CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais

SCRUM APLICADO AO DESENVOLVIMENTO DE JOGOS DIGITAIS

BRUNO FIRMINO DA SILVA

Americana, SP
2010

CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais

SCRUM APLICADO AO DESENVOLVIMENTO DE JOGOS DIGITAIS

BRUNO FIRMINO DA SILVA

brufds@gmail.com

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais, sob a orientação do Prof. Me. Carlos H. Rodrigues Sarro.

Área: Jogos Digitais

Americana, SP
2010

FICHA CATALOGRÁFICA elaborada pela
BIBLIOTECA – FATEC Americana – CEETPS

Silva, Bruno Firmino da
S578s Scrum aplicado ao desenvolvimento de jogos digitais. / Bruno Firmino da Silva. – Americana: 2010.
55f.

Monografia (Graduação em Análise de Sistemas e Tecnologia da Informação). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Me. Carlos H. Rodrigues Sarro

1.Scrum – software de projetos I. Sarro, Carlos H. Rodrigues II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.3.077

Bibliotecária responsável Ana Valquiria Niaradi – CRB-8 região 6203

BANCA EXAMINADORA

Prof. Me. Carlos H. Rodrigues Sarro (Orientador)

Prof.ª Dra. Mariana Godoy Vazquez Miano

Prof. João Sebastião de Oliveira Bueno

AGRADECIMENTOS

Primeiramente, gostaria de agradecer a meus pais, Carlos e Maria, meu irmão Marcelo, que com muito amor e carinho me auxiliaram nessa jornada. Sempre acreditando em mim e no meu potencial. Por todas as horas de conversas e desabafos, com palavras suaves para me animar. A tudo sou muito grato.

Também agradeço meu orientador Carlos Sarro por toda a atenção e colaboração no desenrolar desse trabalho. E ao professor Cleberson Forte pela prontidão e disponibilidade em ajudar sempre.

Aos meus amigos e colegas de classe, meus sinceros agradecimentos também. Por toda a luta nesses anos de estudo, juntos, no companheirismo.

E por fim, agradeço ao Ser supremo: Deus. Grato por me permitir ser o que sou hoje, por todas as dádivas e conquistas.

A todos, meu muito obrigado!

DEDICATÓRIA

Dedico este trabalho à minha grande fonte de motivação,
minha família.

“Stop starting, start finishing!”

Sterling Mortensen

RESUMO

O presente texto visa explorar as faces do desenvolvimento de *software* amparado pelas práticas do processo *Scrum.* Com uma abordagem ampla e ao mesmo tempo simplista, o estudo contido nessa pesquisa traz os aspectos primários no desenvolvimento de *software*, expondo as primeiras metodologias utilizadas desde os primeiros programas de computador escritos, comparando-as com as que foram inseridas recentemente nessa área de pesquisa e trabalho. Por fim, aplica-se o *Scrum* a um exemplo de projeto real (um jogo para celular).

Palavras Chave: *software*, desenvolvimento, metodologias.

ABSTRACT

This paper aims to explore the facets of software development methodologies supported by the practices of Scrum. With a broad approach, while simplistic, the study contained in this research provides the primary aspects in software development, exposing the first methods used since the first computer programs written, and comparing them with those recently entered this area of research and work. Finally, applying Scrum to an example of real project (a mobile game).

Keywords: software, development, methodologies.

SUMÁRIO

1 O *SOFTWARE* E AS METODOLOGIAS DE DESENVOLVIMENTO 10

1.1 *SOFTWARE* – UM BREVE HISTÓRICO 10
1.2 AS METODOLOGIAS DE DESENVOLVIMENTO DE *SOFTWARE* 13
1.3 O MODELO CLÁSSICO 13
1.4 *RATIONAL UNIFIED PROCESS* (RUP) 15
1.5 PROTOTIPAÇÃO 16
1.6 MODELO INCREMENTAL 17
1.7 MODELO ESPIRAL 17
1.8 MÉTODOS ÁGEIS DE DESENVOLVIMENTO DE *SOFTWARE* 18
1.8.1 *EXTREME PROGRAMMING* (XP) E *SCRUM* 19

2 CONHECENDO O *SCRUM* 21

2.1 PRINCIPAIS PAPÉIS NO *SCRUM* 22
2.2 PRODUCT BACKLOG 23
2.3 USER STORIES 24
2.4 SPRINT E A *SPRINT PLANNING* 25
2.5 *DAILY MEETINGS* 27
2.6 *BURNDOWN CHART* 27
2.7 DEMO MEETING 28
2.8 *RETROSPECTIVE MEETING* 29

3 O DESENVOLVIMENTO DE JOGOS DIGITAIS 30

3.1 OS JOGOS E UMA BREVE HISTÓRIA 30
3.2 A CONCEPÇÃO DE UM JOGO 32
3.3 O MERCADO: CRISES E NECESSIDADES 34
3.4 METODOLOGIAS ÁGEIS COMO ALTERNATIVAS 38
3.5 ESTUDO DE CASO: APLICANDO O *SCRUM* EM UM PROJETO 40
3.6 CARACTERÍSTICAS DO JOGO E REQUISITOS BÁSICOS 41
3.7 CRIANDO O *PRODUCT BACKLOG* E FORMANDO A EQUIPE 45
3.8 A *SPRINT PLANNING* E AS PRÓXIMAS FASES 47

4 CONSIDERAÇÕES FINAIS 49

5 REFERÊNCIAS BIBLIOGRÁFICAS 52

LISTA DE FIGURAS E DE TABELAS

Figura 1: Esquema com as fases do Modelo Clássico de Desenvolvimento. 14

Figura 2: As fases do RUP e suas iterações 16

Figura 3: Esquema da prototipação. 17

Figura 4: Modelo Espiral de Desenvolvimento 18

Figura 5: Momento em que os jogadores de *Rugby* se reúnem para a disputa de bola, geralmente na cobrança de penalidades e faltas. No desenvolvimento de *software*, o *Scrum* foi adotado de modo a fazer alusão à reunião da equipe para atingir determinado objetivo 20

Figura 6: Ilustração que demonstra o processo *Scrum*. A partir de um *product backlog* gerado e mantido pelo *product owner*, é definido o *sprint backlog* 26

Figura 7: Gráfico de *burndown*, mostrando a linha de horas gastas no desenvolvimento de um projeto qualquer.....28

Figura 8: Jogador interagindo com o *Kinect*, para *Xbox 360*, apenas com os movimentos do corpo 32

Figura 9: Esquema das etapas na concepção de um jogo 34

Figura 10: Gráfico do mercado de jogos 36

Figura 11: Gráfico da demanda por pessoas para a produção de jogos 36

Figura 12: Quadro Kanban, com as atividades e colunas que descrevem o status atual das mesmas 40

Figura 13: Imagens que ilustram as telas do jogo *Snake*.....42

Figura 14: Tela que demonstra o campo de movimentação da personagem.....43

Figura 15: Demonstração de transposição na tela. 44

Figura 16: Tela do MS Project que mostra as atividades do sprint, estimativas e recursos.....48

INTRODUÇÃO

Este trabalho foi estruturado em três capítulos, sendo que o primeiro conceitua a concepção do *software* como produto da informática e seu histórico, passando pelas principais metodologias de desenvolvimento que o sustentam. Apresenta os principais métodos que surgiram e fazem parte da linha do tempo do desenvolvimento de *software*, suas principais características e valores. Além disso, aborda as metodologias ágeis de forma geral, ressaltando os detalhes que as destacam das demais.

O segundo capítulo traz os detalhes do *Scrum*, seus ritos e ideais; quando o processo surgiu e por quem foi criado, quais são seus focos, exemplificando cada parte do processo detalhadamente. Assim, abre espaço para o terceiro capítulo, que aplica esses conceitos em um exemplo de projeto simples, porém real.

O objetivo central da pesquisa é mostrar quão aplicáveis são as metodologias ágeis, em especial o *framework Scrum* a projetos de desenvolvimento de *software* direcionados aos jogos digitais. O quarto e último capítulo se reserva a expor as melhorias que o *Scrum* pode trazer aos projetos de jogos e destacar os pontos positivos da pesquisa.

1 O *SOFTWARE* E AS METODOLOGIAS DE DESENVOLVIMENTO

1.1 *SOFTWARE* – UM BREVE HISTÓRICO

Os *softwares* ganharam destaque no mercado de tecnologia no momento em que deixaram de ser simples interfaces meramente funcionais e unidirecionais. O que se viu no decorrer dos anos foi uma grande evolução no mercado de tecnologia, gerando grande diversidade de sistemas dedicados às mais variadas funcionalidades, esses que passaram a executar tarefas em conjunto e em tempo real. Com esse avanço, passa-se a personalizar os programas de computador a fim de obter maior ganho em desempenho e com os olhos direcionados aos próprios e reais objetivos dos suários. Foi nesse período em que o *hardware* deixou de ser o ponto chave nas decisões corporativas da indústria de tecnologia, tornando-se essencial, porém não diferencial. Entretanto, quando se fala em *softwares*, pensa-se em produtos essenciais e com características que os tornam diferenciais em usabilidade, desempenho, segurança, confiabilidade etc.

Durante muito tempo os *softwares* foram vistos como objetos secundários e tema de discussões futuras, principalmente no período que chamamos de “primeira era” da informação. No início, quando a maior parte das aplicações funcionavam sob arquiteturas mais robustas (sistemas *batch*), era muito comum haver apenas uma pessoa que fizesse o trabalho de escrever o código, mantendo-o atualizado conforme surgiam novas necessidades, o que se intitulou futuramente de “manutenção de *software*”. Mas esse quadro foi revertido quando se observou a aparição de tendências multiusuário e interdisciplinares nos sistemas. Novos conceitos surgiram e o *software* já despontava para ampla distribuição no mercado [PRESSMAN, R. S., 1985:6].

Em meados da década de 80, muitas revistas e jornais especializados em negócios já publicavam artigos e notícias relacionadas ao futuro dos *softwares* e seu impacto nas relações comerciais. A sociedade já não era apenas industrial, e concretizava-se então a “era da informação”, segundo Naisbitt (1985). Nessa época de descobertas já se discutia muito sobre a importância, as oportunidades e os riscos que os *softwares* trariam para as atividades administrativas. Feigenbaum e McCorduck (1983) já previam que a troca de conhecimento e informações através dos computadores e redes seria o grande foco do século XXI.

> Conhecimento é poder, e o computador é um amplificador desse poder... A indústria americana de computadores tem sido inovadora, vital, bem-sucedida. Ela é, de certa forma, a indústria ideal. Ela cria valor ao transformar o poder cerebral de trabalhadores que detêm conhecimentos, com pouco consumo de energia e de matérias-primas. Hoje [1983], dominamos as ideias e os mercados mundiais nessa que é a mais importante de todas as tecnologias modernas. Mas o que acontecerá amanhã? (FEIGENBAUM, E. e MCCORDUCK, P., 1983:1)

Conforme a demanda por soluções de *software* crescia em paralelo à popularização dos microcomputadores, surgiam também mais estabelecimentos dedicados exclusivamente a esse nicho de mercado. As *software houses* nasciam como grandes promessas lucrativas. E realmente fizeram jus às intenções de seus empreendedores, pois o computador pessoal foi, em âmbito mundial, o grande propulsor e catalisador das empresas de *software* no decorrer dos anos [PRESSMAN, R. S., 1985:8].

