. Dessa forma, outra metodologia se fez necessária.

A evolução natural deu-se com surgimento do Modelo Espiral. Este ataca diretamente a grande falha do Cascata, ou seja, sua linearidade. A proposta então foi o encadeamento de diversos projetos em Cascata sucessivos, ligados por etapas de prototipação. Desta forma, apos a avaliação de cada protótipo, poderia verificar-se o andamento do projeto e direcionar o desenvolvimento da próxima etapa. É importante perceber que o projeto é abordado em sua totalidade a cada etapa, e que internamente o seu fluxo permanece linear.

Embora tenha representado um importante avanço, o Modelo Espiral possui ainda falhas, e como uma evolução desse, foi desenvolvido o Modelo Iterativo. Esta metodologia prega a divisão do projeto em blocos funcionais, definidos e separados, que serão desenvolvidos independentemente, em porções de tempo definidas “iterações”. Cada iteração possui internamente todas as etapas do desenvolvimento, exatamente como presente no Cascata, e permite a avaliação de cada etapa como base para a estimativa de prioridades e padrões para a próxima. Uma grande vantagem é a possibilidade de alteração dos requisitos de cada módulo ao longo do projeto, mas somente quando ocorre o início da iteração relativa

ao mesmo. Porém, curiosamente, este modelo de processo continuou apresentando perfis lineares, não lidando bem com imprevistos, o que tem se demonstrado cada vez mais fatal.

Mesmo que tenham sido soluções relevantes por um bom tempo, os três modelos não são mais adequados para o mundo atual. A velocidade de evolução das necessidades dos clientes é imensa, e uma vez que essas necessidades traduzem-se em requisitos, qualquer método que não esteja preparado para imprevistos rápidos e constantes está quase certamente fadado ao fracasso. Dessa forma, surgiu a proposta do Desenvolvimento Ágil de *Software*, que ao contrário do que o nome remete, não esta ligado à sua velocidade, e sim a sua adaptabilidade a mudanças. Existe uma série de metodologias deste gênero, porém neste trabalho trataremos apenas do *Scrum*.

2.2 O *Scrum*

Scrum é uma metodologia ágil, ou seja, voltada para a rápida adaptação às mudanças do dia-a-dia em um projeto de *software*. O primeiro passo da metodologia é encapsular os processos conhecidos na forma de “caixas pretas", uma vez que são imprevisíveis. Ao invés de voltar-se para o detalhamento em busca da ordem, *SCRUM* aceita que não há como controlar o caos do ambiente e foca em usá-lo ao seu favor. Por isso, dá prioridade máxima para a conceito de comunicação, seja ela entre a equipe ou então entre os desenvolvedores e o cliente/*stakeholder* (detentor do conhecimento do negócio em questão) . É uma metodologia ágil, desenvolvida a partir da observação de grupos pequenos, heterogêneos e bastante dinâmicos, como times de Rúgbi (esporte o qual emprestou o nome de uma de suas jogadas, o *Scrum*, para o processo) .

O cliente participa ativamente de um projeto *Scrum*. Juntamente à ele são combinados períodos de tempo para a entrega de versões do *software*. Cada versão deve possuir no mínimo uma nova funcionalidade complexa, de modo que o cliente possa testá-la e passar o *feedback* para o time de desenvolvimento, priorizando novas funcionalidades para as iterações seguintes. Sendo o empirismo do processo de *software* algo de grande relevância para o *SCRUM*, é necessário atenção aos três conceitos base: Visibilidade (de todos aspectos relevantes ao projeto para todos os envolvidos com o mesmo); Inspeção (O andamento deve ser constante e arduamente avaliado); Adaptação (Capacidade de ajustar o projeto rapidamente à mudanças do ambiente e resultados).

É importante notar que o *Scrum* nunca deve ser visto como um controle do que é criado pela equipe de desenvolvimento, mas sim na forma como o projeto é guiado, sempre voltado para gerar o máximo de valor para o cliente.

Pontos importantes no *Scrum* que o diferenciam das outras metodologias estão geralmente ligados à não-linearidade dos projetos. A metodologia divide o projeto em iterações, chamados *Sprints*, que incrementalmente entregará versões mais completas para o cliente. Porém, só o inicio (planejamento) e fim (revisão) de cada iteração são considerados como bem definidos: a forma de realização de cada incremento é completamente empírica.

Durante a execução de um *Sprint*, é equipe que decide como o projeto deverá ser desenvolvido, e não um gerente. Caberá a elas todas as decisões sobre tecnologias a se utilizar, prazos possíveis para realização de tarefas, formas de contornar problemas, quantidade de itens a se executar em um determinado espaço de tempo. Tudo sempre da melhor forma possível.

Uma vez que o projeto é aberto e mutável até a sua fase final, mantendo disciplina nos processos de inspeção e adaptação a cada iteração, um projeto *Scrum* pode suportar grandes impactos em seu escopo, como alterações de tecnologia, equipe, prazo ou mesmo recursos financeiros. Pode-se dizer que o *Scrum*, além de possuir algumas regras próprias, seja uma combinação das metodologias incrementais e ágeis de desenvolvimento de *software* com as doutrinas *The Art of Possible*[1] e *Less Planning*[2].

O sucesso de um projeto *Scrum* não está vinculado a uma eventual riqueza de metodologia, e sim na fidelidade de seus participantes aos seus princípios, práticas e regras, que, por sua vez, são muito simples, e se traduzem no *Scrum Framework*: 3 PAPÉIS, 3 CERIMÔNIAS e 3 ARTEFATOS.

2.3 Papéis no *Scrum*

É importante perceber a diferença entre os papéis de um projeto *Scrum* e cargos em projetos comuns. A divisão no *Scrum* não ocorre de maneira hierárquica. Não existem papéis superiores ou pessoas que tenham autoridade incondicional. Cada papel possui uma

[1] Mantra informal do gerenciamento de projetos, onde o executor deve focar-se apenas executar o que é possível.
[2] Doutrina informal do gerenciamento de projetos, principalmente em Engenharia Civil, onde se planeja com detalhes apenas o que será feito nas próximas duas semanas.

responsabilidade específica para a execução com sucesso do projeto, e deve respeitar o posicionamento dos outros papeis quando se trata da área de cada um. A seguir serão detalhados os principais papéis do método.

2.3.1 *Product Owner*

Um ponto muito importante do desenvolvimento ágil é a presença constante do cliente no ambiente de desenvolvimento. No *Scrum*, o cliente presente (ou um representante seu altamente responsável) é o principal *stakeholder*[3], e é chamado de *Product Owner* (PO). O envolvimento deste com o projeto deve ser visível – todos devem perceber a presença positiva do PO.

Ele é aquele que alinha o desenvolvimento do projeto, através das suas prioridades. É também o grande responsável pelo valor do negócio, ou seja, o retorno do investimento no produto para a empresa que contratou a equipe de desenvolvimento (ROI, do inglês *Return of Investment*).

A pessoa com o papel de *Product Owner* deve ser bastante pró-ativa e comunicativa. Deve respeitar o espaço da equipe de desenvolvimento, suas decisões para com a execução, seus limites. Não deve de forma alguma se impor como o estereótipo clássico do "gerentão", aquele que segue a máxima "Faça o que digo porque estou mandando".

É o *Product Owner* que, a cada novo *Sprint*, testa e aprova o que foi feito, dando *feedback* para que a equipe possa guiar-se na próxima iteração. Dessa forma, ele é o grande responsável pelos quesitos de inspeção (ao avaliar) e adaptação (ao modificar as prioridades de acordo com o que ocorra a cada *Sprint*). Deve sempre dar suporte ao time de desenvolvedores, reconhecendo o trabalho dos mesmos através de palavras e ações.

Este profissional deve estar altamente atento ao risco das atitudes que toma, bem como das prioridades que estipula, pois ela tem impacto direto no desenvolvimento do projeto. É perfeitamente possível que realize mudanças no escopo ao longo do projeto, mas deve fazer isso da maneira mais justificada possível, repriorizando imediatamente os itens do projeto de acordo com qualquer mudança que ocorra.

[3] Pessoa, geralmente representada pelo cliente, que é a detentora de informações cruciais sobre o projeto.

2.3.2 *Scrum Master*

O *Scrum Master* é o responsável pela aplicação e manutenção das práticas do *Scrum* na equipe de desenvolvimento.

SCHWABER (2004) o define como um "Cão de Guarda". A bem da verdade, o *Scrum Master* deve proteger, garantir o bem estar de toda a equipe. Deve estar a tal ponto comprometido com o projeto que sua equipe veja nele um porto seguro. Como dito por SILVA & LEMOS (2008):

> "O papel do *Scrum Master* pode ser bem exemplificado como um maestro de orquestra. Ambos devem prover orientação e liderança constante a um time de talentosos profissionais que trabalham juntos para criar algo que nenhum deles pode fazer sozinho".

Não deve permitir que um impedimento atrapalhe a equipe: tão logo seja informado de algum problema, deve imediatamente partir para a solução do mesmo. A organização e a disciplina para com os métodos do *Scrum* é essencial para que o *Scrum Master* possa executar sua tarefa de facilitador. Outra atribuição importante do *Scrum Master* é que o mesmo promova uma ótima interação entre os membros da equipe e o *Product Owner*, evitando conflitos e embates.

Para COHN (2007) há uma lista de atributos desejáveis para um *Scrum Master*: Responsabilidade, Humildade, Colaboração, Comprometimento, Influência e Entendimento do assunto.

Este profissional pode ou não participar da execução do projeto propriamente dita. Neste ponto, cada projeto se comporta de uma forma, e autores possuem opiniões diferenciadas: BAKER (2005) discorda, pois acha que o *Scrum Master* não deve dividir sua atenção. Já CHAPIEWSKY (2008) concorda, pois acredita que o envolvimento auxilie sua tarefa de resolver problemas e dar segurança ao time. Ao final de cada *Sprint*, é importante nesse caso pesar os prós e os contras do auxílio do *Scrum Master* na execução, para decidirem juntos se manterão esta atividade na iteração seguinte.

2.3.3 Time

O Time (vindo da expressão em inglês *Team*) é o nome dado à equipe de desenvolvedores comprometidos com o projeto. São a grande estrela do *Scrum*, uma vez que

são eles que, efetivamente, vão executar o projeto e portanto, merecem todo o auxílio para tal.

Internamente, não possui definição de funções especificas como em outras equipes (testador, codificador, refatorador, etc.). É importante que, através de um forte sentimento de união e equipe, todos façam o que for preciso, mirando no objetivo de cada momento.

Esta equipe não deve ser muito grande, de forma a não dificultar a comunicação. O conhecido numero mágico da comunidade de desenvolvimento, 7 +/- 2 é o ideal: Os times devem possuir entre 5 e 9 membros. Caso haja necessidade de times maiores, MOUNTAIN GOAT (2008) sugere o chamado "*Scrum* de *Scrums*". Nesta prática, divide-se os participantes em equipes de 7 +/- 2 participantes, e cada equipe elege um representante. Esses representantes formam uma nova camada de abstração *Scrum*, compartilhando seus objetivos e tomando as decisões entre si seguindo as mesmas regras de um projeto *Scrum*.

É muito importante que não haja interrupções na atenção dos membros do Time. Para isso, todos devem fazer jornada integral, com horários coincidentes, evitando ao máximo faltas. O Time deve sempre buscar as soluções em conjunto, por isso, as pessoas precisam estar presentes. Caso seja altamente necessário, poderá haver um especialista que participe ocasionalmente do projeto, mas apenas se for aprovado pelo *Scrum Master* de forma a não comprometer o bom andamento.

