UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

AGILE GAME PROCESS

METODOLOGIA ÁGIL PARA PROJETOS DE ADVERGAMES

TRABALHO DE GRADUAÇÃO

Aluno: Allan Rodrigo dos Santos Araujo (arsa@cin.ufpe.br)
Orientador: Prof. Hermano Perrelli de Moura (hermano@cin.ufpe.br)

05 de outubro de 2006

ASSINATURAS

Este Trabalho de Graduação é resultado dos esforços do aluno Allan Rodrigo dos Santos Araujo, sob a orientação do professor Hermano Perrelli de Moura, sob o título inicial de "*Metodologia Ágil para o Desenvolvimento de Advergames*", posteriormente modificado para "*Agile Game Process*: *Metodologia Ágil para Projetos de Advergames*". Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

_____

Allan Rodrigo dos Santos Araujo

_____

Hermano Perrelli de Moura

"Navegar é Preciso, Viver não é preciso."

(Pompeu, General Romano)

Dedicado aos meus pais, Alcidésio e Luciene, pela lição de amor e sacrifício.
Dedicado a meu irmão, Diego, pela amizade e fraternidade.
Dedicado a Wanessa, por todo amor, paixão e companheirismo.

Agradecimentos

Agradecer é uma forma de reconhecer a importância das pessoas em sua vida. Entretanto, agradecer é – de qualquer maneira – ser injusto ao se esquecer de pessoas que deveriam ter sido lembradas. A todos estes, vítimas da minha memória problemática, minhas sinceras desculpas.

Inicialmente, gostaria de agradecer a Deus por tudo.

Agradeço a meus pais (Alcidésio e Luciene) pela vida e pela bela lição de amor, sacrifício, honestidade, e educação. Aproveito para agradecer também ao meu irmão, Diego, pela amizade e fraternidade. Sem eles e seu apoio nas horas incertas jamais teria conseguido chegar tão longe.

Agradeço a Wanessa, minha noiva, por todo respeito, amor, paixão, compreensão pelas minhas ausências, por aturar meu mau humor nas horas mais difíceis, por compartilhar dos meus sorrisos e lágrimas e por aceitar enfrentar tantos desafios ao meu lado.

Agradeço a Renata por ter sido uma irmã todos esses anos, mesmo tendo nascido em outra família. Agradeço pela motivação, carinho e todo respeito. Agradeço também a Patrícia por todo carinho e aprendizado sobre a vida e as pessoas. Aproveito para agradecer a Paolinha, amiga tão recente, mas que soube ouvir e provocar enormes risadas, mesmo nos momentos mais difíceis.

Agradeço aos meus compadres (Luis e Daniele) e ao meu afilhado querido, Luis Ricardo.

Agradeço a Hermano, meu orientador, e Scylla, meu co-orientador, pelo apoio e pela oportunidade de desenvolver este trabalho. Agradeço a Anjolina, Ruy, Fábio, Alexandre Mota, Augusto, Adalberto e todos grandes professores que tive estes anos no Centro de Informática. Também a todos os grandes professores que tive em minha vida desde os primórdios, entre eles, Djair e Jorge.

Agradeço aos grandes amigos que tive durante toda a graduação, como Livar, Rafael, Joabe, Renan, Léo, Zé, Jeane, Fernando. Neste momento, agradeço especialmente às grandes caronas de Livar e Rafael, sem as quais eu teria dormido muito menos e esperado muito mais pelo "CDU/Casa Amarela" todos estes anos difíceis. Agradeço especialmente a Joabe também, por todas as madrugadas fazendo projetos ao meu lado. Com outros amigos, tive poucas oportunidades de trabalhar, mas eles têm meu carinho: Ulli, Raquel, Sylvinha, Lais, Carol, Aninha, Carina, além de Geraldo, Carlinhos, Carlos, Farley, Vitão, Ronaldinho, entre tantos outros.

Agradeço a amigos de outras turmas que acabaram sendo muito importantes para meu crescimento durante a graduação. Entre eles cito Rafael (Popis), Rafael (Dudu) e o Bado. Aproveito para agradecer aos amigos do CITI (César, Bruno, etc.) e aos amigos do XPRecife (Marcelo, Ricardo, Raony, etc.)

Agradeço à Jynx Playware, sem seu apoio (através de conhecimento, oportunidades e folgas) eu jamais teria conseguido concluir este trabalho. Aproveito para agradecer aos bons amigos que lá tenho, como Fred, Rui, André, Paulo, Riva, Caio, etc.

Agradeço ao Santa Cruz Futebol Clube por existir. Uma verdadeira lição de amor, vitórias e superação nas horas mais difíceis. Neste momento, aproveito para agradecer a Renato, Fred, Guilherme, Marcelo, Rodrigo (Batata), etc., grandes companheiros de comemorações e lamentações.

Agradeço aos amigos de CEFET que me fazem lembrar muitos momentos de diversão e alegria. Em nome de Daniel, André e Hugo, agradeço a todos os demais.

Agradeço aos amigos do #Cybercalango por tantos anos de amizade. Em nome de Thica, Xellie, Milla, Tita, Ingrid e Nilton, agradeço a todos os demais.

Gostaria de agradecer também a todos os grandes amigos que construí ao longo da minha vida, como Ronaldo, Tarcísio, Diogo, Paulo de Tarso, Barata, Rodolfo, Erick, Camarotti, Thiago Jorge, Marcos André, Roberta (França), Rafaela, Rebeca e todos os demais.

Finalmente, agradeço a você, que teve paciência de ler tantos agradecimentos e ainda encontra coragem para ler o resto deste trabalho.

A todos vocês, muito obrigado!

Resumo

Este trabalho concentra seus maiores esforços em reconhecer os principais desafios enfrentados por uma organização que desenvolve jogos publicitários (*advergames*).

Por isso, se analisa vários métodos (metodologias e processos) utilizados para desenvolver e gerenciar projetos de jogos. Muitos aspectos são analisados para, então, se concluir que nenhuma das metodologias mais utilizadas é completamente aderente às necessidades de um projeto de *advergame*.

Dessa forma, é sugerida uma metodologia - baseada nas melhores práticas das outras metodologias - que se mostra totalmente alinhada com a realidade de desenvolvimento de um jogo publicitário. Ela recebeu o nome de *Agile Game Process* (AGP). Esta metodologia possui os valores defendidos pelos entusiastas das metodologias ágeis e se encontra amplamente detalhada neste trabalho através de padrões para descrições de processos de software, como o SPEM.

Finalmente, vários pontos positivos e negativos da AGP são discutidos. Neste momento, alguns trabalhos futuros são sugeridos como forma de melhorar e refinar a metodologia.

Palavras chave: Gerência de Projetos, Jogos, Advergames, Scrum, XP, SPEM.

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

"What is real? Define real."
Morpheus (Matrix, 1999)

Um jogo é uma atividade estruturada (ou semi-estruturada) onde os jogadores devem alcançar determinados objetivos, respeitando um conjunto de regras, envolvendo estímulos físicos e/ou mentais. Em geral, os jogos têm propósito de entretenimento, porém, podem ser utilizados com outros fins, como educação, treinamento ou publicidade [Wiki061].

"Jogo eletrônico" é o termo designado para representar qualquer espécie de jogo onde o principal meio de jogar seja através de um computador ou um videogame. Atualmente, o mercado de jogos eletrônicos movimenta mais de US$ 12 bilhões (por ano) somente nos EUA [Wiki062].

Atualmente, os jogos são empregados em vários níveis desde o entretenimento à educação, treinamento ou publicidade. *Advergames* são jogos criados com vocação publicitária com o objetivo de passar uma mensagem que contém propaganda. Em geral, a mensagem está intrínseca no jogo [Chen01]. No Brasil, os jogos com fins publicitários representam cerca de 14% do mercado [Abragames05].

Jogos são construídos através de projetos, porque podem ser definidos como produtos únicos (ou exclusivos) produzidos através de um esforço temporário [PMBOK04]. Além disso, desenvolvimento de jogos pode ser considerado como uma instância particular do desenvolvimento de *sistemas computacionais*, dado que um jogo é representado por um *programa*.

Neste contexto, os *advergames* – jogos com vocação publicitária – aparecem como um novo tipo de jogo e, ao mesmo tempo, como aposta do mercado publicitário. O objetivo de um *advergame* é divulgar um produto, uma organização ou um ponto de vista. Seu poder de imersão e interatividade com o usuário, no entanto, torna-se um grande diferencial em relação aos outros tipos de mídia.

No entanto, o desenvolvimento de projetos de criação de *advergames* apresenta uma série de peculiaridades próprias do ambiente e precisa ser suportada por um método que seja capaz de sustentar o desenvolvimento de maneira eficaz e possa alavancar os índices de sucesso dos projetos.

Portanto, se discute o cenário dos projetos de software, os desafios do desenvolvimento de jogos (em particular, *advergames*) e se analisa as metodologias existentes utilizadas para criação de jogos. No momento, então, caso necessário, uma nova metodologia pode ser sugerida para atender aos desafios de se criar um *advergame*.

1.1 Trabalhos Relacionados

Este trabalho é único e sem precedentes sob a ótica de metodologias para o desenvolvimento de *advergames*. No entanto, existem muitos métodos e processos sugeridos para a criação de jogos, como:

- ***Waterfall***: Um método baseado no modelo de ciclo de vida em cascata, sugerido para criação de jogos no livro "Software Engineering for Game Developers" [Flynt05];

- ***Game Unified Process***: Uma adaptação do "Processo Unificado" para o desenvolvimento de jogos, levando em consideração algumas poucas práticas de *Extreme Programming* [Flood03];
- ***Extreme Game Development***: Uma adaptação de *Extreme Programming* (XP) para o desenvolvimento de jogos, onde algumas práticas de XP são discutidas sob o ponto de vista de jogos [Demachy03];
- ***Paper Burns***: Mostra uma nova abordagem para os processos de *Game Design* sob o ponto de vista de metodologias ágeis, como *Scrum* [McGuire06].

Logo, existem alguns métodos para o desenvolvimento de jogos, no entanto, nenhum deles se encontra alinhado às necessidades demandadas pelo desenvolvimento de um *advergame*.

1.2 Objetivos

O trabalho desta monografia está ligado às metodologias de desenvolvimento de software, em particular, softwares que são jogos com fins publicitários. Então, este trabalho apresenta dois objetivos principais:

- Discutir os desafios de desenvolver jogos (*advergames*), analisando as metodologias utilizadas atualmente, e o suporte que oferecem ao desenvolvimento de projetos de *advergames*;
- Sugerir uma nova metodologia capaz de unir as melhores práticas de desenvolvimento e gerenciamento de projetos que possam estar completamente alinhadas às necessidades da criação de um *advergame*.

1.3 Visão Geral

Este trabalho está divido em seis capítulos e um apêndice.

Neste primeiro capítulo encontra-se a descrição da estrutura deste trabalho. No capítulo 2 é apresentada uma introdução sobre o contexto em que este trabalho está inserido, o "Mundo dos Jogos Eletrônicos". O capítulo 3 apresenta o problema que envolve os projetos de jogos, considerando aspectos de desenvolvimento e gerenciamento. Além disso, ele trata – detalhadamente – dos desafios na criação de um *advergame*. O capítulo 4 é responsável por descrever e analisar as metodologias de desenvolvimento que são utilizadas para criação de jogos, isto é, discutir sobre o estado de arte das soluções existentes. Posteriormente, no capítulo 5, é sugerida uma nova metodologia para projetos de *advergames*, capaz de integrar aspectos de negócio e produção, atendendo às principais necessidades e particularidades destes projetos e aumentando as probabilidades de sucesso. Então, o capítulo 6 traz a conclusão sobre este trabalho, analisando vantagens e desvantagens da metodologia proposta, e sugerindo futuros trabalhos para melhorar e refinar a metodologia sugerida. Finalmente, o Apêndice A provê um conjunto de definições para termos técnicos empregados nesta monografia.

2. Mundo dos Jogos Eletrônicos

"Reality is merely an illusion, albeit a very persistent one."
- Albert Einstein

Este capítulo discute os aspectos históricos e mercadológicos dos jogos eletrônicos, como forma de contextualizar e introduzir o ambiente em que este trabalho se concentra.

Os jogos existem desde o início da humanidade e se apresentam através de várias de suas manifestações culturais, sociais e históricas. Os jogos olímpicos na Antiga Grécia servem como exemplo para ilustrar a natureza humana intimamente ligada à competição, disputa e superação de limites.

2.1 Um Pouco de História

Em meados do século XX, o mundo havia enfrentado suas duas maiores guerras e estava polarizado sob a liderança de duas superpotências: os EUA e a URSS. Ambas as nações promoviam uma acirrada corrida tecnológica em busca de uma vantagem hegemônica em relação à outra. Dessa forma, importantes avanços criados no meio científico passaram a ser incorporados pela sociedade. Entre estas invenções, surgia sucintamente o conceito de jogos eletrônicos [Azevedo05].

Assim, o primeiro jogo eletrônico foi "Tennis for Two", criado no Brookhaven National Laboratories, sob a responsabilidade de Willy Higinbotham, que anteriormente havia trabalhado no projeto Manhattam (onde foi desenvolvida a tecnologia da bomba atômica utilizada na II Guerra)

[Wiki062]. Naquela época, o laboratório recebia muitos visitantes fascinados com o poder dos computadores (muito superior aos similares contemporâneos) e o jogo foi uma forma de potencializar o interesse do público em visitar o laboratório.

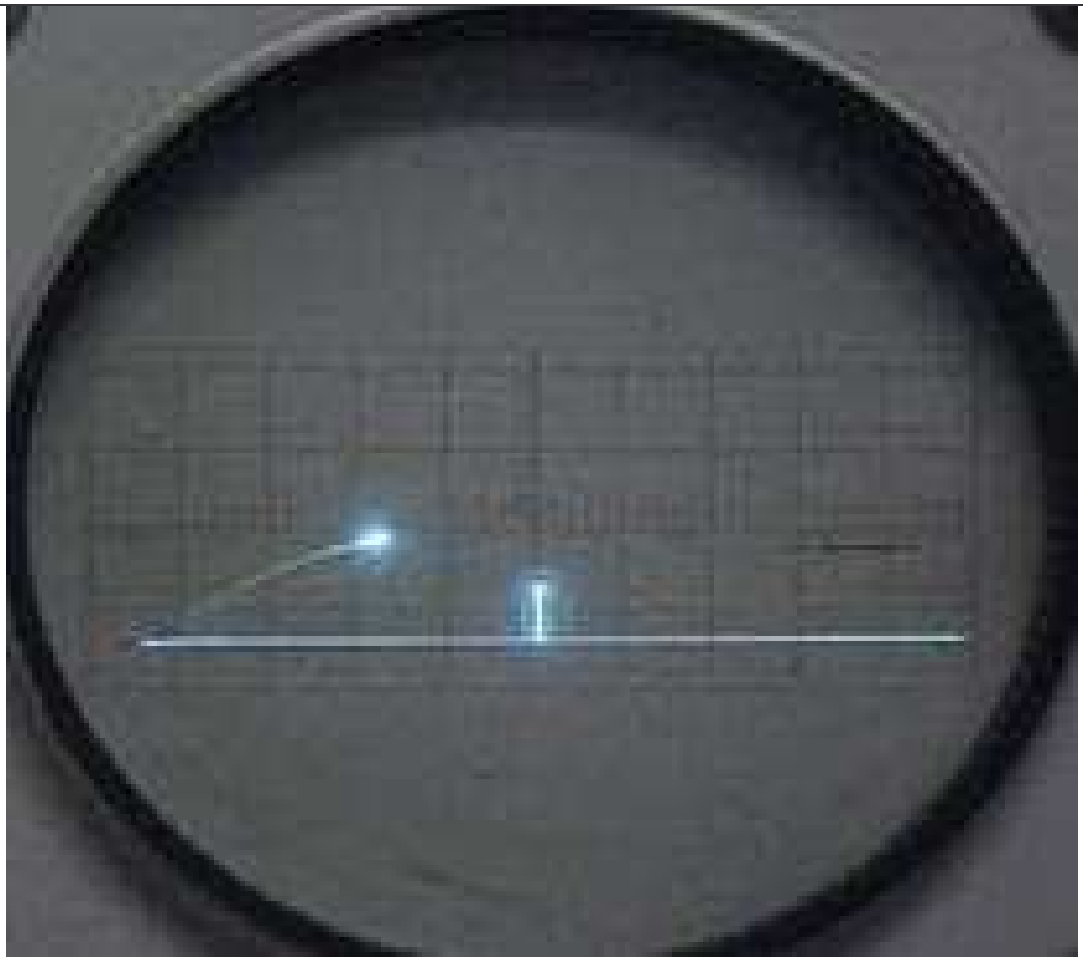


Figura 1 – "Tennis for Two", 1958.

Quatro anos mais tarde, em 1962, nos laboratórios do Massachusets Institute of Technology (MIT), um time liderado por Stephen Russel, munido com um PDP-1 (um assombro tecnológico naquela época), criou "Spacewar!", um jogo que recriava uma batalha entre duas naves, e impressionava pela realidade das propriedades físicas, havendo manifestações de inércia e campos gravitacionais, por exemplo. Assim como "Tennis for Two", seus inventores subestimaram o valor de sua criação e não obtiveram nenhum dividendo financeiro a partir de sua obra [Azevedo05].

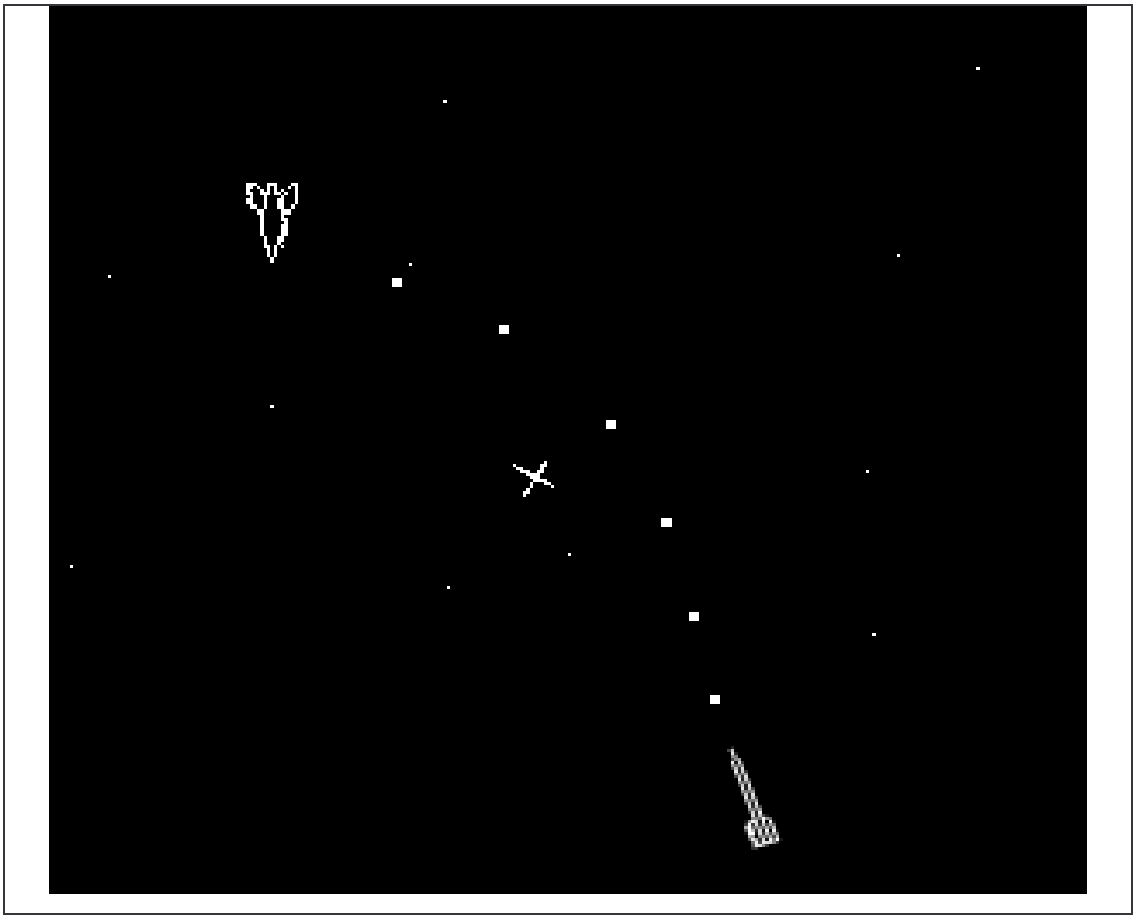


Figura 2 – "Spacewar!", 1962.

Posteriormente, na década de 70, os jogos encontraram sua "idade de ouro". Com a Lei de Moore [Wiki065], a capacidade computacional dos dispositivos evoluía de maneira incrível. Assim, houve uma mudança dos paradigmas fazendo com que os equipamentos eletrônicos saíssem dos laboratórios e estivessem presente no dia a dia, nos lares. Assim, em 1966, Ralph Baer, um pesquisador americano que havia desenvolvido um jogo chamado "Chasing Box", decidiu correr atrás de iniciativa privada diante da falta de apoio governamental para o seu projeto. Muitos anos mais tarde, em 1971, Baer firmou uma parceria com a Magnavox (subsidiária americana da Phillips), e foi lançado o Odyssey-100, o primeiro *console* da história. Em seguida, em 1978, ele foi refinado e lançado como o Odyssey 2. Ao mesmo tempo, em 1972, Nolan Bushnell e Al Alcorn fundaram a Atari. O objetivo de Bushnell era desenvolver um jogo para a plataforma *arcade*, e assim, nasceu um jogo de estilo "ping-pong" chamado "PONG", contando com uma

intuitiva interface, o que transformou o jogo num verdadeiro sucesso e num clássico da era dos *arcades* [Wiki062].

Depois, a década de 80 foi marcada pelo declínio e saturação dos sistemas Atari, e surgimento do Nintendo's Nintendo Entertainment System (NES), que foi responsável pela criação dos mais conhecimentos personagens da indústria dos jogos eletrônicos, como "Super Mario Bros." e tantos outros. Além disso, a Nintendo procurou estabelecer um conjunto de regras com relação aos seus fornecedores de jogos para evitar a saturação de seu sistema e queda de qualidade dos seus jogos, como havia acontecido com a Atari. Contemporaneamente, a Sega lançou seu console, o Master System, que, no entanto, jamais alcançou o sucesso obtido pelo NES [Wiki063].

As últimas duas décadas marcaram a segmentação do mercado de jogos entre consoles, computadores pessoais e dispositivos móveis. Por exemplo, por volta do ano de 1989, a Nintendo lançou seu vídeo game portátil, o Game Boy. Em contrapartida, foram lançados o Sega Game Gear e o Atari Lynx, ambos portáteis e, a exemplo do sucesso no mercado de vídeo game console, o dispositivo da Nintendo – mais uma vez – foi o que mais obteve sucesso no mercado [Azevedo05].

Figura 3 – "Super Mario Bros.".

Nesta mesma época, por volta do ano de 1989, o mercado americano de vídeo games console era dominado pela Sega. No entanto, dois anos mais tarde, o Super NES, da Nintendo, surgiu e provocou um equilíbrio de forças neste mercado. O Super NES foi o primeiro console com suporte de instruções com 16-bits, o que possibilitava um grande avanço em termos de processamento e qualidade dos gráficos. Então, para fazer frente ao avanço da Nintendo, foi lançado o Mega Drive. Naquele momento, os primeiros passos de *interfaces 3D* foram trilhados [Wiki063].

Em torno de 1995, a Sega lançou seu novo console, o Sega Saturn e a Sony decidiu entrar no mercado de consoles lançando seu "Playstation", ambos com sistemas baseados em 32 bits, abrindo, de uma vez por todas, as portas dos gráficos 3D em jogos. Um pouco depois, a Nintendo lançou o Nintendo-64, um console com sistema de 64 bits. No entanto, umas das fornecedoras de uma das séries de jogos de maior sucesso no Nintendo, Final

Fantasy, decidiu lançar seus novos jogos da série sob a plataforma do Playstation. Isto representou apenas um indicativo de que a liderança mundial do mercado de consoles iria passar da Nintendo para a Sony. Ainda assim, o Nintendo-64 conseguiu bastante com alguns jogos, como "Goldeneye". O Sega Saturn nunca ocupou uma posição relevante no mercado mundial de jogos, dado que seu sucesso praticamente se limitou às fronteiras do Japão [Wiki063].

Figura 4 – Final Fantasy VII.

O final da década de 90 foi marcado pelo lançamento do Dreamcast, da Sega e do Game Boy Color, pela Nintendo. Em paralelo, a Sony decidiu lançar a segunda versão de seu console, o PlayStation 2. Além disso, a Microsoft, maior companhia de tecnologia da informação do mundo, decidiu entrar no mercado de consoles e lançou seu XBox, cujo o principal jogo chamava-se "Halo: Combat Evolved". Neste momento, sufocada pela forte concorrência da Sony e Microsoft e – em menor escala – da Nintendo, a Sega decidiu abandonar a produção de consoles e restringiu sua atuação ao desenvolvimento de jogos para as plataformas existentes [Wiki063].

Figura 5 – Halo: Combat Evolved.

Atualmente, uma série de novos lançamentos tem movimentado o mercado de jogos eletrônicos. A Nokia lançou o N-Gage, um híbrido de telefone celular e console portátil. Em paralelo, foram lançados o Nintendo-DS (da Nintendo) e o PlayStation Portable (PSP), dois consoles portáteis. Um pouco depois, no ano de 2005, a Microsoft lançou o XBox 360, seu novo console, que marca o início da sétima geração dos consoles e promete uma verdadeira revolução na maneira de jogar [Wiki062].

Contudo, durante as últimas décadas, o mundo dos jogos eletrônicos não esteve restrito aos dispositivos móveis e aos consoles. Os jogos também estiveram (e estão) presentes nos computadores pessoais. A constante melhoria de processadores, placas de vídeo e capacidade de memória dos computadores têm permitido desenvolver e utilizar jogos com – cada vez mais – poderosos recursos gráficos e de inteligência.

Além disso, é preciso salientar a influência das profundas evoluções tecnológicas e de infra-estrutura ocorridas nos últimos anos. A Internet, comunicação em tempo real, mobilidade, convergência digital são fenômenos que fazem surgir novas possibilidades de jogar. Atualmente, os jogos são utilizados não apenas para fins de entretenimento, mas para simular negócios, auxiliar na educação, divulgar mensagem de propaganda, etc.

Sendo assim, é possível afirmar que deverá haver um grande crescimento no mercado de jogos eletrônicos decorrente do surgimento de novas tecnologias, melhoria da infra-estrutura de comunicação, convergência digital, ampliação da área de atuação dos jogos, entre outros motivos.

2.2 Mercado de Jogos Eletrônicos

O mercado de jogos se encontra em pleno crescimento devido aos motivos expostos previamente. Anualmente, é publicado um estudo – chamado *ESA Facts* - pela *Entertainment Software Association* (ESA), que busca identificar os mais relevantes aspectos com relação ao desenvolvimento de software voltado para o entretenimento. Na edição de 2005 do ESA Facts foi possível distinguir os gêneros de jogos mais vendidos para consoles e para computadores [ESA05]:

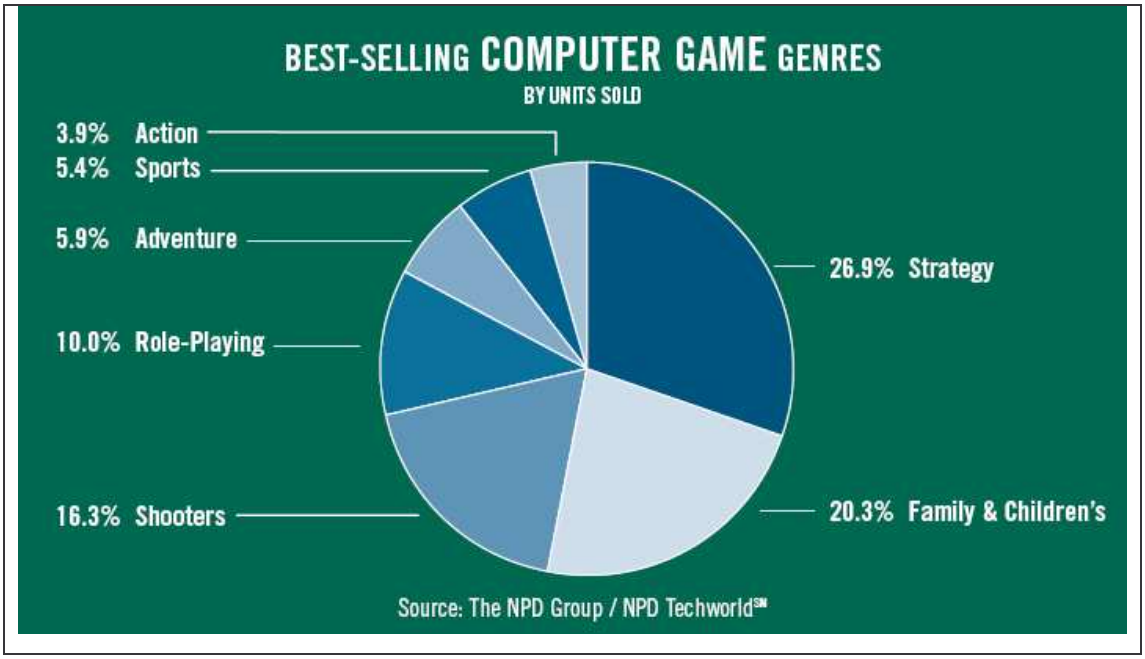


Figura 6 – Tipos de Jogos de Computador mais Vendidos.