Assim como qualquer outro produto, os *softwares* sempre estiveram propensos a falhas. Ao passo que a indústria de *software* crescia, alguns problemas vieram a ser identificados. Muitos estabelecimentos tinham seus próprios projetos internos de desenvolvimento, produzindo dezenas de instruções de programa. Além disso, outros sistemas eram comprados do exterior acrescentando mais funcionalidades, bem como linhas de código extras. Quando apresentadas falhas nessas instruções, mudanças de *hardware* ou alterações de requisitos, os

programadores ficavam encarregados de fazer essas modificações e/ou correções no código fonte do sistema. Porém, percebeu-se que o esforço necessário para se trabalhar nessas manutenções atingia níveis alarmantes. Em alguns casos, pela natural falta de estrutura de alguns desses programas, a manutenção era praticamente impossível e consequentemente inviável, dando origem àquela conhecida como “crise de *software*”. Cabe lembrar também que muitos dos programadores não possuíam sequer técnicas programáticas, e a grande maioria trabalhava sob o sistema de tentativa e erro. [PRESSMAN, R. S., 1985:8].

Ao passo que essas questões e ideias eram colocadas em pauta, pesquisadores passaram a dedicar parte de seu tempo em estudos sobre boas práticas e métodos que pudessem auxiliar a indústria tecnológica a melhorar a qualidade de *software*, reduzir custos e aprimorar o processo de produção de sistemas. Essa foi a premissa para o surgimento de metodologias de desenvolvimento provenientes de técnicas de engenharia de *software* [PRESSMAN, R. S., 1985:10].

Estabelecendo um paralelo entre os sistemas de computador e outros bens de consumo conhecidos por nós, pode-se perceber que há certa distinção quando observados do ponto de vista de produto. Tanto os sistemas de computador quanto os outros bens referidos são consumíveis e frutos do trabalho de pessoas, mas com abordagens completamente diferentes. Em ambos os casos é necessário que existam profissionais executando procedimentos e seguindo processos para que, em determinando momento, surja um artefato palpável (ou não, no caso dos *softwares* como elementos lógicos). No caso dos *softwares*, o custo está estritamente concentrado na engenharia, portanto não podem ser geridos como projetos de manufatura. Para que tudo isso funcione de forma síncrona e em sintonia, é de

extrema importância que se aplique determinados modelos de controle no desenvolvimento desses produtos da tecnologia, são as já conhecidas metodologias de desenvolvimento de *software*.

1.2 AS METODOLOGIAS DE DESENVOLVIMENTO DE *SOFTWARE*

A engenharia de *software* coletou informações relevantes da indústria durante um bom tempo até construir os métodos de desenvolvimento que são conhecidos hoje. Com essas informações em mãos, foi possível discutir formas de controlar o ciclo de vida dos sistemas, desde a concepção dos requisitos até a entrega final. São premissas e modelos que embasam o desenrolar de um projeto nesse âmbito, impactando diretamente na qualidade do produto final.

> Engenharia de *Software* é o estudo dos princípios e sua aplicação no desenvolvimento e manutenção de sistemas de *software*. Assim, tanto a engenharia de *software* como as técnicas estruturadas são coleções de metodologias de *software* e ferramentas. (MARTIN, J. e MCCLURE, C., 1985:23).

No período em que havia a dificuldade de se alocar recursos para manutenção e até mesmo escassez tecnológica, os primeiros *softwares* foram desenvolvidos com base em planejamento e documentação prévia. Essa forma de trabalho é conhecida como Metodologia Clássica.

1.3 O MODELO CLÁSSICO

O modelo clássico foi o pioneiro no processo de desenvolvimento. Constituiu-se com foco no processo sistemático do ciclo de vida de um *software* qualquer, composto por fases em etapas sequenciais (análise, projeto, codificação, testes e manutenção). [MARTIN, J. e MCCLURE, C., 1991:20]

As etapas desse formato são explicitamente requeridas e ligadas a documentos dependentes de aprovação superior. [CASTRO, I. C. A. e MOREIRA, A. M., 2010:2]

Para que um projeto possa caminhar sobre a linha de desenvolvimento clássica, saltando para uma próxima etapa, é necessário que a anterior esteja devidamente completa e de acordo com as diretrizes do processo.

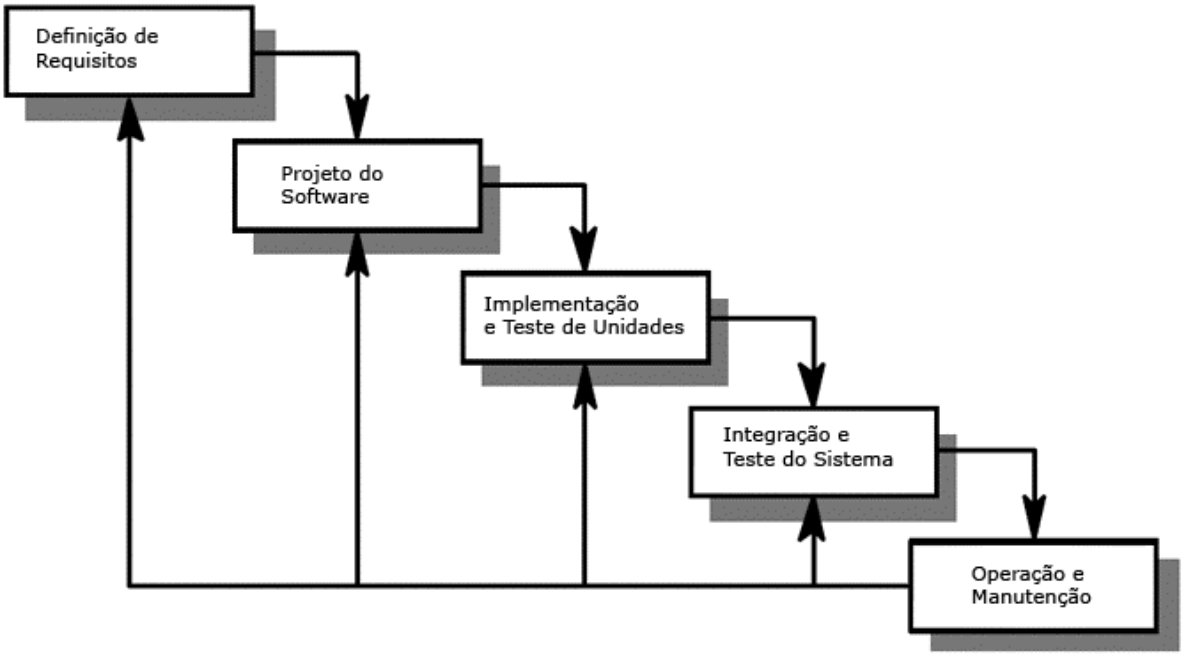

Figura 1: Esquema com as fases do Modelo Clássico de Desenvolvimento.

Apesar das lacunas, o modelo clássico (também conhecido como *Modelo em Cascata*) perdura por muito tempo dentro das empresas. Porém, quando se faz uma análise mais profunda de seus ideais, observa-se que algumas de suas características não condizem com a realidade esperada. Documentações extensas são o melhor exemplo a se destacar. Estima-se um tempo consideravelmente grande ao gerar artefatos e documentos em cada etapa do processo. Esse tempo, em teoria, gera custo e, na maioria das vezes, toda a documentação gerada não agrega o devido valor ao negócio do cliente.

Outro grande e notável problema no modelo clássico é a interdependência entre as etapas do desenvolvimento, causadora da impossibilidade de criação de fases intermediárias ou revisão de fases anteriores antes da finalização do processo. [CASTRO, I. C. A. e MOREIRA, A. M., 2010:3]

1.4 *RATIONAL UNIFIED PROCESS* (RUP)

O denominado RUP (*Rational Unified Process*, desenvolvido pela empresa *Rational Software Corporation*, adquirida pela IBM), é hoje o modelo mais utilizado pelas empresas. Surgiu como uma extensão e adaptação das ideias do Modelo Clássico. Apadrinhado por diversas indústrias especializadas, o então considerado *framework* de desenvolvimento, trouxe uma série de boas práticas vinculadas à qualidade como fruto da experiência das mesmas. Foram estudadas algumas necessidades e melhorias nos processos, e, diante disso, algumas características em destaque moldaram o RUP. Veja abaixo as principais:

1. Cliente interagindo no desenvolvimento do *software*;
2. Requisitos gerenciáveis;
3. Gerenciamento de mudanças;
4. Artefatos visuais (UML);
5. Gerenciamento da qualidade;
6. Necessidade de metodologia de desenvolvimento como base estrutural.

Mesmo com certa maleabilidade, o RUP ainda pertence à categoria das metodologias rigorosas de desenvolvimento e faz uso de documentação robusta. A primeira versão do RUP (5.0) concretizou-se em 1998, como uma evolução de outras tentativas (*Rational Objectory Process*).

O RUP é uma metodologia baseada em fases, desde a modelagem de negócio até o ambiente da aplicação.

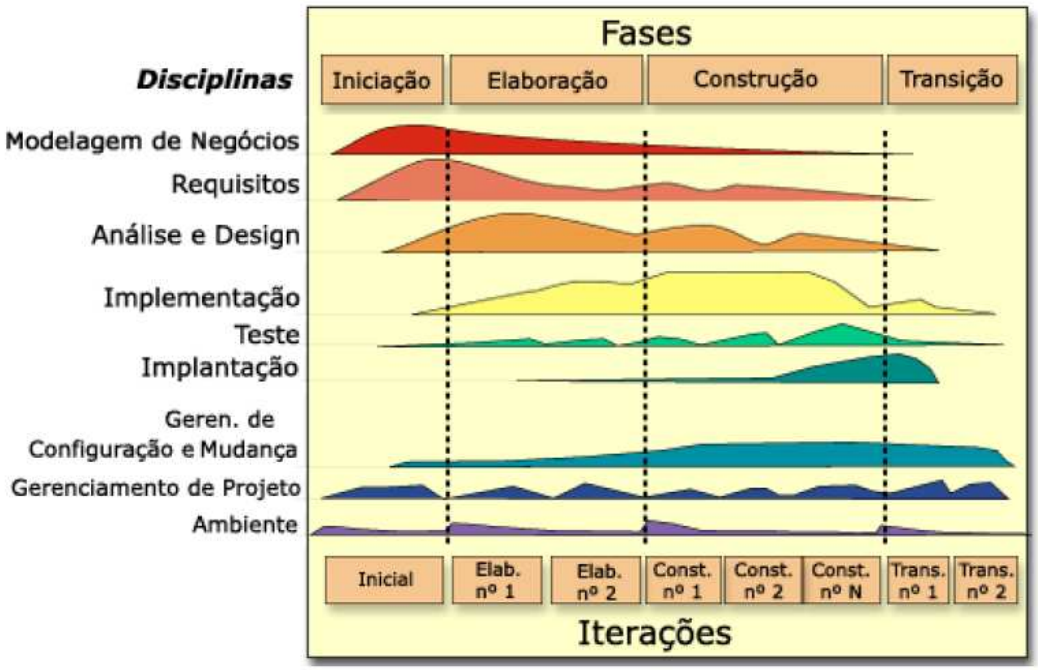

Figura 2: As fases do RUP e suas iterações. (Fonte: wiki.sj.cefet.edu.br)