Em um projeto *Scrum*, os membros do Time devem priorizar o todo, nunca devem agir por orgulho ou ego. O Time deve se auto-organizar, e os participantes não devem possuir títulos internos. Ao ser dado poder de decisão ao time, fortalece-se nele a noção da responsabilidade para com o projeto, melhorando a qualidade das tarefas executadas. SILVA & LEMOS (2008) citam as características desejáveis aos membros do Time: **Credibilidade; Estar em sintonia; Empatia; Interesse; Comunicação; Extroversão.** Outro ponto extremamente importante ao Time é conhecer sua velocidade, ou seja, saber quanto tempo demora para executar uma quantidade determinada de tarefas com dificuldade conhecida (como será dito a seguir).

2.4 Cerimônias

Existem 3 reuniões nominais previstas durante o processo do *Scrum*. Uma delas é diária e as outras duas ocorrem uma vez a cada *Sprint* – uma no início e outro ao fim.

2.4.1 *Sprint Planning Meeting*

A reunião que marca o início de cada *Sprint* foi chamada de *Sprint Planning Meeting*. Como seu próprio nome diz, é nela que ocorre o planejamento (*planning*) da iteração. É importantíssima a presença de todos os membros cruciais do projeto (*Product Owner*, *Scrum Master* e Time). Também é possível a participação de *stakeholders* ocasionais, caso seja interessante e aprovado pelo *Scrum Master*. Por questões de foco, esta cerimônia é dividida em duas partes.

Na primeira parte, também chamada de Primeira Reunião de Planejamento, o *Product Owner* expõe para o Time quais são suas próximas prioridades para o produto, tendo como base o mesmo como um todo. Caberá ao time fazer todas as perguntas que julgar necessárias para elucidar dúvidas relativas aos pedidos do *Product Owner*. Dessa forma, evita-se que haja retrabalho no futuro, o que é altamente indesejável em um projeto *Scrum*.

É definido então um alvo para o *Sprint*. Tal ação deverá ser coletiva e consensual entre todos os participantes. Este alvo, que algumas vezes é conhecido como *Sprint Goal,* será o principal ponto de avaliação do *Sprint*. A avaliação de andamento será com base no quão direcionada para este objetivo a equipe está, assim como a avaliação final do *Sprint* será em relação à semelhança do que for entregue com este alvo. Como será visto em breve, esta etapa equivale a selecionar funções do artefato *Product Backlog*.

Ocorrido isso, parte-se para a segunda parte, ou Segunda Reunião de Planejamento. Neste momento o time se reúne sem a presença do *Product Owner*, apenas do *Scrum Master.* Caberá a ele decidir o **quanto** do que foi pedido poderá ser feito, com base nas capacidades de produção conhecidas pelo Time. É extremamente importante nesta etapa a honestidade dos membros, seja para evitar uma superestimação que resulte em tarefas não completas ao fim do *Sprint* ou uma subestimação que resulte em membros ociosos desnecessariamente. A cada tarefa deverá ser atribuído um **peso** (ex. numa escala de 1 a 10), indicando a dificuldade da tarefa. Estes pesos são comumente chamados de *Story Points.*

Caso seja julgado necessário, após essa seleção, poderá ser negociado com o *Product Owner* as tarefas a serem executadas, de forma a alcançar um consenso geral. É importante lembrar que não é uma guerra entre cliente e equipe, e sim pessoas trabalhando para um resultado comum de êxito. A posição final sobre a quantidade é do Time, mas o mesmo deve obedecer as prioridades estabelecidas pelo *Product Owner.* A participação do *Scrum Master* neste momento deve restringir-se à evitar pressões do *Product Owner* para com o Time. Como será visto posteriormente, essa parte monta o artefato *Sprint Backlog*, base para o andamento do *Sprint.*

2.4.2 Stand Up Meeting

Os dois nomes possíveis para esta cerimônia (*Stand-Up Meeting* ou *Daily Scrum Meeting*) elucidam bastante sobre a sua forma de execução: ela ocorre todos os dias e deve ser feita com todos os seus participantes de pé.

Esta reunião é a grande responsável pelo bom andamento do projeto, uma vez que é ela garante a agilidade e resposta rápida a problemas, pois acompanha dia a dia o que está ocorrendo. Desta reunião devem participar, comumente, o Time e o *Scrum Master.* Caberá a este fazer a cada membro do Time as três perguntas base:

- O que você realizou desde a última reunião?
- O que pretende realizar até a próxima?
- Há algo lhe atrapalhando em concluir suas tarefas?

A resposta a essas perguntas ajudam a cada membro do Time a manter o foco em suas tarefas, bem como na conclusão das mesmas. Os problemas relatados devem ser resolvidos pelo *Scrum Master* o quanto antes ou então devem ser levados àqueles que podem resolvê-lo. A reunião deve ocorrer com os membros de pé para que haja um controle natural do tempo da mesma. Como a mesma não deve durar mais do que 15 minutos, é justamente neste momento que as pessoas começam a apresentar sinais de cansaço e mostra que a reunião deve chegar ao fim. A reunião deve ocorrer todos os dias na mesma hora e local, para gerar em seus participantes o sentido de responsabilidade desejado para o projeto. Ninguém deve atrasar, e caso isto ocorra, a reunião não deve esperar: o atrasado deverá ter uma pequena penalidade.

É essencial a pró-atividade de todos os membros do time durante a respostas às perguntas do *Scrum Master* . Ele não é superior hierarquicamente, por isso é importante que todos se dirijam à todos durante as respostas. Caberá sim, ao *Scrum Master*, evitar a todo

modo a perda de foco durante a reunião, e a ele é dada a autoridade para manter tal. As pessoas devem falar somente quando tem a palavra, mas é altamente desejado que dêem suporte à quem esta falando no momento, caso seja necessário.

2.4.3 *Sprint* Review Meeting

A ultima reunião do *Sprint* também é conhecida como Reunião de Revisão. Em resumo, pode-se dizer que é o momento em que será avaliado se o que foi combinado durante a *Sprint Planning Meeting* de fato foi realizado com êxito. É importante notar que esta reunião deve possuir um clima um pouco diferenciado da reunião de planejamento. Ela deve ser esperada como o momento em que o Time está satisfeito por ter atingido o objetivo, e deve passar esta satisfação ao *Product Owner.*

Ela também pode ser dividida em duas partes, de modo a organizar melhor a passagem da informação, mas é importante notar que deve-se evitar formalidades. Na primeira parte, a mais importante, o Time deve apresentar ao *Product Owner* a nova versão gerada do sistema, com foco no que foi gerado durante o *Sprint.* Não devem ser usados slides, listas, relatórios ou qualquer forma não-ágil para a passagem de dados. Deve ser dada a preferência para uma exposição do sistema rodando com suas novas funcionalidades, enquanto é explicada em paralelo pelo Time ao *Product Owner.*

O objetivo é permitir que o *Product Owner* possa inspecionar o que foi feito, e com base no obtido, planejar suas prioridades para o que irá pedir como prioridade na próxima iteração. Não deve ser um momento de apresentar como o *Scrum* funciona para eventuais *stakeholder*s. Também não cabe ao *Scrum Master* defender a equipe para tarefas que por acaso não tenham sido feitas. A mesma equipe deve assumir a posição de explicação perante o *Product Owner*. Mas cabe sim ao *Scrum Master* manter o foco da equipe na apresentação e evitar a geração de um cenário tóxico com críticas destrutivas de um *Product Owner* distraído.

A segunda parte da reunião é geralmente opcional, mas em equipes maduras e que já trabalhem bem juntas ela se torna de enorme valia. Neste momento todos juntos, Time, *Scrum Master* e *Product Owner* fazem observações sobre o que acharam do *Sprint*, o que agradou, o que não agradou, e todos podem dar idéias, organizadamente, para formas de melhorar o andamento do projeto na próxima iteração. Afinal, todos trabalham juntos para um bom resultado.

2.5 Artefatos

Os Artefatos do *Scrum* são materiais físicos que servem para fortalecer a idéia de agilidade e visibilidade do projeto, onde qualquer informação e/ou material importante deve estar acessível a qualquer membro.

2.5.1 *Product Backlog*

O *Product Backlog* é uma lista que contém tudo que o *Product Owner* considera importante estar presente no sistema final, e pode ser feita em uma planilha. Caberá a ele escrever esta lista com o máximo de completude que for possível no início do projeto, para que se tenha uma noção da escala do mesmo. Porém é sabido que é impossível prever tudo que será necessário no começo (algumas vezes não é nem mesmo saudável para o projeto), desta forma o *Product Backlog* é uma lista que estará em constante evolução, pois o *Product Owner* estará sempre atualizando-a, retirando, incluindo ou descrevendo melhor alguns itens, conforme a sua visão do projeto vá se refinando.

Conforme dito anteriormente, é desta lista que o *Product Owner* seleciona as tarefas para priorizar para o Time a cada *Sprint*. Caso o mesmo ache interessante e de bom valor agregado ao projeto, pode selecionar tarefas relacionadas às priorizadas para executar no mesmo *Sprint*, desde que não atrapalhe as prioridades.

Esta lista pode ser escrita pelo *Product Owner* tanto com cunho técnico como na forma de Estórias (como no *Extreme Programming*), mas o importante é que a mesma seja suficiente para que o Time compreenda corretamente o que deve ser executado. É de inteira responsabilidade do *Product Owner* manter atualizada, priorizada e disponível a todos SCHWABER (2004).

2.5.2 *Sprint* Backlog

Enquanto o *Product Backlog* é a listagem do que deve ser feito no projeto como um todo, o *Sprint Backlog* representa a listagem do que deverá ser feito no *Sprint.* A iteração deverá seguir estritamente o que estiver definido nesta listagem. É interessante que ,quando as funcionalidades forem selecionadas do *Product Backlog* para integrarem o *Sprint Backlog*, elas sejam transcritas com maiores detalhes, de forma a auxiliar sua implementação da forma o mais correta possível.

Existem diversas formas de criar a lista, mas uma bastante interessante é a proposta por KNIBERG (2007): A listagem é criada na forma de um quadro, preso na parede visível a todos. Os itens da lista ficam representados em *post-its*, divididos em três colunas indicando o estado de desenvolvimento daquele item (Aguardando, Em Desenvolvimento e Concluído), e o *post-it* é movido de acordo com a evolução da tarefa. Também pode ser interessante incluir na parede uma planilha contendo informações adicionais sobre cada item e uma indicação de que critérios serão utilizados para testar aquele item específico ao fim do *Sprint.*

O *Sprint Backlog* é mantido pelo Time, e deve ser atualizado diariamente, logo após a *Stand Up Meeting*. Através do andamento das tarefas pelas colunas, tanto o Time como o *Scrum Master* podem avaliar a saúde do andamento do projeto e agir, caso haja algum problema, como por exemplo uma tarefa ficar mais tempo Em Desenvolvimento do que estava pensado inicialmente por sua dificuldade. A disciplina na manutenção desta lista é um grande auxílio para o sucesso do projeto.

2.5.3 Burndown Chart

O *Burndown Chart* é um gráfico cujo objetivo é garantir um *feedback* visual imediato do andamento da execução das tarefas do *Sprint Backlog* ao longo da iteração. É essencial que esteja visível a todos os membros do projeto, por exemplo preso em uma parede.

É composto de dois eixos: Um deles indica a quantidade de dias que compõe o *Sprint* (geralmente o eixo x), enquanto o outro representa a soma de *Story Points* das tarefas presentes no *Sprint Backlog* corrente (geralmente o eixo y). Após a confecção do gráfico, deve ser traçada uma linha partindo da origem (0,0) até o ponto máximo de ambas as escalas. Esta linha é tida como a Linha Ótima, que representa o andamento ideal do projeto ao longo dos dias.

Diariamente, após cada *Stand-Up Meeting*, seguinte à atualização do *Sprint Backlog* pelo Time, caberá ao *Scrum Master* a tarefa de somar os *Story Points* das tarefas realizadas no dia e atualizar o gráfico, traçando uma linha reta do ponto onde ele se encontrava no dia anterior para onde ele se encontra neste momento. O ideal é que, durante o *Sprint*, a linha traçada no dia-a-dia se mantenha o mais próximo possível da Linha Ótima. Caso ela fique muito abaixo, significa que a quantidade de tarefas da iteração foi superestimada e o Time ficará ocioso. Caso contrário, significa que a iteração foi subestimada, e possivelmente o

Time não conseguirá realizar todas as tarefas a tempo. A proximidade da Linha Ótima está diretamente ligada à saúde do *Sprint*, e portanto, do projeto em si.

