Adoção de Metodologias Ágeis para Produção de Jogos Sociais com Times Distribuídos

João E. Ambrósio Gomes, Dhiego A. de Oliveira Martins, Jamilson Batista Antunes
Lenin E. Abadié Otero, Vinicius Cardoso Garcia, Silvio Lemos Meira

Centro de Informática, Universidade Federal de Pernambuco (UFPE) - Recife - PE - Brasil

Resumo

A utilização de metodologias ágeis para desenvolvimento de software tem se tornado uma demanda em times geograficamente distribuídos. Este trabalho tem por objetivo relatar as experiências obtidas em uma fábrica de software *open source* para desenvolvimento de jogos sociais, adaptando a metodologia *Scrum* para o processo de desenvolvimento distribuído.

Palavras chave: Metodologias Ágeis, Desenvolvimento Distribuído de Software, Jogos Sociais.

Contato dos autores:
{jeag, daom, jba, leao, vcg, srlm, ispj}@cin.ufpe.br

1. Introdução

Com o crescente mercado de jogos sociais e as constantes exigências de mais recursos para conseguir atrair um maior número de usuários, empresas necessitam expandir a capacidade de produção e descentralização no desenvolvimento dos seus projetos, uma vez que o desenvolvimento centralizado de software tem se tornado cada vez mais oneroso e menos competitivo.

Como vantagens desse novo modelo de desenvolvimento de jogos têm-se benefícios, como a redução de custos e ganho de produtividade. Contudo, como grande parte dos desafios na Engenharia de Software não é limitada apenas a aspectos técnicos [1], o desenvolvimento distribuído de software (DDS) ainda deixa muitas dúvidas quanto a sua real eficácia, como, por exemplo: times distribuídos têm a mesma eficiência que times centralizados? A comunicação distribuída é tão eficiente quanto à comunicação síncrona? Os processos de software atuais são capazes de lidar com as características do desenvolvimento distribuído enquanto garantem a qualidade do produto?

Neste contexto é possível visualizar os métodos ágeis como solução para auxiliar no desenvolvimento distribuído de jogos, acelerando o rítmo no desenvolvimento e organização dos times e ganhando produtividade. Como exemplo, tem-se um *framework* baseado em princípios ágeis, denominado *Scrum*, no qual foi projetado para acrescentar foco, comunicação, clareza, e transparência para desenvolvimento de software.

Com base nisso, este trabalho apresenta a experiência do desenvolvimento de um jogo social multiplaforma utilizando-se a metodologia *Scrum* para auxilio no gerenciamento de times distribuídos, ferramentas e técnicas apropriadas para gerenciamento e organização dos times.

Este trabalho está organizado conforme se segue: a Seção 2 apresenta uma visão geral de desenvolvimento de jogos sociais *open source*; a Seção 3 descreve a metodologia *Scrum*; a Seção 4 apresenta o estudo de caso: modelo ágil com times distribuídos para desenvolvimento de jogos sociais; a Seção 5 descreve os principais problemas enfrentados e as lições aprendidas; e finalmente, a Seção 6 apresenta as conclusões.

2. Visão Geral de Desenvolvimento de Jogos Sociais Open Source

Segundo Raymond [2], software *open source* é desenvolvido por times auto-organizados e distribuídos, que raramente se reúnem presencialmente, coordenando suas atividades através de ferramentas específicas. O mesmo autor, ainda descreve um trabalho relevante sobre o processo de software *open source* quando associa o desenvolvimento nas comunidades de software livre, desprovido de qualquer processo formalizado, a um "Bazar" no qual as contribuições ocorrem *ad hoc*. Enquanto que o modelo de desenvolvimento tradicional de software está associado a uma "Catedral" e possui um processo formal bem definido.

De acordo com Lovell [3], jogos sociais são aplicativos de entretenimento que utilizam as plataformas da *web social*, com o objetivo de se propagarem. Estes apresentam como principais características a formação de "meta-redes" entre os jogadores (adicionar amigos nas redes sociais), compartilhamento de pontuações, geração de desafios e competições entre os "adversários virtuais".

Conforme descrito por González e Robles em [5], muitos são os benefícios do modelo de desenvolvimento *open source*, tais como: (i) realização dos *releases* mais frequentes; (ii) baixo custo dos projetos, visto que desenvolvedores e testadores trabalham de forma voluntária; e, (iii) alta qualidade e confiabilidade, visto que são muitas pessoas revisando e testando o mesmo código em arquiteturas e ambientes distintos.

3. Scrum

O *Scrum* é um *framework* para criação de produtos complexos que vem sendo utilizado desde a década de 90 [6], e destaca-se dos métodos ágeis existentes pela maior ênfase dada ao gerenciamento do projeto. O *Scrum* é formado por um conjunto de boas práticas de gestão que admite ajustes rápidos, acompanhamento e visibilidade constantes e planos realísticos. Segundo Shuwaber e Sutherland [7], este *framework* de processo ágil se fundamenta na teoria de controle de processos empíricos, utiliza uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e controlar ímpetos**.**

Este *framework* baseia-se ainda em princípios como: equipes pequenas de, no máximo, sete pessoas; requisitos que são pouco estáveis ou desconhecidos; e iterações curtas.