A *Rational Software Coporation*, interessada em findar sua metodologia no mercado, criou também ferramentas de suporte ao desenvolvimento. Um bom exemplo é a UML (*Unified Modeling Language*), que é uma linguagem criada exclusivamente para apoiar o uso do RUP. [CASTRO, I. C. A. e MOREIRA, A. M., 2010:3]

1.5 PROTOTIPAÇÃO

Antes de tratar do processo mais utilizado hoje em dia, cabe comentar sobre outros ciclos de vida reconhecidos pela engenharia de *software*. Dentre eles, tem-se a Prototipação, que consiste no processo de desenvolver um *software* (ou parte do *software*) como objeto de testes. Esse protótipo tem por objetivo coletar informações

precisas sobre as intenções do cliente sobre o *software*. Geralmente, o tempo e o custo que se despendem no desenvolvimento de um protótipo é compensado pelo não desenvolvimento de características desnecessárias ou de forma errônea. [GUSTAFSON, D., 2002:13]

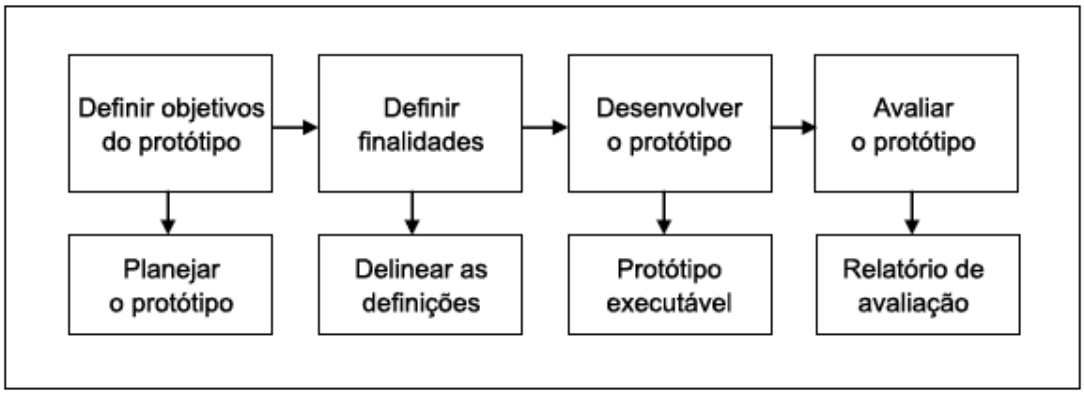

Figura 3: Esquema da prototipação.

1.6 MODELO INCREMENTAL

O Modelo Incremental, criado por D.L. Parnas, visa observar o *software* como módulos funcionais. Assim, uma parte do *software* em condições claras de uso pode ser entregue ao cliente assim que finalizada. Usuários deparam-se com diversos sistemas e funcionalidades exploradas esporadicamente. É possível separar o que é necessário do que não é requerido, porém necessário. Com esse princípio, o Modelo Incremental busca apresentar “pedaços” úteis do *software* e posteriores versões de trabalho. [GUSTAFSON, D., 2002:13]

1.7 MODELO ESPIRAL

Bohem introduziu o Modelo Espiral na engenharia de *software* como alternativa aos ideais tradicionais. Um tanto quanto inovador, o modelo enraizou-se

nas iterações contínuas. A imagem do espiral, que começa pequeno ao centro e vai se expandindo continuamente, representa o crescimento do *software* juntamente às atividades do usuário (comunicação, planejamento, análise de risco, engenharia, construção e implantação, e avaliação do usuário). [GUSTAFSON, D., 2002:13]

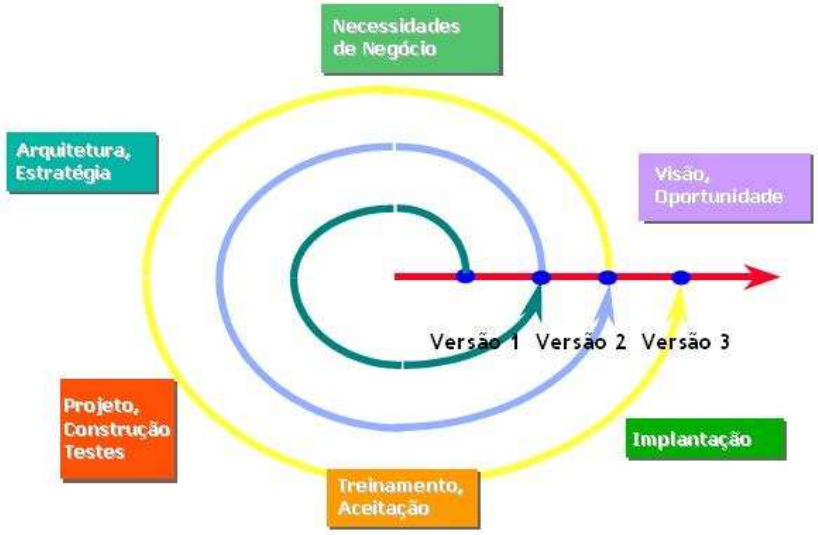

Figura 4: Modelo Espiral de Desenvolvimento. (Fonte: MJV.com.br)

1.8 MÉTODOS ÁGEIS DE DESENVOLVIMENTO DE *SOFTWARE*

O que se vê hoje é a ascensão das metodologias ágeis de desenvolvimento. Estas se baseiam no bom funcionamento do *software*, na colaboração, no valor secundário das documentações, na multidisciplinaridade de times e na comunicação. Nesse tipo de metodologia, normalmente não há fidelidade às ferramentas *CASE* e diagramas UML. O grande foco é nas coisas prontas e entrega de valor. São mais adaptativas e orientadas às pessoas ao invés de processos. [PAPO, J. P., 2010:15]

Projetos sob metodologias ágeis costumam ter retorno rápido. Num curto período de tempo, é possível ter parte do sistema completamente desenvolvida e com a qualidade que é esperada pelo cliente. Porém, observou-se que, para que realmente funcionem, as metodologias ágeis exigem equipes trabalhando em alta

sintonia e empresas que estejam de fato dispostas a mudanças culturais e organizacionais.

Outra característica marcante nas metodologias ágeis, e que difere um pouco das outras, é o fato do cliente ser um membro da equipe de forma literal. Sua participação em todos os processos é essencial. Ele é o responsável pelo negócio e o único que poderá responder pelas características do produto a ser desenvolvido. Sua presença em todas as etapas do desenvolvimento é de suma importância.

1.8.1 EXTREME PROGRAMMING (XP) E SCRUM

Extreme Programming (XP) e o *Scrum* são os dois processos mais usados quando se trata de desenvolvimento ágil. O XP teve início na década de 90, auxiliando pequenas e médias equipes na produção de *software* com pouca ou vaga especificação de requisitos e que sofriam constantes mudanças. Nessa metodologia, no quesito prático, o que mais se destaca é a programação por pares, na qual dois programadores trabalham em conjunto para desenvolver um único código. Um deles faz o papel de par revisor e ajudante, analisando o código escrito a fim de premeditar *bugs* e ajudar o programador a formular toda a lógica programática necessária para comportar as regras de negócio envolvidas em determinado módulo. Outro detalhe importante do XP é a busca pela integração contínua de código, que ocorre ao final de cada dia de desenvolvimento com o objetivo de garantir que o código integrado e testado esteja de acordo com o que era esperado.

Assim como prega o Manifesto Ágil[1], o XP segue o princípio da participação do cliente no decorrer de um projeto. O conceito *On Site Costumer* (cliente presente com a equipe) é aplicado de modo que as questões envolvendo o sistema, seus

[1] Declaração de princípios que fundamentam o desenvolvimento ágil de *software*.

fluxos de trabalho e outras dúvidas que possam surgir em tempo de codificação sejam sanadas imediatamente. Quando o cliente trabalha em parceria com a equipe desenvolvedora, o impacto negativo do tempo de espera por soluções de dúvidas, que acontece comumente nas metodologias clássicas, é eliminado, garantindo assim os prazos contratuais. De acordo com o *Standish CHAOS Report* de 1998, o sucesso de um projeto está embasado no envolvimento do usuário, suporte dos executivos e objetivos de negócio claros. [PAPO, J. P., 2010:20]

O *Scrum* é outro método de desenvolvimento oriundo do Manifesto Ágil. Com suas raízes fincadas nas entregas e no aumento da produtividade, o *Scrum* é muito semelhante ao *Extreme Programming*, pois aborda o desenvolvimento de *software* como uma prática totalmente colaborativa e suscetível às mudanças. Para fins de curiosidade, o termo *Scrum é* original do esporte *Rugby*, que representa o posicionamento dos jogadores em momentos decisivos da partida, reunidos em busca de um único objetivo (a união de um time).

Figura 5: Momento em que os jogadores de *Rugby* se reúnem para a disputa de bola, geralmente na cobrança de penalidades e faltas. No desenvolvimento de *software*, o *Scrum* foi adotado de modo a fazer alusão à reunião da equipe para atingir determinado objetivo. (Fonte: cena de *Invictus,* 2009)

O *Scrum* apenas define os pontos de partida para o inicio de um projeto. Não é comum encontrar guias “do que” levar um projeto com base nesse *framework*, mas

sim “como” tomá-lo como base (esses e outros paradigmas serão esclarecidos nos próximos capítulos). São essas premissas que formam os pilares do *Scrum*, que deixa de ser uma metodologia e passa a ser um conjunto de boas práticas para estimular o gerenciamento criativo de um projeto.

O presente estudo visa colaborar com a indústria de jogos digitais, estudando de forma teórica e prática a abordagem das metodologias ágeis nesse ramo de atividade. Como parâmetro, foi escolhido o *Scrum* como metodologia de desenvolvimento aplicada e serão estudadas com mais afinco as suas faces, descrevendo seus ritos, levantando pontos positivos e negativos de seu uso e sua representatividade no mercado em projetos desse âmbito.

2 CONHECENDO O *SCRUM*

O *Scrum* foi desenvolvido por Ken Schwaber, Mike Beedle, Jeff Sutherland, entre outros colaboradores, na década de 90. O grupo é formado, em grande parte, por profissionais da informática e detentores de um conhecimento extenso na área de desenvolvimento de *software*.

Através de suas experiências anteriores de sucessos e fracassos, decidiram estudar uma nova maneira de produzir *software* como um objeto artesanal e com foco nas pessoas envolvidas, que pudesse ser moldado de acordo com as soluções discutidas pela equipe e adaptável às mudanças durante sua concepção, sem toda a problemática já carregada durante anos por metodologias obsoletas.

O *Scrum* sustenta-se nos quatro pilares do Manifesto Ágil, que são:

- **Indivíduos e interação entre eles,** mais que processos e ferramentas;
- ***Software* em funcionamento,** mais que documentação abrangente;

- **Colaboração do cliente,** mais que negociação de contratos;
- **Responder às mudanças,** mais que seguir um plano.