Algumas equipes se sentem desconfortáveis com o fato do *Burndown Chart* ser um gráfico decrescente, pois psicologicamente pode representar decrescimento de êxito. Caso seja do desejo do Time, o gráfico pode ser invertido, gerando uma espécie de "*Burnup*". A única diferença neste caso é que as regras de avaliação de saúde do projeto ficam invertidas nos quesitos "acima" e "abaixo", merecendo atenção especial a este ponto.

3. Aplicando *Scrum* em projetos de *Games*

Como visto no Capítulo 1, a indústria de *Games* enfrenta um cenário de aumento considerável de recursos necessários para a construção de um produto. Com isso, torna-se natural a busca por outras formas de produção, que possam reduzir tempo e principalmente custo.

3.1 Motivações e pesquisas

O público consumidor de *Games* é altamente exigente, e toma como a sua base de crítica a qualidade de títulos de grande orçamento, os chamados *Triple A* ou *AAA*. Tendo em vista tal cenário, um corte mal planejado nos recursos de um projeto pode impactar em aspectos que sejam muito visíveis no produto final. Um bom exemplo diz respeito à parte artística, que consome boa parte dos custos de um projeto, e é a primeira a ser percebida pelos jogadores. Portanto, a redução de verbas de um projeto precisa ser pensada de modo a não impactar em aspectos importantes.

Técnicas alternativas de *Game Design*[4] podem ser excelentes formas de se criar produtos de qualidade com baixo custo e prazo, porém, este não é o tema deste trabalho. O objetivo é a verificação de que, através da mudança da metodologia clássica de desenvolvimento de *software*, é possível desenvolver um *Game* de maneira mais otimizada. Neste aspecto, as metodologias ágeis de desenvolvimento de *software* naturalmente se apresentam como uma boa alternativa, com *Scrum* destacando-se entre elas. Já há movimentação na comunidade de desenvolvimento de *Games* relativa à pesquisa da aplicação da metodologia *Scrum* em projetos de *Games*. Autores de entidades renomadas apresentam teorias e resultados dignos de nota.

COHN (2007) possui um bom resumo a respeito do paralelo de projetos de *Games* e *software* tradicionais que utilizam *Scrum*. Para o autor, existem muitas semelhanças entre os

[4] Processo responsável por desenvolver toda a mecânica de *game*, interações com jogador e qualquer aspecto relacionado à jogabilidade.

dois tipos de projetos, é importante a transposição de alguns papéis/cargos essenciais da indústria para o *Scrum*. Basicamente, seu estudo traz todos os conceitos relevantes presentes em um projeto de *Game,* bem como aqueles que são dignos de nota em um projeto *Scrum*. Também destaca como muito importante a presença da chamada "visão do jogador", sugerindo que a mesma seja representada pelo *Product Owner*, se for possível.

MILLER (2008) foca em outro aspecto, mais prático. Em seu ótimo artigo para a renomada comunidade de desenvolvedores de *Games* do site Gamasutra.com, apresenta algumas ciladas (*pitfalls*, do original em inglês) que aguardam aqueles que pretendem utilizar *Scrum*. Seu estudo apresenta basicamente problemas originados de uma aplicação direta e não flexível do *Scrum* em um projeto deste tipo. O estudo do autor foi de grande valia para o estudo de caso, que será tratado no tópico seguinte.

Em todo caso, a lição mais importante notada pelos autores é o fato de não ser saudável utilizar *Scrum* para desenvolver um *Game* sem que haja uma adaptação. Projetos deste tipo apresentam consideráveis diferenças de *softwares* tradicionais em vários aspectos, como perfil da equipe (multidisciplinar e muito interessada no produto final), ciclo de vida do produto (uma vez que o produto é lançado, dificilmente permite-se correções), perfil do cliente/consumidor (altamente exigente).

3.2 O ambiente de estudo de caso

A grande valia dos projetos ágeis de *software* está em sua prática e não na teoria. Uma vez que prega a redução máxima da "burocracia de projeto e documentação", pode-se obter mais informação através da execução de uma empreitada ágil do que meramente com estudos bibliográficos.

As duas pesquisas citadas apresentam materiais de grande relevância para um ponto de partida: devemos ter em mente que o *Scrum* não funcionará magicamente para qualquer tipo de projeto, e já se possuía uma lista de pontos a fazer e/ou evitar.

Tendo isso em mente, partiu-se para a procura de um ambiente no qual fosse possível, assim como provavelmente produtivo, o teste prático do que é proposto. Este foi encontrado na empresa Aiyra, uma empresa de desenvolvimento de *Games* que faz parte da equipe da Incubadora de Empresas de Base Tecnológica da Universidade Federal Fluminense (INITIA).

Esta empresa, no fim do ano de 2008, foi vencedora de um edital regional para o desenvolvimento de um *Game*. Os recursos e prazo oferecidos pelo edital eram restritos, mas a empresa gostaria de produzir algo que já possuísse qualidade suficiente para que a mesma pudesse buscar financiamento para o projeto.

O cenário apresentou-se altamente interessante para a aplicação do *Scrum*, e assim foi feito. Ao longo do ano de 2009, o projeto foi executado com grande êxito, estando o produto próximo de sua entrega no início de 2010. O uso da metodologia ágil trouxe um balanço bastante positivo para o projeto mas, como era de se esperar, surgiram problemas que precisaram ser contornados para a continuidade do projeto. Porém, como a própria força do *Scrum* está na sua adaptabilidade à mudanças, não somente o projeto foi adaptado como as próprias peculiaridades percebidas puderam ser estudadas de modo a terem suas origens percebidas e compreendidas.

3.3 Principais peculiaridades notadas

Como citado acima, houve problemas decorrentes do uso do *Scrum*, mas eles foram identificados e estudados. Neste trabalho, optou-se por uma abordagem com foco no que gerou atrito, ao invés de apenas uma listagem do que correu corretamente, por crer-se que a primeira agrega mais valor. A seguir encontram-se listados, por tópicos, as peculiaridades.

Em comparação ao conteúdo apresentado na pesquisa bibliográfica, como está explicitado em cada tópico, alguns aspectos foram coincidentes em ocorrência e importância. Outros, mesmo citados na bibliografia, não foram percebidos no ambiente, porém alguns destes ainda assim foram citados devido à sua importância. Por fim, há aqueles que mesmo tendo sido notados, não apresentaram o aspecto negativo citado na bibliografia.

- **CONFLITO ENTRE A AUSÊNCIA DE DOCUMENTAÇÃO ESCRITA PREGADA PELO *SCRUM* CLÁSSICO E A GRANDE IMPORTÂNCIA DO DOCUMENTO DE *GAME DESIGN* (*GAME DESIGN DOCUMENT* OU GDD):**

    - Um dos pontos de sucesso da agilidade do *Scrum* clássico está na inexistência de uma burocracia para a criação de documentos, gerenciando o que deve ser feito, bem como e quando, através das estórias, que representam bem o que deve ser implementado. Porém, no desenvolvimento de *Games* o equivalente a

documentação de engenharia de *software* tem como paralelo o chamado Documento de *Game Design* (GDD). Uma peculiaridade deste item de documentação se deve ao fato que ele não contém apenas informações técnicas sobre o projeto, como também dados a respeito do clima que o *Game* deve passar ao jogador, clima esse de grande importância para as áreas artísticas do projeto (visual e sonora). Enquanto informações técnicas e de usabilidade podem ser facilmente gerenciadas puramente através de estórias do *Product Backlog*, dados mais subjetivos como clima do *Game*, mensagem a passar ao jogador e definições complexas de mecânica de jogabilidade tendem fortemente a se perder.

- Este item foi citado com importância por MILLER (2008). Coincidentemente, no projeto desenvolvido na Aiyra, após o primeiro mês de projeto, o *Scrum Master* e o *Product Owner* já perceberam alguma perda de informação, e precisaram criar uma documentação compartilhada específica para comportar todas as constantes inclusões no *Game Design*. Com essa medida, o caos foi evitado com sucesso.

- **CONFLITO ENTRE O FOCO NECESSÁRIO NA *STAND UP MEETING* E O INTERESSE POSITIVO DA EQUIPE EM DISCUTIR A RESPEITO DO *GAME DESIGN* E OUTROS ASPECTOS DO *GAME*.**

  - SILVA & LEMOS (2008) reforçam a importância de se manter o foco na reunião diária do *Scrum*. Tal abordagem, além de ser extremamente válida, é muito importante, pois dado o caráter descontraído desejado para a Stand Up Meeting, a reunião poderia facilmente se tornar em uma enorme perda de tempo.

  - Se em projetos tradicionais esta já deve ser uma preocupação, em projetos de *Games* a atenção do *Scrum Master* deve ser dobrada. Durante a aplicação do projeto, constatou-se que quase todos os envolvidos neste tipo de projeto, independente da área em que atuem, possuem interesse em sugerir e discutir principalmente sobre aspectos do *Game Design*. As primeiras *Sprint Review Meeting* e *Sprint Planning Meeting* foram consideravelmente caóticas, pois se tornaram intermináveis discussões sobre as mecânicas do *Game*. Embora este assunto esteja citado parcialmente por MILLER (2008), o *Scrum Master*

precisou proibir este tipo de discussão nas reuniões comuns do *Scrum*, e criando um momento eventual para que todos pudessem ter suas opiniões ouvidas. Além disso, foi necessário remover a participação do *Game Design*er das *Stand Up Meeting*, gerenciando o trabalho do mesmo à parte.

- **ERRO AO FECHAR O *GAME DESIGN* A ALTERAÇÕES DURANTE O PROCESSO DE DESENVOLVIMENTO.**

    - O desenvolvimento de *Games* possui uma etapa que necessita de atenção para exageros: a consideração do *Game Design* como concluído. O primeiro perigo diz respeito a um cenário onde o Time nunca consegue parar de imaginar novas funcionalidades para o *Game*, gerando retrabalho atrás de retrabalho e criando um cenário de constante frustração. Já o segundo perigo reside no fato de se considerar que todo o *Game Design* possa ser imaginado antes do início da implementação do projeto.

    - Neste ponto, uma metodologia ágil não surge como um ponto de ciladas, mas sim como a solução para o problema. A agilidade está no suporte a mudanças, e uma tolerância moderada à alterações/inclusões no *Game Design*. Deve-se ter um ponto de parada, no qual a equipe considera ter o suficiente para desenvolver algo. Após isso, deve-se estar aberto a mudanças percebidas com a experiência de ter o projeto em mãos. Até mesmo mudanças grandes de escopo são suportadas, se forem feitas de maneira correta e cadenciada. No estudo de caso, o *Scrum Master* e o *Product Owner* estavam cientes deste fato, e já iniciaram o projeto com isto em mente, o que se provou de enorme valia ao longo do projeto. É justo dizer que, se este ponto não tivesse sido considerado, o projeto teria alcançado um momento de falha catastrófica.

- **ERRO AO CONSIDERAR DESNECESSÁRIA OU IMPOSSÍVEL A APLICAÇÃO DE METODOLOGIAS A TAREFAS CRIATIVAS, PRINCIPALMENTE COM RELAÇÃO AO *GAME DESIGN*:**

    - Uma expressão comum que se ouve ao tratar-se se de tarefas de criação é “Não há como estimar o tempo que se leva para criar. Tudo depende de inspiração”. Tal conceito está parcialmente correto, porém não é o suficiente para que as

tarefas criativas (arte, som e especial o *Game Design*) corram ao longo do projeto sem controle ou estimativa.