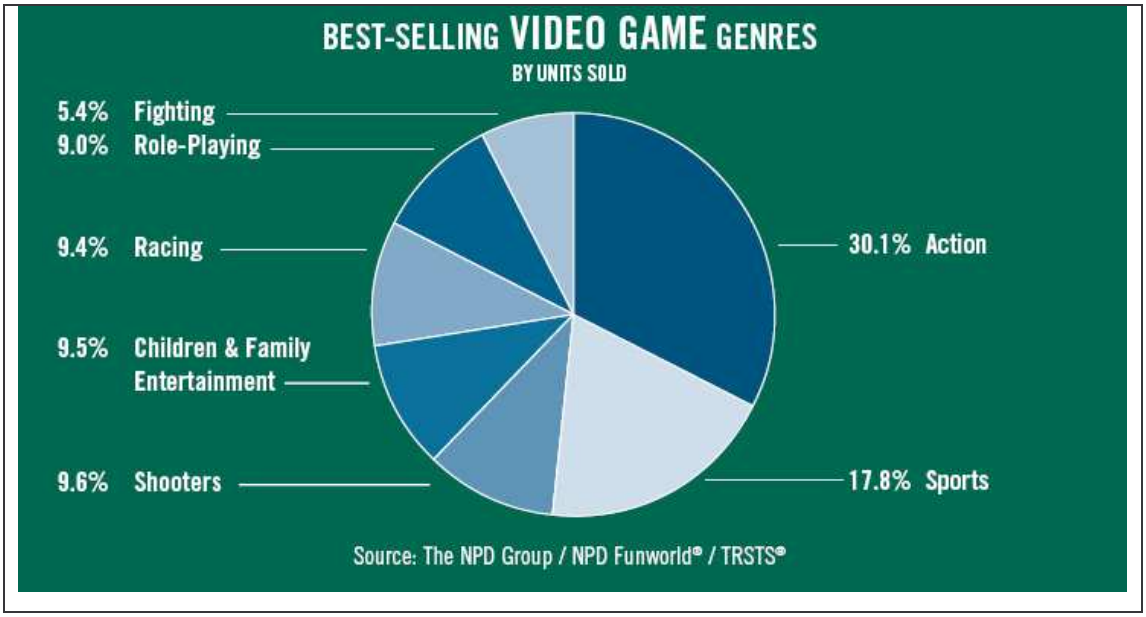


Figura 7 – Tipos de Jogos de Console mais Vendidos.

Além destas informações, o estudo revelou a idade das pessoas que mais compram jogos, os tipos de jogos que são mais jogados "*online*" e foi capaz de apontar uma tendência de crescimento do mercado de jogos eletrônicos nos EUA, ao longo dos anos:

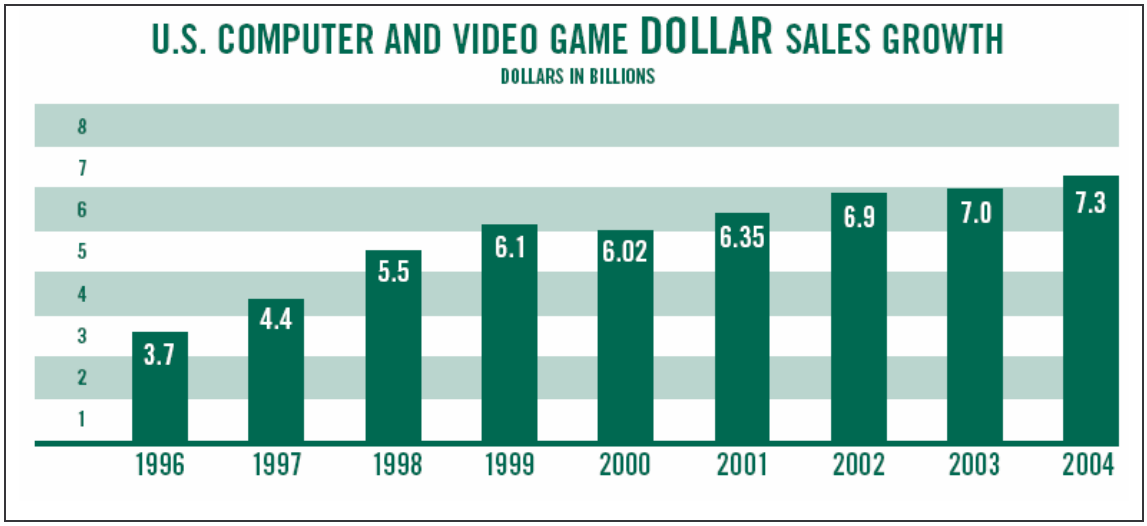


Figura 8 – Mercado de Jogos (em US$) nos EUA.

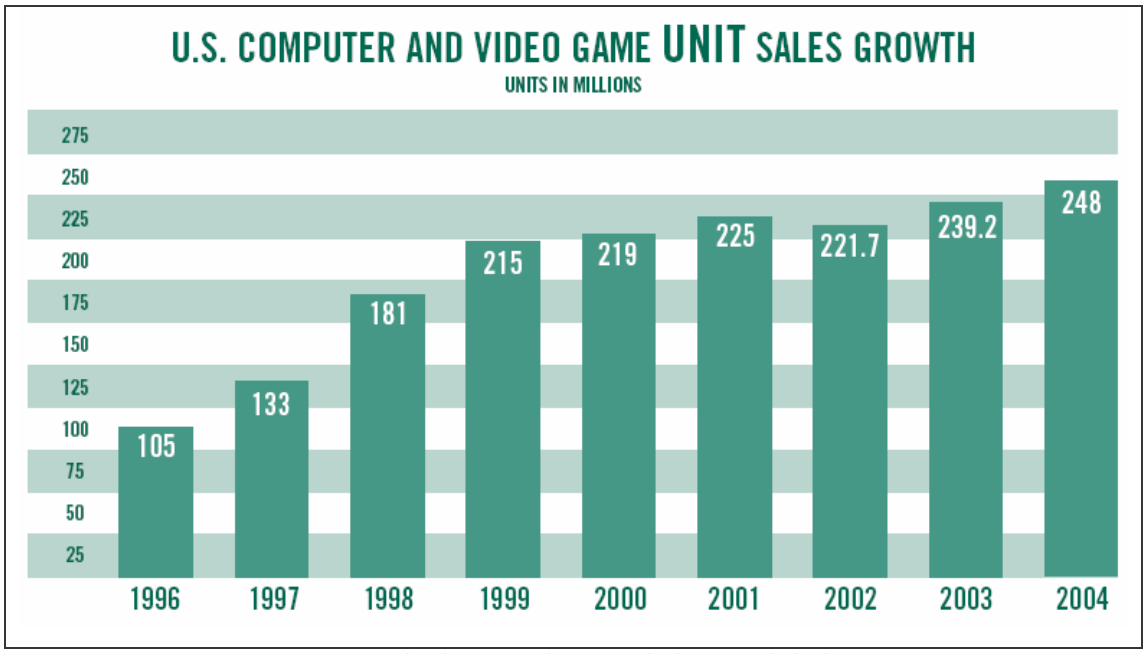
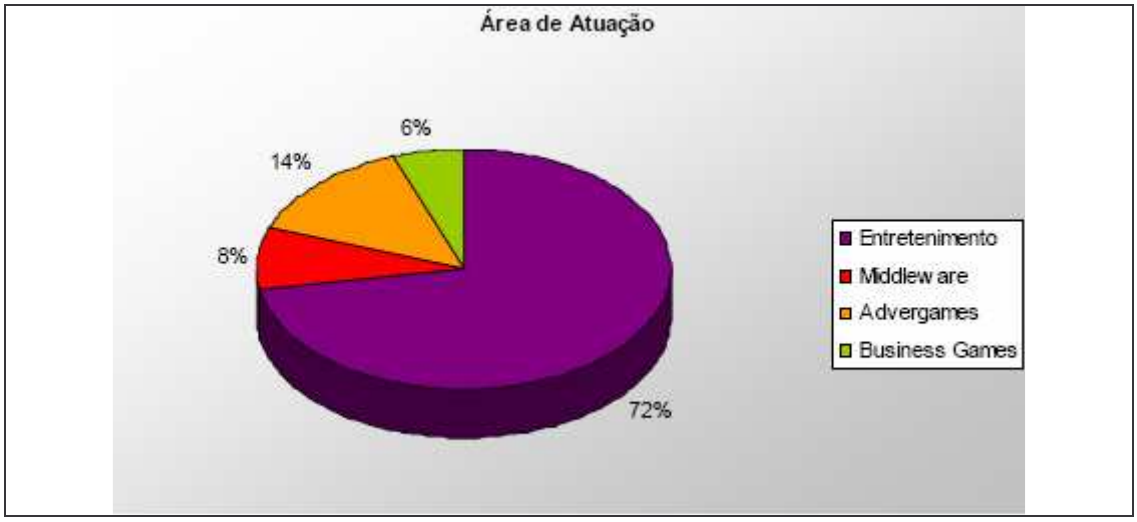


Figura 9 – Mercado de Jogos (em unidades vendidas) nos EUA.

Este crescimento do mercado de jogos eletrônicos, como visto anteriormente, pode ser atribuído a muitas causas. Entre elas, o interesse por outros setores produtivos, como agências publicitárias, tem ajudado a alavancar o crescimento deste segmento da economia. As agências alegam que advergames – jogos com vocação publicitária – serão responsáveis por grande parte das campanhas de propaganda. Atualmente, apenas no Brasil, o desenvolvimento de jogos publicitários representa aproximadamente 14% de todo mercado de jogos [Abragames05].


Área de Atuação
6%
14%
8%
72%
Entretenimento
Middleware
Advergames
Business Games

3. Projeto de Jogos: Desenvolvimento e Gerenciamento

"If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle."
- Sun Tzu

Este capítulo visa discutir os desafios e contextos do desenvolvimento de um jogo. Para tanto, analisa índices de sucesso de projetos de software, aspectos do desenvolvimento de jogos, em particular, *advergames*.

Um jogo eletrônico é desenvolvido para ser um produto distinto, único de alguma forma. Além disso, para criar um jogo é necessário um período de tempo (início e fim) e é preciso utilizar recursos para realizar tal construção. Sendo assim, o desenvolvimento de um jogo eletrônico ocorre através de projetos [PMBOK04].

Outra importante característica é que um jogo eletrônico trata-se de um *programa* computacional. Dessa forma, a criação desse tipo de produto pode ser considerada como uma instância particular dos projetos de desenvolvimento de sistemas computacionais.

3.1 Cenário dos Projetos de Tecnologia da Informação

Nos últimos tempos, a informação tem tido uma relevância cada vez maior em qualquer atividade humana. Através do desenvolvimento da informática (*computação*, *comunicação* e *controle*) [Mahoney88], os sistemas de software estão praticamente em todos os lugares. Softwares são utilizados na indústria, comércio, setor de serviços, setor público, agricultura [Sommerville03], inclusive na educação e no entretenimento.

Apesar de sua larga aplicabilidade, a informática é uma ciência recente (comparada à medicina, por exemplo) que tem seu corpo de conhecimento em construção. Os alicerces da informática começaram a ser delineados através da matemática, eletrônica e programação, com suas principais teorias sendo desenvolvidas somente a partir do século XIX [Mahoney88].

Dentro deste contexto, uma das disciplinas da informática, a engenharia de software, que é responsável pela *"aplicação de uma organizada e sistemática abordagem para o desenvolvimento, operação e manutenção de software"* [Wiki064], ainda caminha para um maior amadurecimento.

Assim, por volta do ano de 1994, o Standish Group começou a publicar – periodicamente – um estudo sobre o sucesso em projetos de software. "Chaos Report" foi o nome que o periódico recebeu. Naquela ocasião, foram analisados os resultados de aproximadamente 175 mil projetos de software em solo norte-americano [Standish94], e obteve-se o seguinte resultado:

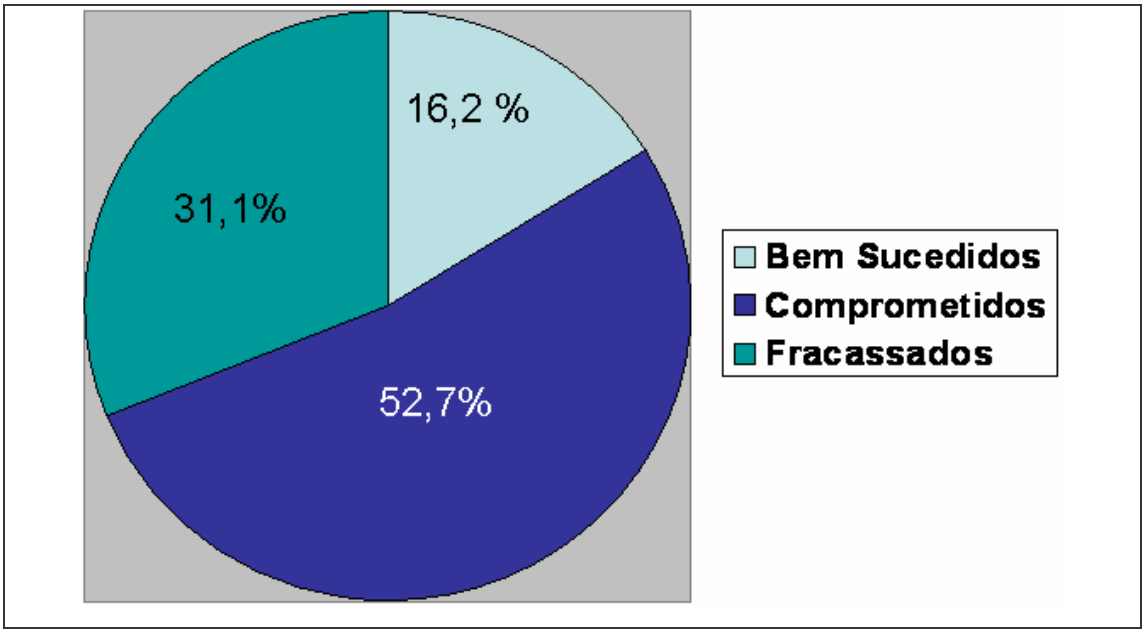


Figura 10 – Chaos Report, Standish Group, 1994.

No resultado do estudo acima, os projetos foram classificados em três categorias:

- Bem Sucedidos: Foram projetos que entregaram seus produtos e/ou serviços de acordo com as expectativas acordadas (escopo), respeitando prazo (tempo) e orçamento (custo);
- Comprometidos: Foram projetos que de alguma maneira desrespeitaram as restrições quanto ao escopo, prazo ou tempo;
- Fracassados: Foram projetos que em algum momento foram cancelados.

Mais tarde, no ano de 2001, o Standish Group novamente publicou o "Chaos Report", avaliando mais de 280 mil projetos de software em solo americano [Standish01], desta vez os números obtidos foram:

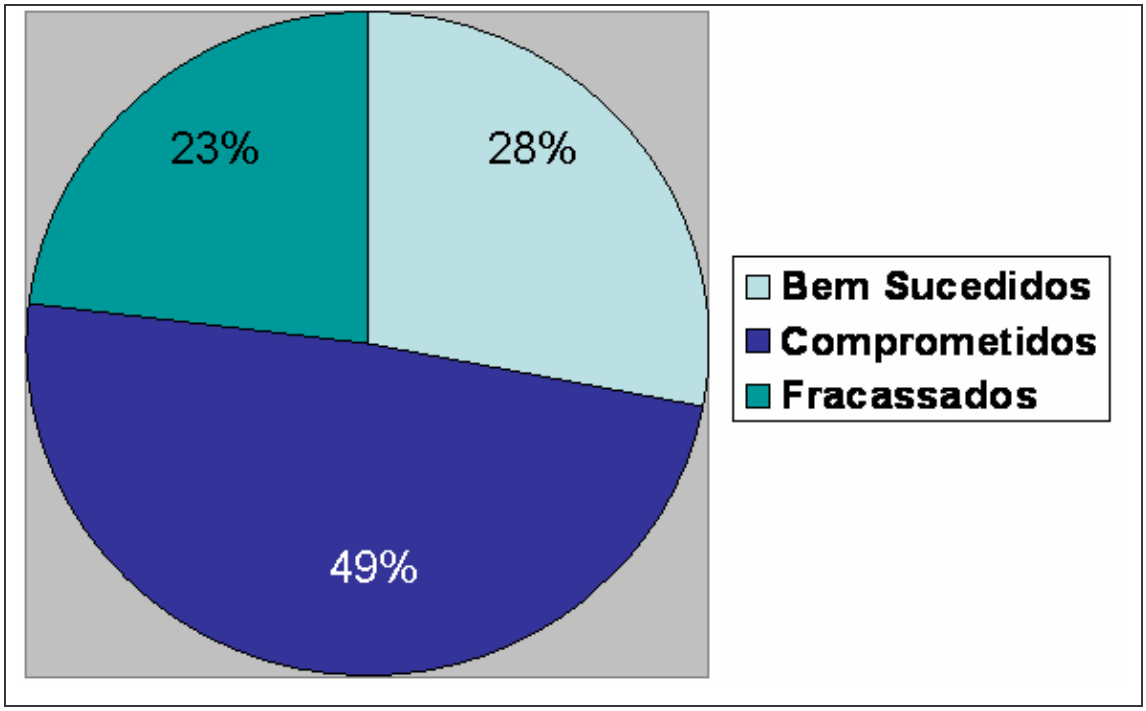


Figura 11 – Chaos Report, Standish Group, 2001.

Os resultados obtidos entre os anos de 1994 e 2001 podem ser analisados de diferentes formas. Primeiramente, pode-se concluir que houve um aumento no número da porcentagem de projetos que alcançam sucesso, e uma diminuição percentual dos projetos que são fracassados. Contudo, numa outra análise, pode-se concluir, também, que a taxa de projetos que são incapazes de alcançar seus objetivos, isto é, respeitar as premissas de escopo, tempo e custo ainda é bastante alta, saindo dos 83,8% (1994) para os 72% (2001), o que pode soar inaceitável para os padrões industriais.

Além disso, durante o mesmo período, surgiram várias propostas de métodos e *frameworks* de "boas práticas" para o gerenciamento e desenvolvimento de produtos de software, tais como: Processo Unificado [RUP], *Project Management Body of Knowledge* [PMBOK00], *Unifying Modelling Language* [UML], etc. Entretanto, todas estas práticas e métodos representam apenas um conjunto de boas práticas que podem ser utilizadas para aumentar a probabilidade de sucesso de um projeto de software e, diante dos resultados

mostrados periodicamente pelo Chaos Report, o impacto da sua adoção sobre o aumento de índice de sucesso dos projetos tem sido pouco relevante.

3.2 Desenvolvimento de Jogos

Como discutido anteriormente, a indústria de games é uma das indústrias geradas pela tecnologia da informação. Embora software seja uma parte substancial da criação de um jogo, o contexto é bem mais amplo. O desenvolvimento de jogos possui uma cultura própria capaz de tornar o projeto de desenvolvimento de um jogo ainda mais crítico que o desenvolvimento de sistemas de informação comuns. Em primeiro lugar, tais projetos envolvem uma enorme diversidade de profissionais com diferentes especialidades, como: artistas, *game designers*, programadores, músicos, etc. Sendo, portanto, bastante difícil integrar todas essas visões e atender todas expectativas [Flynt05].

Sendo assim, o desenvolvimento de um jogo inicia-se com a criação de um *Concept* (ou conceito geral do jogo), que posteriormente, é expandido para um *Game Design Specification (GDS)*, que se trata de um documento que detalha todo o *game-play*, e outras características audiovisuais que o jogo deve ter. Estes documentos são responsabilidades do *game designer*. Uma vez que o GDS é concluído, a equipe de desenvolvimento pode produzir protótipos antes de colocar o jogo em produção definitiva. Estes protótipos, muitas vezes, são chamados de "demo" e podem ser construídos para que os patrocinadores do projeto (por exemplo) possam ter uma visão mais concreta do que deverá ser o produto final. Então, a produção do jogo segue envolvendo um número maior de pessoas: programadores, artistas, músicos, etc. todos focados em criar o jogo descrito pelo *game designer* [Mencher06].

Dessa forma, diante da complexidade de se desenvolver um jogo, uma das maiores evidências da dificuldade de se desenvolver e gerenciar jogos reside no fato de que, em geral, os *postmortems* sempre revelam *"O jogo foi entregue atrasadamente. Ele contém muitos bugs. As funcionalidades originalmente pretendidas não foram entregues. Lançar o jogo levou muitas horas de desenvolvimento com o time sob pressão. Uma vez lançado o jogo, a gerência parecia não estar satisfeita (...)"* [Flood03].

Portanto, criar um jogo demanda bastante esforço, e para que tamanho esforço seja realizado é preciso um processo. Tal processo deve ser capaz de permitir que os envolvidos na criação do jogo possam ser eficientes e efetivos. Assim, nem todas as organizações são capazes de alcançar este nível de maturidade diante de um complexo cenário marcado por grandes projetos, bastante capital investido e muitas pessoas envolvidas na produção de algo que, por muitas vezes, pode ser intangível. Por isso, cerca de uma dezena de companhias monopolizam a maior parte do mercado de jogos [Flynt05].

3.3 Advergames: Contexto e Problemas

Em meio a um novo tempo marcado pela convergência digital e o surgimento de novas mídias, uma das principais apostas do mercado publicitário tem sido a comunicação direta e interativa com os consumidores. Assim, atualmente, os advergames apresentam-se como um dos elementos da comunicação publicitária via Internet, por isso, têm sido utilizados como ferramenta para publicidade massiva e direta com relação ao consumidor [Hafberg04].

De maneira simples, pode-se definir advergames como a integração de propaganda (*advertisement*) e jogos eletrônicos [Wiki069]. Seu principal objetivo é disponibilizar mensagem publicitária para promover um produto, uma organização ou um ponto de vista [Chen01].

Embora a mídia gerada pelos *advergames* seja bastante recente, existe uma forte tendência de crescimento das oportunidades nesta área. O poder de imersão causado pelos jogos e a grande quantidade de capital destinada pelas empresas para fins publicitários indicam que haverá um grande crescimento neste setor [IGDA061].

Há algum tempo, alguns *sites* ofereciam jogos gratuitos e nas regiões próximas à região que o usuário poderia jogar havia alguns *banners* mostrando alguma mensagem publicitária. Contudo, a proposta imposta por um *advergame* é bem mais radical: O *advergame* incorpora diretamente no ambiente do jogo a marca publicitária (ou a mensagem da marca) e, à medida que o consumidor joga, o produto (ou a mensagem) aparece no jogo para que jogadores possam ver ou sentir os benefícios de possuir tal produto, por exemplo. Isto é, a mensagem publicitária é central para o *game-play* do jogo [Hafberg04].

Contudo, ao contrário de jogos de entretenimento, o principal objetivo de um *advergame* não é apenas prover diversão aos jogadores. O maior desafio é transmitir a mensagem publicitária de uma forma que possa justificar o retorno do investimento necessário para criar o jogo [Chen01].

Sendo assim, dentro de um ambiente de negócios extremamente complexo e dinâmico, todo projeto de desenvolvimento de um *advergame* torna-se crítico, instável (suscetível a mudanças) e com fortes restrições de cronograma e custo, como mostrado a seguir:

- **Marca**: É o fator mais importante para o cliente final de um *advergame*. O desenvolvedor (*game designer*, neste caso) precisa identificar rapidamente qual a mensagem a ser transmitida e o tema da campanha publicitária na qual o jogo estará inserido [IGDA061];
- **Cronograma**: Um dos aspectos mais críticos do desenvolvimento de *advergames* é a questão de prazo. Como o jogo, em geral, está inserido numa campanha publicitária, sempre haverá prazos fixos atrelados a campanhas em jornais, TVs e outras mídias. Por isso, um atraso na entrega dos produtos de um projeto de *advergame* pode ter conseqüências desastrosas para uma empresa [IGDA061];
- **Objetivos da Campanha**: Todo *advergame* deve ser capaz de atender objetivos básicos dentro de uma campanha. Entre eles [IGDA061]:
    - **Público**: o jogo deve ter a maior audiência possível. Assim, o jogo terá de ser extremamente fácil, permitindo ao jogador ser efetivo e eficiente. Em geral, devem ter curta duração e algum teor de humor, como forma de encorajar a distribuição viral do jogo;
    - **Experiência**: o jogo deve ser capaz de realizar uma profunda experiência de imersão com o jogador. Assim, o desenvolvedor deve focar na profundidade e variedade do *game-play*.

Um projeto de criação de um *advergame* consome em média de US$ 10mil a US$ 500 mil, a depender do escopo do jogo. Porém, o fator limitante de um projeto de advergame é, quase sempre, o cronograma. Além disso, um projeto de *advergame* é, normalmente, contratado por uma agência ou uma empresa (cliente final) para que uma empresa especializada na produção de jogos execute o projeto. Mas, de maneira geral, os clientes (seja a agência ou a

empresa-cliente final) entendem pouco de jogos e dificilmente sabem exprimir suas expectativas com relação ao produto final [IGDA061].

Normalmente, um *advergame* possui alguns componentes:

Figura 12 – Componentes de um advergame.

- **Comunidade**: Módulo voltado para a integração dos usuários do jogo;
- **Convide Amigo**: Módulo que objetiva fazer a divulgação *viral* do jogo, fazendo com que os atuais usuários possam convidar outros usuários, assim, aumentando o número de jogadores e, conseqüentemente, o público-alvo da campanha;
- **Ranking**: Módulo responsável por criar desafios para os usuários. Melhorar sua pontuação no *ranking* e estar entre os melhores pode ser motivador suficiente para que o usuário jogue inúmeras vezes, aumentando assim, o tempo de exposição da marca por usuário;
- **Cadastro**: Módulo que tem o objetivo de controlar o acesso de usuários ao jogo e estas informações podem ser utilizados para, posteriormente, criar relatórios sobre o perfil e comportamento dos usuários no jogo. Isto é muito importante tanto para o cliente que está expondo sua marca, quanto para a empresa desenvolvedora entender como melhorar seus jogos;
- **Sistema de Administração**: Módulo responsável pelo controle do jogo, pode ser utilizado para obter os relatórios sobre os usuários e também para configurar parâmetros do jogo;

- **Jogo**: Finalmente, o componente mais importante de um *advergame*, o jogo propriamente dito. Sem ele, os demais componentes não têm razão de existir.

Nem todos os projetos de advergames possuem estes componentes, no entanto, a experiência tem mostrado que praticamente todos advergames possuem uma estrutura de componentes similar à apresentada na Figura acima.

Atualmente, os maiores clientes de um *advergame* são as agências de publicidade [IGDA061], que apontam o *advergame* como uma nova sensação do mercado publicitário e – cada vez mais – jogos publicitários ganham importância dentro de uma campanha. Num projeto de criação de um *advergame* existem muitos envolvidos (diretamente) e podem ser classificados conforme a seguir:

- Cliente (Agência de Publicidade, por exemplo)
  - Gerentes de Negócios;
  - Gerentes de Marketing e Propaganda;
  - Consultores (Tecnologia, Jogos, etc.).
- Desenvolvedor
  - Gerentes de Projeto;
  - Gerentes de Negócio;
  - Programadores;
  - Artistas de Interface;
  - Músicos;
  - *Game Designers*;

Além disso, normalmente, o cliente não é especialista em jogos, por isso, a maneira de coleta de funcionalidades ocorre sem muita participação do cliente. Assim, uma agência sempre aponta uma demanda para uma empresa desenvolvedora de jogos (advergames) que, então, desenvolve um *game*

concept. Este *game concept* é um esboço inicial da "idéia" por trás do jogo. Caso este conceito seja aprovado pela agência, um segundo artefato é gerado pela empresa desenvolvedora: uma proposta. Esta proposta contém um detalhamento dos componentes que o jogo poderá conter e, obviamente, um orçamento para que seja criado o jogo. Finalmente, se a agência aprovar a proposta, então, o jogo deverá ser colocado em produção.

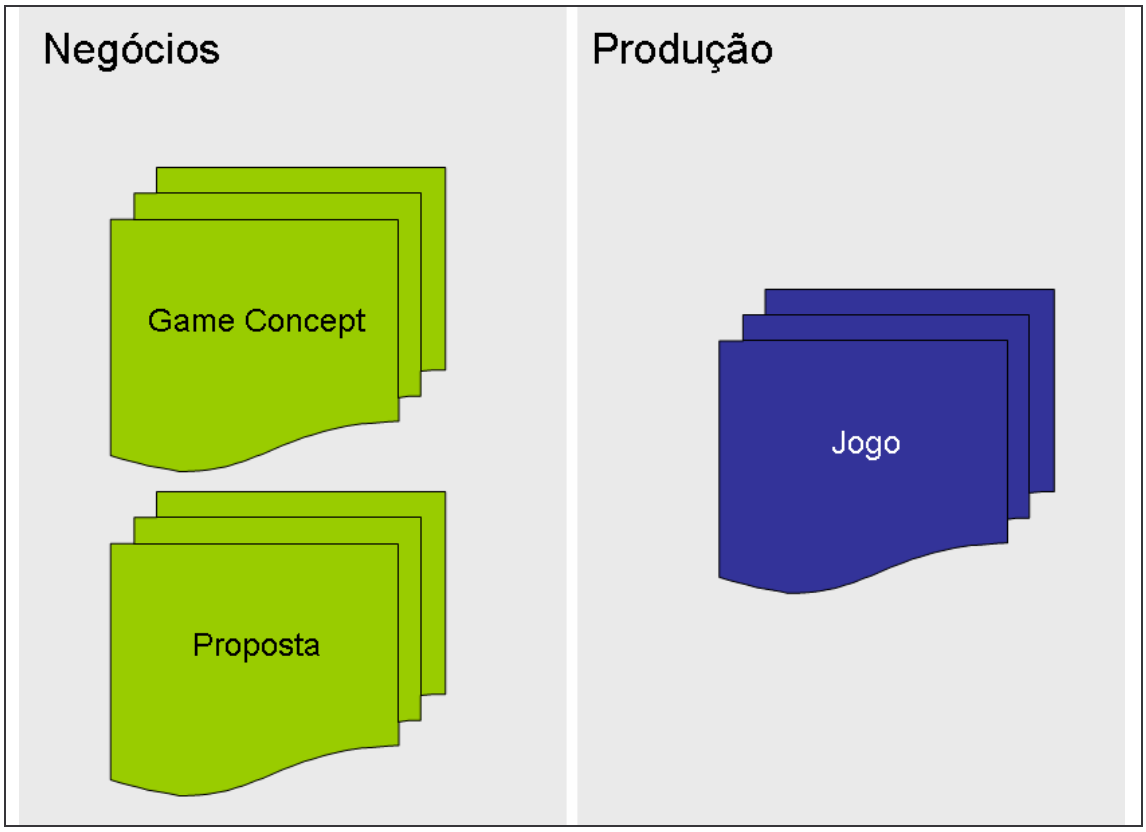


Figura 13 – Ambiente do Desenvolvimento de um Advergame.

Entretanto, esta abordagem apresenta problemas em sua natureza. Em muitos casos, o projeto se inicia apenas após a aprovação da proposta por parte da agência de publicidade. Isto faz com que – muitas vezes – as informações produzidas na fase de "Negócios" sejam ignoradas na fase de "Produção". Assim, essa falha de comunicação pode representar a origem de uma série de problemas que poderão ajudar a comprometer o sucesso de um projeto.