Para Ken Schwaber, cofundador dos conceitos do *Scrum*, ele não é uma metodologia, e sim um *framework*, o que deixa muito claro o fato do *Scrum* não ditar regras, não fazer a imposição de atitudes ou dizer exatamente o que fazer para controlar e gerenciar um projeto. No *Scrum*, todos são responsáveis por adaptar o processo a uma situação específica e ao modo do próprio grupo. [KNIBERG, H., 2007:16]

Mostra-se um processo empírico, ou seja, parte do princípio de que nem todas as variáveis do produto são conhecidas e provavelmente sofrerão mutações no decorrer do desenvolvimento. [CRUZ, L. R. S., 2010:1]

Como um processo de desenvolvimento iterativo e incremental, o *Scrum* pode ser aplicado a qualquer projeto. Não há nada que o restrinja a sistemas de informação. É escalável, podendo ser aplicado em projetos de diferentes complexidades e riscos.

Para entender um pouco melhor as etapas do *Scrum* e do que ele se constitui, serão descritas as suas práticas a seguir.

2.1 PRINCIPAIS PAPÉIS NO *SCRUM*

Antes de conhecer as práticas do processo, é importante deixar claro quais são os principais papéis que ele contempla e suas responsabilidades:

- ***Product Owner*:** como o próprio nome já sugere, é como se fosse o dono do produto. Faz parte da equipe de negócios do cliente e é responsável pelo *product backlog*. O *product owner* é parte da equipe

de desenvolvimento também, e participa das reuniões que encabeçam o processo;

- ***Scrum Master*:** é o membro da equipe que fará as práticas do *Scrum* serem seguidas da forma correta. Como um evangelizador, trabalha a favor da equipe atuando como um facilitador, eliminando bloqueios, mediando entregas e outras burocracias do projeto.
- **Time:** são os membros da equipe que estarão diretamente ligados ao desenvolvimento do projeto, codificando, testando ou executando quaisquer atividades relacionadas à concepção do produto final.

2.2 PRODUCT BACKLOG

> O *product backlog* é o coração do *Scrum*. É aqui que tudo começa. O *product backlog* é basicamente uma lista de requisitos, estórias, coisas que o cliente deseja, descritas utilizando a terminologia do cliente. (KNIBERG, Henrik, 2007:19).

O *product backlog* é considerado o ponto inicial do *Scrum*. Conforme aponta Ken Schwaber, o *product backlog* é a prática responsável pela coleta dos requisitos de um sistema computacional (ou do projeto, esteja ele em qualquer área de atuação). Nessa fase, ocorrem reuniões com os *stakeholders*[2] interessados no projeto, para que sejam levantadas as necessidades de negócio do cliente e requisitos técnicos a serem desenvolvidos. De forma geral, é possível descrever o *product backlog* como uma lista de atividades (contidas nas *user stories*) a serem trabalhadas no decorrer do projeto.

[2] Termo proveniente do PMBoK. Pessoas e organizações envolvidas no projeto, direta ou indiretamente.

2.3 USER STORIES

Utiliza-se o termo *user story* (estória) para referenciar uma funcionalidade do sistema. Grosso modo, é a maneira como o cliente vê a funcionalidade que ele deseja encontrar no produto. Analogamente, quando um telefone celular é adquirido, por exemplo, logo se pensa em fazer ligações com o mesmo. De forma eloquente, pode-se explicar tal fato*: "como usuário do celular, eu gostaria de digitar números no teclado para ligar para meus amigos e poder conversar"*. As estórias são essas sentenças que descrevem o desejo do usuário. Veja alguns exemplos a seguir:

*"Como um **usuário comum**, eu **posso clicar sobre o link Home para retornar à página inicial do site**."*

*"Como um **administrador**, eu **posso acessar a área administrativa do site e revogar o acesso de determinado usuário**."*

*"Como **jogador**, eu **gostaria de apertar um botão que execute uma sequência de golpes no meu adversário para que ele perca pontos de vida**."*

Através dessas sentenças, a equipe de desenvolvimento é capaz de extrair e desenvolver uma sequência de atividades necessárias para que, no final, o usuário possa executar de fato o que deseja.

Boas *user stories* seguem um padrão ao serem criadas. Veja a seguir as informações necessárias:

- **ID:** um campo de identificação, geralmente auto-incremento. Apenas para fins de organização;

- **Nome:** um nome curto e que descreva a estória. Exemplo: “revogar acesso de usuário”. Esse nome deve ser claro o suficiente para que haja entendimento da equipe ao lê-lo;
- **Importância:** toda estória deve receber uma pontuação por importância para o *product owner*, contabilizada em *story points*[3]. Evitamos o termo prioridade;
- **Estimativa inicial:** quanto tempo acredita-se precisar para programar a estória, quando comparada às outras estórias. Observação: normalmente, os pontos da estória determinam a quantidade de homem/dias para se executar uma tarefa. Mas isso nem sempre é válido. [KNIBERG, H., 2007:20].

2.4 SPRINT E A *SPRINT PLANNING*

No *Scrum*, os períodos de iteração são conhecidos como *sprints*. De acordo com os princípios do processo, todo *sprint* deve preencher um *timebox*[4] de duração entre uma e quatro semanas. Caso haja a necessidade de se estender o tempo de desenvolvimento do *sprint* durante o processo de programação, algo já indica que as atividades e estórias foram subestimadas ou mal detalhadas. Os *sprints* incluem as fases tradicionais do desenvolvimento de *software*: requisitos, análise, projeto e entrega [ZANATTA, A. L. e VILAIN, P., 2010:3].

É válido lembrar também que todos os projetos possuem um período para definição do time, entendimento do projeto, conhecimento dos requisitos macro e obtenção das ferramentas necessárias para o desenvolvimento. Esse período é comumente chamado de *sprint zero*.

[3] *Story points* são contados na sequência de Fibonacci. Nada mais são que números utilizados para determinar o grau de importância de uma funcionalidade.
[4] Período de tempo mensurável.

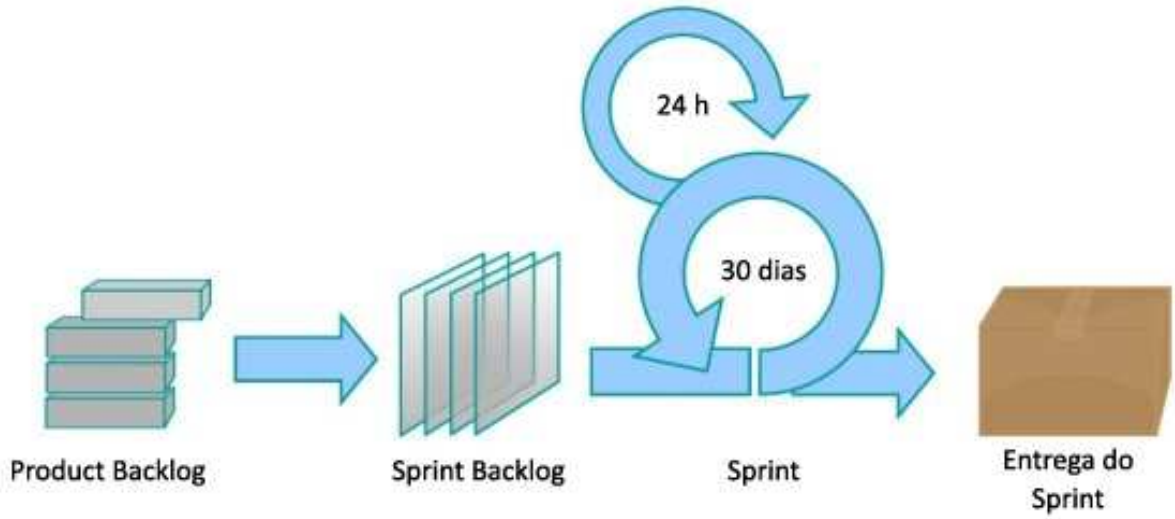

Figura 6: Ilustração que demonstra o processo *Scrum*. A partir de um *product backlog* gerado e mantido pelo *product owner*, é definido o *sprint backlog*. (Adaptado de: *Coderkitty*)

Antes da reunião de planejamento do *sprint* (*sprint planning*) é preciso ter o *product backlog* devidamente organizado e fechado. É imprescindível que se tenha o escopo de desenvolvimento bem claro. Isso não significa que todas as estórias estejam bem definidas, que todas as estimativas estejam corretas ou que as prioridades estejam fixadas. Significa apenas que um *product backlog* existe e pertence a apenas um *product owner*. [KNIBERG, H., 2007:23]

Tanto a equipe quanto o *product owner* devem participar da reunião de planejamento do *sprint*. Dessa reunião sairão informações relevantes, como o objetivo do *sprint*, o *sprint backlog* (lista de estórias inclusas para o *sprint*), data da apresentação do *sprint* e local/data das *daily meetings* (reuniões diárias).

Geralmente, quem inicia uma reunião de planejamento do *sprint* é o *product owner*, destacando os objetivos e as estórias mais importantes do *sprint*. Em seguida, a equipe de desenvolvimento trabalha nas estimativas das atividades.

2.5 *DAILY MEETINGS*

Uma prática obrigatória no *Scrum* são as *daily meetings* (reuniões diárias). Elas se iniciam quando o *sprint* já começou a ser trabalhado. Tem duração de quinze minutos e surgiram no processo como a maneira mais eficaz de manter todos os membros da equipe interados sobre o andamento do projeto. Nas reuniões diárias, três perguntas são respondidas por cada indivíduo de forma objetiva: o que fiz ontem, o que farei hoje e o que está me impedindo de continuar minhas atividades.

Essas reuniões são importantíssimas para a maturidade dos projetos *Scrum.* Elas são os canais para se formar o conhecimento do projeto como um todo. Assim, todos podem ter visibilidade das atividades, dificuldades e problemas que o time está enfrentando no desenvolvimento do trabalho.

2.6 *BURNDOWN CHART*

Os *sprints* começam com uma quantidade de horas de desenvolvimento em razão das estimativas obtidas na *sprint planning*. Conforme os membros da equipe trabalham, as horas e *story points* de desenvolvimento vão caindo, de acordo com a quantidade de tempo utilizada para completar as atividades. Através desses dados entre tempo estimado, total de horas do *sprint* e tempo gasto, gera-se um gráfico denominado *burndown chart*. Com esse gráfico é possível obter métricas muito importantes no final de cada *sprint*, como a *velocity*[5] do time por exemplo.

[5] A *velocity* nada mais é que a medida da capacidade do time, ou seja, *n* membros conseguem executar *n* tarefas por dia, correspondente a *n story points.* Essa métrica auxilia o *scrum master* no planejamento dos próximos *sprints*, que obtém agora o conhecimento da capacidade do time.

Figura 7: Gráfico de *burndown*, mostrando a linha de horas gastas no desenvolvimento de um projeto qualquer.

2.7 DEMO MEETING

No final de cada *sprint*, ocorre a *demo meeting*, que nada mais é que uma reunião com o *product owner* e outros *stakeholders* para que a equipe possa demonstrar na prática o que foi desenvolvido. No caso de um jogo, por exemplo, se no *sprint* atual foi desenvolvido um nível novo, com cenários e personagens diferentes, o que será mostrado na *demo meeting* são esses personagens novos interagindo com os cenários no novo nível criado. Essa reunião existe para que o cliente possa ver o resultado atingido com o trabalho, e garantir que todas suas expectativas foram alcançadas.