- No projeto estudado, inicialmente foi tomado o cuidado de se gerir através do *Scrum* não apenas as tarefas de codificação como também as de arte visual e sonora. Porém, como as tarefas do *Game Design* eram aquelas que gerariam as tarefas dos outros, inicialmente pensou-se que as mesmas não poderiam ou mesmo deveriam ser estimadas. Em dois meses, isso provou-se errado, e as tarefas de *Game Design* começaram a ser tratadas juntamente em ambos os *Backlogs*. O resultado foi excelente, e os problemas que se imaginavam decorrentes anteriormente na verdade foram percebidos como superestimados.

- **PROBLEMAS NAS PARTICIPAÇÕES DO *PRODUCT OWNER* EM UMA *STAND UP MEETING:***

  - SCHWABER (2004) nos diz que no *Scrum* diz que há dois tipos de participação em uma reunião: Há o *Chicken*, aquele que está apenas envolvido com o projeto, e o *Pig*, aquele que está comprometido de fato. Em uma *Stand Up Meeting*, é tido que quem for considerado como *Pig* pode se manifestar, enquanto aquele que for *Chicken* pode apenas ouvir. Sugere também que a participação de *Product Owner*s em uma *Stand Up Meeting* possa ocorrer através da participação como Chicken, ou seja, sem falar efetivamente.

  - No projeto estudado, o *Product Owner* era o *Game Design*er principal do projeto e possuía, nos dois primeiros meses, um auxiliar para as tarefas de *Game Design*. Isto gerou uma série de problemas. Primeiramente o Time se sentia pressionado com a presença do *Product Owner*, o que foi solucionado tranquilamente com a ausência dos mesmos da reunião. Porém, como o auxiliar de *Game Design* participava das mesmas, isso gerava desencontro nas informações, criando ruído na comunicação e vários problemas para a equipe. O problema foi solucionado retirando qualquer participação de *Game Design*er da reunião diária, e chegando-se à conclusão que em projetos futuros a participação deste só poderia se dar através da posição de *Chicken*.

- **ERRO AO INTERROMPER O TRABALHO CRIATIVO PARA A *STAND UP MEETING*:**

    - MILLER (2008) nos atenta para o cuidado de que caso todos interrompam imediatamente o que estiverem fazendo para realizar a reunião diária, isto pode causar grandes perdas de concentração e produtividade. O autor sugere uma flexibilidade quanto ao tratamento da *Stand Up Meeting* quando da aplicação de *Scrum* a projetos de *Games*.

    - Devido à pesquisa bibliográfica, ambos os *Scrum Master* e *Product Owner* estiveram atentos para este fato. Porém, a equipe que executava o projeto conseguiu se adaptar bem à disciplina diária da reunião, sem a necessidade de interrupções. Todos iniciavam as tarefas tendo em mente o tempo que julgavam necessário, dando grande importância para o ritual da reunião diária. Obviamente é importante notar que tal comportamento não deve ser tomado como padrão, e que o mais importante é que o *Scrum Master* conheça aos poucos sua equipe e adapte-se à forma do seu Time trabalhar.

- **OCIOSIDADE DE UM MEMBRO DO TIME COM A JUSTIFICATIVA DE NÃO HAVER NADA QUE POSSA SER DESIGNADO PARA ELE NO *SPRINT BACKLOG* E/OU ESTAR AGUARDANDO PARA O "GERENTE" PASSAR-LHE NOVAS TAREFAS.**

    - Esta peculiaridade pode inicialmente parecer óbvia, mas é bastante comum quando se está se dando o treinamento de novos membros para a metodologia *Scrum*. MILLER (2008) cita esta como uma cilada.

    - Assim como no item anterior, o *Scrum Master* estava preparado, mas o Time de desenvolvimento da Aiyra foi bastante pró-ativo. Por algumas vezes ocorreu de que programadores, artistas ou músicos terminassem suas tarefas antes do fim esperado para o *Sprint*. Porém, ao invés de aguardarem uma nova tarefa passada pelo *Scrum Master*, em boa parte do tempo os membros buscavam pesquisar a respeitos de itens que já conheciam do *Product Backlog*, a serem realizados futuramente e que poderiam possuir certa dificuldade imaginada.

    - Neste ponto pode-se notar como o quesito visibilidade, que é crucial ao *Scrum*, pode tornar possível a pró-atividade do Time. É importante notar que tal comportamento não deve ser tomado como padrão, mas é digno de nota.

- **ERRO AO ESTIMAR ESFORÇOS, *SPRINTS*, CUSTOS E AFINS ANTES DE POSSUIR PELO MENOS ALGO DO GDD JÁ DEFINIDO.**

    - É comum que aficionados por tecnologia queiram definir em que plataforma trabalharão logo de início, assim como clientes queiram saber alguma posição relativa a prazo ou custos do projeto antes do chamado "Dia 0" do projeto. MILLER (2008) cita esta experiência.

    - No projeto estudado na Aiyra, houve ciência deste fato. Enquanto o GDD estava sendo criado, usou-se a motivação do Time de desenvolvimento e a pressão do Edital como motores para executar uma pesquisa por tecnologias com o máximo de flexibilidade possível, de modo a comportar diversas alterações que o projeto poderia sofrer. Deste processo surgiu a escolha de uma ótima *Engine*[5] para o jogo, que se mostrou flexível em diversas situações. Além disso, os artistas e músicos foram trabalhando em paralelo com o *Game Design* para a criação da Arte Conceitual[6],de modo a criar o máximo de material de base possível para a implementação. Tal abordagem foi de extrema valia e pode ser repetida.

[5] Pacote de código reutilizável, geralmente aplicado para gerar as bases de gráfico, física e som de um *game*.
[6] Arte gerada para que se possa dar corpo físico a conceitos e idéias, gerando uma base de referência para todos.

4. Adaptando o *SCRUM* para Games – *Scrum4Games*

Através da análise dos resultados dos estudos de casos, é possível observar que há uma grande quantidade de vantagens na aplicação da metodologia *Scrum* em projetos de desenvolvimento de *Games*. É importante notar que tais vantagens não apenas estão ligadas à qualidade do andamento do projeto em si, como também existe uma sinergia com as novas necessidades do mercado de *Games*, citadas anteriormente.

Não obstante, também é relevante lembrar que a aplicação direta das regras da metodologia gerou, como visto no estudo de caso, alguns problemas. Tais problemas precisaram ser contornados, ora tratados caso a caso, ora tratados em grupo. Embora as soluções adotadas tenham surtido o efeito esperado e corrigido as imperfeições, não é desejável que a aplicação de uma metodologia em um projeto possua tantas exceções e aplicações empíricas, pois tal atitude poderia levar a um ambiente caótico. Em tal ambiente, sem padrões, é praticamente certo que a produtividade volte a baixar, tornando inútil o esforço de aplicação inicial do *Scrum*.

Para evitar tal cenário, o próximo passo importante é a adaptação desta metodologia para projetos de *Games*. Uma vez que *Scrum* é tratado como um *framework*, tal tarefa torna-se mais simples. É preciso analisar cada etapa, regra e artefato do modelo original, considerando os impactos e conseqüências observados, e com isso criar novos métodos, que serão avaliados até que seja comprovada sua eficácia. O objetivo desta nova metodologia é ser um processo que, baseando-se nos padrões originais do *Scrum*, possa ser aplicado a um grande número de projetos de *Games*. Espera-se que com isso, garanta a eles um desenvolvimento ágil, de grande valor agregado e que proporcione um produto de qualidade gerando um melhor retorno de investimento.

O intuito deste trabalho é contribuir para a criação desse modelo. Avaliando os dados obtidos ao longo dos projetos observados nos estudos de caso, observando as soluções adotadas e seus resultados, foi possível criar um paralelo com os preceitos do *Scrum*. A proposta a seguir é um arcabouço que reúne de forma genérica soluções preventivas para

problemas comuns da aplicação de *Scrum* em projetos de *Games*, tornando seu desenvolvimento mais estável e com menos imprevistos. É importante ressaltar que os imprevistos que se deseja evitar não dizem respeito ao projeto, mas ao uso da metodologia: Se inesperadamente a metodologia se tornar dúbia ou falha, o caos pode dominar o projeto. Imprevistos de projeto, por sua vez, são um dos pontos principais da força do *Scrum* (que os considera inevitáveis e por isso, foca em contorná-los), não fazendo assim sentido um esforço em impedi-los.

Para fins ilustrativos, esse framework adaptado será chamado, deste momento em diante, como *Scrum4Games*. Vale ressaltar que neste capítulo serão tratados apenas os elementos do *Scrum* que sofreram modificações ou que necessitaram de atenção especial, permanecendo os restantes inalterados quanto a seus conceitos iniciais.

4.1 Análise da Metodologia Proposta

Por ser uma metodologia focada na simplicidade, agilidade e redução da burocracia, *Scrum* se foca na quantidade reduzida de métodos de trabalho para gerar produtividade. Uma vez que são tais métodos que levam do desenvolvimento do projeto ao resultado em si, são justamente eles que devem ser observados e, posteriormente, alterados conforme a necessidade dos projetos de *Games*. Deve-se evitar, porém, que as alterações descaracterizem os métodos originais: elas devem ser feitas ao redor do *kernel*[7] do framework.

Na seção 4.1.1, encontram-se as análises dos principais ativos de um projeto de desenvolvimento de *Games*, observados sob a ótica do *Scrum*, de modo que seja possível compreender as alterações finais propostas que se encontram após tal seção.

4.1.1 Estrutura da equipe

O ponto principal para a análise de um time que desenvolve um *Game* é notar a grande variedade de profissionais envolvidos em seu "processo central". Como processo central está compreendido todo o processo de desenvolvimento que gera conteúdo essencial ao *Game*, sem o qual o produto final não será completo, ou mesmo funcional.

[7] Termo comumente usado em computação para definir o centro, a parte mais importante, de um software ou conceito.

Em um projeto de *software* comum como, por exemplo, o desenvolvimento de um sistema corporativo de controle de informações, grande parte da equipe é formada por programadores, arquitetos de software , ou seja, engenheiros de *software*. Eventualmente encontram-se funções mais específicas, como *DBA's*, especialistas em segurança, engenheiros de rede, especialistas em criptografia, entre muitas outras áreas da ciência da computação. Embora tais profissionais possuam uma grande especialização em suas funções, um fato importante é que todos estão ligados à mesma área de conhecimento: computação. Essa sinergia é importante para que, mesmo em um projeto onde haja uma série de especialistas, não haja muitos ruídos na comunicação interna ou até mesmo externa da equipe, uma vez que todos compreendem boa parte de todo o processo e objetivos tanto do próprio trabalho como o de outros membros. Ocasionalmente podem haver especialistas com funções menos sinérgicas com a computação, como por exemplo um artista gráfico responsável por uma interface mais atrativa para um programa. Caso tal profissional não tenha formação na área de programação ou semelhantes, sua experiência e campo de trabalho o forjam para que tenha a compreensão do trabalho de seus colegas.

Não obstante, em um projeto de *Games*, o cenário é diferenciado. O "processo central" e o valor agregado de um *Game* não estão apenas em sua programação, como pode ser o caso de um *software* comum, onde é possível experimentar o funcionamento do sistema tendo apenas seu código pronto. Em um *Game*, torna-se necessário o desenvolvimento do material artístico (em duas ou até mesmo em três dimensões) e sonoro, para que o produto final possa ser experimentado e avaliado. Embora trabalhem juntos para a criação de um produto em comum (o qual também é um *software*), esses profissionais não possuem a sinergia natural presente nos times tradicionais de desenvolvimento de *software*. Não faz parte da lógica de raciocínio de um modelador, por exemplo, compreender o que são *floating points* (um tipo de dados). Não é comum para um músico o conceito de *sub-surface light scattering* (um recurso de iluminação 3D para aumentar o realismo). É igualmente estranho, para um programador, entender as curvas e retas que originam uma arte conceitual.