Consiste em um conjunto formado por Times de *Scrum* e seus papéis associados, *Time-Boxes* (eventos com duração fixa), Artefatos e Regras.

Os Times *Scrum* são auto-gerenciáveis, multidisciplinares e trabalham em interação, visando otimizar flexibilidade e produtividade. Cada Time de *Scrum* possui três papeis: 1) O *Scrum Master*: um líder, um facilitador; responsável por garantir que o processo será compreendido e acompanhado; 2) O *Product Owner*, responsável por elevar ao máximo o valor realizado pelo Time de *Scrum*, serve como ponte entre o cliente e o fornecedor; 3) o Time, que tem por função trabalhar na execução das *tasks* que resultarão em uma *release*.

Os principais artefatos do *Scrum* [7] são: *Product Backlog*, que consiste em uma lista priorizada de tudo que pode ser necessário no produto (requisitos); *Sprint Backlog*, uma lista de tarefas para transformar o *Product Backlog*, por uma *Sprint*, em um incremento do produto potencialmente entregável; Um *burndown* é uma medida do *backlog* restante pelo tempo; Um *burndown* de *release* mede o *Product Backlog* restante ao longo do tempo de um plano de *release*; E ainda um *Burndown* de *Sprint* que mede os itens do *Sprint Backlog* restante ao longo do tempo de uma *Sprint*. Já as Regras fazem a ligação entre os *Time-Boxes*, os Papéis e os Artefatos do *Scrum*.

Ao final de cada *Sprint* é realizada a reunião de revisão (*Sprint Review Meeting*) para que o *Team* apresente o resultado alcançado na interação ao *Product Owner*, para que ocorra a inspeção e adaptação das funcionalidades e do projeto. Em seguida, *Scrum Master* conduz a reunião de retrospectiva (*Sprint Retrospective Meeting*), onde os participantes relatam quais foram os impedimentos apresentados durante a sprint, isto com o objetivo de melhorar o processo e/ou produto para a próxima *Sprint*. Na seção 4 apresentaremos com detalhes a forma de planejamento do *Scrum* durante o desenvolvimento do jogo *Catch the Pigeon* e a sua adaptação para times distribuídos.

4. Estudo de Caso: Modelo Ágil com Times Distribuídos para Desenvolvimento de Jogos Sociais

O estudo foi realizado na fábrica de *software* MOSAIC (*Mobile Social Applications in the Cloud*). A fábrica é formada por 23 (vinte e três) alunos do curso de Pós-Graduação do Centro de Informática da Universidade Federal de Pernambuco, dentre eles mestrandos e doutorandos que fazem parte da estrutura organizacional da fábrica. Os alunos estão distribuídos de forma descentralizada, com equipes auto-organizadas e auto-gerenciáveis. Essa estrutura é composta por um Comitê Gestor, responsável pelas principais decisões estratégicas da Fábrica, times de produção liderados pelo *Scrum Master*, o *Product Owner* e os *Stakeholders*.

O projeto consiste no desenvolvimento de um jogo social e *open source* denominado *Catch the Pigeon* [14]. O jogo utiliza os serviços da rede social *facebook*, e foi desenvolvido para mais de uma plataforma, o que permite a sua execução em ambientes *web* ou móveis que utilizam o sistema operacional Android. O objetivo principal do jogo é salvar o pombo correio, impedindo a sua captura enquanto leva mensagens à rede social [14].

O presente estudo é apresentado seguindo os tópicos: metodologia *Scrum*, práticas da metodologia, organização e gerenciamento das fases na fábrica, e por fim a divisão dos times.

4.1 Metodologia

Segundo Keith [6] a utilização da metodologia *Scrum* para desenvolvimento de jogos, apresenta características específicas, e cada *Sprint* é composta pelas iterações: *Concept*, *Design*, *Coding*, *Asset Creation*, *Debugging*, *Optimizing*, *Tuning and Polishing*.

As iterações em uma *sprint* são organizadas dentro do ciclo de vida em cascata. Esta foi uma alternativa adotada pela fábrica MOSAIC com base nos projeto de jogos desenvolvidos pelo C.E.S.A.R (Centro de Estudos e Sistemas Avançados do Recife).

O processo adotado segue as atividades (levantamento de requisitos, análise e projeto, desenvolvimento, testes e operação), bem como a aplicação de práticas de testes de software, como, por exemplo, TDD (*Test Driven Development*). É apresentada na Fig. 1 uma visão geral do ciclo de trabalho.