Nas *demo meetings* costumam aparecer muitas dúvidas e mudanças por parte do *business*. O que ocorre, normalmente, é uma listagem de todas essas solicitações pela equipe, para que possam ser analisadas e desenvolvidas no *sprint* seguinte.

2.8 *RETROSPECTIVE MEETING*

Após a reunião de demonstração, é agendada mais uma reunião chamada *retrospective meeting* (reunião de retrospectiva), na qual todos os membros da equipe se reúnem para discutir e levantar os pontos positivos e negativos encontrados durante o desenvolvimento do *sprint* que está acabando. A ideia dessa reunião é incitar melhorias no projeto.

Vale ressaltar que, para um projeto se enquadrar exatamente no processo *Scrum*, ele deve seguir esse formato básico. Se alguma empresa modifica o processo para adequá-lo à sua realidade corporativa, os conceitos do processo já se tornam dúbios, e temos o que chamamos de *ScrumBUT*[6]*.* Segundo Pedro Mariano, autor do artigo “Metodologias: Às vezes adaptações são necessárias”, a adaptação é natural e muitas vezes necessária, dependendo da necessidade da equipe e do projeto em si. Tanto o *Scrum* como o *Crystal*[7] ou o *Extreme Programming* não foram idealizados com o objetivo de serem intocáveis. Com amadurecimento, podemos sim remanejá-los de acordo com nossas ideias, a fim de atingir resultados melhores com a utilização do processo ou metodologia.

Sabemos que um projeto de *software* não se sustenta somente com uma metodologia ou processo. Existe uma série de outros fatores a serem estudados para que se possa conhecer de forma clara os objetivos que desejamos alcançar com a concepção de um produto. No caso de *game development,* por exemplo, não é válido conhecer somente as práticas *Scrum* que serão aplicadas ao desenvolvimento de um novo jogo, mas sim toda a sistemática que o envolve, desde

[6] Modificar o processo para que ele se enquadre exatamente à realidade do time/produção.
[7] Família de métodos (*Clear*, *Yellow*, *Orange* e *Red*) que podem ser ajustados aos projetos para melhor atendê-los.

a ideia central até sua divulgação. Portanto, veremos, com mais propriedades e de forma macro, como é o processo comum no desenvolvimento de jogos no mercado atual, para depois estudarmos a aplicação do processo *Scrum* nesse âmbito.

3 O DESENVOLVIMENTO DE JOGOS DIGITAIS

3.1 OS JOGOS E UMA BREVE HISTÓRIA

Os primeiros jogos produzidos para os primeiros computadores funcionavam de acordo com o *hardware* dos dispositivos. E quase todos eles se limitavam ao modo *single player*[8], devido à escassez de recursos na época. Tal fato explica a falta de equipes de desenvolvimento nos primórdios. Um único programador era capaz de se dedicar sozinho e conceber um jogo em poucos meses ou mesmo semanas, visto que nada requeria artistas (jogos como o clássico *Pong*, que era monocromático), roteiristas, engenheiros de áudio, entre outros profissionais que estão, atualmente, envolvidos nesse mercado.

Apesar de tal facilidade, não é correto subestimar a complexidade estrutural de um jogo, que, por mais simples que seja, pode abranger cálculos matemáticos, conceitos da física e exigir um conhecimento de programação intermediário ou avançado.

O primeiro jogo de computador a surgir foi o *Tennis For Two*, criado no *Brookhaven National Laboratory* (EUA) por William A. Higinbotham, no ano de 1958. Nada mais era que uma pequena bola que saltava de um campo a outro na tela de um osciloscópio. Mais tarde, apareceu no MIT (*Massachussets Institute of Technology*) o primeiro jogo de guerra interestelar conhecido como *Space War*.

[8] Apenas um jogador por vez interage com o jogo.

Funcionando num dos computadores mais potentes para a época, o *PDP-1*, o jogo veio com o objetivo de testar as capacidades do mesmo. Não demorou muito para que fizesse sucesso no meio acadêmico. [BARROS, R. L. B., 2007:13]

A produção de jogos seguiu a todo vapor desde então. Na década de 80, a *Nintendo* se lançou no mercado com força total, apresentando o primeiro *NES* (*video game* de 8 *bits*). Mesmo com a concorrência de grandes outros nomes como a *SEGA*, *Master System* e a gigante *Atari*, a *Nintendo* conseguiu se destacar quando o assunto era consoles. No final da década, também apresentou o *GameBoy*, console portátil revolucionário. [VELASQUEZ, C. E. L. 2009:31]

Conforme os anos foram passando e a tecnologia caminhando a passos largos, novos equipamentos de *hardware* surgiram, como processadores de alto desempenho, placas de vídeo 3D, dispositivos de áudio extremamente potentes, entre outros periféricos exclusivos à imersão (capacetes com visores LCD, óculos 3D, luvas de toque etc). Com esse avanço, os programadores se sentiram motivados a produzir mais, explorando intensamente os recursos de *hardware* que lhes foram disponibilizados.

Hoje, são três os principais nomes de consoles no mercado mundial: o *Playstation 3*, *Xbox 360* e o *Nintendo Wii*. Alguns desses já possuem inclusive dispositivos capazes de interagir com os jogadores sem quaisquer meios físicos de controle (*Playstation Move* e o *Project Kinect*, do *Xbox 360*).

Figura 8: Jogador interagindo com o *Kinect*, para *Xbox 360*, apenas com os movimentos do corpo. (Fonte: *Microsoft*)

3.2 A CONCEPÇÃO DE UM JOGO

O *software* de um jogo possui características comuns aos sistemas com os quais lidamos no dia-a-dia, programaticamente falando. Porém, obtém um ciclo de vida peculiar e que deve ser tratado de forma diferenciada. Como já mencionado, uma equipe de desenvolvimento de jogos deve, em sua grande maioria, ser multidisciplinar. Isso implica na contratação de recursos com competências distintas, visto que há demanda para diferentes áreas de conhecimento. O processo de criação do produto requer profissionais capacitados e especializados, pois um jogo é um grande e complexo projeto de *software*. [VELASQUEZ, C. E. L., 2009:30]

Antes de dar o primeiro passo na produção de um jogo, é necessário que se conheça o conceito do que ele é. A própria palavra, que vem do latim *iocus*, define brinquedo, passatempo e recreação. Tudo o que é relacionado ao lúdico (do latim

ludus) e que traz entretenimento pode ser considerado uma forma de jogo. [HAMZE, A., 2010]

Os jogos nada mais são que histórias de ficção interativa a serem contadas, nas quais o jogador atua como parte ativa da mesma, interferindo ou não no enredo. Todos os jogos podem ser avaliados pelos seus níveis de interatividade, pois quanto mais interativo, mais intensa a experiência do jogador.

Pensar em desenvolvimento se inicia na concepção do *Game Design Document* (documento do processo de desenvolvimento do jogo). Nesse artefato estão contidas as informações do projeto, como a descrição do jogo, detalhes de programação, seus objetivos no mercado, *sketches* etc. Há uma semelhança aos clássicos documentos de requisitos que são confeccionados nos *softwares* comerciais comuns.

O *Game Design Document* é o ponto chave num projeto dessa alçada, pois tudo se inicia nele, que durante todo o processo será o guia dos envolvidos no projeto. Mas é válido salientar que antes da criação do *Game Design Document* é necessário um estudo para a concepção do jogo, bem como ter criatividade para produzir uma história inovadora e cativante, pois ela será a base para o sucesso do produto. [VELASQUEZ, C. E. L., 2009:36]

Com as informações iniciais em mãos, a empresa responsável pelo projeto analisa se o mesmo será bem aceito no mercado, gerando assim seu conceito de jogo. E, a partir daí, dá-se início à fase de produção do jogo, que se subdivide em pré-produção, produção e pós-produção. [BARROS, R. L. B., 2007:24]

Veja, a seguir, um esquema que demonstra as etapas para a criação de jogos digitais, seguindo uma cadeia hierárquica com base na engenharia de *software*:

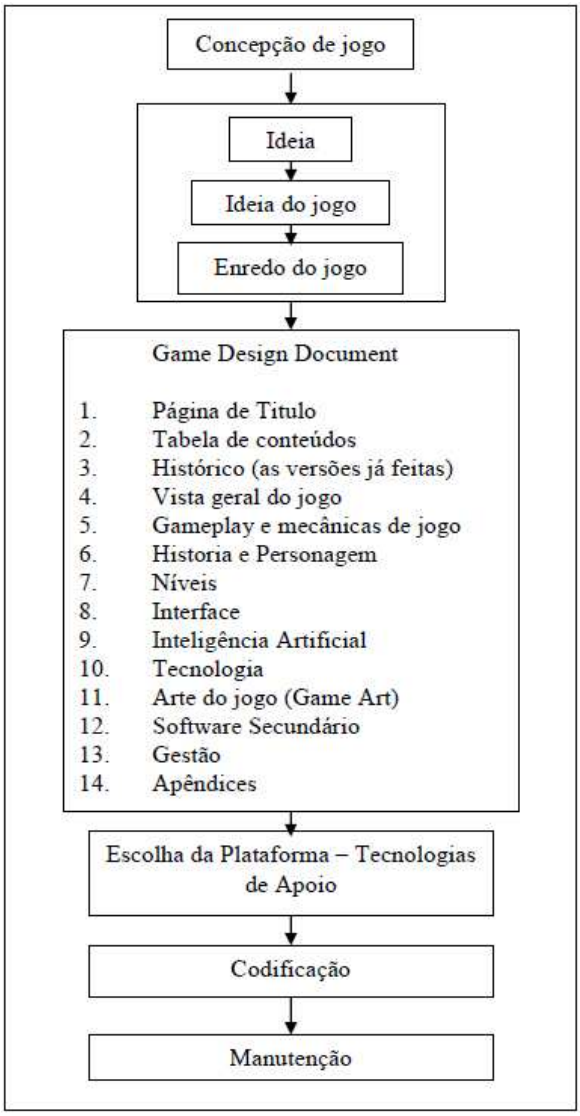

Figura 9: Esquema das etapas na concepção de um jogo. (Fonte: VELASQUEZ)

3.3 O MERCADO: CRISES E NECESSIDADES

Os jogos digitais representaram, em seu início, um nicho de mercado altamente lucrativo. Muitos aventureiros da informática viram uma nova oportunidade para ganhar milhares de dólares com a produção de jogos, que representava um retorno de investimento considerável para a época. Um novo e ascendente mercado se movia rapidamente em direção ao sucesso e arrastava consigo os desafios de se desenvolver *hardwares* cada vez mais potentes.

Assim surgiu um novo conceito de desenvolvimento, conhecido como *Hit-or-Miss* (erro ou acerto). Muitos desses desenvolvedores aventureiros desenvolviam dezenas de jogos simples, a fim de cativar um público e obter lucro. Se um deles emplacasse no mercado, pagaria todos os custos dos outros casos de insucesso. Porém, a porcentagem de projetos que obtiveram lucro nessa empreitada foi muito pequena. Recursos precários e escassez de capital de investimento para criar e manter uma ideia inovadora resultava no insucesso do jogo no mercado.