Deve-se ressaltar que, mesmo pertencendo a funções tão distintas como programação, computação gráfica e composição musical, é muito importante que o time de desenvolvimento de um *Game* tenha uma boa integração, não apenas para a criação e manutenção de um ambiente saudável, como também para a solução de problemas que sejam comuns a duas ou mais áreas diferentes. Um dos exemplos mais recorrentes desse tipo de situação é a de

comunicação entre diferentes tecnologias: Pode ser necessário, por exemplo, que os dados gerados por um programa de modelagem 3D tenham que ser exportados em um formato específico para que a importação pelo código seja possível e correta. Caso não haja boa comunicação dentro do time e esse processo de exportação/importação não seja tão simples, este será um grande empecilho para o desenvolvimento do projeto, podendo até mesmo impossibilitá-lo.

JAMES (2008) cita a importância da multidisciplinaridade em um time *Scrum*. No caso do desenvolvimento de *Games*, a multidisciplinaridade é ainda mais forte, com a presença de especialistas em todos os projetos. Não se possui membros de disciplinas diferentes apenas em uma área, mas sim, de diversas áreas. É bastante comum, inclusive, que a quantidade de membros de outras áreas, somados, seja superior à quantidade de programadores. Tal fato, se tratado da maneira correta, pode ser um grande aliado para a aplicação do *Scrum* em projetos de *Games*. Percebe-se que, enquanto *softwares* convencionais possuem raras vezes especialistas de outras áreas durante seu desenvolvimento, para *Games* essa presença especializada é literalmente obrigatória na maioria dos casos, como já citado anteriormente.

Porém, a diversidade de áreas não deve fazer com que o time seja separado em times de artes, código e som, respectivamente. Além de haver perda na riqueza de comunicação, problemas como o de interface de tecnologia seriam dificílimos de se obter soluções em prazo hábil, pois cada equipe pesquisaria independentemente, e somente sobre sua área de conhecimento. Da comunicação do time podem surgir soluções para os mais variados problemas, seja por um obstáculo comum ou pelo valor eventual de um olhar não-técnico sobre uma questão específica. Conclui-se, portanto, que o conceito do *Scrum* de equipe multidisciplinar deve ser mantido, formando o que é chamado de "Time de Desenvolvimento", composto por profissionais das áreas de arte, som e código.

Outra disciplina altamente relevante é o *Game Design*. Porém, esta merece atenção especial por sua natureza. Como será explanado nas próximas seções, é comum a boa parte dos profissionais envolvidos com *Games* o interesse pela participação nas escolhas das mecânicas que irão compor o *Game*, bem como seu roteiro, clima, entre outros. Como citado por SCHUYTEMA (2008), o *Game Design* é composto de parcelas de arte e ciência, sendo a primeira responsável pelas boas idéias e a segunda responsável pela transformação das boas idéias em bons *Games*. Todo profissional é capaz de ter boas idéias que ocasionalmente levem a excelentes *Games*. Porém, poucos possuem a habilidade necessária para a efetiva

transformação da idéia em conceito de *Game*, do conceito em documento de design, e do documento de design em *Game* efetivamente. E é justamente esse o papel do time do(s) *Design*er(s). Uma vez que o interesse de todos pelo *Game Design* é grande, caso não seja controlada, essa prática pode trazer problemas ao projeto.

Portanto, a primeira adaptação proposta pelo *Scrum4Games* diz: Todo projeto de *Games* que utilize *Scrum* deve possuir dois times independentes e paralelos, o Time de Desenvolvimento e o Time de *Game Design*. Cada time deve ser construído seguindo as regras *Scrum*, ou seja, possuindo uma variação de cinco a nove membros, com um time auto-gerenciável, um membro com a função de facilitador do andamento do projeto (*Scrum Master*) e um responsável pelo produto (*Product Owner*).

Essa estruturação faz com que o time para desenvolver um *Game* tenha um mínimo de dez pessoas, ainda sem considerar os *Product Owner*s, o que contrasta com o padrão do *Scrum*, que pedia no mínimo cinco pessoas acompanhadas de seu *Product Owner*. Embora pareça desnecessário, tal quantidade é essencial para que um projeto de *Games* desenvolvido com *Scrum* corra bem e que gere um produto de qualidade.

Há diversas observações importantes a respeitos dos papéis do *Scrum* com essa nova organização, que serão explicadas na seção 4.1.2. A Figura 1 ilustra a aparência do time básico, ainda sem as observações. Cada quadrado representa um time de 9 membros; o quadrado externo representa seu respectivo *Product Owner*; o quadrado destacado internamente representa seu respectivo *Scrum Master*:

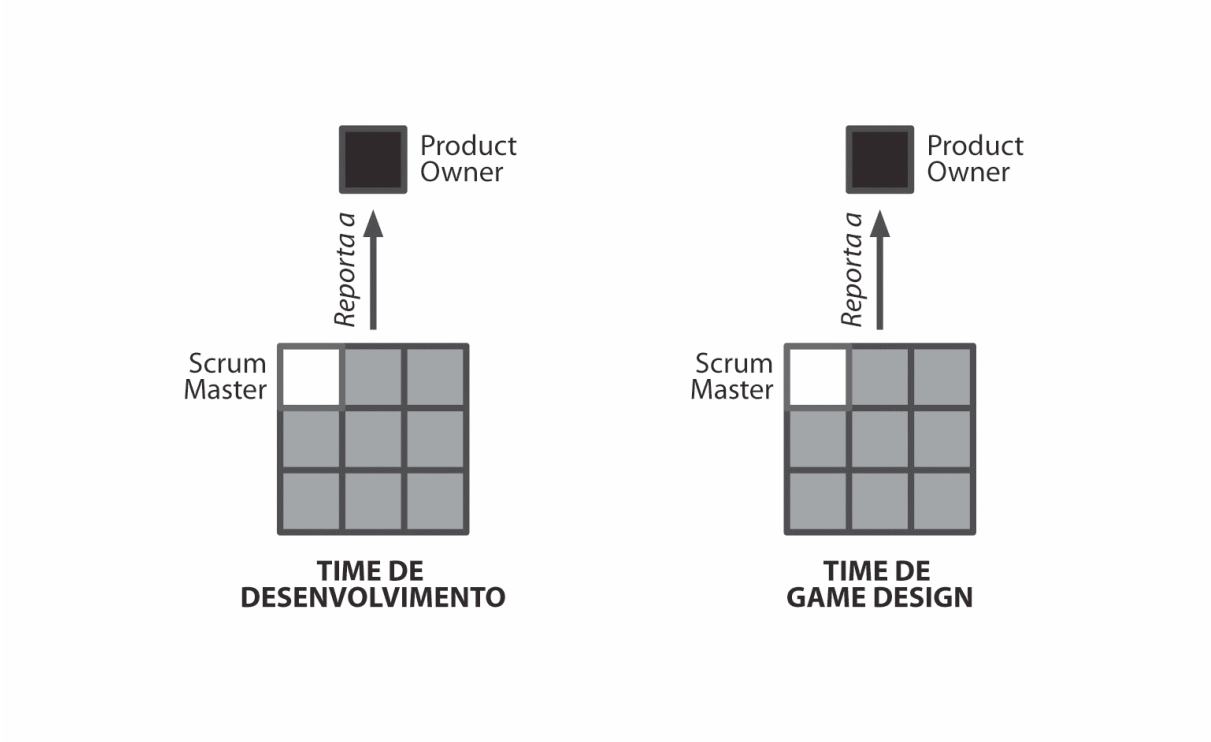

Figura 1 - Os dois times básicos do *Scrum4Games* com seus papéis

Uma vez que já está compreendida a divisão essencial das equipes do *Scrum4Games*, devemos nos voltar agora ao comportamento dos papéis do *Scrum* nessa nova abordagem. Conforme dito anteriormente, cada um dos times será organizado de acordo com o padrões do *Scrum*.

4.1.2 Adaptações de papéis – Time de Desenvolvimento

O Time de desenvolvimento possuirá seu *Scrum Master* e seu *Product Owner*. É importante que o *Scrum Master* do time de desenvolvimento (que deverá ser referido como *Development Scrum Master*, para fins de organização do processo) possua algum conhecimento nas áreas de arte gráfica, codificação, música e *Game Design*. Um grande ponto de discussão entre os autores que estudam *Scrum* é se o *Scrum Master* deve ou não possuir tarefas de desenvolvimento. O principal argumento para uma resposta negativa é de que ele poderia não dar atenção suficiente para o time enquanto estivesse realizando suas funções de desenvolvedor. Já o argumento mais forte para um posicionamento positivo cita que o envolvimento do *Scrum Master* com o projeto no papel de desenvolvedor garante que ele esteja alinhado com o trabalho do seu time, estando pronto para resolver problemas rapidamente.

A proposta do *Scrum4Games* é a união dos pontos positivos de ambos os casos: atribuir ao *Development Scrum Master* a tarefa de resolução de problemas que estejam atrasando ou mesmo impossibilitando o avanço do time. Uma vez que este profissional deverá ter conhecimento básico de todas as áreas, ele poderá ser responsável por tarefas como: pesquisa, escolha e uso de ferramentas específicas; verificação de soluções de para barreiras tecnológicas e/ou de interface; instalações e obtenções de licença, avaliação de ferramentas com base em informações de custo/beneficio (custo financeiro, esforço de utilização, aceitação do mercado, resultados possíveis, entre outros).

Esse posicionamento quanto às funções do *Development Scrum Master* é benéfico em vários sentidos. Em primeiro lugar, ele estará constantemente a par do status de todos os membros de seu time, obrigatoriamente, já que será sua responsabilidade pesquisar por bloqueios que possam surgir. Ao realizar as pesquisas, estará se integrando cada vez mais ao ambiente de seu time, reduzindo ruídos de comunicação e inspirando mais confiança na equipe. Além disso, sempre que houver alguma questão atrapalhando o desenvolvimento do projeto, o *Development Scrum Master* tomará para si a pesquisa e, se possível, a resolução, deixando o time livre para continuar seu trabalho em paralelo.

Em relação à figura do *Product Owner* do desenvolvimento (*Development Product Owner*), embora possa-se associá-lo à função de fornecer a chamada "visão do jogador", tal abordagem é arriscada. Há uma enorme gama de perfis de jogadores e, uma vez que o *Product Owner* com tal posição teria grande tendência para imprimir sua opinião como jogador para o projeto (o que não é desejável, principalmente em um projeto *Scrum*), isso representa um risco de insucesso para o produto final. Também não é interessante considerar o *Product Owner* como o único ponto de vista representativo do público-alvo, uma vez que seria difícil a simulação, por parte dele, dos diversos gostos que permeiam os jogadores do mundo.

Sabendo disto, é correto assumir que o *Development Product Owner* possua o papel de *Game Design*er, ou mesmo líder da equipe de *Game Design*. Tal conclusão se dá não somente por estas características como também por este exercer, em seu papel, a posição de defesa do projeto visto como um produto, comportamento altamente compartilhado por *Game Design*ers e *Product Owner*s, e profissionalmente desejado em ambos. Esse profissional híbrido será melhor tratado na seção 4.2.1.

Tendo em vista o time como dito anteriormente, é sabido que, ao escolher a profissão de desenvolvedor de *Games*, boa parte dos profissionais se destaca por ter grande interesse no produto final, não apenas em sua construção. Pode-se dizer que, enquanto boa parte dos desenvolvedores de *software* padrão está preocupada com os meios, os que trabalham com *Games* estão preocupados com o fim.

Para manter a motivação destes profissionais, que podem ser das áreas de código, artes gráficas ou mesmo efeitos sonoros e composição, é importante fornecer aos mesmos a possibilidade de opinar e participar da construção do *Game Design* do *Game* no qual estão trabalhando. Caso esta participação seja feita de forma organizada, não apenas será altamente gratificante para o desenvolvedor, aumentando sua motivação para com o projeto e conseqüentemente sua produtividade, como também garantirá uma diversidade de pontos de vista sobre *Game Design*, o que é altamente salutar. Sugere-se que esta participação se dê através de uma reunião específica para tal, como será tratado na seção 4.2.3.