Outra importante característica inerente a todo desenvolvimento de *advergame* é que o ambiente de negócios em que está envolvido o projeto é extremamente dinâmico e, portanto, sujeito às mudanças.

Portanto, o desenvolvimento de um *advergame* deve seguir um processo (ou metodologia) capaz de sustentar toda produção (desde o fluxo de negócios) e fazer com que os índices de sucesso de projetos desenvolvidos por uma organização sejam satisfatórios.

4. Processos de Desenvolvimento de Jogos

"Failure is simply the opportunity to begin again,
this time more intelligently ".
- Henry Ford

Este capítulo visa discutir alguns dos processos e metodologias de desenvolvimento utilizados na criação de jogos e analisar seu alinhamento com as necessidades demandadas pela criação de um *advergame*.

Os processos de desenvolvimento de jogos foram – geralmente - inspirados em alguma metodologia ou modelo de ciclo de vida da indústria de software. Portanto, torna-se inevitável discutir sobre processos de desenvolvimento de jogos (*advergames*, em particular) sem abordar aqueles utilizados amplamente na criação de sistemas de informação.

4.1 Waterfall

O modelo de ciclo de vida em "cascata" ou *waterfall* tratou-se do primeiro modelo para desenvolvimento de software, por volta de 1970 [Sommerville03]. Posteriormente, o método passou a ser utilizado para criação de jogos também. Ele recebeu este nome devido à sua estrutura baseada em fases bem definidas que acontecem em seqüência linear, como mostrado abaixo:

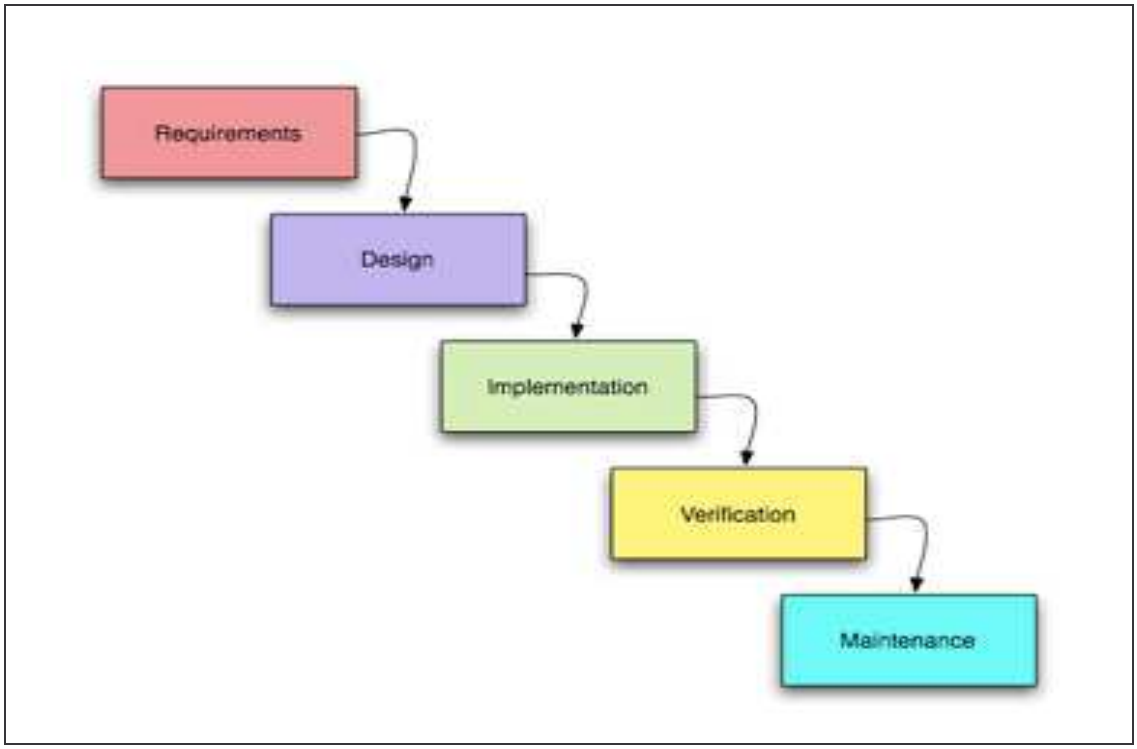


Figura 14 – Modelo Cascata.

O processo acontece fase a fase, como mostrado acima. Nem sempre, as fases exibidas acima são as utilizadas, depende bastante da organização e do produto que está sendo desenvolvido. Em geral, temos as seguintes fases [Sommerville03]:

- **Requisitos**: responsável por descobrir e especificar os requisitos necessários para o desenvolvimento do sistema;
- **Design**: responsável por analisar, descobrir algumas particularidades nos requisitos definidos e propor uma solução inicial e isolada para cada um deles. Além disso, descobrir uma estrutura robusta e capaz de realizar não apenas a realização dos requisitos individualmente, mas de suportar uma solução completa;
- **Implementação**: responsável por codificar e desenvolver a solução proposta para os requisitos definidos e analisados;
- **Verificação**: responsável por averiguar se a implementação condiz com tudo aquilo previamente definido. Nesta fase,

requisitos são validados e possíveis inconformidades são identificadas;

- **Operação e Manutenção**: responsável por implantar o software nas operações onde irá funcionar e realizar manutenções necessárias.

Além disso, a transição de uma fase para outra acontece através da geração de artefatos (comumente, um documento) e nenhuma fase pode ser iniciada sem que estes artefatos gerados na fase anterior possam ser aprovados. É preciso considerar que – idealmente – o cliente deve conhecer todos os requisitos e saber descrevê-los detalhadamente, para que mudanças nos requisitos não ocorram em outras fases.

Sendo assim, no desenvolvimento de jogos eletrônicos, quando se utiliza algum processo baseado no modelo de ciclo de vida em cascata, primeiramente, o *game designer* (ou vários *game designers*) cria um documento inicial contendo o conceito do jogo. Neste momento, há pouca interação com os demais membros da equipe responsável pelo jogo. Posteriormente, o documento é utilizado para extrair um conjunto de requisitos (funcionalidades ou características do jogo). Depois, os requisitos são detalhados em forma de "*assets*" de animação, programação, efeitos sonoros, arte e efeitos gráficos, qualidade, etc. Então, o desenvolvimento prossegue com cada parte da equipe focada apenas no seu trabalho. Ou seja, artistas preocupam-se apenas em produzir os "*assets*" de arte, programadores preocupam-se apenas com os módulos que têm de desenvolver, e assim por diante. Nesta forma de trabalho existe pouca ou quase nenhuma colaboração entre os membros da equipe.

Dessa forma, cada fase deste processo linear expõe o produto que está sendo desenvolvido a um conjunto limitado de envolvidos. Por exemplo, as pessoas responsáveis por executar os testes do jogo jamais verão o produto

até que a equipe de implementação tenha concluído suas tarefas. Portanto, a própria natureza do modelo de ciclo de vida introduz uma série de problemas ao desenvolvimento do produto, entre eles:

- **Feedback tardio**: Os clientes e membros da equipe têm visão limitada e passam grande parte do desenvolvimento sem opinar ou conhecer o estado do produto [Flood03];
- **Ausência de Colaboração**: Neste tipo de estrutura, cada indivíduo tem um papel bem definido e se preocupa apenas com suas tarefas, promovendo a falta de colaboração entre os membros da equipe [Schwaber04];
- **Inflexibilidade**: Por ser linear, sua capacidade de responder às mudanças de maneira ágil se torna praticamente nula, havendo um grande índice de re-trabalho [Sommerville03]. Isto pode gerar um "efeito cascata", fazendo com que muito trabalho de fases anteriores seja retomado e aprovado novamente, o que pode deixar o processo extremamente lento e caótico;
- **Orientação a Documento**: A transição entre as fases deste tipo de modelo de ciclo de vida se dá através da aprovação de artefatos, em geral, documentos [Sommerville03]. Ou seja, muitas horas, a equipe de desenvolvimento fica mais focada em produzir um documento para servir de defesa contra o cliente, do que em produzir valor, de fato, para o cliente.

Por sua vez, cada um destes problemas pode gerar um conjunto de conseqüências catastróficas para o sucesso de um projeto:

- **Escopo**: o produto final poderá não conter muitas funcionalidades previstas e esperadas pelos clientes;
- **Tempo**: o produto poderá ser entregue muito depois da data final definida no cronograma. No caso de um advergame (jogo publicitário), esta poderá ser a mais importante variável do

projeto, porque advergames fazem parte de uma campanha publicitária bem mais ampla;

- **Custo**: Diante das possíveis inconformidades de escopo e tempo (cronograma), o custo do projeto tende a ser maior que o valor previsto no início do mesmo.

Portanto, desenvolver software em ambientes de negócio bastante complexos e dinâmicos (mudanças inerentes), utilizando algum processo baseado no modelo de ciclo de vida em cascata, traz significativos problemas que podem influir de maneira decisiva no sucesso do projeto.

4.2 Processos Incrementais

A abordagem de desenvolvimento incremental foi sugerida no início dos anos 80, como alternativa ao modelo de ciclo de vida em cascata, com o objetivo de diminuir o re-trabalho e proporcionar aos clientes oportunidade de postergar algumas decisões, para que estas possam ser baseadas numa experiência com alguma versão intermediária do produto final [Sommerville03].

Assim, a idéia é que versões - cada vez mais completas - do produto final sejam entregues durante todo transcorrer do projeto. Para tanto, é preciso que o cliente identifique as funcionalidades e classifique-as de acordo com seu grau de importância. Então, as funcionalidades mais prioritárias são entregues antecipadamente aos clientes, agregando valor imediatamente aos negócios que dependem do produto que está sendo construído. Por isso, cada "estágio" do desenvolvimento deverá ser capaz de entregar um "incremento" ao cliente. Estes novos "incrementos" são integrados aos "incrementos" entregues em "estágios" anteriores, de modo que cada "estágio" seja capaz de melhorar a funcionalidade do sistema como um todo.

Entre os principais e mais difundidos processos baseados na abordagem incremental destaca-se o Processo Unificado, que se trata de um *framework* de processo de desenvolvimento de software. Ele foi desenvolvido a partir da experiência de desenvolvimento de software da Rational (e empresas que a Rational incorporou) ao longo de cerca de vinte anos. Durante este tempo foi possível identificar algumas das principais causas do insucesso de alguns projetos de desenvolvimento de software [Wiki066]:

- **Gerenciamento Ad-hoc de Requisitos**: O processo de gerenciamento de requisitos é pouco padronizado e desorganizado;
- **Comunicação Falha**: Os canais e maneiras de comunicação permitiram ambigüidades e imprecisão em todos os pontos do desenvolvimento;
- **Arquitetura Frágil**: A arquitetura, isto é, a estrutura da solução proposta para o software mostrava-se extremamente frágil e pouco robusta;
- **Inconsistências**: Houve inconsistências nos requisitos, design e implementação;
- **Testes Insuficientes**: A quantidade e a cobertura dos testes durante o desenvolvimento realizados foram insuficientes e ineficazes;
- **Controle Subjetivo**: O controle do status do projeto era realizado qualitativa e subjetivamente, sendo impossível obter números precisos sobre o status do projeto;
- **Gerenciamento de Riscos Falho**: Os riscos não foram devidamente identificados, e atacados. Dessa forma, qualquer atitude tomada era meramente reativa, e não proativa;
- **Propagação de Mudanças Sem Controle**: Havia sempre muitas mudanças que não eram devidamente tratadas por um sistema

de controle de mudanças, fazendo com que o caos pudesse emergir no desenvolvimento;

- **Automação Insuficiente**: O desenvolvimento não era devidamente suportado por ferramentas que pudesse ajudar o processo a ser eficiente (automatizado).

Assim, ao mesmo tempo em que foram identificadas as causas de insucesso de projetos de software, foram testadas "as melhores práticas" de desenvolvimento de software moderno [RUP]:

- **Desenvolvimento Iterativo e Incremental**: Esta abordagem permite "dividir para conquistar", transformando problemas complexos em vários outros problemas menores, que podem ser resolvidos de maneira mais simples;
- **Gerenciamento de Requisitos**: Permite que o problema possa ser analisado, as necessidades de todos envolvidos sejam consideradas, o sistema possa ser definido, o escopo do sistema possa ser gerenciado, que a definição do sistema possa ser melhorada e que mudanças nos requisitos possam ser controladas. Tudo isto faz com que o processo de requisitos seja padronizado e claro para todos envolvidos.
- **Arquitetura Componentizada**: A componentização da arquitetura faz com que esta seja facilmente extensível, compreensível e reusável;
- **Modelagem Visual**: Abstração é uma eficiente forma de construir soluções. A utilização de representação gráfica ajuda a comunicar e entender melhor a representação da solução proposta. Um modelo representa, ao mesmo tempo, a visualização e a simplificação de um design complexo. Existem vários modelos como diagramas de caso de uso, diagramas de seqüência, diagramas de estados, diagramas de classe, entre

outros. Todos utilizam a notação UML, um padrão para modelagem visual de software orientado a objetos;

- **Contínua Verificação da Qualidade**: A verificação da qualidade é uma das falhas mais comuns em projetos de software, visto que muitas vezes é postergada para o final do processo ou realizada por outro time que não desenvolveu o produto. O processo unificado assume que cada membro do time é responsável pela qualidade e provê *workflows* de testes para medir esta qualidade;
- **Controle de Mudanças**: Todos os projetos de software sofrem mudanças. O Processo Unificado define métodos para controlar, acompanhar e monitorar mudanças.

Além disso, o Processo Unificado fornece um conjunto de relações entre atividades, artefatos, e papéis para que uma organização possa usá-los conforme suas necessidade e cultura Estes artefatos, atividades e papéis encontram-se distribuídos em diferentes disciplinas que buscam tratar diferentes aspectos do desenvolvimento [RUP]:

- **Modelagem de Negócios**: Objetiva estabelecer um melhor entendimento e comunicação entre a engenharia de negócios e a engenharia de software. Isto é, os engenheiros de software devem entender a dinâmica e a estrutura da organização em que o software funcionará, os problemas existentes e possíveis soluções;
- **Requisitos**: Objetiva descrever o que o sistema deve fazer e permite que desenvolvedores e clientes concordem com esta descrição;
- **Análise e Design**: Busca mostrar como o sistema será desenvolvido na fase de implementação;
- **Implementação**: Busca definir a organização e implementação do código através de classes, objetos e componentes, além de

testá-los (unitariamente) e integrá-los do modo que produza uma versão executável do sistema;

- **Testes**: É responsável por verificar a interação entre objetos, a integração de todos componentes do software, a conformidade dos requisitos implementados (em relação aos requisitos descritos), identificar e assegurar que defeitos sejam tratados antes da implantação do software;
- **Implantação**: É responsável por produzir versões externas do software, entregar o software aos seus clientes finais e colocá-lo em funcionamento no ambiente de produção. Além de prover toda ajuda e assistência aos usuários do sistema;

Além destas, existem disciplinas auxiliares ao desenvolvimento de software dentro do Processo Unificado, tais como Gerência de Configuração e Mudanças, e Gerência de Projetos.

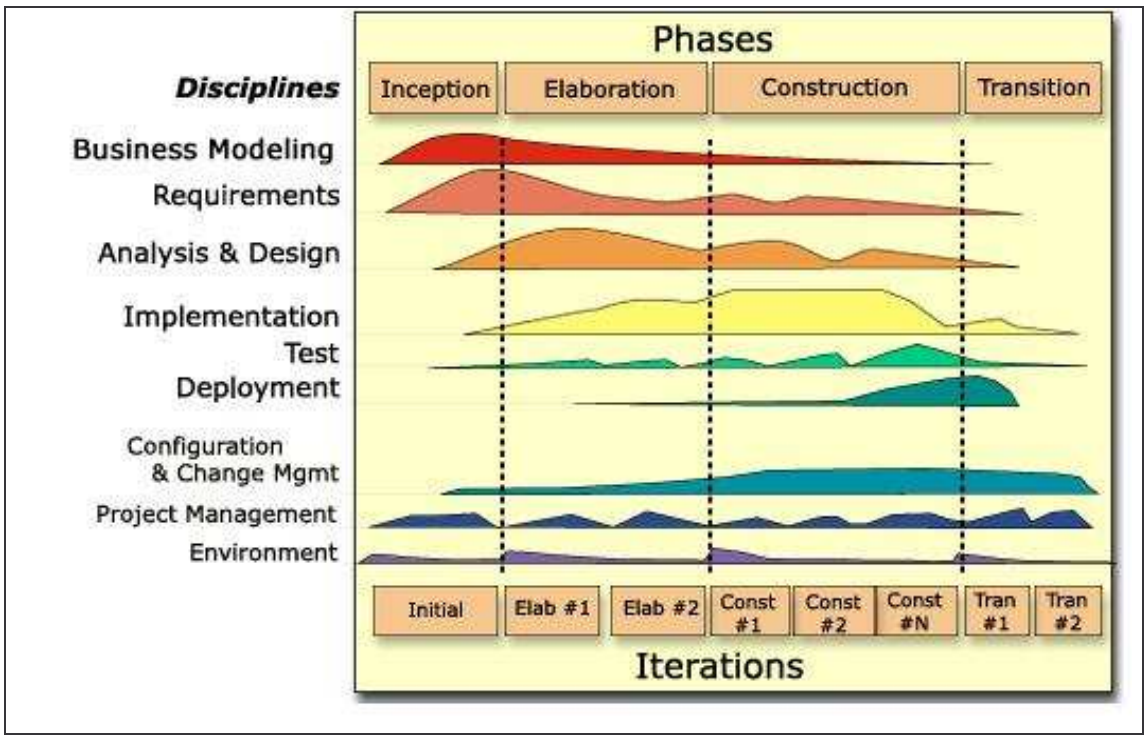


Figura 15 – Processo Unificado Overview.

Estas disciplinas ocorrem com maior freqüência de acordo com a fase em que o desenvolvimento esteja. Assim, cada disciplina é tratada e ocorre durante todo o projeto, no entanto, com ênfase diferente de acordo com a fase.

Cada fase de um ciclo do Processo Unificado possui características bem distintas:

- **Concepção**: Marcada pelo estabelecimento do *business-case* que inclui contexto do negócio, fatores de sucesso (retorno esperado, reconhecimento de mercado, etc.), e tendência financeira. Além disso, é criado um modelo de casos de uso básico, plano de projeto, avaliação inicial dos riscos, e descrição do projeto;
- **Elaboração**: É responsável por "dar forma" inicial ao projeto. Realiza a análise do problema do domínio e a arquitetura do projeto obtém sua estrutura básica;
- **Construção**: Objetiva desenvolver os componentes e outras funcionalidades do sistema projetado. Esta é a fase que possui maior ênfase em codificação. Em projetos muito grandes esta fase pode ter muitas iterações, como forma de dividir os casos de uso em partes gerenciáveis e que possam gerar protótipos demonstráveis;
- **Transição**: É responsável pela mudança do produto do ambiente de desenvolvimento para a organização-alvo (usuários finais). As atividades desta fase incluem treinamento dos usuários finais, manutenção do produto e "beta-testes" do sistema para validá-lo diante das expectativas dos usuários.

Portanto, o Processo Unificado oferece um conjunto de mecanismos e ferramentas, como o gerenciamento de riscos e o gerenciamento de requisitos, para lidar com o desenvolvimento de software complexo, fazendo com que as probabilidades de sucesso de um projeto de software sejam maiores. Entretanto, a natureza do processo se mostra extremamente linear, o que pode tornar o processo pouco flexível. Além disso, o processo sugere a criação de uma série de artefatos que não focam na criação de valor mais rapidamente para o cliente final.

4.3 Metodologias Ágeis

Em 2001, um grupo de renomados profissionais da indústria de software se reuniu em Utah (EUA) para discutir novas formas de desenvolver software, em detrimento à antiga ordem marcada por processos excessivamente documentados e "pesados". Em comum, todos possuíam experiência com metodologias alternativas que buscavam ser "ágeis" [Agile01].

Neste encontro, se decidiu adotar o nome "ágil" designar uma metodologia de desenvolvimento de software que respeitasse estivesse de acordo com o Manifesto Ágil, o principal resultado da reunião. O Manifesto Ágil é uma declaração explícita de valores [Agile01]:

Indivíduos e interações sobre processos e ferramentas
Software funcionando sobre documentação detalhada
Colaboração do cliente sobre negociação de contrato
Responder a mudanças sobre seguir um plano

Enquanto as metodologias tradicionais de desenvolvimento de software procuravam valorizar os itens à direita, as metodologias ágeis buscam enfatizar valor nos itens à esquerda. Além destes valores, o Manifesto ainda inclui um conjunto de princípios [Agile01]:

- Nossa maior prioridade é satisfazer o cliente através da rápida e contínua entrega de software de valor;
- Mudanças (de requisitos) são bem vindas, mesmo que tardiamente no processo de desenvolvimento. Processos ágeis tiram se aproveitam da mudança para vantagem competitiva do cliente;
- Entrega freqüente de software funcionando, entre duas semanas a dois meses, com preferência na menor escala de tempo;

- Pessoas de negócios e desenvolvedores devem trabalhar juntas diariamente durante o desenvolvimento dos projetos;
- Construa projeto em torno de pessoas motivadas. Dê-lhes o ambiente e todo suporte necessário, e confie que eles podem realizar o trabalho;
- A mais eficiente e eficaz maneira de obter informação sobre e para a equipe de desenvolvimento é a conversa franca, aberta e cara a cara;
- Software funcionando é medida primária de progresso;
- Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um passo sustentável indefinidamente;
- Contínua atenção com a excelência técnica e de projeto melhoram a agilidade;
- Simplicidade – a arte de maximizar a quantidade de trabalho não feito – é essencial;
- As melhores arquiteturas, requisitos, e design surgem em times auto-organizáveis;
- Em intervalos regulares, o time deve refletir sobre como se tornar mais efetivo. Assim, melhora seu comportamento de acordo com as decisões tomadas.

Existem várias metodologias de desenvolvimento que possuem tais valores e práticas como fundamentos, entre elas:

- *Extreme Programming;*
- *Scrum;*
- *Lean Software Development;*
- *Feature Driven Development*.

Neste trabalho, serão discutidos aspectos de duas das mais proeminentes metodologias ágeis: *Extreme Programming* e *Scrum*. A primeira

enfatiza aspectos de desenvolvimento, a segunda foca em aspectos gerenciais do desenvolvimento, sendo, portanto, complementares.

4.3.1 Extreme Programming (XP)

Em meados de 1996, Kent Beck foi designado como líder de um projeto na Chrysler (indústria de automóveis) para desenvolver um sistema de folha de pagamento interna. Ele também seria responsável por melhorar a metodologia de desenvolvimento de software da organização. Em 1999, foi publicado o livro "Extreme Programming Explained", de autoria de Kent Beck [Beck99]. Este livro descreve XP como:

- Uma tentativa de reconciliação entre humanidade e produtividade;
- Um mecanismo para mudança social;
- Um caminho para melhoria;
- Um estilo de desenvolvimento;
- Uma disciplina de desenvolvimento de software.

Extreme Programming é das mais difundidas metodologias ágeis de desenvolvimento de software. Ao contrário das metodologias tradicionais, XP valoriza a adaptabilidade em detrimento à predictibilidade. Os idealizadores de XP acreditam que mudanças de requisitos são inerentes no desenvolvimento de software, e, por isso, é preferível adotar uma abordagem flexível.

Entre as principais diferenças de XP em relação às metodologias tradicionais encontra-se uma declaração de valores:

Figura 16 – Valores de XP.

- **Comunicação**: É um dos valores mais importante em qualquer trabalho em equipe. A maior parte dos problemas ocorre porque "alguém esqueceu de avisar alguém sobre alguma coisa". É comum que os "desenvolvedores reportem algum fato aos gerentes e estes os punam". Assim, desencorajando qualquer tipo de comunicação. XP busca a comunicação franca através de desenvolvedores e gerentes através de um conjunto de práticas que só podem ser realizadas com bastante comunicação;
- **Simplicidade**: XP procura sempre fazer com que a equipe pense sobre: "Qual a coisa mais simples que possa funcionar?". Simplicidade não é fácil. É muito difícil não olhar o que você

precisa implementar amanhã ou próxima semana. Pensar – compulsivamente – adiante faz com que surja o medo do custo exponencial da mudança. XP entende que é melhor fazer uma coisa simples hoje e gastar um pouco para ajustá-la (se necessário), do que construir uma coisa extremamente complicada que pode ser jamais utilizada. Simplicidade e comunicação podem ser mutuamente combinadas: Quanto mais comunicação, mais claro poderá ser enxergado sobre o que precisa ser feito e o que não precisa. Isto é, quanto mais comunicação, mais simples discernir entre o que é simples e o que não é;

- **Feedback**: Conhecer o real estado atual do sistema é uma das maneiras mais eficazes de exercer controle sobre o desenvolvimento. Para tanto, casos de testes podem ser escritos e sempre executados para avaliar se o sistema se encontra em conformidade. Em geral, o otimismo é um grande mal da programação. *Feedback* é o tratamento para este mal. Feedback pode funcionar em várias escalas de tempo. Primeiramente, na escala de minutos e dias, através da criação de *testes unitários* para todas as unidades lógicas do sistema que possam falhar. Assim, os desenvolvedores obtêm – minuto a minuto – *feedback* concreto sobre o estado do sistema. Quando clientes escrevem novas *estórias* (*User Stories*), os desenvolvedores – imediatamente – estimam-nas para que os clientes possam ter *feedback* sobre a qualidade de suas estórias. Pessoas que acompanham o progresso do trabalho monitoram a realização das tarefas para dar ao time *feedback* (uma visão geral) informando se eles tendem a acabar o trabalho que se comprometeram no período de tempo definido. Além disso, *feedback* funciona na escala de semanas e meses. Os clientes e testadores escrevem testes funcionais para todas as estórias implementadas pelo sistema.

Assim, com estes testes implementados, é possível que testadores e clientes obtenham *feedback* concreto e real sobre o estado do sistema. Os clientes também podem revisar o cronograma em períodos curtos de tempo (duas ou três semanas) para avaliar se a velocidade de evolução do trabalho do time coincide com o plano. Se necessário, são realizados ajustes. Outra importante forma de obter *feedback* é colocar o sistema em produção o mais cedo possível. Então, uma das estratégias no planejamento é colocar em produção as estórias mais prioritárias (que têm maior valor do ponto de vista do cliente). Isto fornece *feedback* aos desenvolvedores com respeito à qualidade das decisões tomadas. Portanto, *feedback* concreto trabalha juntamente com a comunicação e a simplicidade. Quanto mais *feedback* você tem, mais fácil é comunicar. Por exemplo, se alguém tem uma objeção em relação a algum código construir por outra pessoa, ao invés de discutir por indefinidas horas, basta que seja mostrado um caso de teste em que o código falha. Se todos os testes executam com sucesso, então, o desenvolvimento pode ser considerado como finalizado;

- **Coragem**: Muitas práticas de XP reforçam a coragem. Uma delas é sempre codificar e fazer design pensando apenas no que deve ser realizado hoje. Este esforço é muito importante para evitar atribulações para projetar (design) e codificar uma grande quantidade de funcionalidades. A coragem permite aos desenvolvedores sentirem-se confortáveis em ter de *refatorar* seus códigos quando necessário, ou seja, reavaliar o sistema existente e mudar sua estrutura para que possa se tornar mais flexível, organizada, compreensível, etc. Outra manifestação de coragem se dá quando é necessário "jogar fora" algum código que esteja obsoleto, independente de quanto esforço tenha sido gasto para criar aquele pedaço de código. Coragem também

significa persistência. Por exemplo, se um programador se vir tomado por um problema por um dia inteiro, se ele for persistente, pode facilmente resolver o problema no outro dia;

- **Respeito**: Se manifesta através de várias formas. Os membros de um time respeitam-se uns aos outros porque ninguém jamais submeterá um código que não *compila*, que faz com que os testes unitários já implementados falhem, ou jamais atrasará o trabalho de seus parceiros. Os membros respeitam seus trabalhos através da exigência de alta qualidade e busca pelas melhores soluções. Portanto, o respeito é um valor que une todos os demais valores.

Além disso, XP descrever quatro atividades fundamentais que devem ser realizadas por cada membro de uma equipe:

- **Codificar**: XP entende que codificar é a atividade capaz de produzir valor para o cliente. Sem codificação não há nada. No entanto, o conceito de codificação preconizado por XP é bem mais amplo que o convencional. Codificar pode ser desenhar diagramas que poderão gerar código, construir *scripts* para um sistema *web* ou codificar um programa que precisa ser compilado. Codificação também pode ser utilizando para descobrir qual a melhor solução para um problema através da programação das diferentes soluções e validação delas diante de um conjunto de casos de teste. Além disso, uma das maneiras mais eficientes de se comunicar é através de código porque - mesmo diante de problemas complexos - é possível discutir sobre eles através de código obtendo feedback a partir de outros programadores.
- **Testar**: Não existe certeza sem testes. Quando um pedaço de código é construído, a única maneira de saber se ele realmente implementa o que havia sido previamente definido. O

programador deve definir um conjunto de casos de testes unitários em que o código poderá falhar. Se, no entanto, todos os testes unitários funcionarem, então, o código pode ser testado diante de testes de aceitação (definidos pelo cliente).

- **Ouvir**: Programadores não devem – necessariamente – entender sobre o negócio do sistema em desenvolvimento. As funcionalidades do sistema devem ser determinadas pelas pessoas de negócio. Os desenvolvedores, no entanto, precisam ouvir – sempre – atentamente as pessoas de negócios. Isto para entender sobre o problema e também para ajudar a estas pessoas melhorarem seu entendimento sobre o problema.
- **Projetar**: Inicialmente, o projeto pode andar muito bem sem que haja preocupação com design (projetar). No entanto, a medida que o sistema cresce (em complexidade), deve haver esforços para organizar melhor a estrutura do sistema que está sendo construído, permitindo que boas características de arquitetura sejam inseridas, como componentização, legibilidade, reusabilidade, etc. Portanto, um dos mais importantes mecanismos de projetar um sistema é a refatoração, que permite que – continuamente – o design do sistema possa ser melhorado.