Ainda assim, segundo estudos da *Electronic Entertainment Design Research*, as vendas dos jogos eletrônicos cresceram em média 10% ao ano, de 1998 a 2009. Um número favorável aos investidores e profissionais da área. Mas o que se obervou foi um déficit de profissionais capacitados no mercado, bem como ferramentas necessárias para esse trabalho. Isso gerou um problema ainda mais agravante: equipes pequenas, *software* complexo e certeza de alto custo de desenvolvimento. [KEITH, C., 2010:7]

Os custos de desenvolvimento de jogos cresceram proporcionalmente ao desenvolvimento das tecnologias e à expansão das equipes. O custo dos jogos também aumentou cerca de 25%, o que fez com que o modelo *Hit-or-Miss* fosse descontinuado. Veja os gráficos a seguir que demonstram essa proporção:

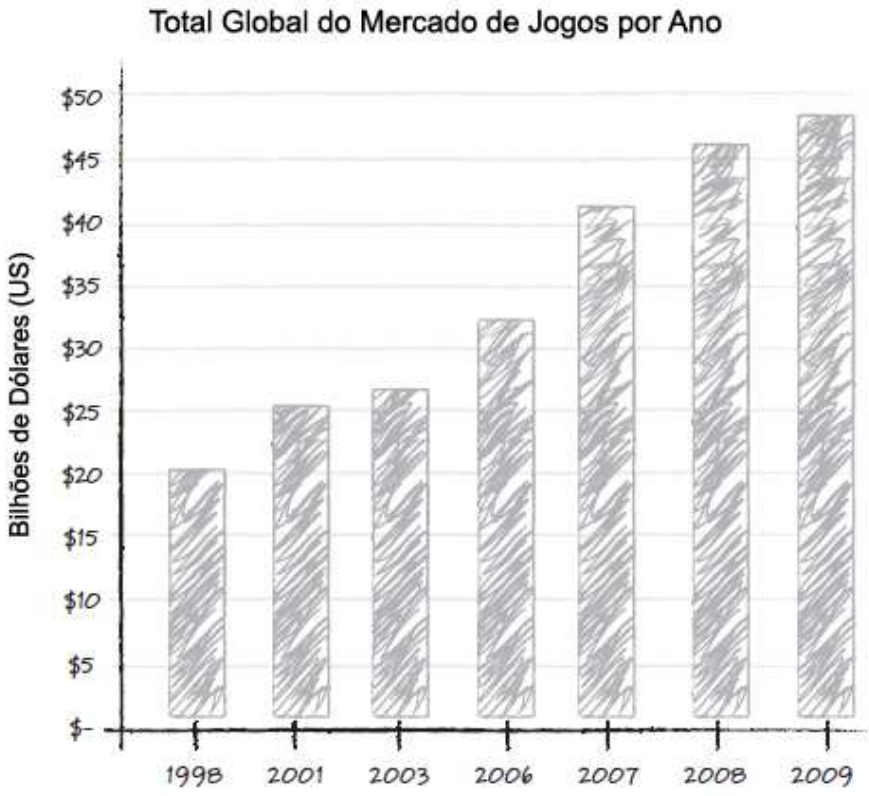

Figura 10: Gráfico do mercado de jogos. (Fonte: KEITH)

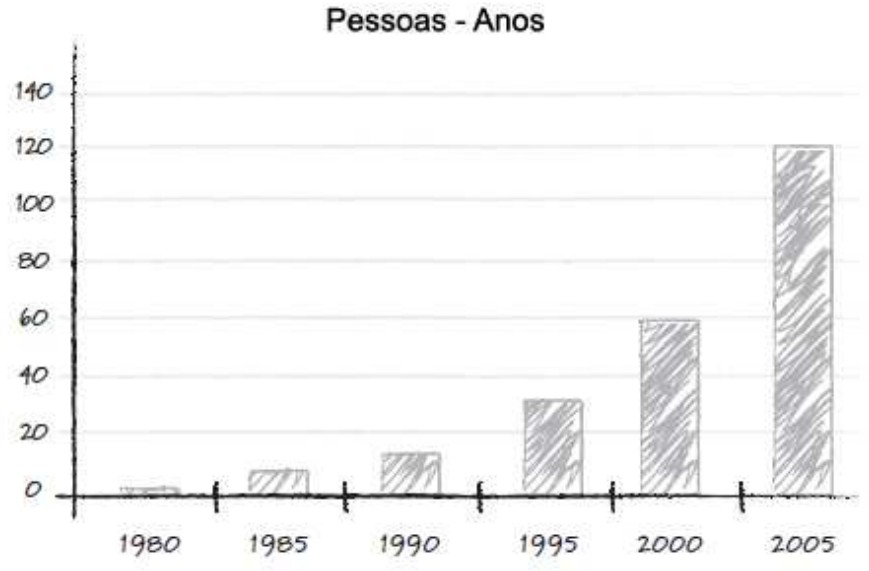

Figura 11: Gráfico da demanda por pessoas para a produção de jogos. (Fonte: KEITH)

Observou-se, com o crescimento do mercado, que boa parte dos jogos não são mais rentáveis o suficiente para manter equipes com cerca de cem

desenvolvedores e custos excedendo os milhões de dólares. Esse problema vem sendo impulsionado por três principais fatores:

- **Falta de inovação:** não se inova mais nos jogos. Os usuários já estão esgotados de ver jogos com histórias parecidas, personagens semelhantes e finais iguais. Não é possível criar *hits*[9] sempre, porém, a indústria dos jogos baseia-se na inovação e a ela deve fazer jus;
- **Redução de custo:** a redução de custos levou à baixa qualidade e à redução de conteúdo dos jogos. Consequentemente, o tempo de jogo também diminuiu, o que leva os consumidores a pesarem o custo-benefício;
- **Ambiente de trabalho hostil:** muitos desenvolvedores deixam seus empregos devido às horas excessivas de trabalho, que são, às vezes, não condizentes com seus salários. Isso afeta diretamente a indústria. De acordo com o IGDA (*International Game Development Association*), 34% dos desenvolvedores esperam deixar a indústria em cinco anos e 51% em dez anos. Tornou-se inviável elaborar estratégias de desenvolvimento profissional, liderança e novos métodos de gerenciamento de projetos. [KEITH, C., 2010:10]

No Brasil, de acordo com dados estatísticos de 2008 obtidos através de uma pesquisa da ABRAGames (Associação Brasileira de *Games*), a indústria nacional de jogos eletrônicos é responsável por 0,16% do faturamento mundial nessa área e emprega cerca de 560 profissionais, distribuídos entre programadores e artistas gráficos.

[9] Chamamos de *hit* algo que se destaca dentre coisas semelhantes.

Observa-se claramente números significativos e que denotam o crescimento do mercado brasileiro nesse âmbito. Segundo as pesquisas da ABRAGames, há um prospecto de que a produção de jogos no Brasil cresça ainda mais nos próximos anos. E ao conhecer um pouco dos problemas e crises que esse mercado enfrenta no exterior, será possível moldar nosso território para minimizarmos as dificuldades.

Algumas universidades já estão investindo em cursos especializados para essa área (como as FATEC, SENAC e a Unisinos), visando suprir a carência de profissionais capacitados, o que já vem a ser um ponto extremamente positivo. O FINEP (Financiadora de Estudos e Projetos) também tem colaborado com empresas novas e que atuam nessa vertente da tecnologia.

Não é possível enxergar um desgaste na indústria de jogos nacional, pois essa se encontra em sua infância plena e ainda tem muito por caminhar. Mas quando analisamos a face norte-americana, de acordo com os dados já mencionados, fica clara a necessidade de uma reformulação na forma de trabalho. A seguir, será demonstrada então uma abordagem de como o *Scrum*, como processo de desenvolvimento de *software*, pode ser parte positiva na grande busca pelo cenário ideal na indústria de entretenimento eletrônico, colaborando ativamente para o sucesso de novos projetos.

3.4 METODOLOGIAS ÁGEIS COMO ALTERNATIVAS

Os primeiros projetos de jogos não utilizavam nenhum tipo de metodologia de desenvolvimento de *software* como base, pois todos eles eram feitos de forma independente e muitas vezes não lucrativa. É o caso do jogo *Space War*, mencionado anteriormente, que foi desenvolvido apenas para fins de estudo.

Posteriormente, quando os jogos entraram de fato no meio computacional, agregando complexidade de desenvolvimento e ganhando espaço no mercado tecnológico, percebeu-se a necessidade de se adaptar os projetos correntes aos modelos de desenvolvimento que a engenharia de *software* havia proposto. Muitos dos jogos produzidos desde a década de 80 foram trabalhados sob o Modelo Clássico, também conhecido como cascata. A minimização de riscos em projetos que apresentavam prospectos de alta complexidade, com equipes cada vez maiores e de difícil gerenciamento, foram os principais motivos a levar as companhias à adoção dessa metodologia.

As metodologias clássicas, apesar de efetivas, possuem pontos que se chocam com a indústria de jogos. Conhecemos toda a parte burocrática do desenvolvimento de um *software*, desde a coleta de requisitos até sua liberação para produção. Hoje, há uma busca pela mudança nos hábitos empresariais, visando processos mais lúdicos e que possam gerar prazer aos trabalhadores. O *Scrum* possui esse lado descontraído e que pode auxiliar a indústria a produzir mais e fazer com que os profissionais encarem o trabalho com mais diversão e dedicação. [KEITH, C., 2010:7]

Parte das equipes que trabalham hoje com o *Scrum* usam métodos visuais e extremamente eficazes para controle dos projetos. O mais conhecido é o *Kanban*, que consiste num quadro branco em que são colados papéis adesivos com as atividades que compõem o desenvolvimento. À medida que o *status* da atividade é alterado (*para fazer*, *fazendo* e *feito*), os papéis adesivos são movidos entre as colunas. A Figura 12 ilustra um exemplo real de *Kanban.*

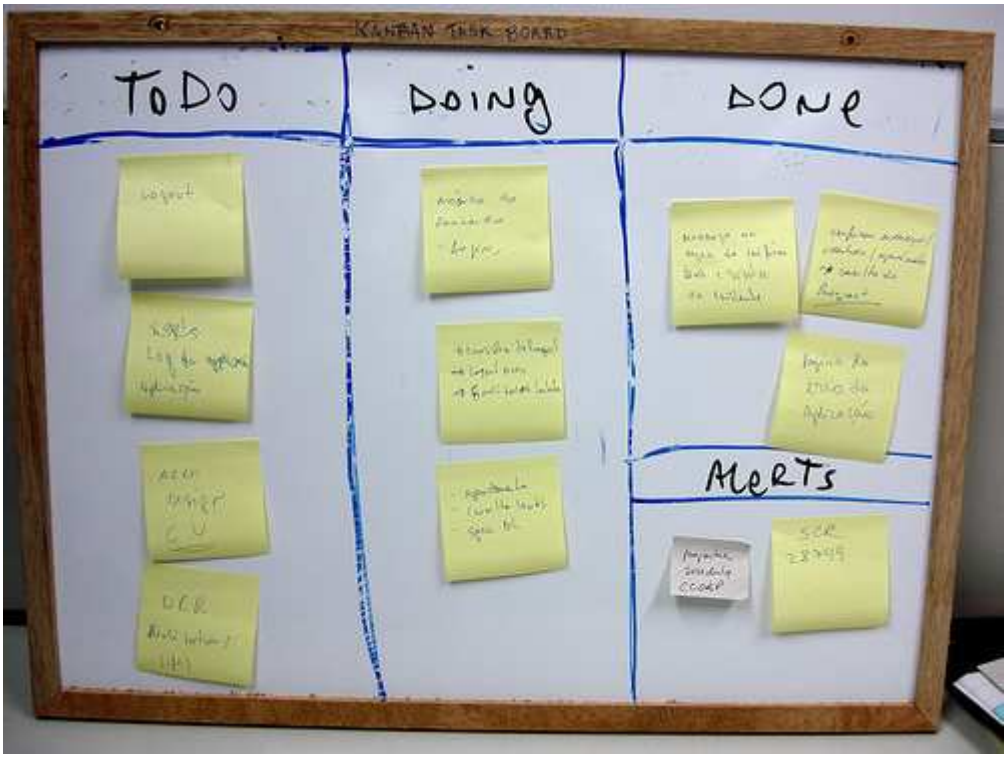

Figura 12: Quadro Kanban, com as atividades e colunas que descrevem o status atual das mesmas. (Fonte: Boaglio.com)

A seguir será exposto um caso simples de como o *Scrum* pode ser utilizado para produzir um jogo para celular. Um exemplo simples, mas que poderá ser utilizado como guia para outros projetos mais complexos.