O time de desenvolvimento deve ser formado de acordo com as necessidades do projeto em questão. É comum a todos os projetos de *Games* que haja a necessidade de especialistas nas áreas de artes gráficas, codificação e efeitos sonoros, como já dito anteriormente. A quantidade de profissionais em cada uma dessas áreas, porém, irá variar com a complexidade do projeto na respectiva parte de abrangência, permitindo uma maleabilidade nessas quantidades. Porém, para um projeto de dificuldade padrão e complexidade balanceada (basendo-se no custo do mesmo), considerando toda a produção feita pelo time interno (sem terceirizações), sugerem-se os seguintes *templates* para a formação do time de desenvolvimento. Tal sugestão foi baseada na experiência do estudo de caso e extrapolada para outras configurações:

- 5 membros: 2 Programadores + 2 Artistas Gráficos + 1 Músico
- 6 membros: 3 Programadores + 2 Artistas Gráficos + 1 Músico
- 7 membros: 3 Programadores + 3 Artistas Gráficos + 1 Músico
- 8 membros: 4 Programadores + 3 Artistas Gráficos + 1 Músico
- 9 membros: 4 Programadores + 3 Artistas Gráficos + 2 Músicos

A Figura 2 mostra a evolução da compreensão a respeito dos times e papéis do *Scrum4Games*. Em uma equipe de 9 membros, tem-se 4 programadores (verde), 3 artistas (amarelo) e 2 músicos (azul). O profissional destacado em branco é *Development Scrum Master*. Pode-se perceber que a indicação do *Development Product Owner* agora aponta para o Time de *Game Design*, numa indicação de que ele pode ser considerado o "cliente". O quadrado mais claro representa o *Scrum Master*, que também é um dos programadores.

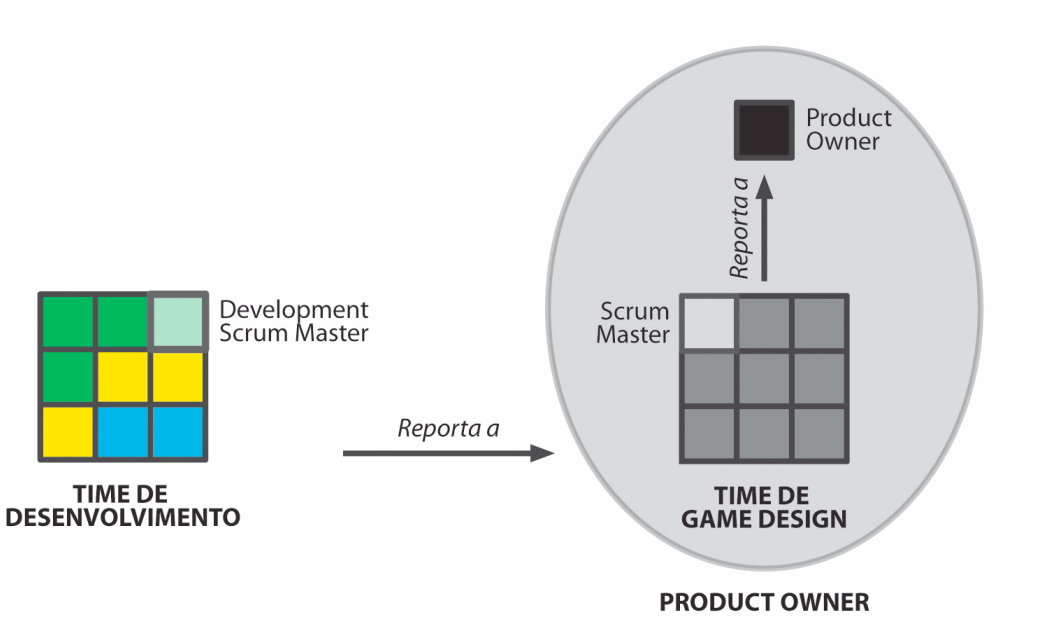

Figura 2 - Os dois times do *Scrum4Games*, com detalhamento do Time de Desenvolvimento e tendo o Time de *Game Design* como *Product Owner*

4.1.3 Time de Game Design

Como dito, uma abordagem interessante para o desenvolvimento do projeto de *Games* é a de se considerar o(s) *Game Designe*r(s) como “clientes”. Tal postura trará exatamente o posicionamento necessário para o bom andamento da metodologia, reforçando a vantagem da existência de dois times separados. Deve-se tratá-los como clientes presentes, que estão instruídos e treinados na maneira correta de documentar e passar informações relevantes ao projeto. São profissionais altamente associados ao valor do negócio e que compreendem o fato de que limitações tecnológicas podem influenciar na modelagem inicial do produto definida por eles. Possuem naturalmente a posição dos *stakeholder*s de “defesa das funcionalidades do sistema, dando grande valor à presença das mesmas, não importando a forma como foram desenvolvidas”.

É provável que o projeto exija apenas a presença de um *Game Design*er, como visto na Figura 3. Caso seja esta a opção, este deverá exercer o papel de *Development Product Owner* e principal *stakeholder* do projeto, para com o time de desenvolvimento. Nesse caso então, o processo será bastante semelhante ao *Scrum* tradicional. Bastará, porém, que o *Game Design*er esteja atento a todos os outros conceitos do *Scrum4Games* e aplique-os no seu projeto.

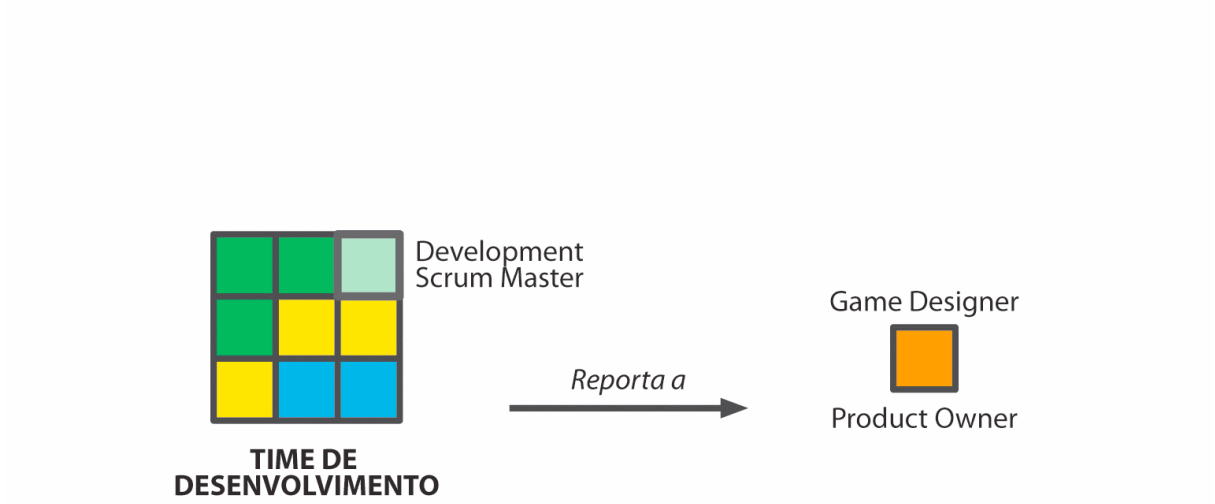

Figura 3 - *Scrum4Games* onde o *Game Design* é feito apenas por um indivíduo, tratado como *Development Product Owner*

No caso de haver uma equipe para o *Game Design* e não apenas um indivíduo, recomenda-se uma adaptação do "*Scrum* de *Scrums*": Enquanto o time de desenvolvimento possui seu *Development Scrum Master*, o time de *Game Design* deverá possuir também o seu *Game Design Scrum Master*. Este profissional terá um papel tão importante para o projeto quanto o *Development Scrum Master*. Ele será responsável por liderar a equipe de *Game Design*, garantindo que toda a construção do conceito do *Game* esteja de acordo com os objetivos do projeto. Esse profissional é o responsável direto pela qualidade total do *Game* final, portanto, caberá a ele a responsabilidade do ROI (*Return of Investment* ou Retorno do Investimento) proveniente do *Game*, após seu resultado de vendas e avaliação por parte do mercado.

Caberá também ao Time de *Game Design* eleger um representante para lidar com a equipe de desenvolvimento, exercendo a função de *Development Product Owner* para com a mesma. É muito interessante que tal representante seja o próprio *Game Design Scrum Master*, porém, não é obrigatório. Este assunto será melhor tratado na seção 4.2.1.

A respeito de sua formação, observa-se que o time de *Game Design* deve ser composto de membros com perfil técnico, representados pelos próprios *Game Design*ers, somados a alguns com perfil não-técnico. Estes últimos devem, principalmente, ser jogadores de diversos perfis convidados para participarem do projeto (representantes do público alvo do *game*), com o intuito de testar tanto a concepção quanto a implementação do *Game*. Todos os membros deverão contribuir, em suas responsabilidades específicas, para a construção do *Game Design*, garantindo assim a heterogeneidade de pontos de vista.

Caso a opção do projeto seja por um time para ter responsabilidade sobre o *Game Design*, a formação da equipe deve seguir os padrões do *Scrum*, possuindo entre cinco e nove participantes. Sugere-se que para uma equipe com cinco integrantes, dois deles sejam técnicos e os outros três, não-técnicos (referenciando o que foi dito anteriormente). No caso de uma equipe máxima, com nove integrantes, a sugestão passa a ser de quatro técnicos e cinco não técnicos.

É importante a presença de um maior número de membros não-técnicos (jogadores/*testers*) neste time, pois dessa forma haverá a garantia de que o *Game Design* não sofra um problema bastante comum no desenvolvimento de *Games*: a paixão de seus criadores sobre seus *Games*. Muitas vezes, os *Game Design*ers se motivam em demasia com as suas criações, o que acaba criando nos mesmos uma espécie de comportamento hostil a críticas e alterações. Com a presença de vários jogadores, poderão ser feitas opiniões e análises mais leves e descompromissadas, muito construtivas ao *Game* e ao projeto em si.

A respeito das regras de formação para o time de *Game Design*, sugerem-se as seguintes distribuições:

- 5 membros: 2 *Game Design*ers + 3 Jogadores Testers
- 6 membros: 3 *Game Design*ers + 3 Jogadores Testers
- 7 membros: 3 *Game Design*ers + 4 Jogadores Testers
- 8 membros: 4 *Game Design*ers + 4 Jogadores Testers
- 9 membros: 4 *Game Design*ers + 5 Jogadores Testers

Um último ponto relevante em relação ao time de *Game Design* diz respeito ao seu *Product Owner*. Uma vez que o produto do trabalho do time é o próprio *Game Design*, este não pode ser tratado novamente como cliente. Faz-se necessário então uma figura superior ao *Game Design*, em termos de análise de valor do projeto. Percebe-se então que esta figura é a representação do Produtor do *Game*, transportada para dentro do modelo *Scrum*. O produtor terá praticamente as mesmas atribuições de um produtor de um projeto de *Game* tradicional, como planejamento de marketing, prospecção de mercado, comunicação com canais de venda e afins. Graças a essas atribuições, o *Scrum4Games* o coloca em uma posição de alto valor agregado ao projeto: a avaliação do produto através do *Game Design*.

É importante notar que o *Game Design Product Owner* não ditará como o *Game* deve ser, mas sim que objetivos ele deve alcançar. E este profissional, por estar altamente associado à pesquisas de mercado e às necessidades do mesmo, poderá fazer uma avaliação junto à equipe de *Game Design* de quão próximos o time está de atingir o objetivo da empresa.

A Figura 4 mostra o novo passo de evolução do modelo. Agora, além de termos o time de desenvolvimento bem detalhado, temos também o time de *Game Design* na mesma situação: Os *Game Design*ers estão marcados em laranja, os jogadores/testers em marrom. O *Game Design Scrum Master* é mostrado em vermelho, e o *Game Design Product Owner* é agora a figura do Produtor, vista em preto.