Dessa forma, os valores de XP e suas atividades básicas são implementados por um conjunto de práticas que se destacam entre as práticas de maior sucesso na engenharia de software. Entre elas:

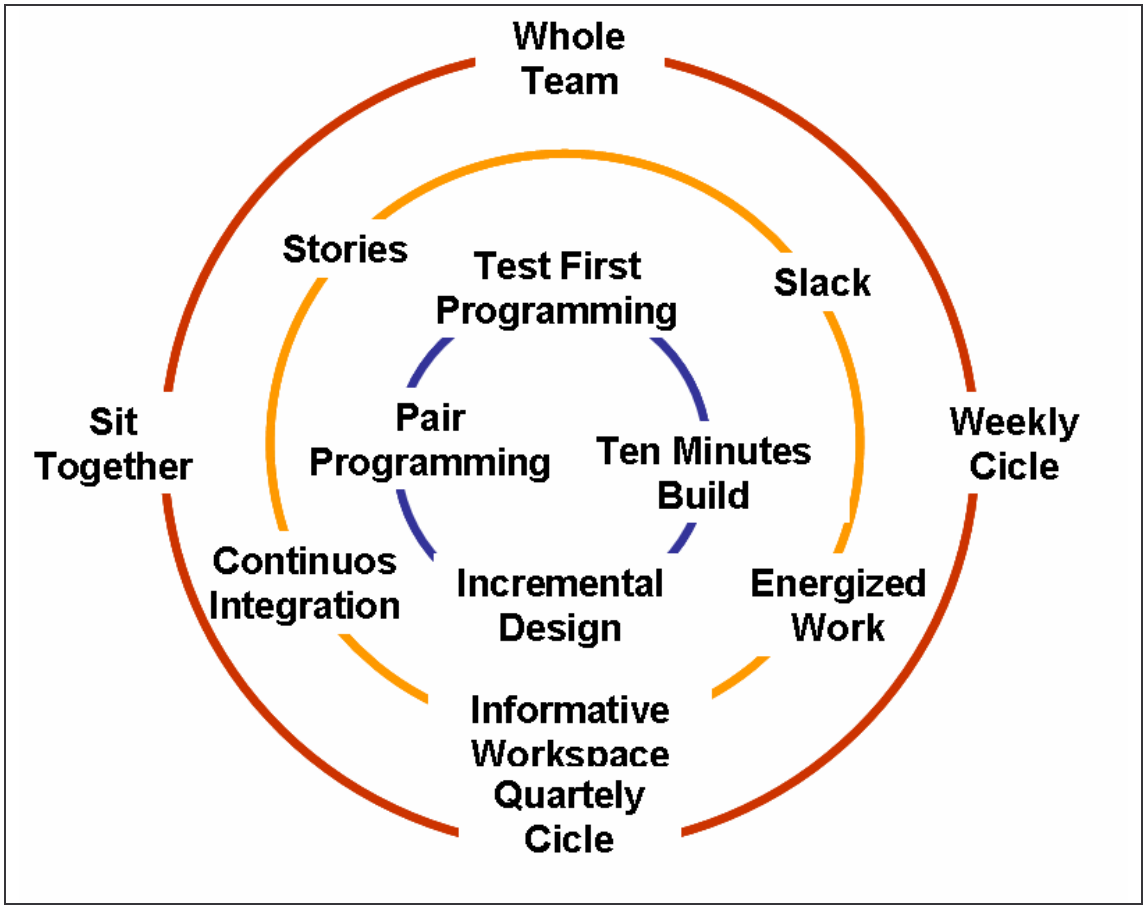


Figura 17 – Práticas de XP.

Posteriormente, no próximo capítulo, cada uma dessas práticas que for utilizada na metodologia de desenvolvimento proposta será discutida mais detalhadamente.

4.3.2 Scrum

O nome *Scrum* tem origem no esporte conhecido com *rugby*, onde os indivíduos colaboram entre si para alcançar o objetivo, mover a bola contra o campo adversário [Rising00]. *Scrum* é uma metodologia ágil que busca uma forma empírica de lidar com o caos, em detrimento a um processo bem definido [Schwaber04]:

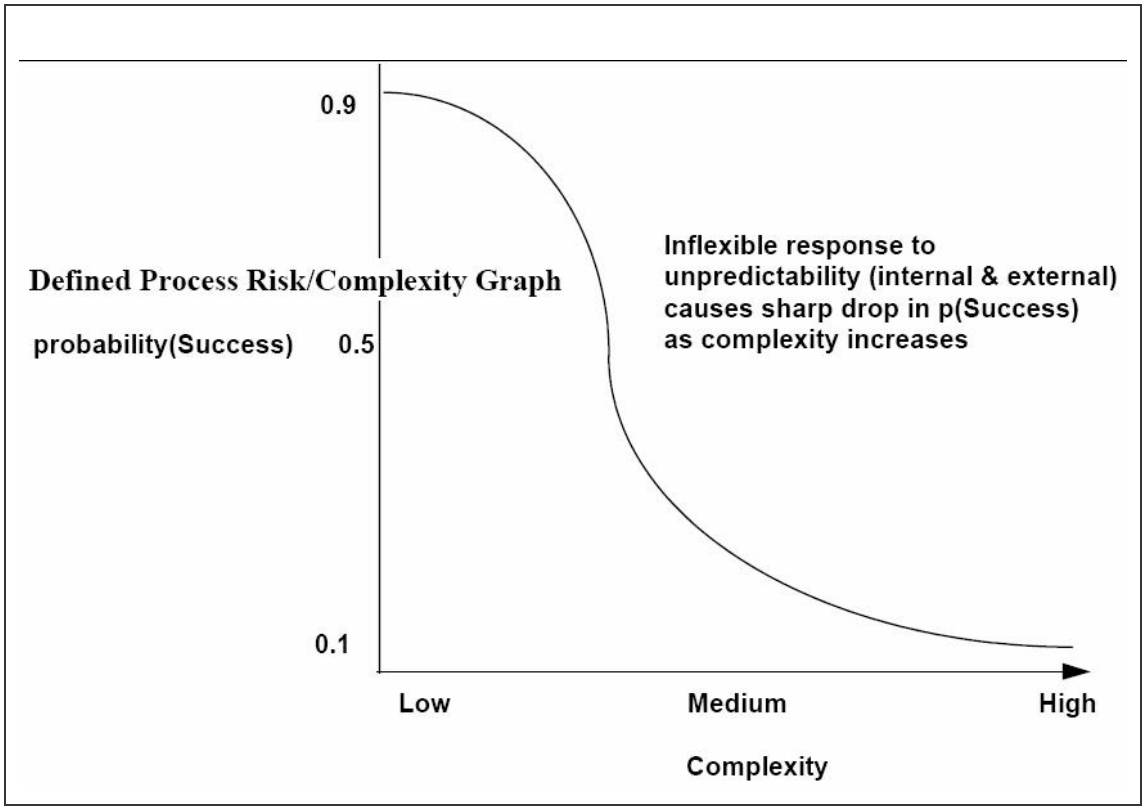


Figura 18 – Complexidade e Probabilidade de Sucesso com Métodos Preditivos.

Em ambientes complexos, um "processo bem definido" tem sua probabilidade sucesso drasticamente reduzida quando aplicado ao desenvolvimento de um produto. Uma das principais idéias do *Scrum* para atacar este problema é a implantação de um controle descentralizado, que é capaz de lidar mais eficientemente com contextos pouco previsíveis [Schwaber04].

Para tanto, o gerenciamento é distribuído através de três agentes independentes [Beedle04]:

- ***Product Owner***: É o responsável pelo retorno sobre o investimento do projeto. Seu papel é garantir que o produto entregue atenda aos anseios do patrocinador do projeto e priorizar quais funcionalidades devem ser entregues primeiramente, por agregar mais valor ao mesmo;

- **Time**: São todos integrantes comprometidos em desenvolver o produto alvo do projeto, de acordo com suas próprias decisões para alcançar os objetivos estabelecidos;
- ***ScrumMaster***: É responsável por guiar o time, intermediar negociações entre o *Product Owner* e o Time. Seu principal papel é ensinar e acompanhar a utilização do *Scrum*.

Embora o principal foco seja entregar valor ao cliente, os membros de um time de projeto *Scrum* devem utilizar alguns artefatos como apoio ao gerenciamento descentralizado e simples [Schwaber04]:

- ***Product Backlog***: É uma lista priorizada (pelo *Product Owner*) de todas as funcionalidades que o produto deve conter;
- ***Sprint Backlog***: É uma lista das funcionalidades mais relevantes, quebrada em tarefas com suas respectivas estimativas de duração do presente momento, até o fim do *Sprint*, isto é, da iteração. É definida pelo time, e negociada com o *Product Owner.* O time deve procurar manter esta lista sempre atualizada;
- ***Impediment List***: É uma lista que contém os impedimentos para o time alcançar os objetivos. Tais impedimentos devem ser resolvidos pelo *ScrumMaster*.
- ***Reports***: São relatórios capazes de mostrar o andamento do projeto. Existem vários, como o *Bug History, Bug Count, Impediment List Report*, etc. No entanto, destacam-se:
    - ***Sprint Burndown***: Permite visualizar a velocidade e o progresso do trabalho das atividades no *Sprint* atual. Serve para motivar a equipe e ajudar na tomada de decisões de planejamento. Sua atualização é feita diariamente (um ou mais vezes) pelos integrantes do Time;
    - ***Product Burndown***: Permite uma visualização do projeto como um todo, não apenas do *Sprint*, informando as funcionalidades que têm sido entregues. Serve de apoio às

decisões relativas ao projeto como um todo, não apenas restritas ao *Sprint*.

Além destes, o principal artefato que um time de Scrum pode entregar ao final de um *Sprint* (iteração) é um *Product Increment*, que se trata - idealmente - do conjunto de funcionalidades definidas para aquele *Sprint* e que pode ser considerado uma versão intermediária do produto final.

Outras características do *Scrum* é que o processo é extremamente simples e adaptativo. Em um projeto, existem basicamente as seguintes etapas [TeamSystem]:

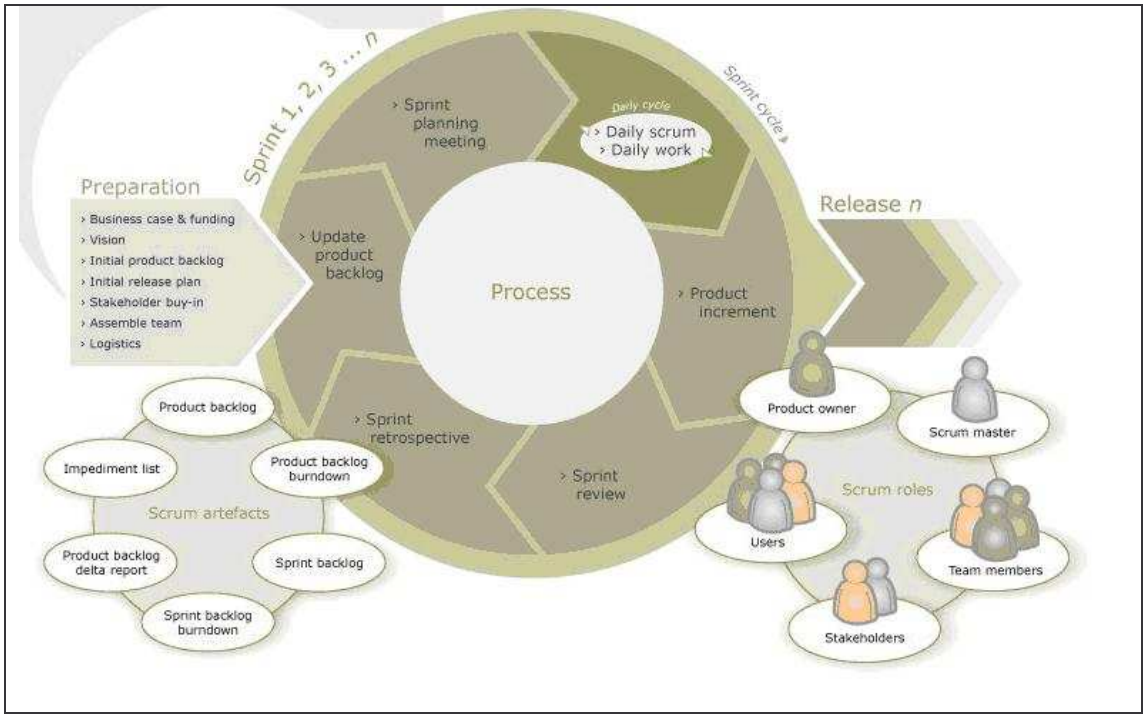


Figura 19 – Scrum Overview [TeamSystem].

- **Preparação**: Onde se define *o business-case*, *visão*, *Product Backlog* inicial e outras premissas ligadas aos projetos;
- **Sprints**: São iterações realizadas, uma após a outra, para entregar gradativamente funcionalidades capazes de agregar valor ao negócio do cliente. Dentro de cada *Sprint* são realizadas algumas atividades:
    - ***Scrum Planning Meeting***: É o início de um *Sprint*, onde o *Product Owner* tem a oportunidade de atualizar a priorização dos itens

do *Product Backlog* e definir juntamente com a equipe um *Product Increment* a ser entregue ao cliente ao final do *Sprint*;

- ***Daily Scrum Meeting***: É um encontro diário realizado pelo time onde os membros discutem: tarefas em que trabalharam, tarefas em que irão trabalhar, e possíveis impedimentos que estejam atrapalhando o progresso do trabalho. Este encontro é uma maneira eficiente de manter os membros cientes dos objetivos e impedir que o projeto "saia do rumo";
- **Criação do *Product Increment***: A finalização das funcionalidades definidas para um determinado *Sprint* marca a realização de um *Product Increment*;
- ***Sprint Review***: Neste momento, o time exibe o *Product Increment* construído ao *Product Owner*, que é responsável por validar e/ou solicitar ajustes para que o produto se torne adequado aos anseios do cliente;
- ***Sprint Retrospective***: Objetiva identificar os pontos positivos e negativos do *Sprint* que entregou o último *Product Increment* e busca corrigir os problemas encontrados.
- **Atualização do *Product Backlog***: O *Product Owner* é responsável por re-priorizar toda lista de itens do *Product Backlog* para que um próximo *Sprint* possa ser iniciado motivado pelos mais prioritários itens.

- **Encerramento**: É a última etapa de um projeto utilizando *Scrum*. Ocorre após a finalização de todos *Sprints* e é marcada pela entrega do produto final que motivou a criação do projeto.

Portanto, é sobre tudo isto que trata *Scrum*. É sobre reconhecer que mudanças são inerentes e aceitá-las. É sobre o desafio de fazer a ordem emergir do caos. E, principalmente, é sobre promover a maturidade das pessoas e satisfazer as necessidades do cliente.

4.4 Análise das Metodologias Existentes

O desenvolvimento de advergames demanda uma série de particularidades que dificilmente são consideradas pelas metodologias existentes para criação de jogos (ou software, de mais maneira mais ampla).

A criação de um advergame sempre está inserida num ambiente dinâmico e complexo de negócios. Sendo, portanto, sujeito às mudanças provenientes do ambiente. Além disso, todo projeto de advergame envolve fortes restrições de cronograma.

Outro importante fator é que todo desenvolvimento de jogos é realizado por uma equipe multidisciplinar composta por pessoas de negócios (clientes, publicitários, etc.) e de produção (programadores, game designers, artistas de interface, músicos, etc.), assim, exigindo formas eficazes de comunicação para que todos possam se manter informados sobre o andamento e as necessidades do produto que está sendo criado pelo projeto.

Assim, para uma empresa que produz advergames é preciso que exista uma metodologia que suporte o desenvolvimento e gerenciamento de projetos de advergames respeitando as particularidades da criação do produto, os diferentes *stakeholders*, e integrar a visão de negócios à de produção. Do contrário, as estatísticas relativas à produção de advergames poderão ser ainda mais assustadoras que os números trazidos pelo Chaos Report.

Dessa forma, analisam-se as metodologias descritas aqui através de vários critérios:

I. **Flexibilidade**: Trata-se da capacidade de lidar com ambientes dinâmicos e complexos, estando apto para responder às mudanças, quando necessário;

II. **Comunicação**: Trata-se da capacidade de integrar toda a equipe (negócios, produção) e os envolvidos no projeto (*stakeholders*) através de mecanismos de comunicação simples e eficazes, fazendo com que todos estejam informados sobre o andamento e as necessidades do projeto;

III. **Suporte à Multidisciplinaridade**: Trata-se da capacidade de integrar indivíduos de diferentes especializações numa mesma equipe;

IV. **Gerenciamento Descentralizado**: Trata-se da capacidade de delegar algumas decisões a agentes independentes mais aptos a escolherem o melhor caminho para alcançar determinado objetivo;

V. **Tratamento de Riscos**: Trata-se da capacidade de agir proativamente para evitar que os riscos se concretizem e, quando eles ocorrerem, estar apto para responder a eles;

VI. **Valor:** Trata-se da capacidade de gerar valor antecipadamente durante o processo de desenvolvimento;

VII. **Suporte ao Desenvolvimento**: Trata-se da capacidade de prover boas práticas, nas diferentes áreas, para suportar o desenvolvimento.

	I	II	III	IV	V	VI	VII
Waterfall	•	•	••	•	•	•	•
PU	••	••	•••	•	•••	••	••
XP	••••	••••	•••	•••	••••	••••	••
Scrum	••••	••••	••••	••••	••••	•••	•
• Ruim •• Regular ••• Bom •••• Excelente							

Tabela 1 – Análise das Metodologias.

Além disso, é possível analisar – separadamente - as adaptações das metodologias supracitadas através dos seguintes trabalhos:

- ***Paper Burn - Game Design with Agile Methodologies*** [Mencher06]: Sugere o Scrum como metodologia de desenvolvimento de jogos, no entanto, a visão da aplicação do *Scrum* encontra-se bastante limitada pela visão de *Game Design*. Assim, poderia ter uma maior ênfase em aspectos de organização e auto-gerenciamento da equipe e focalizar nas demais áreas, como arte e programação;
- ***Game Unified Process*** [Flood03]: Se baseia no modelo do Processo Unificado para o desenvolvimento de jogo e a única inovação é a inclusão de algumas práticas de *Extreme Programming*, como *Pair Programming*. Ou seja, traz consigo a maior parte dos problemas do Processo Unificado;
- ***Extreme Game Development – Right on Time, Every Time*** [Demachy03]: É uma adaptação do Extreme Programming, estando, portanto, orientada a enfatizar os aspectos de programação. Seria necessária uma maior discussão sobre os demais aspectos do desenvolvimento de um jogo, como gerenciamento, arte e *game design*;
- ***Waterfall*** [Flynt05]: Se baseia no modelo de ciclo de vida em cascata e traz consigo todos os problemas existentes no modelo em cascata, como ausência de colaboração, inflexibilidade, *feedback* tardio, etc.

Portanto, diante dos resultados obtidos acima, conclui-se que uma metodologia para desenvolvimento de advergames deve ser baseada nos valores e princípios do manifesto ágil. Além disso, será preciso considerar as melhores práticas – alinhadas a estes valores – que são capazes de tratar dos problemas demandados pela criação de um advergame.

5. Agile Game Process, Uma Nova Ordem

Prediction is very difficult, especially if it's about the future.
- Niels Bohr

Este capítulo é responsável por discutir e sugerir uma nova metodologia, chamada *Agile Game Process* (AGP), baseada em soluções existentes, como forma de melhorar o desenvolvimento e gerenciamento de projetos de jogos (em particular, advergames), aumentando suas chances de sucesso.

"Sucesso". Provavelmente deve ser este o objetivo de toda metodologia sugerida para projetos. No entanto, esta palavra pode soar de maneira subjetiva para algumas pessoas. Isto é, a definição de sucesso não está - em muitos casos - precisamente definida. Muitos podem interpretar que sucesso é entregar o produto ao cliente, outros podem pensar que é respeitar o cronograma, alguns podem defender que é atender às restrições de orçamento. Uma outra visão pode entender que sucesso é superar as expectativas do cliente ou mesmo satisfazer seus colaboradores. Existem outros possíveis critérios: ROI (*Return on Investiment*), importância estratégica, etc. Enfim, todo projeto tem um conjunto de envolvidos (*stakeholders*) e todos possuem diferentes expectativas com relação a um produto (ou serviço) criado por um projeto. Por isso, pode ser um desafio extremamente difícil conciliar todas essas expectativas numa visão de projeto que possa trazer satisfação a todas as partes envolvidas.

A metodologia proposta neste trabalho defende que sucesso é realizar o projeto que melhor possa atender as expectativas de todos *stakeholders*. Isto é, fazer com que o projeto responsável pela criação do produto seja capaz de satisfazer as necessidades do cliente, e também satisfazer as necessidades da organização desenvolvedora e de seus colaboradores.

Grande parte das causas de insucesso de um projeto é atribuída a "razões políticas". Pessoas, em geral, costumam utilizar esta expressão para designar qualquer problema relacionado ao fator **pessoas**. Por exemplo, problemas de relacionamento com algum membro da equipe, com o chefe, com os clientes, questões motivacionais, etc. podem ser todos definidos como problemas de fator pessoal. Tratam-se basicamente de problemas de **comunicação** [DeMarco99].

Ambientes **complexos** e **dinâmicos** não são eficientemente tratados por metodologias preditivas. Ou seja, metodologias de desenvolvimento que buscam determinar previamente todos os passos do desenvolvimento e gerenciamento de projetos apresentam poucas probabilidades de sucesso em projetos em tais ambientes. Isto porque nestes ambientes as ferramentas e capacidade de predizer todas as possibilidades e tratar todos os possíveis eventos durante o ciclo de vida de criação de um produto são ineficazes [Schwaber04].

Além disso, o principal foco da metodologia é gerar **valor** para o cliente, através de *feedback* concreto e constante, diminuindo os riscos de insucesso do produto do projeto.

Por todas estas razões, a metodologia deverá ser baseada em metodologias ágeis, que possuem um conjunto de valores e práticas que se

encontram completamente alinhados com as necessidades demandadas pela criação (gerenciamento e desenvolvimento) de um *advergame*.

5.1 Objetivo da Metodologia Proposta

O objetivo da metodologia é maximizar a satisfação dos envolvidos no projeto através da proposta de uma nova metodologia capaz de tratar os maiores problemas e desafios demandados pela criação de um advergame através da otimização da comunicação entre todas as partes interessadas no produto do projeto.

5.2 Essência da Metodologia

Esta metodologia utiliza como valores aqueles descritos no Manifesto Ágil [Agile01]. Sua estrutura principal é baseada no *Scrum* (ver Seção 4.3.2 – "Scrum"), por ser uma metodologia ágil com grande ênfase nas ferramentas gerenciais. O *Scrum* defende a existência de um modelo de gerência descentralizada e executada por agentes independentes que são capazes de tomar decisões. Assim, os mecanismos gerenciais sugeridos por esta metodologia estão fortemente baseados nas ferramentas providas pelo *Scrum*. As práticas de desenvolvimento têm forte inspiração em metodologias ágeis, principalmente *Extreme Programming* (ver Seção 4.3.1 – "Extreme Programming"). Contudo, o foco não é valorizar as práticas ágeis, mas sim, praticar os valores. Por isso, relembram-se os valores ágeis:

Indivíduos e interações sobre processos e ferramentas
Software funcionando sobre documentação detalhada
Colaboração do cliente sobre negociação de contrato
Responder a mudanças sobre seguir um plano

Estes valores são suficientemente amplos para que possam ser utilizados de diferentes formas durante a criação de um *advergame*. Assim, esta metodologia é modelada em função de maximizar estes valores durante todo o ciclo.

Além destes valores, existe um conjunto de princípios e características que esta metodologia pretende possuir:

- **Desenvolvimento Iterativo e Incremental**: O modelo de desenvolvimento se baseia na constante melhoria do jogo que está sendo produzido;
- **Orientação a Testes**: Ao contrário de algumas metodologias que buscam criar um conjunto de modelos diferentes do software a ser implementado, esta metodologia se preocupa com ênfase nos testes como forma de permitir à equipe foco no problema, garantia da qualidade (através dos testes) e simplicidade.
- **Gerenciamento Descentralizado**: O antigo papel do gerente de projetos é substituído por um modelo de gerenciamento de projetos distribuído entre os vários papéis da metodologia. Esta forma de gerenciamento se mostra bastante eficiente em ambientes complexos e dinâmicos onde as habilidades gerenciais são incapazes de predizer todo o andamento do projeto. Ao distribuir a responsabilidade do gerenciamento entre vários agentes, esta metodologia acredita no crescimento e maturidade das pessoas.

A principal idéia é maximizar e otimizar a comunicação entre todos envolvidos no projeto, em detrimento à antiga ordem, onde os momentos de "Negócios" e de "Produção" eram tratados como esforços totalmente dissociados. A idéia atual é integrar a todos num processo extremamente colaborativo, aumentando a sinergia da equipe formada por todos envolvidos.

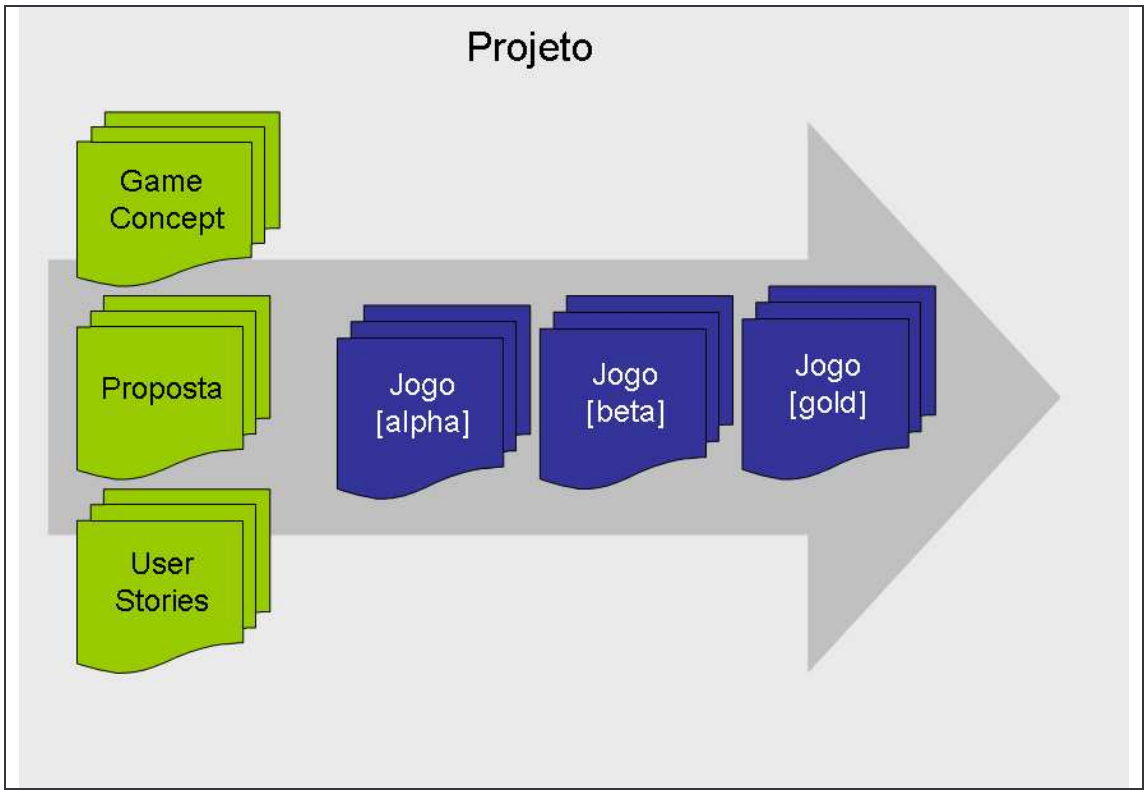


Figura 20 – Nova Ordem: Integração Total.

Logo, a maneira encontrada para integrá-los desde o início é considerar o esforço de negócios como parte do projeto. Assim, o projeto se inicia com a apresentação da demanda por parte do cliente, e não, pela aprovação da proposta para início da produção do produto. Então, são iniciados os *Sprints* (pequenas iterações), com objetivos diferentes, de acordo com a fase do projeto. Os primeiros *Sprints* têm finalidade de descobrir as motivações do projeto, sugerir as idéias iniciais do produto e estabelecer o *business-case*. Os *Sprints* posteriores têm finalidade de produzir o produto, refiná-lo e implantá-lo em seu ambiente de operação.

Além disso, o *feedback* antecipado é um dos principais mecanismos de otimização da comunicação com os clientes e diminuição das probabilidades de ocorrência dos riscos. Por isso, a metodologia sugere que o desenvolvimento (propriamente dito) de *advergames* seja capaz de produzir três diferentes versões:

- ***Alpha***: Geralmente, é uma primeira versão do jogo onde tem os principais elementos de *jogabilidade*, mecânica, no entanto, nem todo o conteúdo está incluso;
- ***Beta***: É uma segunda versão do jogo, onde todos os aspectos de conteúdo (como som, ajuda, etc.) estão implementados;
- ***Gold Master*:** É a versão final do jogo onde constam todos os melhoramentos e refinamentos necessários ao jogo.

Cada um desses ciclos possui três fases: Concepção, Construção e Pós-Construção. Em cada fase existe um foco separado. A concepção busca alavancar o projeto, a Construção busca produzir o produto de fato, e a Pós-Construção foca nos aspectos de implantação e melhoria dos jogos, etc.