3.5 ESTUDO DE CASO: APLICANDO O *SCRUM* EM UM PROJETO

O jogo escolhido para se trabalhar foi o clássico *Snake*, já conhecido por toda a geração de celulares, desde os primórdios e monocromáticos aparelhos, até os *smartphones* de última geração, com maior capacidade de processamento de dados e gráficos. O objetivo do projeto é planejar com as práticas *Scrum* o desenvolvimento de uma réplica do jogo, com as funcionalidades mais básicas de controle do jogo (movimentação e pontuação), de forma a ressaltar os pontos positivos do processo.

3.6 CARACTERÍSTICAS DO JOGO E REQUISITOS BÁSICOS

- **Nome do jogo:** *Snake;*
- **Tipo de jogo:** estratégia;
- **Gráficos:** 2D;
- **Jogadores:** *single player*.

O jogo consiste em controlar uma cobra sobre um terreno plano a fim de alcançar maçãs espalhadas pelo mesmo. O objetivo principal é comer as maçãs para aumentar a pontuação.

No decorrer do jogo, na medida em que a cobra come as maçãs, seu tamanho duplica. O que também faz com que a complexidade na movimentação dentro do cenário aumente e, consequentemente, a dificuldade do jogo.

O jogo será composto de três telas:

- **Menu:** nessa tela o usuário terá um botão para iniciar o jogo ou sair da aplicação, retornando às atividades comuns do celular;
- **Palco do jogo:** essa é a tela onde o jogo acontecerá. Todas as ações do usuário serão refletidas nessa tela. Possui um *display* com o nível atual de dificuldade e a quantidade de maçãs que restam para que o próximo nível seja alcançado;
- **Tela de finalização (*Game Over*):** quando o usuário perde o jogo, essa é a tela que surge, informando sua pontuação final.

A seguir, um exemplo de todo o contexto gráfico do jogo (considerando a resolução de um celular comum).

Figura 13: Imagens que ilustram as telas do jogo *Snake*.

No menu principal, para alternar o botão em foco, o usuário deve utilizar as teclas de sentido do celular (↑ e ↓). Quando selecionado o botão desejado, para tomar a ação, o botão do meio do teclado (*OK*) deve ser pressionado.

Caso o usuário escolha o botão “Sair”, o aplicativo é encerrado e o celular volta às funções comuns. Se a escolha for por “Iniciar Jogo”, a segunda tela aparece e o jogo começa.

O jogador deve utilizar as seguintes teclas para movimentação de seu personagem no palco:

- **Tecla 2:** muda a direção da cobra para cima (↑);
- **Tecla 8:** muda a direção da cobra para baixo (↓);
- **Tecla 4:** muda a direção da cobra para esquerda (←);

- **Tecla 6:** muda a direção da cobra para direita (→).

Quando a personagem estiver em determinada direção e o usuário pressionar a tecla correspondente à direção contrária (exemplo: a personagem está indo para a esquerda e o usuário pressiona a tecla 6), nada deve acontecer. Os movimentos são feitos considerando o sentido horário ou anti-horário. A tela será uma espécie de matriz quadriculada que será “pintada” durante o jogo (veja imagem ilustrativa).

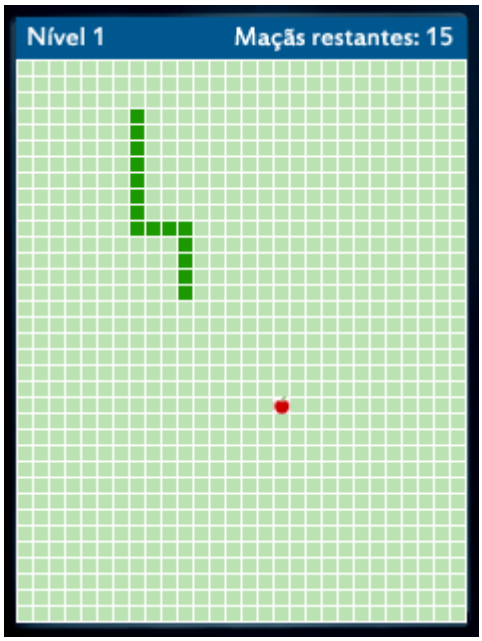

Figura 14: Tela que demonstra o campo de movimentação da personagem.

Outro ponto muito importante a ser considerado na programação do jogo é o comportamento caso a personagem atinja as extremidades da tela. Ela deverá aparecer na extremidade contrária (exemplo: atravessou o topo da tela, aparece novamente em sua parte inferior). Veja abaixo a imagem ilustrativa:

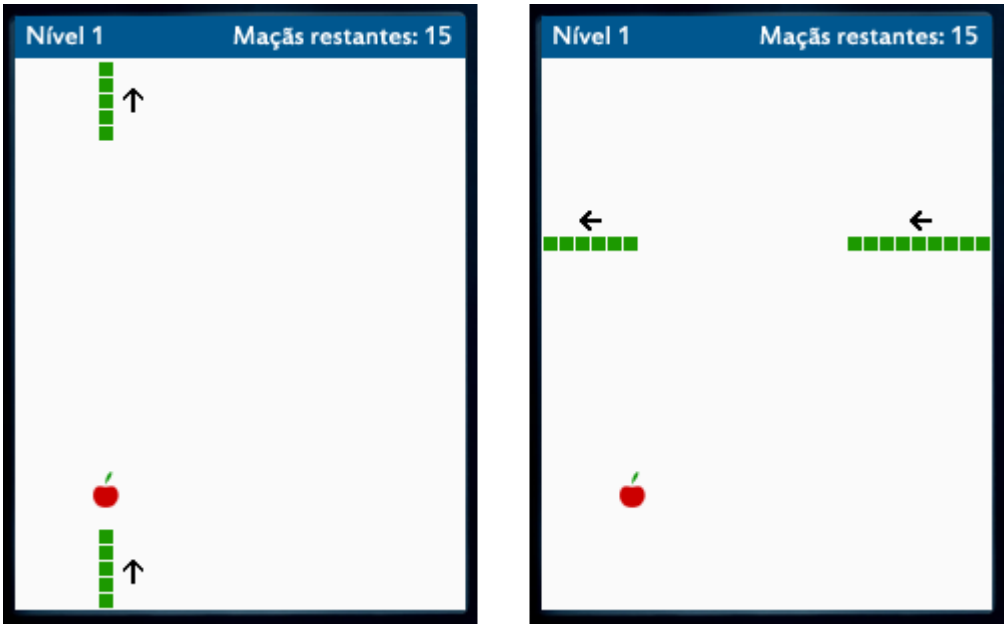

Figura 15: Demonstração de transposição na tela.

Quando a personagem se choca com a maçã, a imagem da maçã é removida da tela, acrescenta-se um novo gomo no fim da personagem e o número de maçãs restantes decresce um. Quando esse número chegar a zero, soma-se um ao nível do jogo (veja parte superior da tela).

O nível do jogo indica a velocidade com que a cobra se movimenta na tela. Conforme os níveis mudam, a cobra se move mais rápido, fazendo com que fique cada vez mais difícil controlá-la no cenário.

Lógica de perda: o jogador perde o jogo a partir do momento em que encosta a cabeça da personagem em qualquer parte do corpo da mesma. E, quando perde, a tela final aparece, mostrando a pontuação total do jogo (que é calculada em *background* e mostrada somente no encerramento). Essa tela final ficará exposta por 10 segundos, e, em seguida, o jogador volta para o menu principal, onde pode escolher se deseja jogar novamente ou sair.

A pontuação é calculada de acordo com a quantidade total de maçãs comidas. A cada maçã comida, soma-se 10 pontos à variável de pontuação.

3.7 CRIANDO O *PRODUCT BACKLOG* E FORMANDO A EQUIPE

Nesse exemplo, visualizamos ambos os papéis, tanto do lado dos clientes como dos fornecedores, em diferentes situações. Ao observar o jogo de exemplo a ser estudado, consegue-se montar um *product backlog* contendo as estórias de acordo com as prioridades exigidas pelo cliente.

ID	Nome	Importância	Estimativa
1	Iniciar um novo jogo pelo menu	100	16h
2	Sair do jogo através do menu	5	4h
3	Movimentar a personagem usando o teclado	80	24h
4	Atingir as maçãs para ganhar pontos	70	16h
5	Perder o jogo com o choque da personagem	50	24h
6	Visualizar tela de *Game Over* e pontuação	10	8h

Tabela 1: Estórias.

Pode-se perceber, com esse simples *product backlog,* as funcionalidades macro do jogo em questão. As atividades que compõem a mesma serão identificadas na *sprint planning*, na qual serão analisadas todas as estórias individualmente.

As estimativas não são as mesmas que surgirão com o planejamento dos desenvolvedores, porém, são necessárias no processo de venda de um projeto. No caso estudado, supõe-se que foram vendidas 92 horas de desenvolvimento, o equivalente a duas semanas de projeto (considerando 8 horas diárias de jornada de trabalho).

A seguir, pode-se notar quais são as estórias extraídas do *product backlog*:

*“Como **jogador**, eu **gostaria de selecionar o botão “Iniciar” no menu do jogo para poder iniciar uma nova partida**.”*

*“Como **jogador**, eu **gostaria de selecionar o botão “Sair” no menu do jogo para poder voltar às funções normais do celular**.”*

*“Como **jogador**, eu **gostaria de controlar a personagem do jogo ao pressionar as teclas do teclado do celular, para assim poder movimentá-la**.”*

*“Como **jogador**, eu **gostaria de levar a personagem até as maçãs para conseguir ganhar pontos**.”*

*“Como **jogador**, eu **gostaria de chocar a personagem com ela mesma para assim perder o jogo**.”*

*“Como **jogador**, eu **gostaria de visualizar a tela de Game Over no fim do jogo, para assim visualizar minha pontuação final**.”*

Com essas estórias, parte-se então para o planejamento dos *sprints.* É definida a equipe de desenvolvimento e iniciado o planejamento do *sprint* com a *sprint planning*.