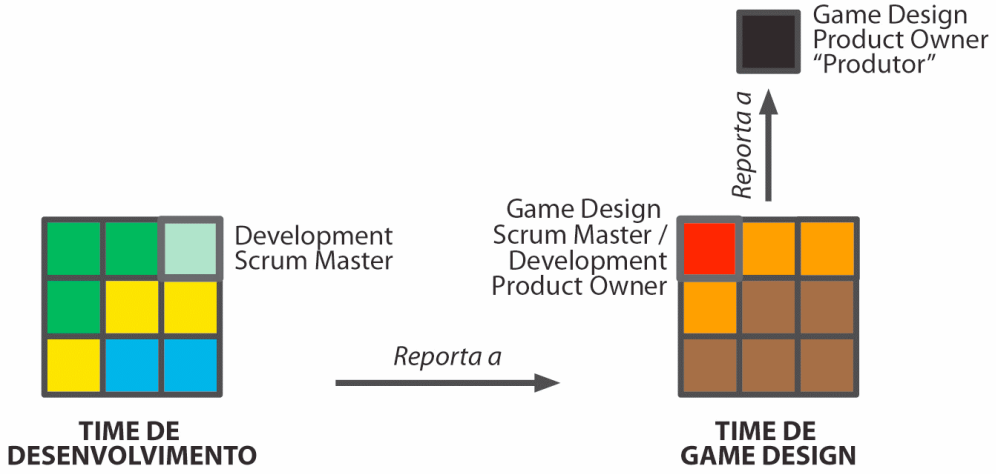

Figura 4 - Os dois times básicos do *Scrum4Games*,
com detalhamento em ambos os times

4.1.4 Observações Gerais

Após todas as considerações anteriores, já é possível formular o funcionamento básico do *Scrum4Games*. Porém, há alguns pontos relevantes percebidos durante os estudos de caso que merecem citações, como que um guia de "boas práticas" do *Scrum4Games*:

- Uma das funções principais de um *Scrum Master* é defesa da equipe. Ele não deve defender o projeto, o produto, funcionalidades, pretensões e expectativas para com o mesmo. Este é o papel do *Product Owner*. Desta forma, é altamente desaconselhável que o Development *Scrum Master* seja também o Development *Product Owner*, ou seja, o responsável pelo *Game Design* do projeto. Possuindo tal acúmulo de funções, ele pode tomar posturas ofensivas à equipe em prol do projeto ou produto, prejudicando o bom andamento do mesmo. Esta prática é parcialmente evitada com o modelo de dois times do *Scrum4Games*, porém em empresas com poucos funcionários é bastante comum tal pensamento ocorrer. Entretanto, caso seja imprescindível tal cenário de acumulo de funções, é essencial que o então *Scrum Master* recorra à consultoria de terceiros para que o seu *Game Design* seja criticado.

- Outra observação relevante diz à posição do *Scrum Master* dentro do quadro dirigente da empresa. É altamente desejável que ele não faça parte da equipe administrativa ou do quadro societário, tanto para que sua atenção não seja dividida em tarefas muito distintas e igualmente importantes (prejudicando assim o seu desempenho em ambas), como para que não assuma ocasionalmente uma posição hostil em relação ao time. Tal posição poderia ser tomada frente a uma avaliação de valor do projeto, ou atraso de produção. O *Scrum Master* deve ser exatamente o profissional que defende o time desse tipo de investida, e não o que as causa. Porém, caso seja imprescindível tal cenário de acumulo de funções, é essencial que o então *Scrum Master* deixe de lado sua posição hierárquica em tudo que tange ao projeto, seja no posicionamento administrativo quanto ao projeto ou mesmo na tomada de decisões executivas.

- No caso de desenvolvimentos externos de *Games*, o *Game Design Product Owner* deverá ser um representante do cliente e também o mais importante *stakeholder*. O desenvolvimento externo pode ser exemplificado com a contratação da empresa por parte de outra companhia, para desenvolver para a mesma um *advergame* (*Game* com o intuito de divulgar uma marca ou conceito). Um exemplo para esta situação é ilustrado na Figura 5.

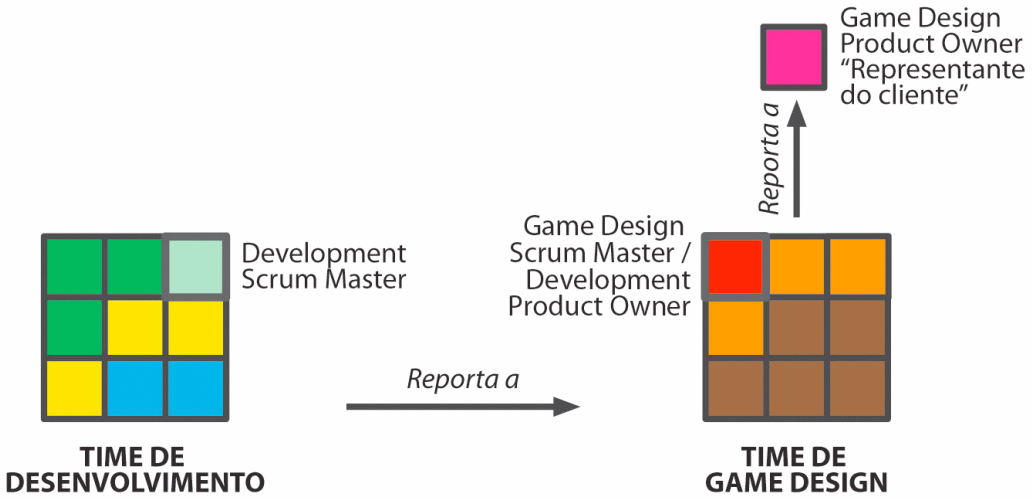

Figura 5 - Os dois times básicos do *Scrum4Games*, substituindo o *Game Design Product Owner* por um representante de um cliente.

4.2 Adições do *Scrum4Games*

Uma vez que toda as adaptações do *Scrum* para a criação do *Scrum4Games* foi finalizada, devemos agora nos voltar para os pontos onde é necessária alguma adição de conceitos e/ou métodos para a finalização da metodologia. Este é um ponto crucial, pois a estrutura de papéis/artefatos/cerimônias do *Scrum* foi desenvolvida de forma a ter o máximo de produtividade, com o mínimo de burocracia, visando principalmente a agilidade.

Após profunda análise, concluiu-se que a melhor prática seria incluir adições apenas nos pontos que gerassem alto valor agregado ao projeto e/ou solucionassem um (ou vários) problema(s) percebido(s) durante os estudos de caso. Limitou-se à regra de uma adição por classe para manter a simplicidade, que é a chave do *Scrum*.

4.2.1 Papel Adicional - *Game Design Master*

Conforme citado na seção 4.1.3, é extremamente interessante que o profissional responsável pelo time de *Game Design* (*Game Design Scrum Master*) seja o mesmo a fazer a ponte com a equipe de desenvolvimento (*Development Product Owner*). A união desses dois cargos deverá ser tratada como um novo cargo, e não como um acúmulo de funções. Tal

profissional, devido à natureza de seu papel, deve ser referido como *Game Design Master*. Tal mudança visa resolver problemas de comunicação e aumentar o valor agregado do projeto.

O *Game Design Master* será o responsável pelo valor agregado do projeto, internamente ao desenvolvimento. Com isso, reduz-se a burocracia de comunicação com o time de desenvolvimento, torna-se mais clara e rápida a comunicação interna do time de *Game Design*, elimina-se completamente o ruído de comunicação dos times com o *Game Design Product Owner*/Produtor. Este papel terá única e exclusivamente a função de garantir a concepção de um *Game Design* que atenda exatamente as demandas do Produtor, garantindo que as informações sejam passadas para o time de *Game Design* na forma de desejos do *Product Owner* e para o time de desenvolvimento na forma de necessidades do projeto. Como são todas realizadas pelo mesmo profissional, não há perda de informação no caminho.

A vantagem na escolha deste indivíduo híbrido é que a mesma posição de defesa e cobrança que ele terá como *Product Owner* em relação ao produto para com o time de desenvolvimento se refletirá como uma posição de proteção ao trabalho do seu próprio time (de *Game Design*), quando houver a defesa do conceito do *Game* para o executivo superior (Produtor). Isso o caracteriza exatamente como um *Scrum Master* para sua equipe. Porém, é muito importante lembrar que a escolha de um profissional deve atender às necessidades de perfil de ambos os papéis.

4.2.2 Artefato Adicional - *Game Design Wiki*

Como visto no capítulo 2, embora o *Scrum* defenda a diminuição de documentação e o impacto desta prática no aumento da agilidade, no desenvolvimento de *Games* esta é uma prática perigosa, conforme citado por MILLER (2008).

Da mesma forma que a equipe de desenvolvimento de um *Game* se difere da de um *software* comum pela diversidade dos profissionais que a compõe, a documentação de um *Game* se difere da usual pela diversidade de conteúdo. Um documento de *Game Design* não possui apenas as especificações do *software*, mas diversas informações relevantes que não podem ser representadas através das *user stories* ou mesmo dos *backlogs*.

Porém, é necessário perceber que, como dito no Capítulo 3, o *Game Design* está em constante modificação, motivo para o qual o *Scrum* é ideal para este tipo de projeto. Para isto, faz-se necessário uma ferramenta capaz de gerar uma documentação dinâmica e receptiva a

mudanças. Foram escolhidas então, para fazer parte do arcabouço do *Scrum4Games*, ferramentas de edição de textos livre e colaborativa, como por exemplo um *Wiki.*

Através delas, deverá ser criado o artefato *Game Design Wiki*. Este documento será de responsabilidade primária do *Game Design Master* e secundária dos membros do Time de *Game Design*. O *Game Design Master* deverá monitorar o *Wiki* procurando por alterações e tomando atitudes quando necessário. Seu papel quanto ao texto é análogo ao editor de uma revista: possui a noção do todo, supervisiona e gerencia a construção e edição da mesma. Ele também é responsável por atribuir aos membros do time de *Game Design* as tarefas de edição. Já o papel dos membros do time será análogo ao dos redatores de uma revista: tem responsabilidade sobre um conteúdo e devem transformá-lo em uma matéria na publicação

Outra vantagem do *Game Design Wiki* diz respeito à comunicação interna no projeto. Tanto membros do time de desenvolvimento como o *Game Design Product Owner*/Produtor devem acessá-lo periodicamente e, a cada ponto que não compreendam, incluir comentários explicando suas dúvidas para que elas possam ser tiradas pelo *Game Design Master*. É importante que comentários pessoais e opiniões não sejam feitos no *Wiki*, para evitar caos, mas sim que sejam feitos durante a *Weekly Game Design* Meeting, que será vista na seção a seguir.

O objetivo principal desta adição é a resolução do problema de documentação do projeto, muito comum em projetos passados que utilizaram *Scrum* para o desenvolvimento de seus *Games*.

4.2.3 Cerimônia Adicional - *Weekly Game Design Meeting*

Esta cerimônia tem como principal objetivo unir a motivação da equipe ao aumento do valor agregado do produto. Como dito anteriormente, os membros do time de desenvolvimento possuem grande interesse em participar do *Game Design*, com idéias, e se sentem frustrados quanto isso não ocorre.

A *Weekly Game Design* Meeting (WGDM), como seu próprio nome diz, ocorre semanalmente (quinta-feira mostrou-se um bom dia no caso estudado) e pode possuir duas formas: livre ou cadenciada.

Uma WDGM livre é aquela onde todos os participantes do projeto, independente do seu time ou papel, podem dar idéias, sugestões, críticas, comentários. É recomendado que seja

tratada como um *brainstorm* (discussão livre de idéias). O grupo participante deve eleger um mediador (qualquer um, não necessariamente um *Scrum Master* ou *Product Owner*). Discussões sobre as idéias levantadas são permitidas, porém caso se tornem inflamadas demais, devem ser interrompidas pelo mediador. O material gerado por estar reunião não deve ser anotado, e sim gravado, para não inibir a dinamicidade desejada. Caso seja gerado algum material gráfico, o mesmo deve ser guardado ou fotografado. Posteriormente, caberá ao time de *Game Design* a avaliação e aproveitamento deste material.