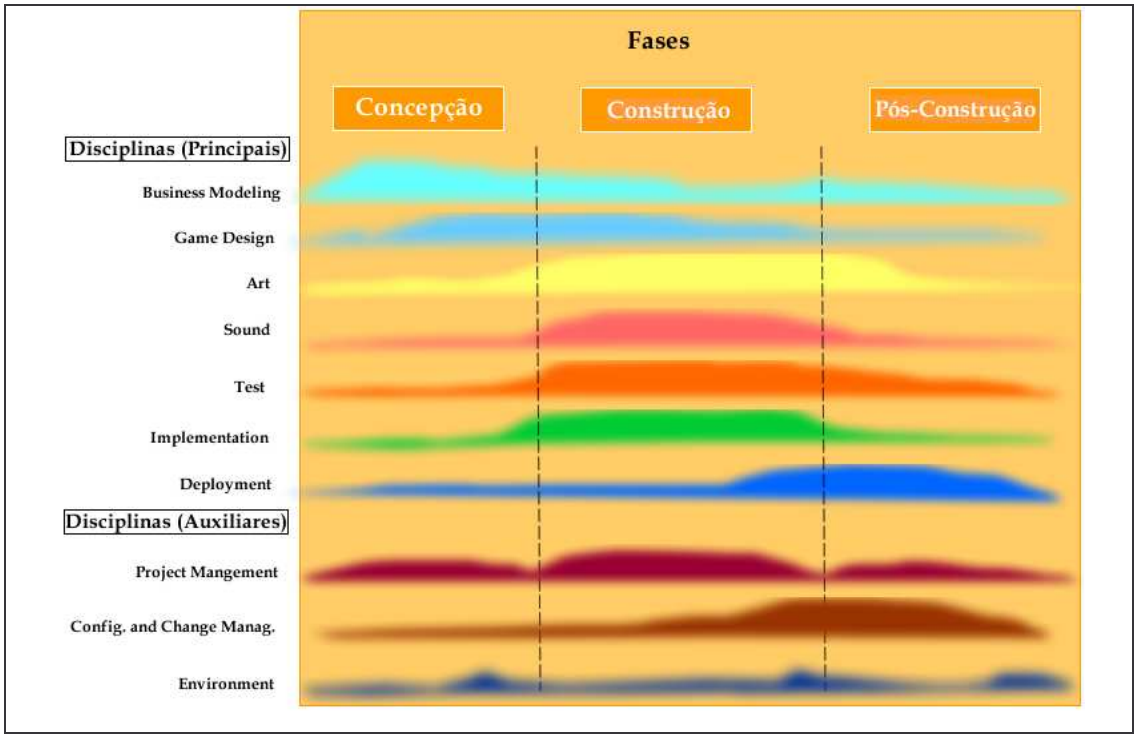


Figura 21 – AGP Overview – Fases/Disciplinas.

O gráfico acima se torna mais alto ou mais baixo de acordo com ênfase que cada disciplina deve ter em cada fase do ciclo de desenvolvimento. Durante cada uma dessas fases, um conjunto limitado de disciplinas é executado:

- *Business Modeling;*
- *Game Design;*
- *Art;*
- *Sound;*
- *Test;*
- *Implementation;*
- *Deployment;*
- *Project Management;*
- *Configuration and Change Management;*
- *Environment.*

Nas próximas seções discute-se detalhadamente a notação adotada para modelar o processo (ou metodologia), suas fases, disciplinas e práticas sugeridas.

5.3 SPEM – Software Process Engineering Metamodel

SPEM (*Software Process Engineering Metamodel*) tem se tornado um padrão para descrição, instanciação e qualquer forma de representação de um processo de software. O padrão é sugerido pela OMG (*Object Management Group*). A notação é baseada nas teorias de objetos, apresentando-se como uma extensão (*profile*) de UML [UML]. Ou seja, é um conjunto de uma ou mais extensões da semântica de UML com a intenção de customizá-la para um domínio ou propósito particular. Por exemplo, para modelagem de

processos no caso de SPEM. Por isso, pode utilizar uma série de diagramas de UML (como diagramas de caso de uso, de atividade, etc.) para modelar visualmente o processo. Qual a vantagem desta abordagem [SPEM]?

- Facilitar o entendimento do processo;
- Facilitar a adaptação do processo;
- Facilitar gerência do processo.

No entanto, os diagramas de UML para descrever processos através do SPEM utilizam uma série de estereótipos, entre eles:

- **Artefatos (tradução de *Work Product*)**: Uma descrição de algo que contém informação ou é uma entidade física produzida ou usada por atividades do processo. Ex: modelos, planos, documentos, etc.;
- **Atividade**: Descreve uma determinada atividade que um papel realiza dentro de um processo;
- **Papel (tradução de *Process Role*)**: Descreve os papéis, responsabilidades e competências que um determinado indivíduo tem dentro do processo;
- **Disciplina**: É um agrupamento coerente de elementos do processo (artefatos, papéis, atividades) cujas atividades são organizadas segundo algum ponto de vista ou tema comum (Ex: Análise e Projeto, Teste, Implementação, etc.);
- **Fluxo de Atividades (Tradução de *Workflow*):** Determina um fluxo seqüencial e lógico envolvendo papéis, artefatos e atividades;
- **Documento:** É um tipo particular de artefato. Nesta metodologia o conceito de documentação é bem mais amplo. Não precisa ser um documento formal, basta ser algo que possa comunicar de maneira eficiente, como uma apresentação, ou um diagrama;

- **Guias (tradução de *Guidance*):** É um elemento do modelo que pode ser associado a atividades para ajudar ou instruir na sua realização. Pode representar técnicas, *guidelines*, *templates*, etc.

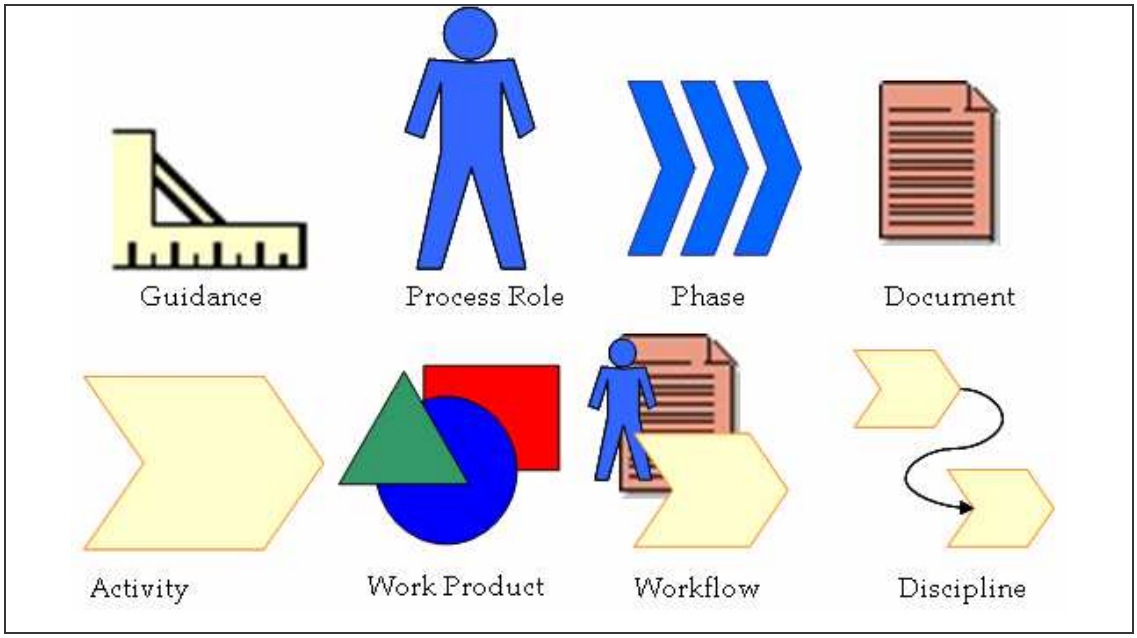


Figura 22 – Estereótipos de SPEM.

Nas próximas seções iremos discutir cada um dos principais aspectos da metodologia: papéis, artefatos, disciplinas. Atividades, fluxo de atividades e documentos serão discutidos apenas quando houver o detalhamento das disciplinas. Os guias (*guidance*) ou práticas recomendadas serão apenas exibidas no detalhamento das disciplinas e, posteriormente, serão abordadas extensivamente. No entanto, é preciso salientar que os conjuntos de artefatos, atividades e guias sugeridos neste trabalho são apenas recomendações.

Afinal, a metodologia apresentada aqui é adaptativa e não preditiva, e com a presença de agentes independentes capazes de escolher os melhores caminhos e tomarem decisões de acordo com o andamento do trabalho, objetiva-se apenas sugerir um conjunto de práticas nas diferentes áreas de conhecimento envolvidas na criação de um *advergame*. Esta é uma importante contribuição desta metodologia, visto que nem todos envolvidos no projeto de um jogo publicitário (*advergame*) conhecem as atividades envolvidas na criação do mesmo, assim, podem utilizar as disciplinas descritas aqui como

forma de criarem um *advergame* de maneira organizada e sistemática. Porém, os membros do time (ou *Team Members*) são aconselhados a discutir a eficácia dessas atividades e práticas e, então, modificar o caminho trilhado pela equipe para alcançar os objetivos definidos para o projeto.

Portanto, para um melhor entendimento de como esta metodologia pode ser expressa através do SPEM, é exibido o diagrama a seguir:

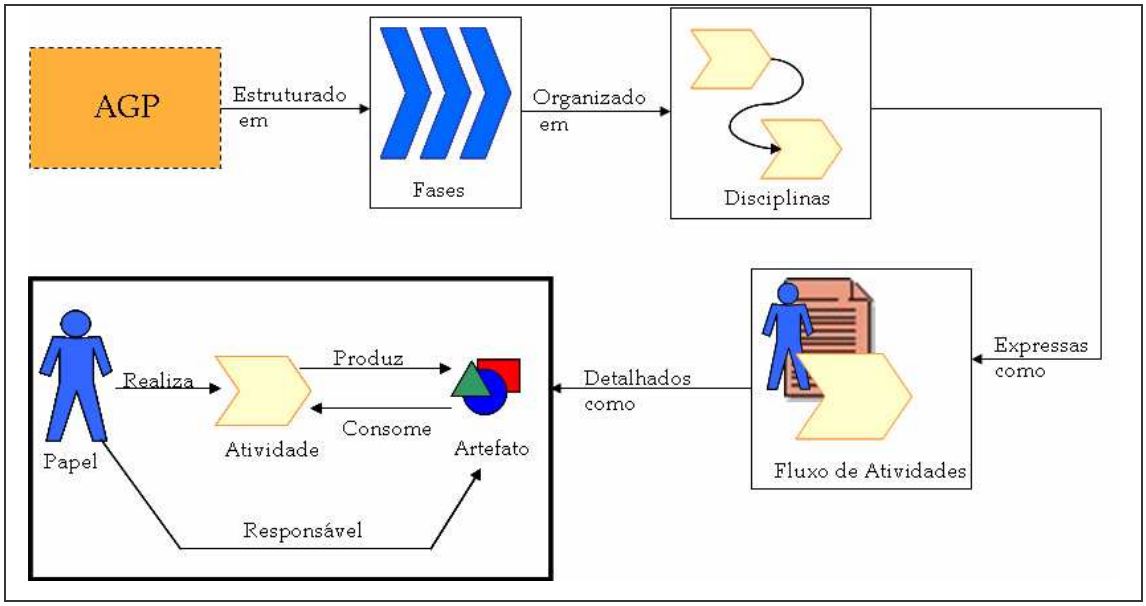


Figura 23 – Organização da Metodologia sob o SPEM.

5.4 Metodologia – Papéis

Os papéis da metodologia descritos nesta Seção se baseiam na definição sugerida pelo *Scrum* e ajustes realizados para estarem de acordo com o cenário da criação de um *advergame*.

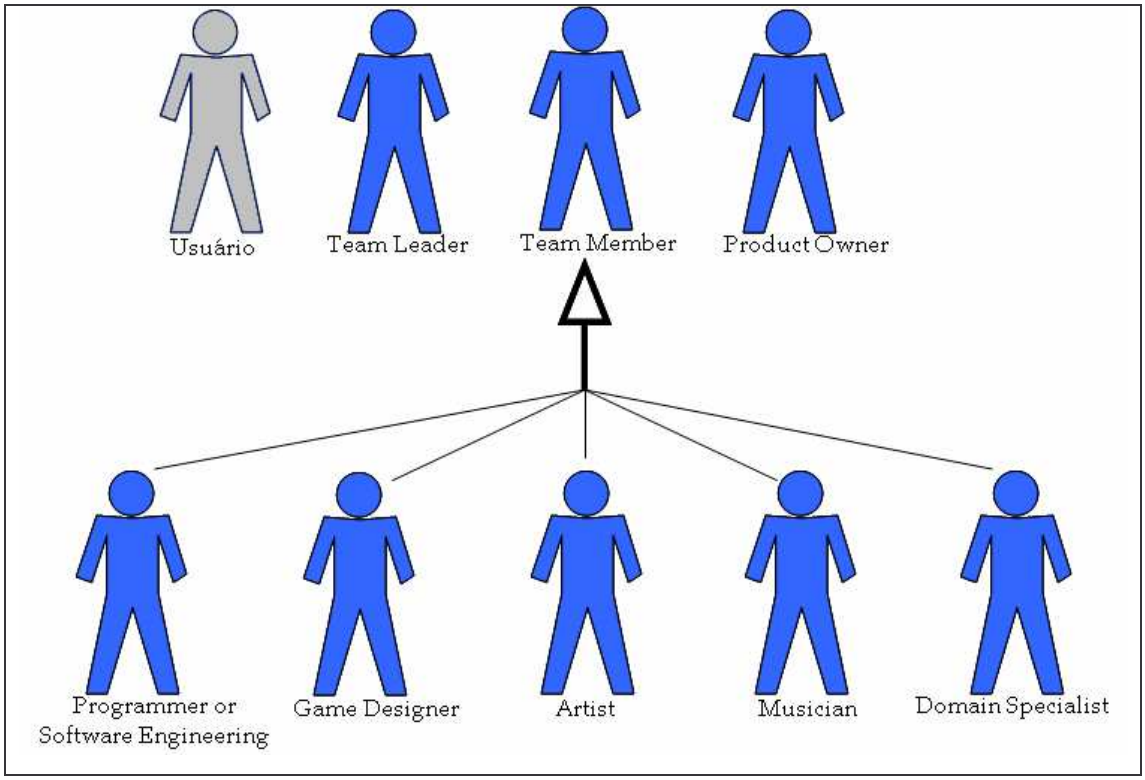


Figura 24 – Papéis definidos para esta metodologia.

Assim, a sintaxe de UML para diagramas de casos de uso foi utilizada para descrever os papéis existentes numa equipe de projeto de *advergame* dentro desta metodologia. Notadamente, existem três papéis básicos: *Team Leader*, *Product Owner* e *Team Member*, onde este último papel é realizado por atores (papéis) mais especializados, como engenheiros de software, artistas (de interface gráfica), *game designers*, músicos e, eventualmente, um especialista do domínio relativo àquele jogo. A atuação do usuário se restringe a participar dos testes de *jogabilidade*.

5.4.1 Product Owner

Em projetos de software – normalmente – os clientes é que são responsáveis por pelos requisitos, enumerando um conjunto de problemas ou necessidades que precisam ser tratados pelo produto do projeto. Entretanto, em projetos de *advergames*, os clientes (agência de publicidade) têm pouco (ou

limitado) conhecimento sobre jogos. Por isso, sua atuação na etapa de requisitos torna-se superficial e restrita a algumas diretrizes informando qual o público-alvo do produto, temática da campanha publicitária associada ao jogo, etc.

O *Product Owner* é o responsável pelo retorno sobre o investimento (ROI) do projeto e, por isso, representa o cliente internamente na organização desenvolvedora de um *advergame*. Além disso, prioriza as funcionalidades que apresentam maior valor para o cliente, fazendo com que o risco do projeto seja diminuído e o cliente possua valor mais cedo durante o processo de desenvolvimento. Outra importante atribuição do *Product Owner* é informar as restrições (cronograma, por exemplo) e objetivos de um projeto a toda equipe.

Idealmente, deve ter um bom conhecimento sobre a área de jogos para que seja capaz de realizar negociações com a equipe e manter a confiança do cliente alta em relação ao sucesso do projeto. E, principalmente, deve ser transparente com o cliente e membros da equipe, jamais interferindo no trabalho de uma equipe durante a realização de um *Sprint*. Sua atuação se dá basicamente ao longo do processo da seguinte maneira:

- Priorização e atualização dos itens do *Product Backlog*;
- Validação e/ou solicitação de alteração de itens de *Product Backlog* que foram entregues em determinado Sprint;
- Participação no *Restrospective*, onde são discutidos os aspectos positivos e negativos de um *Sprint* e, então, são tomadas algumas medidas para melhorar os resultados dos próximos *Sprints*.

Assim, durante um *Sprint* a atuação de um *Product Owner* se resume ao esclarecimento de dúvidas ou participações em *Meetings* realizados pela equipe, quando for convidado.

5.4.2 Team Leader

O *Team Leader* está para uma equipe assim como um cão pastor está para um rebanho de ovelhas. Ele é alguém escolhido entre os membros da equipe e é responsável por liderar a mesma:

- ***Mentoring/Coaching***: Ensinar o processo a todos envolvidos no projeto, incluindo membros da equipe e o (s) *Product Owner* (s). Acompanhar a utilização do processo e corrigir eventuais imperfeições;
- **Remoção de Obstáculos**: Impedimentos são todos e quaisquer fatos ou eventos que impeçam o avanço do trabalho de uma equipe dentro de um projeto. O *Team Leader* é responsável por remover todos esses obstáculos que possam impedir que uma equipe alcance os objetivos definidos para um *Sprint* ou para um projeto inteiro. Inclusive, é bastante importante fazer com que o *Product Owner* não possa interferir no trabalho da equipe. Tudo isto faz com que a equipe se mantenha focada nos objetivos com os quais se comprometeu;
- **Comunicação**: O *Team Leader* deve ser o grande responsável por maximizar a comunicação entre as partes envolvidas no projeto. AGP fornece um conjunto de práticas que buscam a comunicação franca e transparente entre as pessoas. O *Team Leader* deve garantir que as práticas sejam utilizadas da maneira correta, fazendo com que a comunicação seja mantida de maneira satisfatória.

O *Team Leader* é sempre alguém escolhido entre os membros da equipe (entre os *Team Members*) onde sua principal preocupação não é desenvolver o produto, mas facilitar ao máximo o trabalho da equipe e otimizar a comunicação entre as partes interessadas do projeto.

5.4.3 Team Members

Os membros de uma equipe possuem responsabilidade direta com o desenvolvimento do produto de um projeto, neste caso, com a criação do *advergame*. Contudo, a principal diferença com relação a outras metodologias é que – assim como *Scrum* – esta entende que algumas decisões devam ser tomadas por membros da equipe, por terem maior conhecimento sobre sua produtividade, área de trabalho, etc. Assim, estabelecendo um modelo de gerenciamento descentralizado (distribuído) realizado por agentes independentes. Isto faz com que cresça a maturidade dos membros do time.

O caráter de uma equipe de projetos para criação de *advergames* é essencialmente multidisciplinar. Isto é, a equipe é heterogênea e formada por pessoas com diferentes habilidades e conhecimentos. Em geral, a equipe é formada por:

- ***Game Designers***: São os criadores da idéia do jogo. Definem todos os aspectos com relação aos cenários, *jogabilidade*, mecânica do jogo, elementos sonoros e visuais, personagens, etc. Além disso, na criação de um *advergame* são responsáveis por prover um jogo que possa emitir a mensagem publicitária de maneira eficiente, fazendo com que a marca possa ser percebida;
- **Programadores ou Engenheiros de Software**: São responsáveis por implementar o programa que contém o jogo, todos os seus

módulos, e executar todos os testes para confirmar se o jogo realizará as funcionalidades que tinham sido definidas para ele;

- **Artistas**: Devem implementar toda a parte de interface e arte dos jogos. Ou seja, construir todos os aspectos visuais do jogo;
- **Músicos**: Devem implementar toda a parte de som. Isto é, construir todos os aspectos sonoros do jogo;
- **Especialistas do Domínio**: São pessoas que possuem conhecimento específico sobre a área que um jogo abordará. Por exemplo, um médico pode ser utilizado como consultor e fonte de conhecimento para jogos que levem em consideração o domínio conhecido como saúde.

Em todo o desenvolvimento, os membros do time têm papel decisivo:

- **Estimativas**: Criação das estimativas iniciais das funcionalidades do projeto e de cada uma de suas tarefas. Cada membro participa das estimativas da tarefa, por isso, aumenta o seu comprometimento para com aquilo planejado para o projeto;
- **Controle do Projeto**: Todos os dias, os membros da equipe são responsáveis por atualizar as funcionalidades e tarefas concluídas e as estimativas daquelas tarefas que ainda precisam ser executadas. Isto é fundamental, pois permite uma visualização da situação do projeto em tempo real;
- **Execução**: Os membros da equipe executam as tarefas necessárias para alcançar os objetivos.

Ao contrário de outras metodologias, esta sugere um conjunto limitado de papéis que, na verdade, representam a justaposição de muitos papéis existentes em outras metodologias. Esta decisão foi tomada levando em conta o pequeno tamanho das equipes que desenvolvem *advergames* e a

impossibilidade de haver recursos para ocupar todos os papéis normalmente existentes.

5.4.4 Usuário

O papel de usuário se restringe apenas à necessidade de identificar alguns comportamentos dos usuários com relação ao jogo. Assim, o papel de usuário aparece apenas na disciplina de *Test* (Teste).

5.5 Metodologia - Fases

A descrição da metodologia AGP em fases é capaz de capturar aspectos dinâmicos da mesma. O ciclo de desenvolvimento nesta metodologia possui basicamente três fases: Concepção, Construção e Pós-Construção. Como visto anteriormente, sugere-se que o desenvolvimento de advergames seja realizado de maneira iterativa e incremental, disponibilizando versões intermediárias do produto final (*Alpha, Beta e Gold Master*). Assim, existem *Sprints* (de Preparação) iniciais cujo objetivo é desenvolver o *business-case*, propostas e aspectos de negócios que levam à aprovação da continuação do projeto por parte do cliente. Os demais *Sprints* (de Produção) focam na produção do jogo propriamente dito.

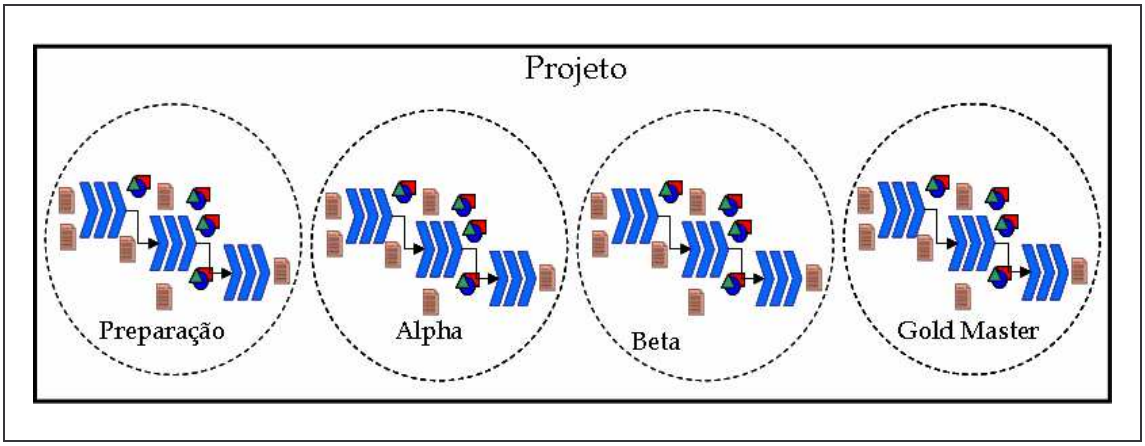


Figura 25 – Ciclos da Metodologia.

Em cada ciclo, as fases (e suas disciplinas) são executadas sempre de modo que o desenvolvimento seja iterativo e incremental, visando melhorar o produto que está sendo construído.

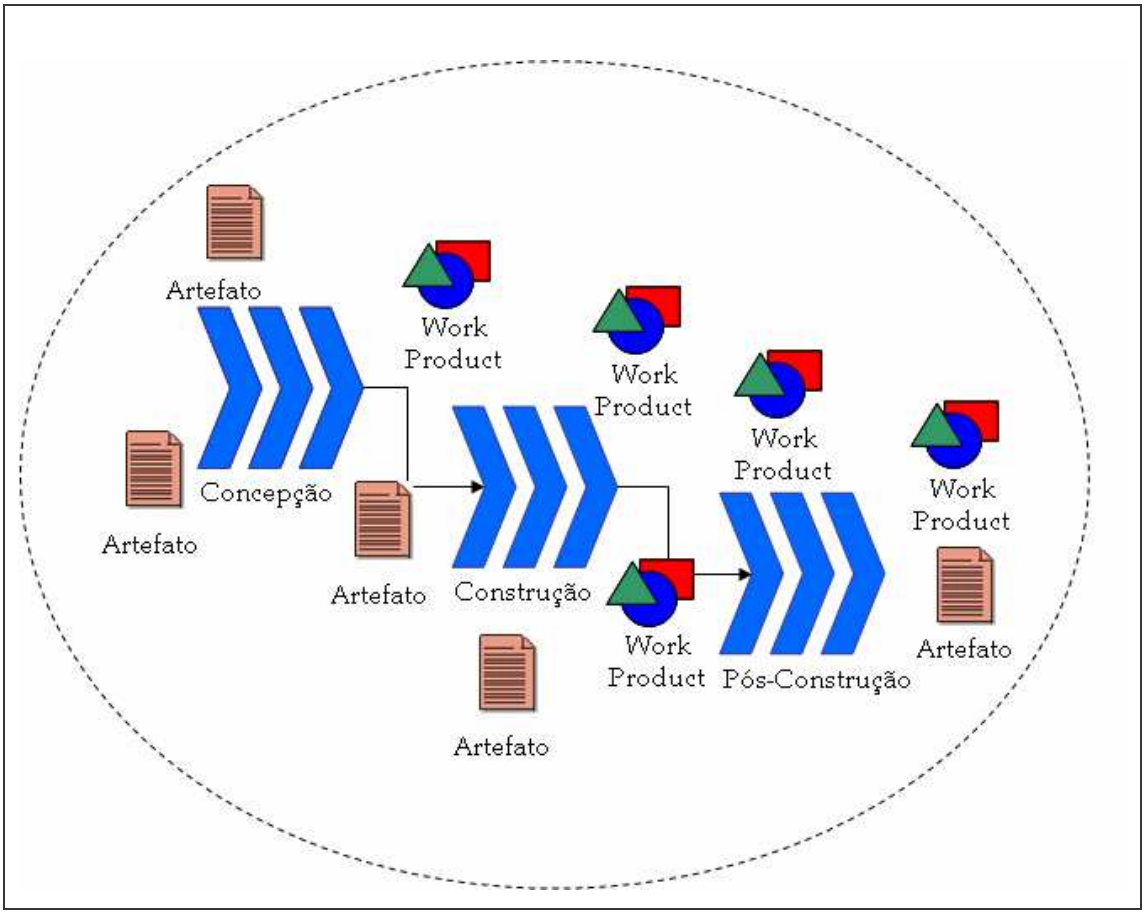


Figura 26 – Fases da Metodologia.

Assim, as fases são executadas consumindo e produzindo artefatos e documentos. As estruturas das fases nesta metodologia são continuas, isto é, a

transição de uma fase para outra não se dá de maneira brusca, mas de modo gradativo. Devido ao pequeno número de fases e sua sobreposição em muitos momentos, o problema da linearidade do modelo *Waterfall* ou do Processo Unificado (ver Seção 4.2 – "Processos Iterativos") é praticamente inexistente. As fases também podem ter objetivos diferentes, de acordo com o tipo do ciclo de desenvolvimento em execução. Se o ciclo for de "Preparação", por exemplo, o objetivo deve ser criar artefatos de "Proposta". Caso seja um ciclo de "Produção", o objetivo deve ser produzir uma versão do jogo.

Além disso, em cada ciclo podem ser realizados inúmeros *Sprints*. O número e a duração dos *Sprints* dependerão das restrições de cronograma previstas para o projeto.

5.5.1 Concepção

A fase de concepção é a primeira entre as fases do ciclo de desenvolvimento de um *advergame*. Num ciclo de "Preparação", ela inicia-se com a apresentação da demanda de criação de um jogo e identificação dos objetivos do jogo. Num ciclo de "Produção", ela se inicia com a criação de um *Product Backlog* que contém um conjunto de funcionalidades (estórias) previstas para aquele ciclo.

	**Objetivos**	Criar a motivação para a criação de um *Game Concept* e de uma Proposta. Ou criar um *Product Backlog* para guiar um ciclo de desenvolvimento.
	Milestones	*Business-case* ou *Product Backlog* (criado ou atualizado).

Tabela 2 – Fase de Concepção (Resumo).

5.5.2 Construção

A fase de Construção tem o objetivo de produzir a demanda atual. Assim como a Concepção, esta fase pode ter caráter diferente de acordo com o tipo de ciclo de desenvolvimento em execução num projeto. Se o ciclo for de "Preparação", deve-se criar um conceito (*Game Concept*) e uma Proposta. Caso o ciclo seja de "Produção", uma versão ou *Release* (*Alpha, Beta, Gold Master*) do jogo deverá ser entregue.

	**Objetivos**	Criar um artefato como um Game Concept (ou Proposta) ou entregar uma versão do jogo ao Product Owner.
	Milestones	Game Concept, Proposta, ou versão do Jogo.

Tabela 3 – Fase de Construção (Resumo).

5.5.3 Pós-Construção

A fase de Pós-Construção se preocupa com os aspectos de avaliação, melhoria e implantação dos artefatos entregues na fase de Construção. Assim como na metodologia *Scrum*, o principal objetivo neste momento é avaliar.

Se o ciclo for de "Preparação", o objetivo deve ser refinar o artefato entregue na fase de Construção. Caso o ciclo seja de "Produção", o objetivo deverá ser implantar a versão do jogo produzida na fase de Construção em seu devido ambiente de operação.

	**Objetivos**	Melhorar o artefato entregue na fase de produção, deixando-o em estado de ser utilizado pelo usuário final.
	Milestones	Game Concept (refinado), Proposta (refinada), ou versão do jogo implantada.

Tabela 4 – Fase de Pós-Construção (Resumo).

5.6 Metodologia - Disciplinas

Esta metodologia define um conjunto de 10 disciplinas, entre as quais, 7 (sete) são diretamente ligadas ao desenvolvimento do produto. As 3 (três) disciplinas restantes funcionam como apoio ao desenvolvimento e ao processo. Segue abaixo a segmentação:

- Disciplinas Principais
    - *Business Modeling;*
    - *Game Design;*
    - *Art;*
    - *Sound;*
    - *Test;*
    - *Implementation;*
    - *Deployment.*
- Disciplinas de Apoio
    - *Project Management;*
    - *Configuration and Change Management;*
    - *Environment.*

A seguir, cada uma dessas disciplinas é discutida detalhadamente.

5.6.1 Business Modeling

Esta disciplina está diretamente relacionada a entender as necessidades do cliente, e sugerir uma solução baseada em jogo que possa fazer com que ele tenha seus objetivos alcançados.