A equipe é formada por seis membros. Um *Scrum master*, peça chave no desenvolvimento do projeto, auxiliando os demais membros da equipe com suas atividades (não atua tecnicamente, e sim como o facilitador). Como se trata de um projeto relativamente pequeno, com pouca codificação, dois desenvolvedores são

mais do que o bastante. Também é necessária a participação de um artista gráfico para elaborar as imagens utilizadas no jogo (criação dos personagens e cenários), bem como um testador que possa validar o produto final.

O P.O. (*product owner*) é o membro da equipe por parte do cliente. Normalmente, alguém ligado às áreas interessadas no projeto.

Figura 15: Equipe responsável pelo projeto.

3.8 A *SPRINT PLANNING* E AS PRÓXIMAS FASES

Da *sprint planning* obtém-se as estimativas reais geradas através do *planning poker*[10] dos membros da equipe. Com essas estimativas, pode ser feito o planejamento do projeto, estipular suas metas e marcos. Geralmente é utilizada a ferramenta *MS Project* (ou semelhante) para se controlar as atividades do *sprint*, recursos envolvidos, estimativas, precedência e relação entre atividades, calendário do projeto, custos e muito mais.

A seguir, uma figura ilustrativa do artefato gerado após a *sprint planning* do projeto *Snake:*

[10] *Planning poker* é um baralho utilizado para estimar as atividades nas *sprint plannings*.

Snake		
Sprint 1	12 dias	
Menu e início de jogo	1 dia	
Criar imagem do menu	2 hrs	Artista Gráfico
Criar chamadas para os botões "Iniciar" e "Sair"	4 hrs	Programador 1
Criar cenário	2 hrs	Artista Gráfico
Criar imagem do personagem e das maçãs	6 dias	Artista Gráfico
Movimentação	2 dias	
Criar lógica das teclas	12 hrs	Programador 2
Criar lógica de movimentação	16 hrs	Programador 2
Criar lógica para transposição da personagem na tela	8 hrs	Programador 1
Criar lógica de colisão	12 hrs	Programador 1
Pontuação	6h	
Criar arquivo para armazenar ranking	2 hrs	Programador 1
Criar lógica para armazenar pontuação no arquivo	4 hrs	Programador 1
Tela de Game Over	1 dia e 2h	
Criar imagem da tela de game over	2 hrs	Artista Gráfico
Criar chamada para tela de game over	4 hrs	Programador 2
Lógica para mostrar pontuação total do jogador	4 hrs	Programador 2
Testes	3 dias	
Criar plano de testes	8 hrs	Testador
Executar plano de testes	16 hrs	Testador
Fechamento do sprint	1h	
Demo meeting	1 hr	Artista Gráfico;Programad

Figura 16: Tela do MS Project que mostra as atividades do sprint, estimativas e recursos.

Observa-se que as estimativas se adequaram ao que já era previsto no *product backlog*. Doze dias seriam necessários para executar o projeto por completo, o equivalente a duas semanas e dois dias de trabalho. Um *sprint* atende àquilo que o projeto requer. A partir daqui, com o planejamento feito e o cronograma em mãos, os próximos passos seriam:

- **Desenvolvimento:** a codificação do jogo e criação das imagens;
- ***Daily meetings*:** diariamente e durante todo o processo de execução, reunir toda a equipe por 15 minutos para se discutir o *status* do projeto, as atividades executadas, o que será ainda executado e se existem bloqueios;

- **Testes:** execução dos testes do jogo num aparelho celular compatível;
- **Correção de defeitos e reteste:** testador reporta os defeitos numa ferramenta que servirá de banco de dados para armazenar todos os erros encontrados no jogo. Os desenvolvedores acessam essa ferramenta para conhecer os defeitos e corrigi-los. O testador retesta o *software* e o valida;
- ***Demo meeting:*** demonstração do jogo, feita pela equipe de desenvolvimento para o cliente (*product owner* e *business*);
- ***Retrospective meeting:*** reunião para levantar os pontos positivos e negativos do projeto, a fim de propor melhorias.

4 CONSIDERAÇÕES FINAIS

A partir da pesquisa feita, percebe-se uma necessidade iminente de mudanças no mercado tecnológico com relação ao desenvolvimento de *software* para jogos digitais. Apesar do enorme lucro que essa indústria gera, existem problemas pelos quais ela vem passando. Produção desenfreada, recursos não assegurados, que, encontram-se insatisfeitos com o meio em que atuam, falta de criatividade para conceber produtos inovadores, projetos com alto custo (alguns beirando os milhões de dólares), entre outras dificuldades.

Os baixos salários e a carga horária de trabalho em excesso levam a uma exaustão que só tende a crescer. É difícil encarar essa realidade quando analisamos o mercado de jogos digitais como algo muito amplo, e que, emprega milhares de pessoas ao redor do mundo. Não só profissionais da informática, mas também pessoas de diversas outras áreas, das ciências exatas às humanas. Essa diversidade mostra o quão representativa é essa indústria. E é fácil de perceber tal

fato, basta fazermos um paralelo entre o mercado de entretenimento digital e as outras vertentes da informática.

No primeiro capítulo deste documento, é explicitada a história do *software* e sua produção, bem como os estudos da engenharia nesse meio. Portanto, fica claramente visível que o *software* que compõe um jogo, em sua essência, não difere totalmente dos outros programas de computador. Existem processos e procedimentos que são semelhantes (ou, em muitos casos, idênticos). Tanto que as primeiras empresas que decidiram adotar a produção de jogos como principal fonte de renda, utilizaram em seus projetos as metodologias tradicionais, propostas pela engenharia de *software* (como o Modelo Clássico e o RUP, por exemplo).

Partindo desse ponto e com base nessa análise, nota-se que é perfeitamente possível aplicar o *Scrum* ao desenvolvimento de jogos digitais. Utilizamos um exemplo muito simples, porém nada impede o uso do *framework* em projetos de maior complexidade. Caso haja a necessidade de se gerir mais de uma equipe (muito comum em projetos grandes), pode-se dividir o trabalho em pequenos projetos distintos, aplicando o *Scrum* individualmente, junto a cada equipe em particular (*Scrum* de *Scrums*).

Usando o *Scrum,* além de tratarmos o desenvolvimento do *software* de forma mais dinâmica e menos burocrática, pois, é o que o próprio *framework* já visa, conseguimos garantir que o ambiente de trabalho se torne mais agradável. O processo pode se tornar menos cansativo. É possível ter maior controle sobre os projetos, garantindo que os recursos se sintam beneficiados e mais motivados, estimulando a criatividade e produção. Metodologias ágeis conseguem embutir maleabilidade nos projetos, sem deteriorar a qualidade do produto final.

Apesar da necessidade de adaptação cultural e organizacional, o *Scrum* pode sim vir a determinar o primeiro passo para o sucesso de projetos futuros e melhorias na indústria de jogos de forma geral.

5 REFERÊNCIAS BIBLIOGRÁFICAS

A indústria brasileira de jogos eletrônicos - Um mapeamento do crescimento do setor nos úl6mos 4 anos. **ABRAGames**, 2008. Disponivel em: <http://www.abragames.org/docs/Abragames-Pesquisa2008.pdf>. Acesso em: 06 Novembro 2010.

BARROS, R. L. B. Análise de Metodologias de Desenvolvimento de Software aplicadas ao Desenvolvimento de Jogos Eletrônicos, Recife, p. 69, 22 Agosto 2007. Disponivel em: <http://www.cin.ufpe.br/~tg/2007-1/rlbb.pdf>.

CASTRO, I. C. A.; MOREIRA, A. M. Metodologias de Desenvolvimento: Um Comparativo Entre Extreme Programming e Rational Unified Process, Salvador, p. 12, 2010. Disponivel em: <http://www.frb.br/ciente/BSI/BSI.CASTRO.%20et%20al%20.F2%20.pdf>. Acesso em: 10 Outubro 2010.

COHN, M. Scrum For Video Game Development. **Mountain Goat Software**, 2007. Disponivel em: <http://www.mountaingoatsoftware.com/presentations/50-scrum-for-video-game-development>. Acesso em: 06 Novembro 2010.

CRUZ, L. R. S. Desenvolvimento de Software com Scrum. Disponivel em: <http://www.assembla.com/spaces/projeto_ambap_ufal/documents/csaP3ANYqr3A1_ab7jnrAJ/download/DesenvolvimentodeSoftwarecomScrum.pdf>. Acesso em: 07 nov. 2010.

FEIGENBAUM, E.; MCCORDUCK, P. **The fifth generation:** artificial intelligence and Japan's computer challenge to the world. 1. ed. Stanford: Addison-Wesley, 1983. 275 p.

GUSTAFSON, D. A. **Teoria e Problemas de Engenharia de Software**. 1. ed. Santana: Bookman, 2002.

HAMZE, A. O jogo educativo como fato social. **Canal do Educador**, 2010. Disponivel em: <http://www.educador.brasilescola.com/trabalho-docente/o-jogo-educativo-como-fato-social.htm>. Acesso em: 06 Novembro 2010.

ITTERHEIM, S. Scrum in Game Development. **Gaming Horror**. Disponivel em: <http://www.gaminghorror.net/scrum-resources/scrum-in-game-development/>. Acesso em: 11 Novembro 2010.

KEITH, C. Beyond Scrum: Lean and Kanban for Game Developers. **Gama Sutra**, 2008. Disponivel em: <http://www.gamasutra.com/view/feature/3847/beyond_scrum_lean_and_kanban_for_.php>. Acesso em: 06 Novembro 2010.

KEITH, C. **Agile Game Development With Scrum**. 1. ed. Boston: Addison-Wesley, 2010.

KNIBERG, H. **Scrum e XP direto das Trincheiras**. 1. ed. [S.l.]: C4Media, 2007.

LARAMÉE, F. D. **Game Design Perspectives**. 1. ed. Massachussets: Charles River Media, 2002.

MARIANO, P. Metodologias: Às vezes adaptações são necessárias. **InfoQ**, 13 Julho 2010. Disponivel em: <http://www.infoq.com/br/news/2010/07/metodologia-adaptacao>. Acesso em: 1 Novembro 2010.

MARTIN, J.; MCCLURE, C. **Técnicas Estruturadas e Case**. 1. ed. São Paulo: Makron Books, 1991.

PAPO, J. C. Metodologias Ágeis e Extreme Programming(XP) - Uma Introdução. **Spin SP**, 2010. Disponivel em: <http://www.spinsp.org.br/apresentacao/SpinXP.pdf>. Acesso em: 12 Outubro 2010.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books do Brasil Editora Ltda., 1992.

SUTHERLAND, J. et al. Manifesto para o desenvolvimento ágil de software. **Manifesto Ágil**. Disponivel em: <http://www.manifestoagil.com.br/>. Acesso em: 12 Outubro 2010.

VELASQUEZ, C. E. L. Modelo de Engenharia de Software para o Desenvolvimento de Jogos e Simulações Interactivas, 29 Dezembro 2009. Disponivel em: <https://bdigital.ufp.pt/dspace/bitstream/10284/1361/3/DM_CarlosVelasquez.pdf>. Acesso em: 03 Novembro 2010.

ZANATTA, A. L.; VILAIN, P. Uma análise do método ágil Scrum conforme abordagem nas áreas de processo - Gerenciamento e Desenvolvimento de Requisitos do CMMI, 2010. Disponivel em: <http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER05/alexandre_zanatta.pdf>. Acesso em: 12 Outubro 2010.