A WDGM cadenciada possui o mesmo objetivo da livre, porém se difere em seu funcionamento. Ao invés de ser tratada como um *brainstorm*, ela será como uma entrevista com cada membro interessado em participar dela. O seu mediador deverá ser obrigatoriamente o *Game Design Master* (ou um membro do time de *Game Design*, no caso de sua ausência). Cada membro presente na reunião irá expor todas as suas opiniões e afins, que serão anotadas (não gravadas) pelo mediador. Caso haja algum material extra que o participante tenha trazido, deverá ser entregue ao mediador. Tal procedimento é feito com todos os participantes, um a um. A vantagem desta reunião é que permite uma exposição mais detalhada da opinião de cada membro.

Um ponto crucial para evitar problemas provenientes da WGDM é manter nos seus participantes a consciência de hierarquia, não internamente ao projeto, mas com relação à empresa. É bastante comum haver confusões, por parte de funcionários, quando liberdade demais é fornecida à ele. Portanto, cabe ao mediador manter tal imagem, sem inibir.

Outro ponto importantíssimo é que a participação da reunião não é compulsória a nenhum membro. Eles devem se sentir livres para participar apenas quando acharem válido. Há muito mais valor em uma reunião com poucos participantes, mas que possuem interesse do que uma reunião com diversos desinteressados, presentes apenas por serem obrigados.

O *Game Design Master* deve saber balancear, de acordo com sua experiência, a execução de WDGMs livres ou cadenciadas, de acordo com as necessidades do projeto e/ou motivação da equipe.

4.2.4 Composição Completa da Equipe

Considerando todos os conceitos da proposta do *Scrum4Games*, a imagem abaixo representa a união dos papéis e times explicitados ao longo deste capítulo. Através dela, é possível ter uma noção total da corrente de comunicação do projeto gerenciado através do *Scrum4Games*.

Com isso, é importante concluir que o *Scrum4Games*, através da correção de problemas comuns e tratamentos de afinidades, mantém a simplicidade criando uma equipe voltada ao valor do *Game* com apenas 2 times (*Game Design* e Desenvolvimento) e 3 papéis de destaque muito bem definidos (*Development Scrum Master, Game Design Master* e *Produtor*).

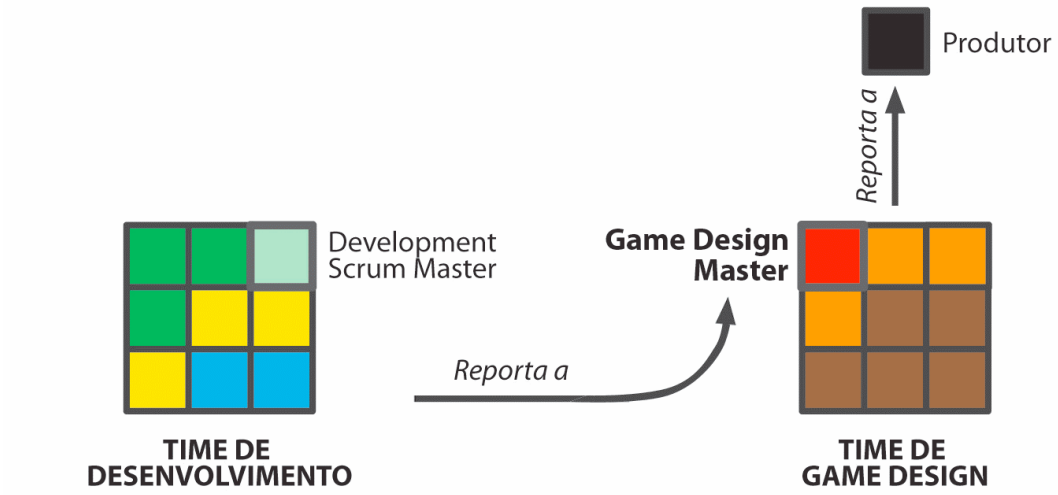

Figura 6 - A composição completa dos papeis do *Scrum4Games*

5. Considerações Finais

O mercado de desenvolvimento de *softwares*, cada vez mais, lida com projetos de alta complexidade. São diversas as particularidades que abrangem essa área, e um projeto com falhas na metodologia de gerenciamento poderá ter um resultado caótico, que não cumpre os requerimentos exigidos para tal, ou mesmo não consegue ao menos chegar a um fim. Devido à referida possibilidade de falhas, ao longo do tempo, foram desenvolvidos diversos modelos metodológicos que se destinam a suprir essa necessidade de gestão. O *Scrum,* abordado nesse trabalho, mostra-se, no campo de desenvolvimento de *software*, uma ferramenta bastante adequada, devido à sua agilidade e proposição para adaptações rápidas.

Tratando-se de uma área mais específica, e objeto direto da pesquisa desse trabalho, o desenvolvimento de *games* possui características únicas, que o diferenciam da produção de *softwares* em geral. O questionamento levantado foi até que ponto seria possível, viável e proveitoso estabelecer a aplicação da referida metodologia na produção desses produtos voltados ao entretenimento. E nesse cenário, verificou-se, através tanto de pesquisa bibliográfica quanto do estudo de caso realizado, que a aplicação "nua" da metodologia *Scrum* em projetos de desenvolvimento de *games* possui problemas em sua eficácia.

Para a realização desse projeto, procurou-se trabalhar em um estudo de caso na empresa Aiyra, que trabalha com o desenvolvimento de *games* – e da qual o autor deste trabalho é sócio-fundador. Como a empresa foi vencedora de um edital para desenvolvimento de um *game,* ela se constituiu em um cenário bastante adequado, já que criou a possibilidade de, a partir de revisão bibliográfica, observar, na prática, o estabelecimento do *Scrum* nesse projeto. Além disso, a presença do autor dentro do já mencionado projeto, enalteceu a capacidade de fazer modificações, experimentações e testes de funcionalidade. Por isso, pode-se dizer que a escolha por esse estudo de caso mostrou-se acertada.

A aplicação do *Scrum* em *games* – o *Scrum4Games* – mostrou-se vantajosa. No entanto, carecia de adaptações, à medida que possui peculiaridades, como funções distintas trabalhando integradas, assim como a presença do *Game Design* em sua construção. As proposições aqui abordadas procuram dar conta dessas questões. O resultado prático foi a identificação, observação e sugestão de atitudes corretivas a problemas encontrados tanto por

parte de autores referenciados quanto da observação empírica. Da mesma forma, o trabalho decorreu em uma sistematização e organização das diversas funções presentes na produção de um *game* de acordo com a lógica da metodologia *Scrum*.

O trabalho mostrou-se particularmente relevante para o autor da pesquisa, já que possibilitou a conjunção de suas pesquisas teóricas com a prática vivenciada na empresa onde trabalha. Essa união gerou grande benefício e riqueza de conhecimento para ambos os setores. Considera-se, por fim, que as proposições apontadas são relevantes para o tema proposto e o trabalho aqui apresentado cumpriu com seu objetivo de contribuir para a pesquisa já existente na área de adaptação do *Scrum* para o desenvolvimento de *games*.

Porém, é importante ter em mente que este trabalho não é definitivo, tão pouco a realidade do projeto exemplo possa ser tomado como padrão para toda a comunidade. Ainda é necessário que a metodologia adaptada do *Scrum4Games* seja aplicada na maior variedade possível de projetos, ambientes e equipes. Dessa forma, poderá verificar-se sua eficácia e eventuais falhas, tornando possível a adaptação e evolução da metodologia para um contexto mais geral.

Futuramente, pretende-se não apenas realizar esta aplicação em outros projetos da própria Aiyra, como também incentivar a comunidade, através da distribuição livre do conteúdo, utilizando como ferramentas palestras, *websites* colaborativos, eventos, programas de acompanhamento voluntário de projetos e etc. O objetivo deste trabalho é iniciar um trabalho colaborativo que contribua para melhorar as bases da indústria mundial de desenvolvimento de *games*, podendo assim conseqüentemente elevar a qualidade dos títulos produzidos.

6. Referências Bibliográficas

BAKER, S; POWER, G. - **Agile in Action: Being an effective Onsite Customer or *Product Owner*,** Disponível em: <http://www.think-box.co.uk/blog/2005/08/being-effective-onsite-customeror.html>. Acesso em 08 de fevereiro de 2009..

BAKER, S; POWER, G. - **Agile in Action: Your *Scrum* team needs you,** Disponível em: <http://www.think-box.co.uk/blog/2005/12/your-*Scrum*-team-needs-you.html>. Acesso em: 08 de fevereiro de 2009.

COHN, M. - **Leader of the Band: Six Attributes of a Good *ScrumMaster***, Disponível em: <http://www.*Scrum*alliance.org/articles/36-leader-of-the-band>. Acesso em: 22 de maio de 2009.

COHN, M. - **Sprint Planning: Better Together,** Disponível em: <http://www.*Scrum*alliance.org/articles/19-spring-planning-better-together>. Acesso em: 22 de maio de 2009.

COHN, M. - ***Scrum* for Video Game Development***,* 2007 – Disponível em http://sites.austin-cs.org/web/archive/*Scrum*-for-video-game-development, Acesso em 08 de fevereiro de 2009

CHAPIEWSKI, G. - ***Scrum Master* técnico?,** Disponível em: <http://gc.blog.br/2008/04/06/*ScrumMaster*-tecnico/>. Acesso em: 08 de fevereiro de 2009.

CHAPIEWSKI, Guilherme. - **Como estamos indo com a adoção de *Scrum* na Globo.com,** 2008. Disponível em: <http://gc.blog.br/2008/05/27/como-estamos-indo-com-a-adocao-de-*Scrum*-na-globocom/>. Acesso em: 15 de fevereiro 2009.

KNIBERG, H. - ***Scrum* and XP from the Trenches – How we do *Scrum*.** s.e. Estados Unidos: C4Media Inc, 2007

MILLER, P. - **Top 10 Pitfalls Using *Scrum* Methodology for Video Game Development,** 2008 – Acessado em 17 de janeiro de 2009, Disponível em : http://www.gamasutra.com/view/feature/3724/top_10_pitfalls_using_*Scrum*_.php

MOUNTAIN GOAT SOFTWARE. - **Learning *Scrum* - The *Product Backlog*,** Disponível em: <http://www.mountaingoat*software*.com/product-backlog>. Acesso em 20 de janeiro de 2009.

MOUNTAIN GOAT SOFTWARE. - **Learning *Scrum* - The *Sprint* Review Meeting,** Disponível em: <http://www.mountaingoat*software*.com/*Sprint*-review-meeting>. Acesso em 20 de janeiro de 2009.

MOUNTAIN GOAT SOFTWARE. - ***Sprint Planning Meeting*****,** Disponível em: < http://www.mountaingoat*software*.com/*Sprint*-planning-meeting>. Acesso em 20 de janeiro de 2009.

MOUNTAIN GOAT SOFTWARE. - **The *Scrum* Team,** Disponível em: <http://www.mountaingoat*software*.com/*Scrum*-team>. Acesso em 20 de janeiro de 2009.

JAMES, M. et al. - **Who is on the *Scrum* Team?,** Disponível em: <http://danube.com/blog/michaeljames/who_is_on_the_*Scrum*_team>. Acesso em: 13 maio de 2009.

SILVA, W. ; LEMOS, L**.** - ***SCRUM*****: Uma nova abordagem no desenvolvimento de *Software* frente à demanda competitiva do cenário atual,** - Monografia (Bacharelado em Ciência da Computação) – Universidade Federal Fluminense, Niterói, Rio de Janeiro. 2008.

SCHUYTEMA, Paul. - **Design de Games: Uma Abordagem Prática,** Editora Cengage Learning: São Paulo, 2008.

SCHWABER, Ken. - **Agile Project Management with *Scrum*****,** Microsoft Press, 2004.