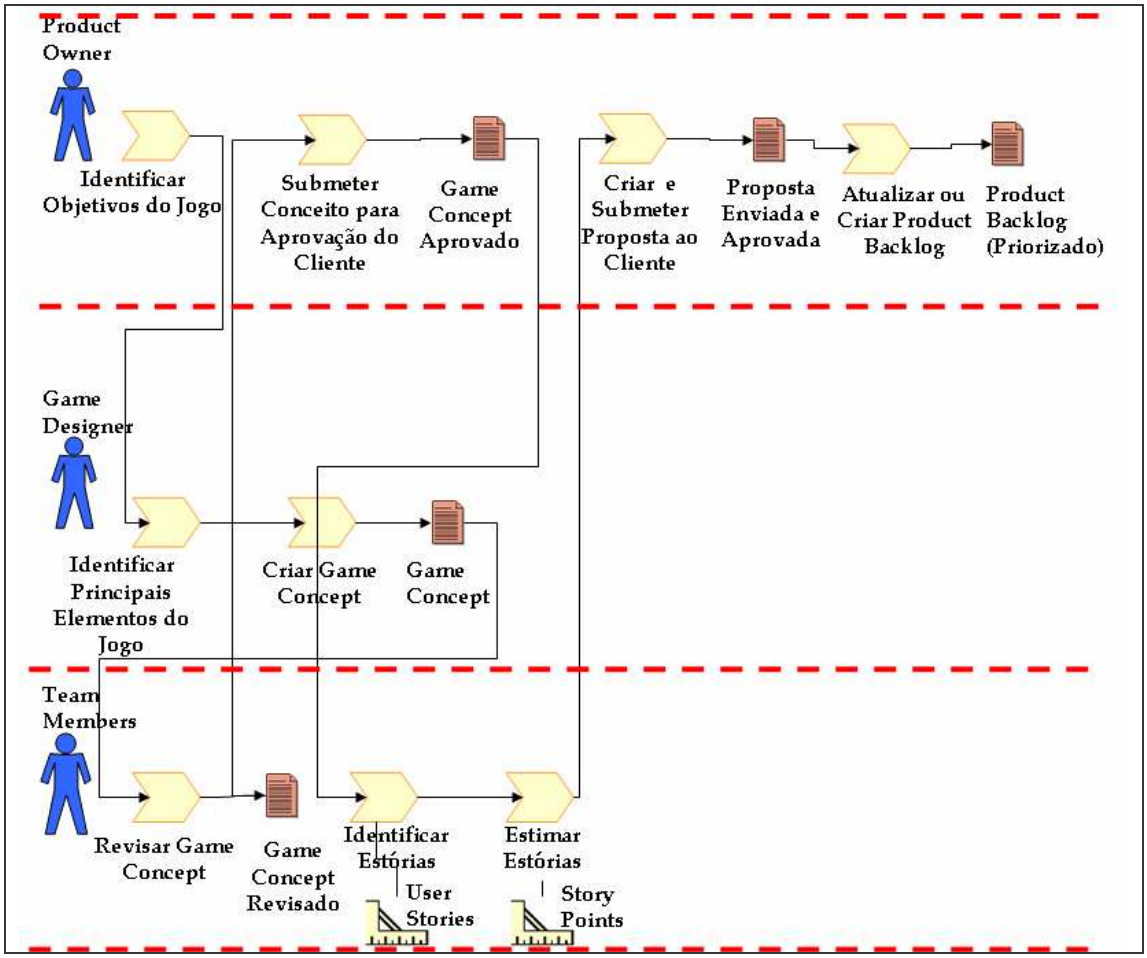


Figura 27 – Fluxo de Atividades - Business Modeling.

Durante o fluxo inteiro são gerados alguns artefatos importantes. Assim, o fluxo transcorre gerando artefatos da seguinte maneira:

- ***Game Concept***: um *game designer* da equipe é responsável por buscar referências e sugerir um conceito de jogo que possa atender àquelas necessidades apresentadas pelo cliente. Posteriormente, o *game designer* discute com os demais membros do time e o conceito é melhorado e refinado. Logo, após a aprovação interna (na organização desenvolvedora) o conceito é enviado para o cliente. Assim, se o cliente desejar alguma alteração com relação ao conceito, o processo de criar o conceito é reiniciado (focado apenas nas alterações). Caso o conceito seja aceito, então, inicia-se a criação da proposta;

- **Proposta**: Com o *Game Concept* aprovado, a equipe pode continuar seu trabalho através da identificação de estórias (ver Seção 5.7 - "Práticas Sugeridas", onde se discute o uso de estórias ou *User Stories*). Então, após identificar as estórias, a equipe estima o esforço para realizar cada uma dessas estórias. Finalmente, o *Product Owner* visualiza o esforço para completar cada uma das estórias necessárias ao jogo e cria uma proposta (detalhando aspectos de escopo, custo e cronograma do projeto) e envia ao cliente;
- ***Product Backlog***: Quando finalmente a proposta é aprovada, o *Product Owner* pode criar o *Product Backlog* do projeto. O *Product Backlog* é um forte instrumento de comunicação para o desenvolvimento e gerenciamento de projetos. Trata-se de uma lista contendo todas as funcionalidades que devem ser providas por um sistema. No caso desta metodologia, o *Product Backlog* deve ser uma lista com todas as estórias identificadas para o projeto do jogo. Além disto, o *Product Owner* deve priorizar cada um desses itens. A priorização é extremamente fundamental. Através da priorização o *Product Owner* informa – explicitamente – à equipe quais são os itens mais críticos do ponto de vista de: valor agregado ao cliente, risco, etc. Dessa forma, o time trabalha sempre focado em produzir – de maneira mais simples possível – aquilo que irá possuir mais valor para o cliente. Esta alternativa também representa uma forma eficiente de ataque aos riscos. Quando todos os itens do *Product Backlog* tiverem sido realizados, o projeto poderá ser considerado como finalizado. Então, o ciclo de vida de operação do produto é iniciado.

Os papéis importantes à execução deste fluxo de atividades estão exibidos na imagem anterior. O papel de *Team Members* foi incluso para

expressar que todos os membros da equipe (programadores, músicos, artistas, etc.) devem realizar aquelas atividades descritas. Em casos em que as atividades devem ser realizadas apenas por um *game designer*.

5.6.2 Game Design

A disciplina de *Game Design* está diretamente voltada com a definição dos aspectos de *jogabilidade*, de comportamento de personagens e objetos, e características sonoras e visuais do jogo.

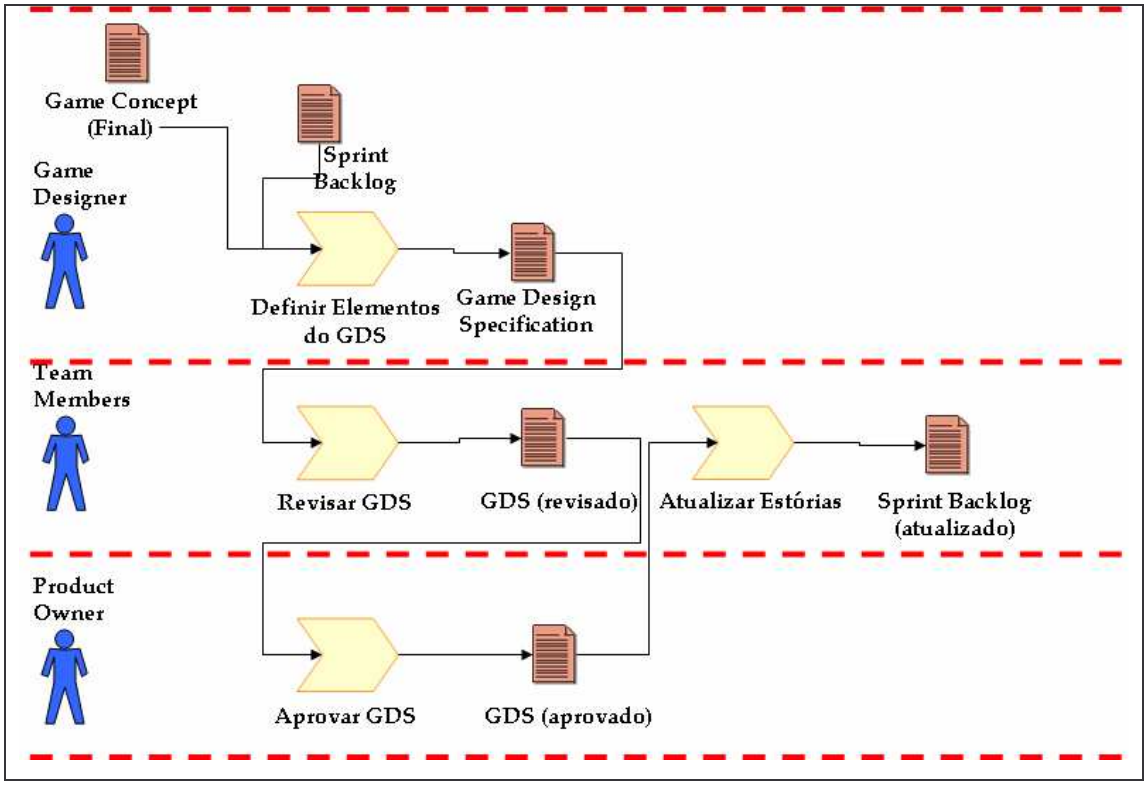


Figura 28 – Fluxo de Atividades – Game Design.

Os principais artefatos utilizados pela disciplina de *Game Design* são o *Sprint Backlog* (lista que contém todas as estórias do *Sprint* atual) e o *Game Design Specification (GDS)*.

Inicialmente, o *Sprint Backlog* contém as estórias do usuário, entretanto, nem todas estão descritas num nível de detalhes satisfatório. Por isso, alguns artefatos precisam ser gerados para melhorar o poder de comunicação das estórias escolhidas para o Sprint. Entre eles:

- ***Game Design Specification***: O GDS é um documento comumente utilizado nas organizações que desenvolvem jogos. Seu papel é extremamente crucial na produção de um jogo, afinal este documento define todos os aspectos ligados à *jogabilidade* e mecânica (*game-play*), personagens, atributos visuais e sonoros do jogo. Enfim, um GDS está para um jogo assim como um documento de requisitos está para um processo de desenvolvimento de um sistema de informação de propósito geral. Entretanto, nesta metodologia, o GDS passa a ocupar um papel menos crítico e menos orientado a documentação. O *game designer* deve se preocupar em definir as estórias e o documento deve ser orientado às prioridades definidas pelo *Product Owner*. Assim, o novo papel do GDS é meramente servir de apoio para aumentar o poder de comunicação das estórias.
- ***Sprint Backlog* (atualizado)**: É o mesmo *Sprint Backlog* inicial, só que enriquecido com as informações detalhadas obtidas a partir do GDS;

Em geral, documento de *Game Design Specification* tem uma estrutura bem definida e o *game designer* trabalha sequencialmente para definir todos os itens necessários a um documento de *Game Design*. Na imagem acima, a atividade "Definir Elementos do GDS" não foi suficientemente detalhada porque que itens do GDS o *game designer* deverá abordar serão determinados pelas prioridades das estórias definidas pelo *Product Owner*. Logo, não é possível afirmar que "o *game designer* deve trabalhar primeiramente na mecânica do jogo para, então, tratar os aspectos sonoros", por exemplo.

Mais uma vez, os papéis se mostram muito bem definidos com o papel *Team Members* assumindo responsabilidades que devem ser todos os membros de uma equipe. O *game designer* aparece à parte porque existem atividades que devem ser realizadas exclusivamente por ele mesmo. O *Product Owner*, assim como no *Scrum*, não deve interferir no trabalho da equipe durante um *Sprint*. Por isso, a aprovação do GDS é uma atividade meramente para comunicar ao *Product Owner* dos detalhes das estórias definidas por ele e, se necessário, identificar alguma inconsistência com aquilo que ele deseja para o produto entregue ao final daquele *Sprint*. Salienta-se que neste momento ele (o *Product Owner*) não pode solicitar alterações. Se ele realmente precisar realizar modificações, o *Sprint* pode ser cancelado e um novo *Sprint* é remarcado.

5.6.3 Art

A disciplina de *Art* (arte) está diretamente ligada à construção dos aspectos visuais do jogo. O principal responsável pela criação dos elementos visuais e artísticos do jogo são artistas. O fluxo de atividade da disciplina ocorre conforme abaixo:

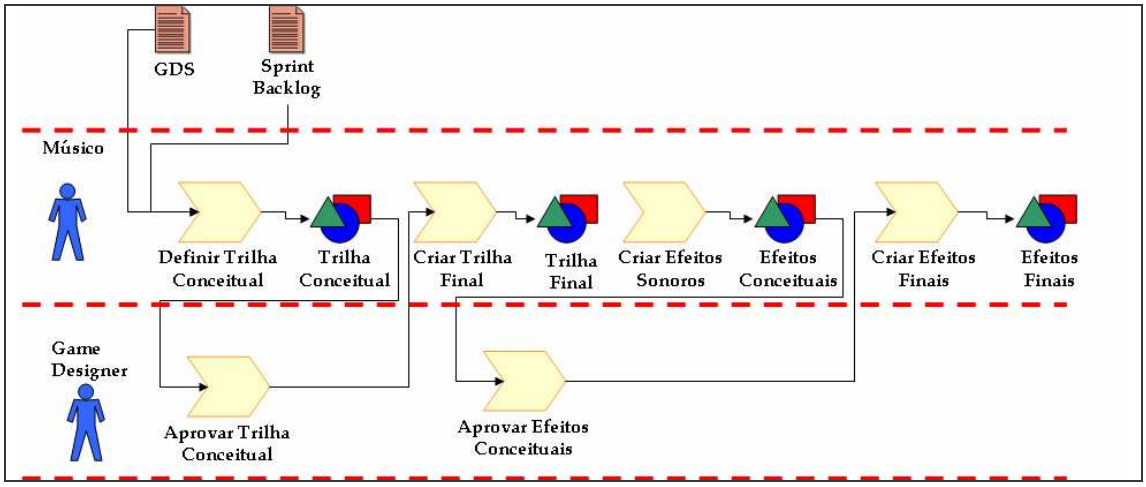


Figura 29 – Fluxo de Atividades – Art.

Os principais artefatos que servem para a construção dos elementos visuais são o *Sprint Backlog* e o *Game Design Specification*. O primeiro contém todas as estórias definidas para aquele *Sprint*. O segundo contém descrições detalhadas, extensas e – possivelmente – referências que o artista deve levar em consideração ao criar os elementos visuais do jogo.

Assim, mais uma vez, as atividades de arte foram citadas de maneira abstrata, devido ao fato que a criação dos elementos de arte deve estar orientada à prioridade das estórias definidas pelo *Product Owner*. Por esta razão, é impossível afirmar que "o artista deve primeiramente criar os elementos da *game-screen* e, somente depois, criar a tela de ranking", por exemplo.

Portanto, o artista cria um conjunto de *assets* para aquela estória em que a equipe está focada no momento, chamado "Arte Conceitual". Então, o *game designer* que descreveu aqueles elementos pode aprovar ou solicitar alterações. Caso solicite alterações, a atividade de "Definir Arte Conceitual" é reiniciada, do contrário, o artista produz os "Assets Finais de Arte", principal artefato desta disciplina.

5.6.4 Sound

A disciplina de *Sound* (Som) está diretamente ligada ao desenvolvimento das características sonoras de um jogo. Esta disciplina se preocupa com a trilha do jogo e também com outros efeitos sonoros que servem para melhorar a interatividade do jogo.

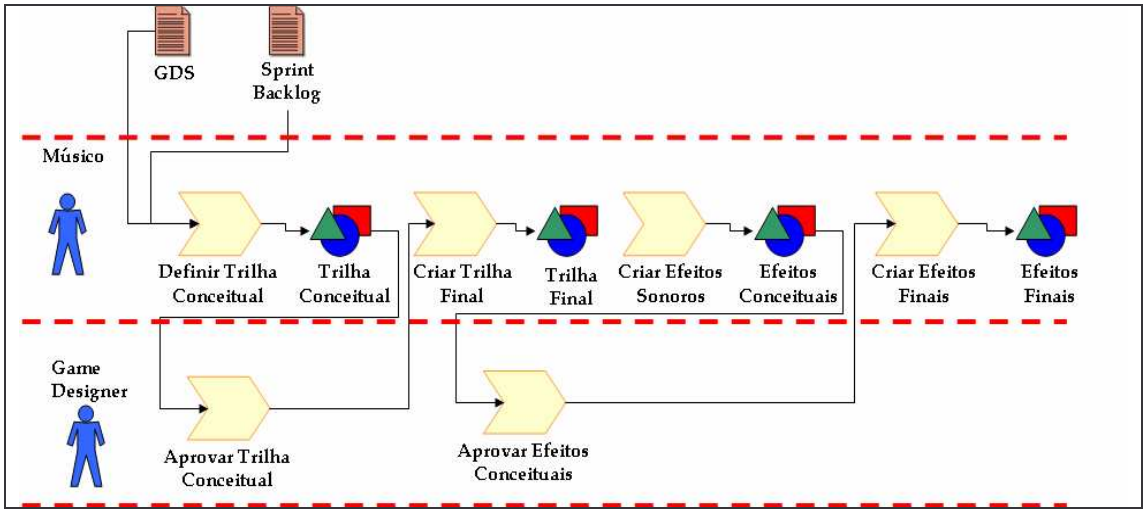


Figura 30 – Fluxo de Atividades – Sound.

As principais fontes de informação para que um músico possa criar os *assets* sonoros de um jogo são o *Sprint Backlog* e o GDS. O primeiro contém todas as estórias definidas para o *Sprint* atual e o segundo contém descrições detalhadas e referências sonoras que o músico pode utilizar na construção dos elementos sonoros.

Cada estória pode conter elementos sonoros, por isso, para cada estória sugere-se criar primeiramente a trilha sonora (que é mais custosa e demanda maior trabalho) e posteriormente criar os efeitos sonoros, que podem ser apenas alertas sonoros curtos e baratos de se fazer. Mais uma vez, esta metodologia não está apta a sugerir que o "artista desenvolva os efeitos sonoros ligados aos personagens antes de desenvolver os efeitos sonoros ligados ao cenário". Por isso, essas escolhas dependerão das prioridades das estórias determinadas pelo *Product Owner*.

Portanto, inicialmente, o músico realizada a atividade "Definir Trilha Conceitual", que deve ser aprovada pelo *game designer*. Caso ele não aprove, "Definir Trilha Conceitual" é mais uma vez realizada pelo músico. Do contrário, ele finalizada a criação da trilha final através da atividade "Criar Trilha Final". Depois, o músico foca em produzir os efeitos sonoros daquela

estória. Assim, cria os efeitos conceituais, que devem ser aprovados pelo *game designer*. Se não houver aprovação, o músico deve reiniciar "Criar Efeitos Sonoros". Quando houver aprovação dos efeitos conceituais, então, o músico realiza a atividade "Criar Efeitos Finais" para produzir os efeitos sonoros finais para aquela estória em que a equipe está trabalhando.

5.6.5 Test

A disciplina de *Test* (Teste) é uma das mais importantes disciplinas desta metodologia. Em sua essência, a metodologia AGP (*Agile Game Process*) é *Test-Driven*, isto é, orientada a testes (para saber mais sobre *Test-Driven Development* ver Seção 5.7 "Práticas Sugeridas – Test-Driven Development e Continuous Design"). Existem vários tipos de testes:

- **Testes de Aceitação**: Em geral, são escritos pelo usuário. No entanto, nesta metodologia, os testes de aceitação são escritos pelo *game designer*. Os testes de aceitação determinam as maneiras que o sistema (jogo, neste caso) deve se comportar diante da intervenção do usuário. Estes testes são extraídos a partir das estórias escritas. Se o jogo passa em todos testes de aceitação, ele pode ser considerado finalizado e coerente com aquilo que deve ser seu propósito;
- **Testes de Integração**: São testes de sistema, que visam averiguar o sucesso da integração de um novo módulo aos módulos já existentes de um jogo;
- **Testes Unitários**: São testes locais, realizados por programadores, para testar se unidades (*classes, objetos e métodos*) funcionam da maneira devida;

- **Testes de Jogabilidade**: São testes de jogos em que o objetivo é analisar o comportamento do usuário para que, então, possam ser sugeridas melhorias nos jogos.

Por simplicidade e facilitar a legibilidade, o fluxo de atividades de *Test* em duas imagens, conforme abaixo:

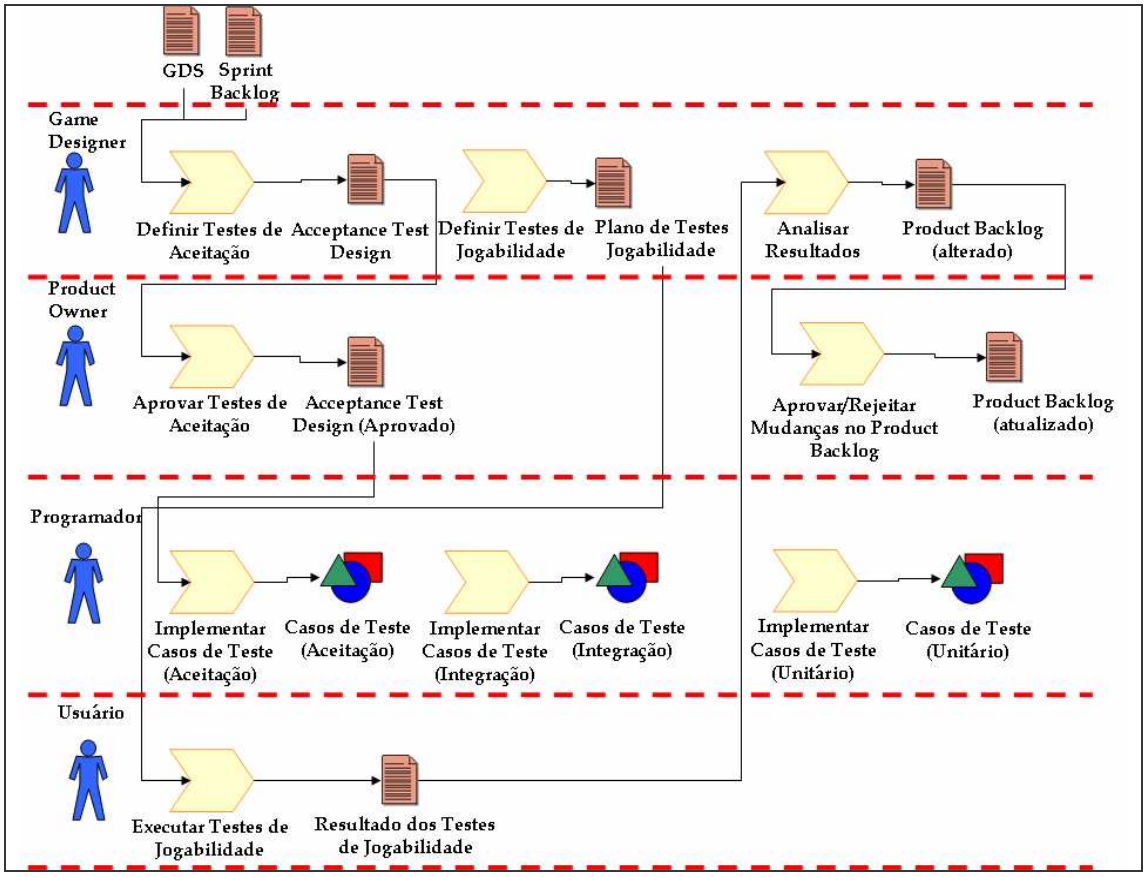


Figura 31 – Fluxo de Atividades – Test (Parte I).

Mais uma vez, a principal fonte de informações para a disciplina de testes é o *GDS* e o *Sprint Backlog*, por conterem preciosos detalhes sobre as estórias.

O fluxo de atividades se inicia com a criação do *Acceptance Test Design* (ATD), que contém todos os casos de teste de aceitação. Este documento serve de base para que os programadores consigam implementar os testes de aceitação, integração e unidade. Além disso, a disciplina também foca em testes de *jogabilidade*. O *game designer* produz um plano de testes de *jogabilidade*

que basicamente define um conjunto de usuários (representativos do público-alvo final) e situações para poder coletar resultados sobre o comportamento dos usuários. Estes resultados são analisados pelo *game designer*, que pode solicitar algumas alterações no *Product Backlog*, de modo que o jogo desenvolvido possa ser melhorado para alcançar os objetivos que motivaram aquele projeto. No entanto, essas modificações para somente serão inclusas no *Product Backlog* mediante aprovação do *Product Owner*.

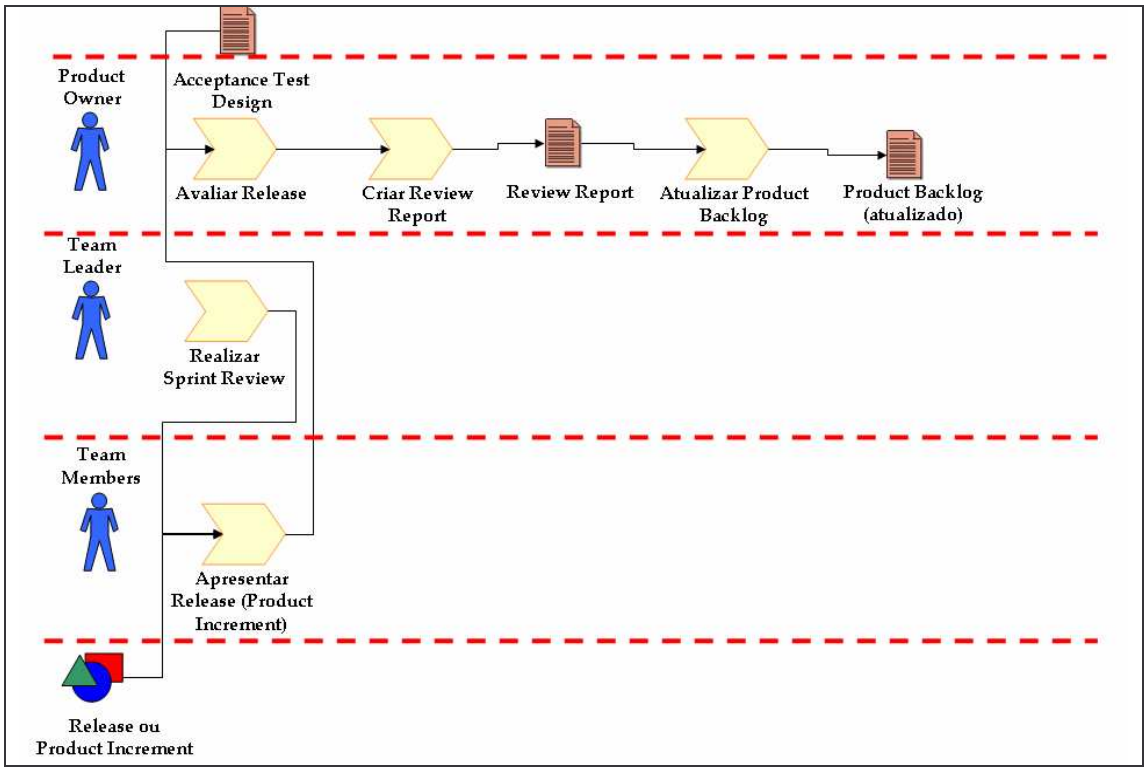


Figura 32 – Fluxo de Atividades – Test (Parte II).

Após a finalização de um *Sprint*, o *Team Leader* promover um encontro para apresentar o resultado gerado (*Release* do jogo ou *Product Increment*) por um *Sprint*. Este resultado deve ser avaliado pelo *Product Owner*. Ele testa o produto gerado e aprova, ou solicita mudanças. Estas mudanças são atualizadas no *Product Backlog*.

5.6.6 Implementation

A disciplina de *Implementation* (Implementação) é responsável pela criação do programa (sistema computacional) que representa o jogo. A construção se dá orientada a testes. Assim, os *módulos, funções, entidades e componentes* desenvolvidos devem ser desenvolvidos para que possam ser aprovados em um conjunto de casos de testes. Nesta disciplina, os programadores desenvolvem iterativa e incrementalmente o jogo, e fazendo – passo a passo – que cada estória seja realizada. Assim, ao contrário de metodologias MDD (*Model-Driven Development*), nesta disciplina a implementação não se apóia em práticas que criam modelos gráficos do sistema, como diagramas de caso de uso, diagramas de análise, etc.

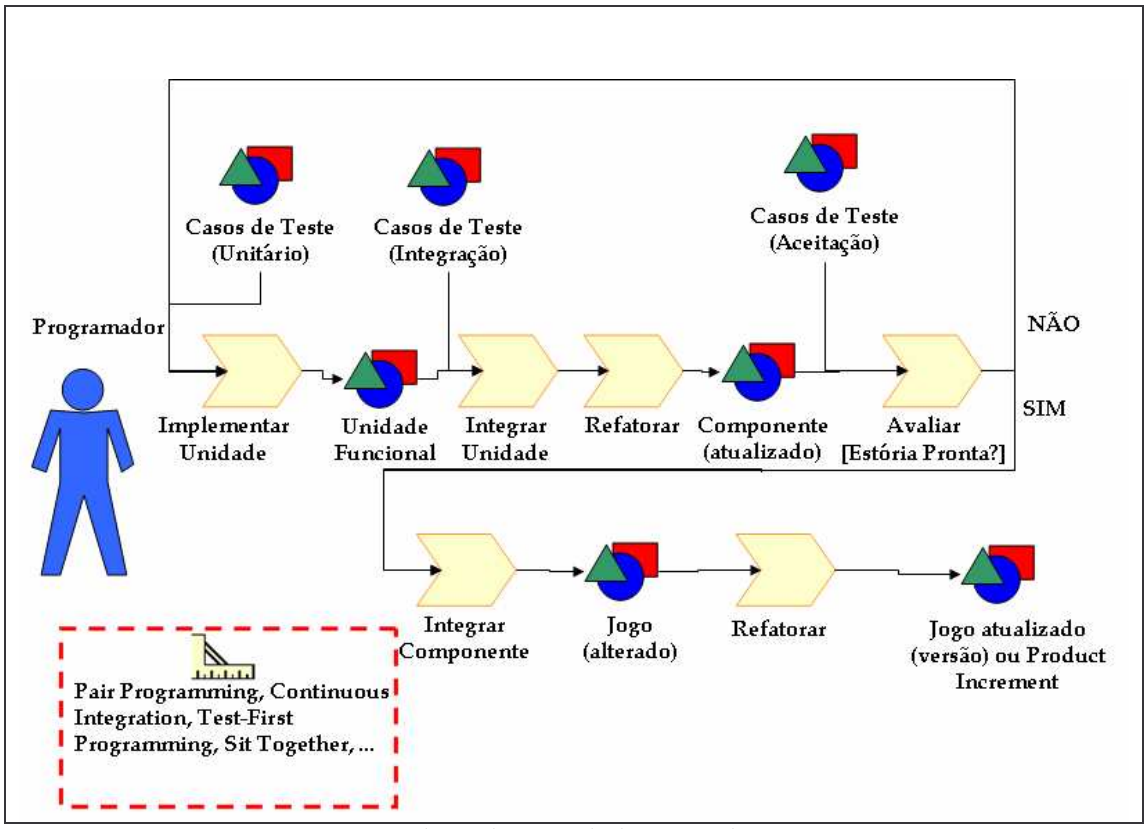


Figura 33 – Fluxo de Atividades – Implementation.

Dessa forma, a principal fonte de informação para os programadores desenvolverem seus códigos são os casos de teste. Assim, para implementar

uma estória, o programador utiliza uma estratégia *bottom-up* baseada na simplicidade. Seu objetivo é que seu código passe nos casos de teste. O trabalho é iniciado implementando uma unidade lógica que provê algum tipo de funcionalidade. Então, tendo escrito os testes unitários, ele (programador) faz com que suas classes (objetos e métodos) passem nestes testes. Depois, quando a unidade funcional testada for aprovada, ele integra-a ao restante do *componente* em que ele está trabalhando. Mais uma vez, ele tenta fazer com que o novo *componente* (modificado com a inclusão da unidade funcional) seja aprovado pelos testes de integração. Neste momento, é realizada uma avaliação, conduzida pelos testes de aceitação (daquela estória), para saber se a estória foi concluída. Se ela não tiver sido finalizada, deverá ser implementada a próxima unidade funcional, isto é, o fluxo de atividades é reiniciado. Do contrário, caso a estória tenha sido concluída, o componente atualizado é integrado ao restante do sistema (que representa o jogo) objetivando atualizar o sistema (jogo) existente, com a nova estória completamente integrada.

Durante todos estes passos, o programador deve estar sempre focado em produzir a "coisa mais simples que possa funcionar". Logo, sua concentração deverá estar totalmente voltada para o problema em que ele está trabalhando. Por isso, ao tentar fazer integrações ao longo do fluxo de atividades de *Implementação*, pode ser necessário fazer alguns ajustes no código. Esta atividade de ajuste recebe o nome de *refactoring* (ou refatoração) e visa melhorar a estrutura do código existente através do aumento da *coesão*, diminuição do *acoplamento*, e melhoria da *compreensibilidade* do código.

Por simplicidade, o estereótipo "Guia" (ou *Guidance*) foi omitido em cada atividade no diagrama de fluxo de atividades exibido acima. Entretanto, é sugerido o uso de muitas práticas (detalhadas na Seção 5.7 - "Práticas Sugeridas" deste capítulo), como *Pair-Programming*, *Continuous Integration*, etc.

Além disso, para implementar unidades funcionais, o programador (ou programadores) pode necessitar de elementos de arte ou sonoros, por isso, a comunicação com a equipe pode facilitar bastante, fazendo com que estes elementos estejam prontos para serem usados quando o programador precisar implementar a estória que contém tais elementos.

Portanto, os principais artefatos gerados na disciplina de *Implementation* podem gerar valor imediatamente para o cliente ou para o *Product Owner*. O grande esforço em testes visa garantir que o jogo realmente realiza as funcionalidades desejadas. Esses fatores aumentam a eficiência de *feedback* e a probabilidade de sucesso de um projeto.

Finalmente, quando todas as estórias definidas para o *Sprint* tiverem sido realizadas, um *Product Increment* é gerado. Caso tenha funcionalidades suficientes para se criar um *Release* (*Alpha, Beta, ou Gold Master*) do jogo, então ele será criado, devendo assim, ser preparado para o ambiente em que será utilizado pelos usuários finais.

5.6.7 Deployment

A disciplina de *Deployment* (ou Implantação) é responsável por entender o ambiente de funcionamento do jogo para o usuário final e disponibilizar um *Release* preparado para aquele ambiente.

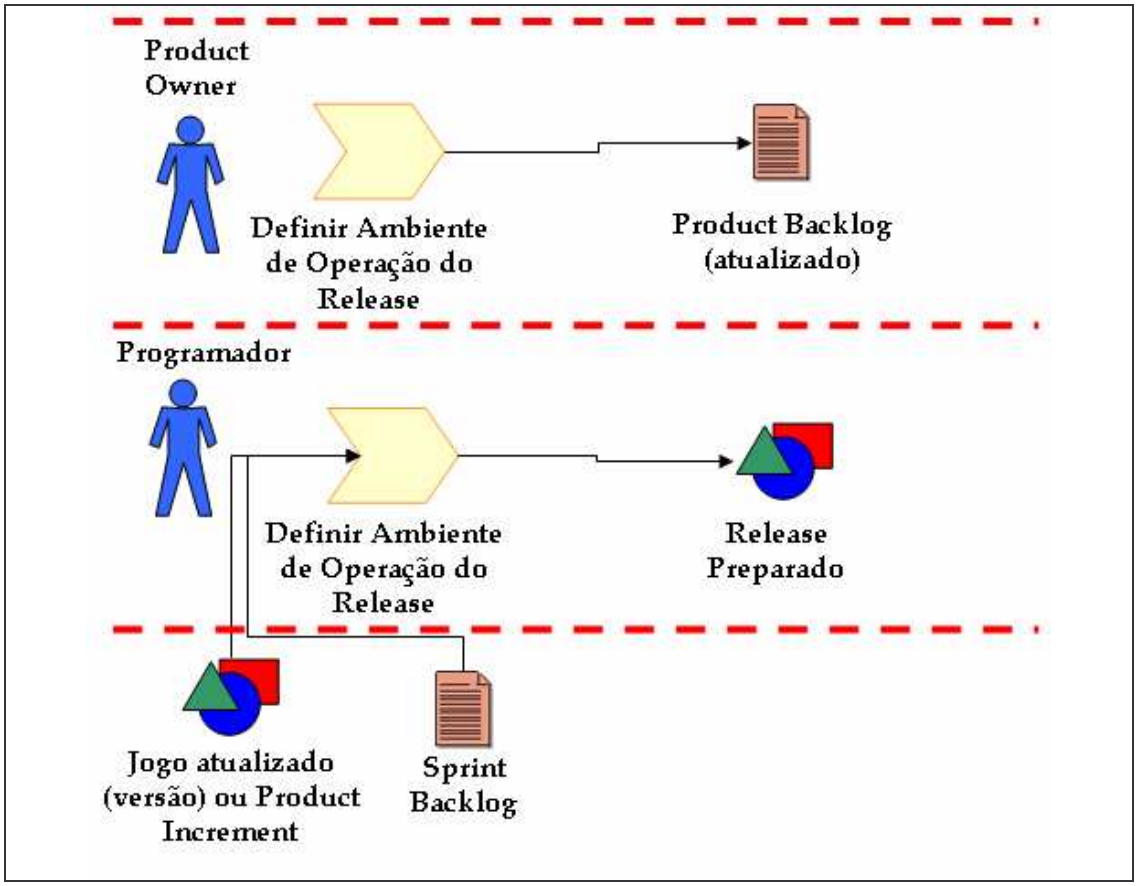


Figura 34 – Fluxo de Atividades – Deployment.

Na "Definição do Ambiente de Operação do Release" o *Product Owner* deverá ser capaz de entender o ambiente em que o usuário final utilizará o jogo. Por exemplo, se será uma aplicação instalada em seu computador, ou o jogo estará disponível numa página *web*. Este tipo de definição é bastante importante porque afeta a forma como o trabalho será desenvolvido por vários membros da equipe.

Quando um *Release* do jogo é gerado, então, o programador analisa o *Sprint Backlog* para descobrir informações sobre o ambiente de operação do jogo e, recebendo o *Release* do jogo (versão do ambiente de desenvolvimento), realiza a atividade "Preparar Release". Esta atividade encontra-se definida de maneira abstrata devido às diferentes possibilidades de ambientes para operação daquele *Release*. Assim, a preparação do *Release* pode ser configurar um servidor de aplicação e implantar o jogo para que ele possa ser acessado

via *web*, ou mesmo criar um instalador para que as pessoas possam fazer *download* ou adquiri-lo através de um *DVD*. Quando a preparação do release for finalizada, então, teremos um *Release Preparado*, que é a versão do jogo implantada.

5.6.8 Project Management

A disciplina de *Project Management* (ou Gerenciamento de Projetos) nesta metodologia está ligada aos aspectos de planejamento, controle do andamento do trabalho e encerramento do projeto. A disciplina de *Business Modeling* (Modelagem de Negócios) trata os aspectos de iniciação do projeto e a execução do projeto simplesmente ocorre através de disciplinas como *Game Design*, *Test*, *Sound*, *Art*, *Implementation*, *Deployment*. Para facilitar a compreensão, os fluxos de atividade desta disciplina foram separados em fluxos de planejamento, controle e encerramento.

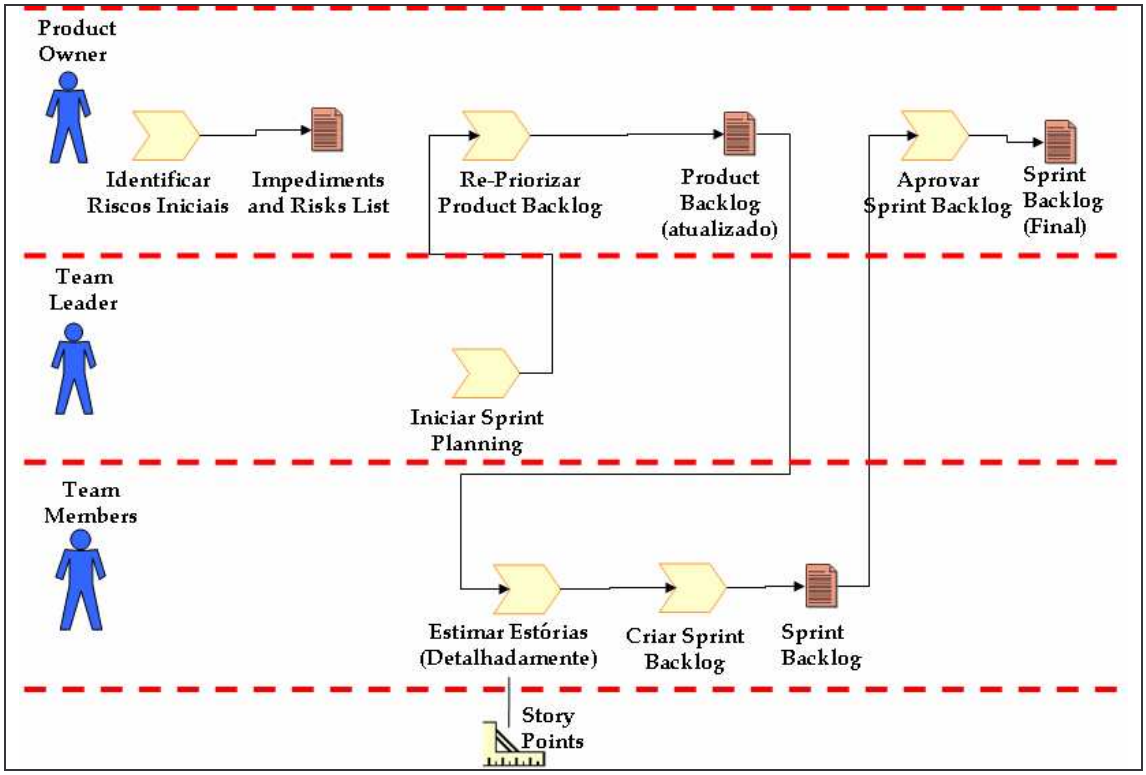


Figura 35 – Fluxo de Atividades – Project Management - Planejamento.

Nos fluxos de atividade de planejamento, o mais importante artefato para realização de um *Sprint* é o *Sprint Backlog*. Assim como um *Product Backlog*, o *Sprint Backlog* é uma lista de estórias que devem ser implementadas num determinado período de tempo, numa iteração, isto é, num *Sprint*. Um projeto – em geral – possui alguns *Sprints*. Antes de cada *Sprint* é construído um *Sprint Backlog* através de um encontro entre a equipe e o *Product Owner*, chamado *Sprint Planning*. Este encontro é dividido em dois momentos:

- **Apresentação e Priorização**: Neste primeiro momento, o *Product Owner* fala sobre a importância daquele projeto e passa a motivação necessária para o time acreditar na possibilidade de sucesso do projeto. Além disso, o *Product Owner* informa ao time os mais prioritários itens que devem ser realizados;
- **Estimativa e Acordo**: No segundo momento, o time então parte para estimar cada estória mais detalhadamente. Antes, havia apenas uma estimativa de esforço para completar a estória. Agora, o foco é quebrar a estória em várias tarefas que ajudam a realizá-la. Assim, tarefas são segmentadas por áreas: arte, programação, *game design*, etc. Entretanto, a colaboração é mantida, afinal, o objetivo do time é finalizar cada estória e, para tanto, terão de colaborar entre si para poderem alcançar as metas. O *Product Owner* ainda deverá estar presente na reunião, para que possa clarificar qualquer dúvida existente. A equipe, então, sabendo a quantidade de esforço que pode realizar num *Sprint* (que tem um tamanho predeterminado) sugere ao *Product Owner* quantos – entre os mais prioritários – itens do *Product Backlog* ela será capaz de entrega ao final daquele *Sprint*. Finalmente, inicia-se um acordo onde as partes (*Product Owner* e equipe) buscam chegar a um denominador comum. Quando as partes chegam a um acordo, o *Sprint* é – de fato – iniciado.

Durante todo o *Sprint*, o *Sprint Backlog* é o fator unificador da comunicação entre as pessoas do time. Cada membro da equipe deve diariamente manter o foco em produzir os itens do *Sprint Backlog* que possuem maior prioridade.

Além disso, o fluxo de atividades de planejamento é responsável pela identificação, classificação (qualitativa e quantitativamente) e planejamento dos riscos. Isto porque o gerenciamento dos riscos tem caráter preventivo.

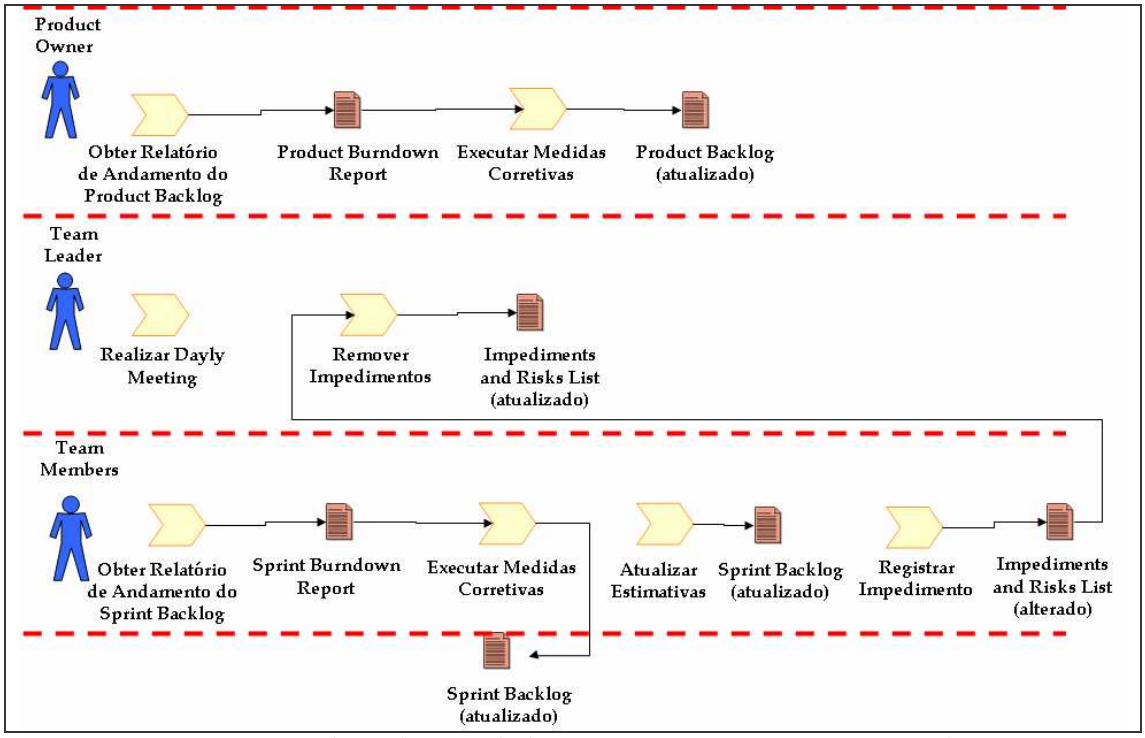


Figura 36 – Fluxo de Atividades – Project Management - Controle.

Nos fluxos de atividade de controle existem dois relatórios que são fundamentais para o controle eficiente do projeto. São eles:

- ***Product Burndown Report***: é um importante mecanismo de controle de projetos. Este *report* - ao longo de um projeto - fornece uma visão ampla de como os itens do *Product Backlog* têm sido realizados em função do tempo. Além disso, dependendo do software utilizado, é possível visualizar um gráfico com uma linha que mostra a tendência informando

quando todos os itens do *Product Backlog* terão sido finalizados, de acordo com a tendência com que eles têm sido entregues. O *report* pode ser extremamente útil fazendo com que o *Product Owner* possa tomar algumas decisões corretivas se, por exemplo, a tendência da data de finalização do projeto (itens de *Product Backlog*) apontar uma data inaceitável para o fim projeto. A ferramenta também serve para comunicar às partes interessadas o andamento do projeto sob um ponto de vista mais geral.

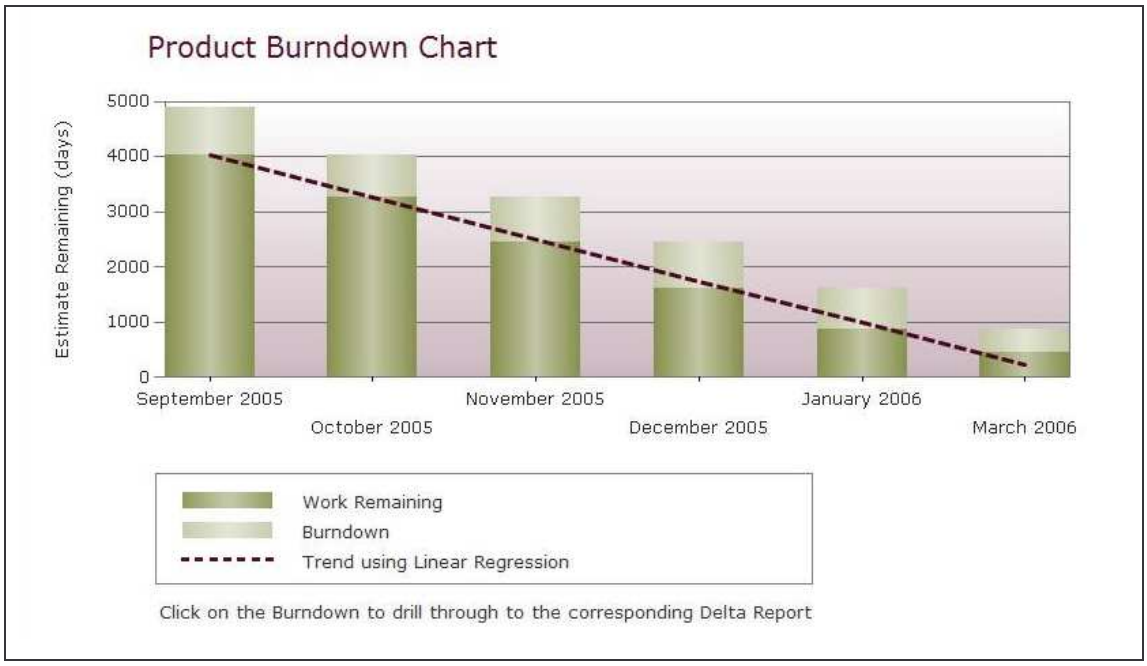


Figura 37 – Exemplo de Product Burndown [TeamSystem].

- ***Sprint Burndown Report*:** é uma poderosa ferramenta de comunicação e controle. Diariamente, os membros de um time devem atualizar a estimativa de esforço restante para finalizar cada tarefa. Assim, a qualquer momento, é possível visualizar a situação atual do projeto. Além disso, de acordo com a tendência de realização de esforço dos últimos dias, é possível projetar uma data final em que todos os itens do *Sprint Backlog* estariam finalizados. Dessa maneira, o *Sprint Burndown Report* serve para comunicar a todos envolvidos no projeto a evolução do trabalho ao longo de um *Sprint*. Isto garante confiança aos

membros do time quando vêem que o trabalho caminha para ser finalizado conforme o comprometimento assumido junto ao *Product Owner*. Outra importante utilização deste relatório é para auxiliar na tomada de decisão. Por exemplo, se a tendência aponta para finalização dos itens do Sprint Backlog apenas após a data combinada com o Product Owner, os membros da equipe podem decidir realizar alguns turnos-extras para retomar o controle do projeto diante do compromisso assumido. Ou seja, este *report* é uma forma eficaz de combater uma série de riscos.

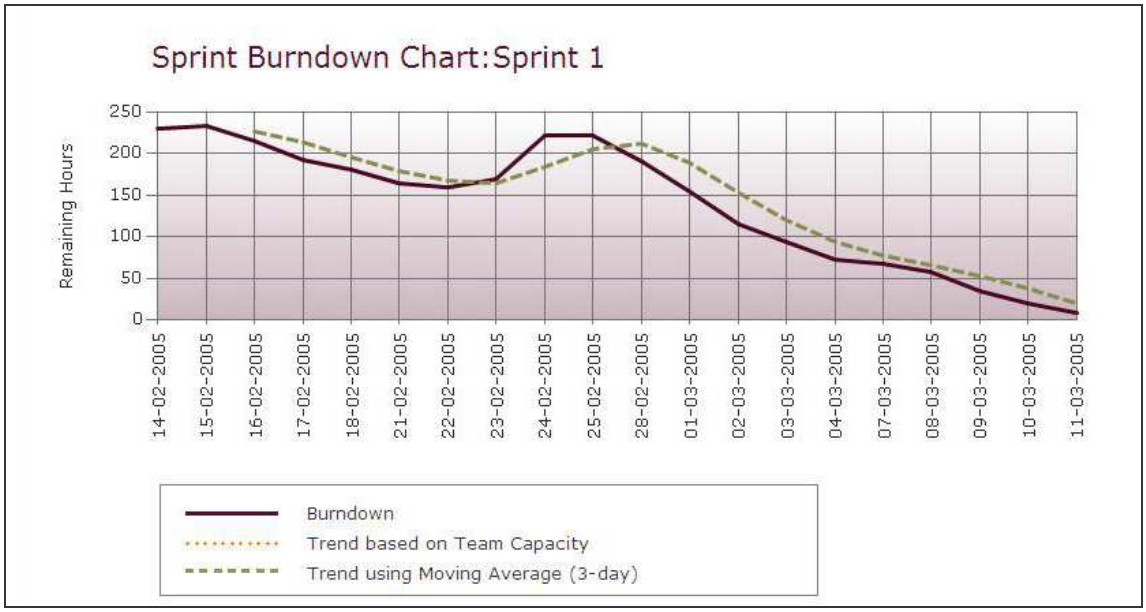


Figura 38 – Exemplo de Sprint Burndown [TeamSystem].

Além destes relatórios, os membros da equipe podem atualizar o *Impediments and Risks List* registrando impedimentos, que são problemas que aparecem durante um *Sprint* e impedem o avanço do trabalho dentro do mesmo. São identificados e registrados pelos membros da equipe. No entanto, o *Team Leader* é quem tem o papel de remover todos esses obstáculos, fazendo com que o time possa avançar e cumprir com os objetivos determinados para o *Sprint*. Outra importante – porém menos tangível – forma de controle é a realização do *Daily Meeting* um encontro diário que visa discutir o trabalho recentemente realizado, o trabalho que será iniciado e eventuais impedimentos. Por isso, o *Daily Meeting* funciona muito mais como uma

ferramenta de comunicação do que como uma ferramenta de controle dos projetos.

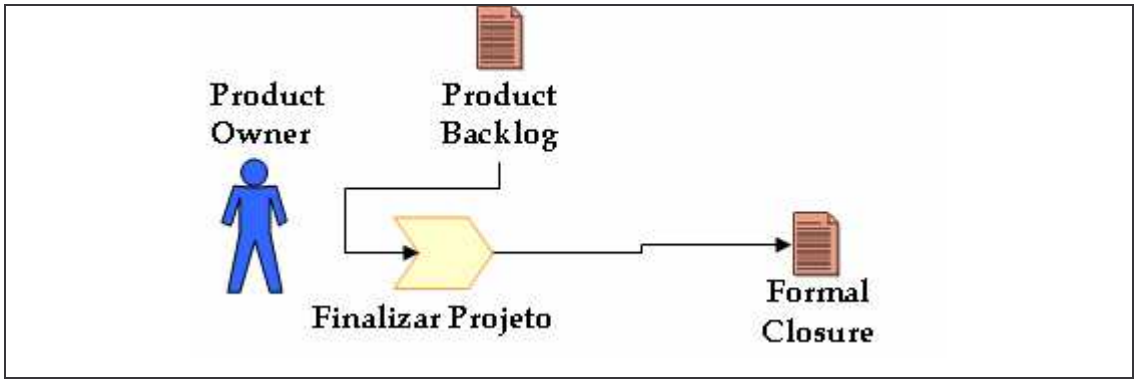


Figura 39 – Fluxo de Atividades – Project Management - Encerramento.

O fluxo de atividades de encerramento da disciplina de *Project Management* trata apenas do encerramento formal do projeto. O Product Owner, ao ver que todos os itens do Product Backlog foram realizados, inicia a finalização do projeto através da liberação de recursos (equipe), aprovação formal do cliente do projeto, etc.

5.6.9 Configuration and Change Management

A disciplina de *Configuration and Change Management* (CCM) é responsável por monitorar e tratar as mudanças ocorridas em tudo utilizado ou produzido durante o ciclo de desenvolvimento. Esta disciplina fornece forte suporte ao desenvolvimento do projeto, por isso, deve receber atenção especial do *Team Leader*.

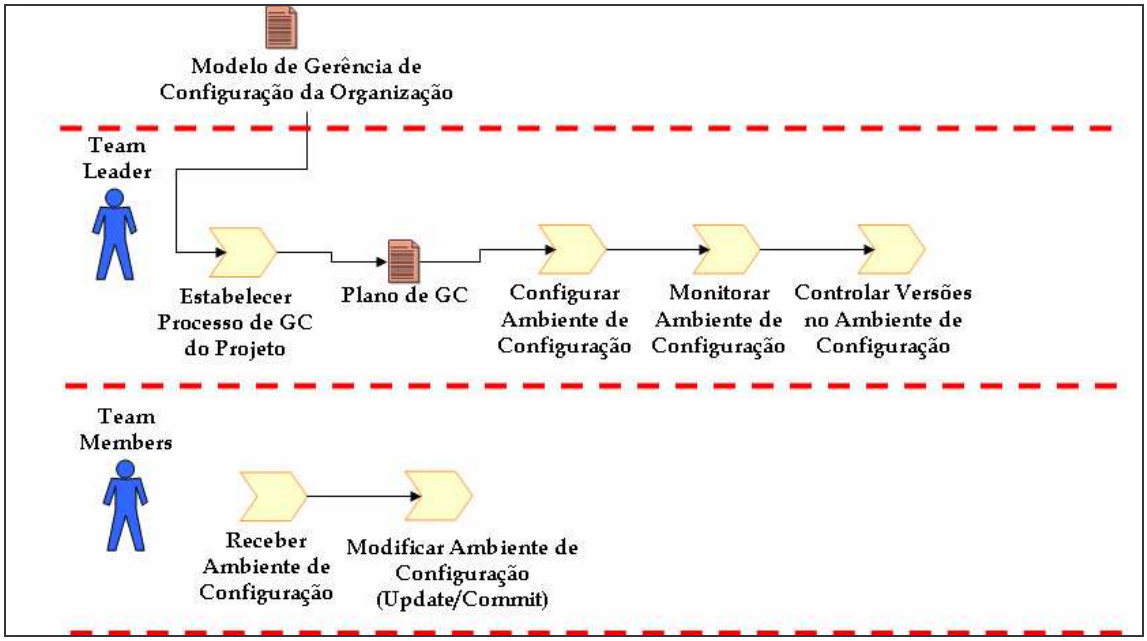


Figura 40 – Fluxo de Atividades – Config. and Change Management.

A principal fonte de informação para estabelecer os parâmetros de CCM é o "Modelo de Gerência de Configuração da Organização", isto é, o corpo de conhecimento da organização na área de CCM. Todo o trabalho de preparação, monitoramento, e controle do ambiente de configuração é responsabilidade do *Team Leader*. Afinal, caso o sistema de configuração seja falho, provavelmente irá ocorrer inúmeros impedimentos no trabalho da equipe. Os membros da equipe devem se preocupar unicamente em receber o ambiente e atualizá-lo/modificá-lo através de operações de *update/commit*, respectivamente.

5.6.10 Environment

A disciplina de *Environment* (Ambiente) é responsável pelo suporte ao processo de desenvolvimento e gerenciamento através da melhoria continua da maneira de executar projetos numa organização. Ao final de cada *Sprint* é realizado um *Sprint Retrospective*, cujo objetivo principal é avaliar a forma como o projeto foi executado no último *Sprint*.

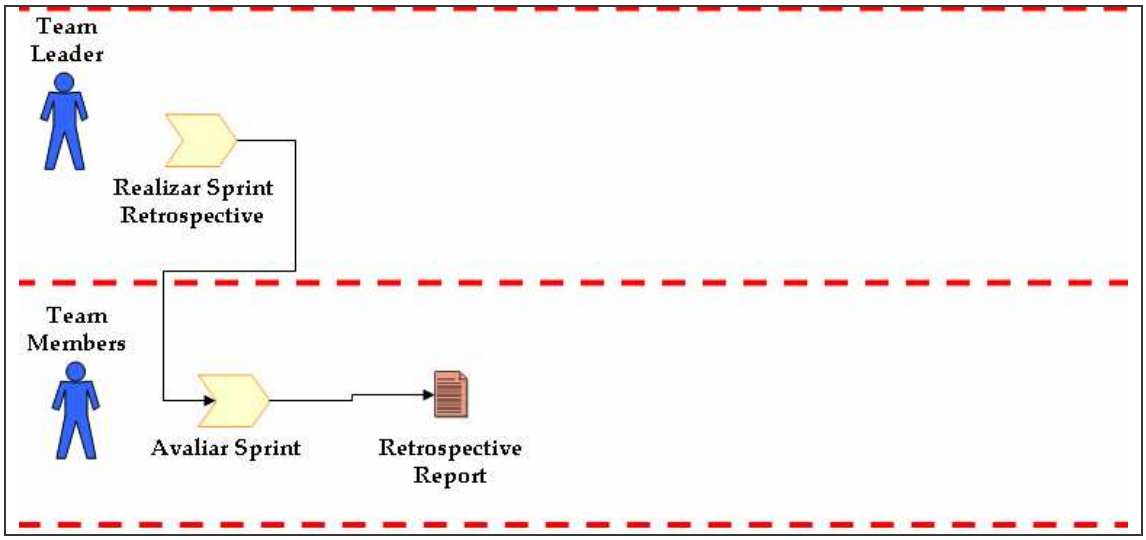


Figura 41 – Fluxo de Atividades – Environment.

Este encontro envolve a todos que participaram da realização do último *Sprint*. O convite ao *Product Owner* é opcional. O principal resultado deste encontro é o *Retrospective Report*. Cada membro contribui com a construção deste artefato através da discussão sobre os pontos positivos e negativos do último *Sprint*. Além disso, a equipe pode decidir tomar medidas para resolver os problemas enfrentados e maximizar o sucesso daquilo que funcionou bem. Dessa maneira, o *Retrospective Report* funciona como uma poderosa ferramenta de aprendizado da equipe e da organização. É bastante similar aos *postmortems* realizados ao final de projetos. No entanto, estes encontros são realizados várias vezes durante o ciclo de desenvolvimento, o que permite que o aprendizado ocorra de maneira eficaz ainda durante o desenvolvimento do jogo.

Portanto, este artefato é um mecanismo de crescimento da maturidade da equipe através do reconhecimento de suas fraquezas e adoção de ações que permitam todos enfrentarem essas deficiências.

5.7 Metodologia - Práticas Sugeridas

Esta Seção apresenta um conjunto das melhores práticas de desenvolvimento e gerenciamento ágil de software. Muitas delas foram inspiradas em práticas de *Scrum* [Schwaber04] e *Extreme Programming* [Beck04], e de metodologias ágeis em geral.

5.7.1 User Stories

Estória do usuário (ou *User Story*) é uma forma de expressar requisitos de sistemas de software através de uma ou duas sentenças em linguagem natural. Em geral, as estórias são escritas por um usuário (ou cliente) do sistema final [Wiki067]. Contudo, nesta metodologia, pelo fato dos clientes não necessariamente entenderem suficientemente de jogo, as estórias são criadas pela equipe do projeto utilizando como referência o *Game Concept* criado no início do projeto. Posteriormente, estas estórias são enriquecidas através da criação do *Game Design Specification*. Entende-se que boas estórias são [Cohn04]:

- **Independentes**: Devem ser suficientemente independentes para evitar introduzir dependências entre as estórias. Isto pode causar problemas de priorização e planejamento. Por exemplo, uma estória de altíssima prioridade pode apresentar dependência com uma estória de prioridade insignificante. Assim, a priorização fica deficiente;
- **Negociáveis**: Estórias não são contratos ou extensos documentos de requisitos. Por isso, em momentos posteriores, suas descrições podem ser mais detalhadas e, assim, sofrerem negociações;

- **Relevantes**: Estórias devem ser relevantes para os usuários finais (ou clientes). Cada estória deve possuir um valor tangível para estes envolvidos;
- **Estimáveis**: Estórias devem ser passíveis de serem estimadas. Uma estória deverá ser implementada e, para tanto, o esforço para realizá-la deve ser considerado nas atividades de planejamento;
- **Pequenas**: O tamanho de uma estória influi no planejamento. Logo, se você tem estórias muito grandes ou estórias muito pequenas. O ideal é que você tenha estórias curtas, coesas e simples. Dessa maneira, a capacidade de estimar será privilegiada;
- **Testáveis**: Uma estória deve ser testável. Cada estória possui um conjunto de testes sob os quais ela deve ser executada e aprovada.

Estórias é uma forma eficiente de tratar requisitos sem que haja necessidade de lidar com extensos documentos e bastante esforço para mantê-los consistente. A intenção é responder rapidamente às mudanças sem muito *overhead*. Inicialmente, estórias são descrições informais de requisitos, entretanto, posteriormente, testes de aceitação mais formais são escritos para que possa ser estabelecida a realização ou não da estória.

Um dos principais desafios do uso de estórias do usuário é como estimá-las. Uma técnica bastante difundida nos últimos tempos para estimativa em estórias de usuário é a técnica de ***Story Points***.

5.7.2 Story Points

Story points é uma técnica de estimativa para estórias. Atualmente, não existe um padrão para a utilização de *Story Points*. Uma equipe pode definir um *story point* como um dia ideal de trabalho, que significa um dia de trabalho sem interrupções, sem e-mails, sem reuniões, etc. Uma outra equipe pode definir um *story point* como uma semana ideal de trabalho, que é uma semana onde todos os dias de trabalho são ideais. Para facilitar o gerenciamento e desenvolvimento de estórias, esta metodologia sugere que um *story point* seja definido como um dia ideal, isto é, oito horas de trabalho ininterrupto. Assim, cada estória pode ser estimada em termos de *story points* [Cohn04].

Uma outra maneira de estimar o tamanho de estórias através de *story points*, de forma análoga sugerida funciona da seguinte forma [Bado05]:

- Primeiro *Release*
  - Classificar as estórias de acordo com a sua complexidade em níveis;
  - Atribuir à estória mais simples 1 (um) ponto. Atribuir à estória mais difícil 5 (cinco) pontos;
  - Atribuir às outras estórias pontuações que estão entre 1 (mais fácil) e 5 (mais difícil). Fazer isso comparando as novas estórias com as que já possuem pontuação atribuída. Se, por exemplo, uma nova estória é 2 vezes mais difícil de ser implementada do que a estória mais fácil, atribua a esta estória 2 (dois) pontos.
- *Releases* Seguintes
  - Atribuir pontos às novas estórias comparando-as com as estórias previamente implementadas.

A utilização de *story points* como medida de tamanho em projetos baseados em estórias é extremamente vantajosa por provê um método sistemático e rápido. Além disso, quando se descobre erros em estimativas, o seu ajuste é extremamente simples.

5.7.3 Test Driven Development e Continuous Design

Test-Driven Development (TDD) é uma técnica de programação, ou estilo de desenvolvimento de software que envolve – repetidamente – escrever um caso de testes e, então, implementar apenas o código necessário para passar no teste. TDD provê feedback rápido a quem está programando. Praticantes de TDD alegam que não se trata apenas de um método de teste, mas de *projeto de software*. Afinal, você tem que pensar no problema (modelando-o) e implementar a solução [Wiki068].

Esta estratégia confere forte caráter de simplicidade ao desenvolvimento, fazendo com que o programador se preocupe em criar "aquilo mais simples que possa funcionar". Por isso, o *projeto do software* é continuamente criado (***Continuous Design***) e refinado. Bastante refatoração (melhoria da estrutura do código) é realizado como forma de melhorar o projeto, provendo assim, qualidade ao projeto de software construído.

5.7.4 Comunicação Transparente

A comunicação entre todos envolvidos no projeto deve ser transparente. Aconselha-se que a equipe de projeto diariamente realize um

encontro de curta duração que sirva para cada membro expor o trabalho recentemente realizado por ele, o trabalho que está por começar e possíveis impedimentos para este trabalho. Este encontro ajuda a manter o foco de todos os membros e trata de problemas sem esperar que eles possam trazer conseqüências catastróficas para o projeto. O encontro diário receber o nome de ***Daily Meeting***.

5.7.5 Ambiente de Trabalho Otimizado

O ambiente de trabalho pode ser decisivo para o sucesso de um projeto. A primeira recomendação realizada quanto ao ambiente de trabalho é trabalhar junto. Pessoas de um mesmo projeto devem – idealmente – sentarem-se perto umas das outras para facilitar a comunicação. Esta prática é conhecida em *Extreme Programming* como ***Sit Together***.

O aspecto de um gráfico de *Sprint Burndown Report* é similar ao de uma escada, indicando que indivíduos não costumam trabalhar nos finais de semana. Kent Beck, em sua primeira edição do "Extreme Programming Explained" defende que se uma equipe tiver que realizar turnos extras por mais de uma semana é porque houve erro de planejamento. Turnos extras podem acontecer, mas não como costume. Beck ainda defende que o "trabalho sem descanso provoca uma queda drástica na produtividade do individuo". Esta prática de trabalhar semanalmente apenas a carga horária norma recebe o nome de ***Energized Work***.

Além disso, o ambiente deve ter a cara de uma equipe de desenvolvimento de projetos de software (jogos, neste caso). Quadros para rascunho, quadros de aviso, e toda forma de estrutura do ambiente que possa

ajudar a maximizar a quantidade de informação passada por ele. Esta prática recebe o nome de ***Informative Workspace***.

5.7.6 Feedback Constante e Ciclos Curtos

O ***feedback* constante** deve ser uma obsessão da equipe. O *feedback* pode vir de companheiros de equipe, do *Product Owner*, ou outros envolvidos. Entretanto, devido às dificuldades do dia, por exemplo, um *Product Owner* pode não estar envolvido no dia-a-dia da produção do jogo. Por isso, recomenda-se que os ciclos de desenvolvimento sejam realizados em ***ciclos curtos*** para que o *feedback* não seja tardio. Esta prática facilita o planejamento e diminui os riscos de re-trabalho, colaborando diretamente com o *feedback* constante.

5.7.7 Pair-Programming

Pair-Programming ou Programação Colaborativa é quando programadores se organizam em duplas e sentam-se no mesmo computador. Um dos dois assume o papel de "piloto" e o outro assume o papel de "co-piloto". Enquanto o primeiro se mantém focado no problema em que está trabalhando, o segundo tem uma visão mais ampla do problema. Alguns defendem que o uso de programação em dupla consome o dobro de recursos para se realizar uma funcionalidade.

Entretanto, muitos projetos têm utilizado programação em duplas com amplo sucesso. Embora possa consumir mais recursos numa primeira observação, a programação em pares faz com que o tempo absoluto de entrega de uma funcionalidade seja menor do que quando realizada por um

programador individualmente. Além destes motivos, muitos outros motivos fazem a adoção de *pair-programming* ser fortemente sugerida. Entre alguns deles [COCKBURN&WILLIAMS]:

- **Econômicos**: Software produzido utilizando programação em pares costuma apresentar muito menos erro. Assim, o tempo de depuração é bem maior em projetos que não utilizam esta prática;
- **Satisfação**: A maioria dos programadores prefere o trabalho colaborativo e com feedback e comunicação maximizados;
- **Qualidade de Projeto**: A solução final apresentava melhor estrutura do ponto de vista de boas práticas de projeto, como acoplamento, coesão e legibilidade, em projetos que adotaram *pair-programming*;
- **Aprendizado**: O aprendizado se mostra muito mais eficiente e rápida nos casos em que programação em pares é utilizada.

Portanto, por todos estes motivos, o uso de *pair-programming* é extremamente aconselhado.

6. Conclusão

"Imagination is more important than knowledge."
-Albert Einstein

Este capítulo discute as conclusões obtidas a partir deste trabalho e sugere possíveis melhorias futuras para tornar este trabalho melhor e mais completo.

6.1 Conclusões

Advergames, isto é, jogos publicitários é uma grande aposta do mercado de propaganda. Cada vez mais, as mídias necessitam ser interativas e emitir a mensagem publicitária de maneira eficiente, por isso, muitos apostam no poder de imersão de jogos eletrônicos como forma de alcançar maior eficiência e tamanho de público em suas campanhas.

Atualmente, projetos de *advergame* podem custar de US$ 10mil a US$ 500mil, sendo extremamente críticos do ponto de vista de estratégia e custo para as diversas organizações envolvidas em tais projetos. A criação de um *advergame* precisa ser suportada por um método de desenvolvimento e gerenciamento capaz de tratar as maiores dificuldades apresentadas pelo

desafio demandas por tal tipo de projeto. Entre os principais desafios destes projetos está:

- Fortes restrições de cronograma;
- Ambiente complexo e dinâmico de negócios;
- Equipes multidisciplinares;

O cenário crítico dos resultados de projetos de jogos foi identificado analisando-se resultados da indústria de software como um todo, e alguns *postmortems* de projetos de desenvolvimento de jogos. O discurso quase sempre era o mesmo: "O projeto está atrasado, o produto não contém as funcionalidades especificadas e apresenta muitos *bugs*, além disso, o orçamento já ultrapassou o previsto (...)". Enfim, as taxas de sucesso de projetos de jogos mostravam-se extremamente insatisfatórias para os padrões industriais.

Dessa forma, na ausência de pesquisas que revelassem detalhadamente as causas de falhas de projetos de jogos, decidiu-se analisar metodologias de desenvolvimento de software utilizadas para projetos de jogos. Muitos processos (ou metodologias) utilizados se mostravam problemáticos em sua estrutura ou em vários outros aspectos que podem ser resumidos como problemas de comunicação. Alguns processos eram pouco flexíveis, uns desconsideravam a característica multidisciplinar da equipe, e outros abordavam, por exemplo, somente aspectos gerenciais.

Logo, a solução encontrada foi sugerir uma metodologia apropriada e completamente aderente às necessidades da criação de um *advergame*. Neste contexto, *Agile Game Process* (AGP) surge como uma alternativa ao antigo modo de desenvolvimento de jogos. AGP é uma metodologia adaptativa baseada nos valores ágeis e com características próprias à realidade deste tipo de projeto. A organização da metodologia (em papéis, artefatos, atividades, fluxos de atividades, disciplinas e guias) provê um corpo de conhecimento

muito rico do ponto de vista de ferramentas gerenciais e de desenvolvimento. Este corpo de conhecimento é suficientemente abstrato (para poder ser adaptado a diferentes realidades) e, ao mesmo tempo, detalha elementos dinâmicos e estáticos da metodologia sendo rico em informação para os usuários desta metodologia.

Utilizando os mesmos critérios (relevantes para projetos de *advergames*) definidos para a análise de metodologias utilizadas para desenvolvimento (e gerenciamento) de projetos de jogos (na Seção 4.4), temos os seguintes critérios e resultados obtidos:

I. **Flexibilidade**: Trata-se da capacidade de lidar com ambientes dinâmicos e complexos, estando apto para responder às mudanças, quando necessário;
II. **Comunicação**: Trata-se da capacidade de integrar toda a equipe (negócios, produção) e os envolvidos no projeto (*stakeholders*) através de mecanismos de comunicação simples e eficazes, fazendo com que todos estejam informados sobre o andamento e as necessidades do projeto;
III. **Suporte à Multidisciplinaridade**: Trata-se da capacidade de integrar indivíduos de diferentes especializações numa mesma equipe;
IV. **Gerenciamento Descentralizado**: Trata-se da capacidade de delegar algumas decisões a agentes independentes mais aptos a escolherem o melhor caminho para alcançar determinado objetivo;
V. **Tratamento de Riscos**: Trata-se da capacidade de agir proativamente para evitar que os riscos se concretizem e, quando eles ocorrerem, estar apto para responder a eles;
VI. **Valor:** Trata-se da capacidade de gerar valor antecipadamente durante o processo de desenvolvimento;

VII. **Suporte ao Desenvolvimento**: Trata-se da capacidade de prover boas práticas, nas diferentes áreas, para suportar o desenvolvimento.

	I	II	III	IV	V	VI	VII
AGP	••••	••••	••••	••••	••••	•••	•••
• Ruim •• Regular ••• Bom •••• Excelente							

Tabela 5 – Análise da AGP.

Entre as inúmeras vantagens da metodologia proposta, encontra-se seu completo alinhamento com as necessidades da um projeto de *advergame* e a integração e aderência entre processos de negócio e de produção. Além destes fatores, a metodologia maximiza a comunicação e colaboração entre os envolvidos no projeto, fazendo com a sinergia da equipe aumente e as probabilidades de sucesso do projeto sejam bastante altas.

Uma das desvantagens da metodologia apresentada é que ela não foi devidamente utilizada num amplo número de projetos para que se possa medir o sucesso dos resultados dos projetos e medir o próprio processo em si. Por isso, os trabalhos futuros são muito importantes para refinar a metodologia proposta neste trabalho. Outro importante ponto negativo nesta metodologia é a falta de ferramentas integradas que possam suportar todo o processo de desenvolvimento e gerenciamento. Além disso, acredita-se que o conjunto de práticas utilizadas possa ser amplamente melhorado com a utilização da metodologia em diversos projetos de maneira contínua.

6.2 Trabalhos Futuros

A metodologia apresentada neste trabalho pode ser melhorada em muitos aspectos. Entre eles:

- **Medir Resultados do Projeto**: Utilizar a metodologia numa organização com informações históricas sobre o desenvolvimento de jogos publicitários. Capturar os resultados dos projetos após a adoção da AGP. Então, comparar os dados para averiguar a eficiência (ou não) prometida pela AGP;
- **Ferramentas**: A criação de ferramentas integradas que possam suportar todo o processo de desenvolvimento e gerenciamento de projetos utilizando AGP pode ser alvo de muitos trabalhos futuros;
- **Gerenciamento de Portfólio**: AGP atualmente não suporta o gerenciamento de portfólio, isto é, de múltiplos projetos numa mesma organização. Estes projetos, em geral, compartilham inúmeros recursos e o gerenciamento de recursos compartilhados é um dos problemas mais críticos do contexto de gerenciamento de projetos.
- ***Framework***: Criação de um *framework* (ou ferramenta) onde as pessoas possam instanciar sua própria metodologia baseada na estrutura da AGP;
- **Melhoria do Processo (ou Metodologia):** AGP disponibiliza em sua estrutura uma disciplina voltada para a melhoria contínua do processo. Esta disciplina se chama *Environment* e busca melhorar tanto as práticas gerenciar e de desenvolvimento, como a própria estrutura do processo. Com a execução de muitos projetos utilizando AGP, a melhoria do processo ocorrerá de maneira natural e praticamente transparente. Assim, chegaremos a um ponto onde será possível, diante da maturidade da equipe, detalhar a metodologia num nível de atividades mais concretas. Um trabalho atual neste sentido, onde as atividades são muito bem definidas, é o trabalho para desenvolver uma "Metodologia Preditiva para Desenvolvimento de Jogos de Computador" [Carvalho06], em

que o processo é mais robusto, complexo, e mais detalhado. Isto se deve ao fato de que jogos de computador têm um ciclo de vida diferente de *advergames* e que suas restrições também não são as mesmas.

Referências

[Wiki061] Wikipedia – Enciclopédia Online. Jogos. Disponível em: http://en.wikipedia.org/wiki/Game. Acessado em: Agosto/2006.

[Wiki062] Wikipedia – Enciclopédia Online. Jogos de Computador. Disponível em: http://en.wikipedia.org/wiki/Video_game. Acessado em: Agosto/2006.

[Abragames05] A Indústria de Desenvolvimento de Jogos Eletrônicos no Brasil. Pesquisa publicada pela Abragames, Associação Brasileira das Desenvolvedoras de Jogos Eletrônicos. Ano: 2005.

[PMBOK04] Project Management Body of Knowledge. Project Management Institute (PMI). Edição: 3ª. Ano: 2004.

[Flynt05] FLYNT, J. Software Engineering for Game Developers. Premier Press. Ano: 2005.

[Flood03] FLOOD, K. Game Unified Process. Disponível em: http://www.gamedev.net/reference/articles/article1940.asp. Acesso em: Julho/2006.

[Demachy03] DEMACHY, T. Extreme Game Development: Right on Time, Every Time. Disponível em: http://www.gamasutra.com/resource_guide/20030714/demachy_01.shtml. Acessado em: Julho/2006.

[McGuire06] McGUIRE, R. Paper Burns: Game Design with Agile Methodologies. Disponível em:

http://gamasutra.com/features/20060628/mcguire_01.shtml. Acessado em: Julho/2006.

[Azevedo05] AZEVEDO, E.; et. al. Desenvolvimento de Jogos 3D e Aplicações em Realidade Virtual. Editora Campus. Edição: 1ª. Ano: 2005.

[Wiki065] Wikipedia – Enciclopédia Online. Lei de Moore. Disponível em: http://en.wikipedia.org/wiki/Moore's_law. Acessado em: Setembro/2006.

[Wiki063] Wikipedia – Enciclopédia Online. História dos Jogos de Vídeo Game. Disponível em: http://en.wikipedia.org/wiki/History_of_video_game_consoles. Acessado em: Setembro/2006.

[Esa05] 2005 Essential Facts about Computer and Video Game Industry. Pesquisa publicada pela Entertainment Software Association (ESA). Ano: 2005.

[Mahoney88] MAHONEY, M. The History of Computing in the History of Technology. Disponível em: http://web.mit.edu/STS.035/www/PDFs/mahoney.pdf. Acessado em: Agosto/2006.

[Sommerville03] SOMMERVILLE, I. Engenharia de Sofware. Addison-Wesley Press. Edição: 6ª. Ano: 2003.

[Wiki064] Wikipedia – Enciclopédia Online. Engenharia de Software. Disponível em: http://en.wikipedia.org/wiki/Software_Engineering. Acessado em: Setembro/2006.

[Standish94] Chaos Report 1994. Pesquisa publicada pelo The Standish Group International. Disponível em: http://www.standishgroup.com/sample_research/chaos_1994_1.php. Acessado em: Julho/2006.

[Standish01] Chaos Report 2001. Pesquisa publicada pelo The Standish Group International. Disponível em: http://www.standishgroup.com/chaos/introduction.pdf Acessado em: Julho/2006.

[RUP] JACOBSON, I.; et. al. The Unified Software Development Process. Addison-Wesley Press. Edição: 1ª. Ano: 1999.

[PMBOK00] Project Management Body of Knowledge. Project Management Institute (PMI). Edição: 1ª. Ano: 2000.

[UML] GUEDES, G. UML, Uma Abordagem Prática. Editora Novatec. Edição: 1ª. Ano: 2004.

[Mencher06] MENCHER, M. In the Beginning: A (Very) Brief History of The Videogame. Disponível em: http://www.gamasutra.com/features/20060724/mencher_01.shtml. Acessado em: Julho/2006.

[Hafberg04] HAFBERG, K.; et. al. Discover Iceland Through Advergame. Ano: 2004.

[Wiki069] Wikipedia – Enciclopédia Online. Advergames. Disponível em: http://en.wikipedia.org/wiki/Advergames. Acessado em: Agosto/2006.

[Chen01] CHEN, J.; et. al. Can Advergames Be the Future of Interactive Advertising? Ano: 2001.

[IGDA061] 2006 Casual Game White Paper. Pesquisa publicada pela International Game Developers Association. Disponível em: http://www.igda.org/casual/IGDA_CasualGames_Whitepaper_2006.pdf#search=%22IGDA%20White%20Paper%202006%22. Acessado em: Setembro/2006.

[Schwaber04] SCHWABER, K. Agile Project Management with Scrum. Microsoft Press. Edição: 1ª. Ano: 2004.

[Wiki066] Wikipedia – Enciclopédia Online. Rational Unified Process. Disponível em: http://en.wikipedia.org/wiki/Rational_Unified_Process. Acessado em: Agosto/2006.

[Agile01] BECK, K.; et. al. Agile Manifest. Disponível em: http://agilemanifest.org. Acessado em: Agosto/2006.

[Beck99] BECK, K; et. al. Extreme Programming Explained. Addison-Wesley Press. Edição: 1ª. Ano: 1999.

[Rising00] RISING, L. The Scrum Software Development Process. Disponível em: http://members.cox.net/risingl1/articles/IEEEScrum.pdf. Acessado em: Agosto/2006.

[Beedle04] BEEDLE, M. Agile Software Development with Scrum. Prentice Hall. Edição: 1ª. Ano: 2004.

[TeamSystem] Scrum for TeamSystem White Paper. Disponível em: http://scrumforteamsystem.com. Acessado em: Agosto/2006.

[DeMarco99] DeMARCO, T.; LISTER, T. Peopleware. Dorset House Publishing. Edição: 2ª. Ano: 1999.

[SPEM] Software Process Engineering Metamodel Specification. Disponível em: http://www.omg.org/technology/documents/formal/spem.htm. Acessado em: Setembro/2006.

[Beck04] BECK, K; et. al. Extreme Programming Explained. Addison-Wesley Press. Edição: 2ª. Ano: 2004.

[Wiki067] Wikipedia – Enciclopédia Online. Estórias de Usuário. Disponível em: http://en.wikipedia.org/wiki/User_Story. Acessado em: Agosto/2006.

[Cohn04] COHN, M. User Stories Applied for Agile Software Development. Addison-Wesley Press. Edição: 1ª. Ano: 2004.

[COCKBURN&WILLIAMS] COCKBURN, A.; WILLIAMS, L. The Costs and Benefits of Pair Programming. Disponível em: http:// collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF. Acessado em: Setembro/2006.

[Wiki068] Wikipedia – Enciclopédia Online. Test-Driven Development. Disponível em: http://en.wikipedia.org/wiki/Test_driven_development. Acessado em: Setembro/2006.

[Bado05] MENEZES, M. Estimativas Ágeis de Software com Extreme Programming. Trabalho de Graduação, Graduação em Ciência da Computação, Centro de Informática (CIn), Universidade Federal de Pernambuco. Ano: 2005.

[Carvalho06] CARVALHO, G. Metodologia Preditiva para o Desenvolvimento de Jogos de Computador. Trabalho de Graduação, Graduação em Ciência da Computação, Centro de Informática (CIn), Universidade Federal de Pernambuco. Ano: 2006.

Apêndice A - Glossário

Acoplamento – medida da quantidade de relações entre entidades computacionais. Quanto mais acoplado, mais difícil de modificar e entender um componente computacional.

Arcade – plataforma de videogames bastante difundida nos anos 70 e 80.

Assets – um ou mais elementos de um determinado tipo. Um conjunto de sons, por exemplo, formam os assets sonoros do jogo.

Banners – Espaço reservado para publicidade.

Bottom-up – Estratégia de resolução de problema que busca resolver vários problemas menores e ir ajustando-os, como forma de resolver um problema maior.

Bugs – Defeitos.

Business-case – Um caso de negócios, onde estão explicitas informações sobre a motivação do projeto, os objetivos, etc.

Casos de teste – Um caso de teste define um conjunto de situações em que um sistema precisa ser testado para avaliar a realização ou não de uma determinada funcionalidade.

Classes – Um modelo que descreve como objetos computacionais se comportam através de suas operações e suas características.

Coesão – Medida de quão coesa é uma entidade computacional. Uma entidade computacional é coesa se ela realizar apenas os serviços que se entende que ela deveria realizar.

Commit – Operação de submeter um conjunto de alterações em artefatos para um repositório remoto.

Compilar – Transformar um arquivo escrito numa linguagem de programação em um conjunto de códigos binários que possam ser executados por um computador.

Componentes – Partes integrantes de um software.

Compreensibilidade – Capacidade de entendimento e legibilidade de um conjunto de componentes computacionais.

Computação – capacidade de utilizar funções para transformar entradas em saídas.

Comunicação – Capacidade de transmitir dados entre dispositivos de informática.

Concept – Conceito.

Console – Dispositivo criado para propagação do videogame como instrumento do entretenimento doméstico.

Controle – Área da informática voltada para a coordenação entre dispositivos computacionais.

Diagramas de Caso de Uso – Diagrama de UML [UML] voltado para identificar as formas de iterações de agentes com um sistema, sob um ponto de vista estático.

Diagramas de Classe – Diagrama de UML [UML] voltado para exibir a estrutura de um sistema.

Diagramas de Estados – Diagrama de UML [UML] que mostra o sistema como uma máquina de estados.

Diagramas de Seqüência - Diagrama de UML [UML] voltado para identificar as formas de iterações de agentes com um sistema, sob um ponto de vista dinâmico.

Download – ato de baixar algum arquivo (ou dado) de um endereço remoto para um endereço local.

DVD – Digital Versatile Disc, um novo formato de mídia.

Framework – São estruturas genéricas que podem ser utilizadas e adequadas a várias situações do desenvolvimento de projetos de software.

Game Designer – Profissional responsável pela criação da idéia por trás de um jogo.

Game-Play – ver jogabilidade.

Game-screen – Tela principal de um jogo. É onde o jogo se dá.

Interfaces 3D - Interfaces gráficas que impõe uma sensação de três dimensões espaciais aos usuários.

Jogabilidade – Conjunto de regras, escolhas e objetivos que o jogador (usuário) poderá adotar.

Métodos – Operações que objetos computacionais podem realizar.

Objetos – São abstrações computacionais para representar entidades do mundo real.

Online – status de estar conectado à Internet.

Overhead – Um custo adicional em processamento ou armazenamento que, como conseqüência, piora o desempenho de um programa ou de um dispositivo de processamento. Uma analogia pode ser feita para o uso da palavra "overhead" em outros contextos.

Portfólio – Conjunto de produtos, serviços, projetos ou programas de uma organização.

Postmortem – Designa o nome do encontro ou artefato gerado ao final de um projeto para avaliar a forma como a execução aconteceu, identificando pontos positivos e negativos.

Programa – Conjunto de arquivos executáveis que – juntos – provêem um ou mais serviços.

Projeto de Software – Disciplina de desenvolvimento de software que cria a arquitetura do sistema.

Release – Versão de um software.

Report – Relatório.

Scripts – Rotinas para execução de alguma atividade.

Sistemas Computacionais – Sistemas de computador.

Sistemas de Informação – Um programa que provê informação a outros programas ou a usuários.

Sites – Página de Internet.

Stakeholders – Envolvidos no projeto que podem, de alguma forma, afetar/ ser afetados o/pelo projeto, respectivamente.

Update - Operação de receber um conjunto de alterações em artefatos de um repositório remoto.

Viral – Massiva.

Web – World Wide-Web, meio mundial que permite o acesso a páginas de Internet.

Workflows – Conjunto de Atividades.