Nos próximos tópicos serão abordadas as práticas adotadas do *Scrum* e organização dos times, bem como a adaptação das ferramentas de gerenciamento e configuração para essa metodologia.

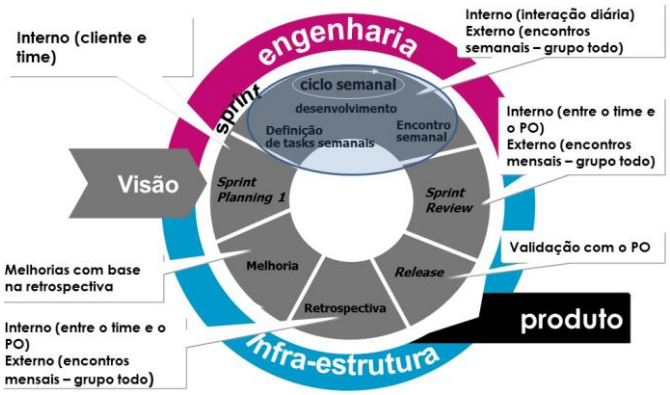

Fig. 1. Visão geral do ciclo de trabalho da Fábrica MOSAIC.

4.2 Práticas

Seguindo a organização proposta pelo *Scrum* para o gerenciamento, o processo da Fábrica MOSAIC é composto das seguintes práticas:

Sprint Planing Meeting : reunião com o objetivo de definir e identificar os requisitos que serão desenvolvidos na *sprint*.

Sprint: os itens definidos no *Product Backlog* serão implementados pela equipe, em períodos de tempo com duração de no máximo duas semanas (14 dias).

Daily Scrum: reuniões diárias, com duração de aproximadamente 5 minutos, entre os envolvidos do projeto. Através dessa reunião é possível ter conhecimento das atividades que estão sendo realizadas e da existência de um possível impedimento.

Sprint Review Meeting: reunião para revisão dos resultados obtidos na *Sprint* por parte dos envolvidos no projeto. Caso novos requisitos surjam durante a revisão, estes são adicionados ao *Product Backlog* para serem desenvolvidos na próxima *Sprint*.

Release: uma versão do jogo com melhorias agregadas e potencialmente entregável, que implementa as funcionalidades previstas para uma determinada *sprint*.

Sprint Retrospective: uma reunião realizada pela equipe na sequência da revisão da *sprint*, para refletir sobre a eficácia da equipe no desenvolvimento da *Sprint*, analisando formas de melhorar as suas práticas. Na fábrica MOSAIC, este ítem envolve o *scrum master* de cada time (*scrum de scrum's)*, juntamente ao Comitê Gestor, *Product Owner* e *Stakeholders*.

4.3 Fases

Em função das características específicas da aplicação, as *sprints* são compostas pelas atividades ilustradas na Fig. 2 e descritas a seguir:

Concept: idéias são geradas, possivelmente prototipadas, e textualizadas em uma base regular. Exemplo: a criação do documento de game design, com a descrição do *core* do jogo (roteiro, cenários, personagens) e especificação de requisitos.

Design: projeto das ideias geradas, estudo de viabilidade. Neste estágio, além de projetar como as funcionalidades serão implementadas, pode-se projetar testes de unidade para cada uma delas. Exemplo: os programadores irão definir o projeto (classes, métodos, relacionamentos, entre outros) de como as ações referentes ao vôo dos personagens serão implementadas em cada plataforma.

Coding: codificação de acordo com o que foi projetado na fase anterior. Exemplo: codificação das ações relacionadas ao vôo das personagens, na linguagem específica de cada ambiente (móvel e web). Com a utilização de TDD, os testes de unidade e os seus códigos associados são construídos em conjunto.

Asset creation: componentes relacionados à arte (como as figuras das personagens, cenários, entre outros), aos sons do jogo (quando uma personagem é atingida ou morre, por exemplo), e às animações (bater de asas das personagens, explosões, entre outras), ou implementações obtidas a partir da codificação, podem ser componentes (ou pacotes) que agregam determinadas funcionalidades a aplicação. Exemplo: um pacote que implementa o vôo de uma determinada personagem, como um pombo do mal.

Debugging: busca minimizar a ocorrência de bugs nas funcionalidades implementadas, por meio da aplicação de testes, identificação e correção de defeitos. Os erros encontrados precisam ser corrigidos antes da implementação da funcionalidade ser categorizada como concluída.

Optimizing: este estágio visa melhorar a aplicação de forma incremental, procurando corrigir eventuais problemas relacionados a requisitos não-funcionais, requisitos gráficos, requisitos de inteligência, entre outros.

Tuning and polishing: neste estágio, características relacionadas ao game design, por exemplo, podem ser refinadas para melhorar a experiência do usuário. Esta fase é dedicada a verificar o quão divertido é o jogo, ajustando a jogabilidade quando preciso. Tarefas de "polimento" podem ser alocadas no *Backlog*.

Todas estas fases são realizadas durante o ciclo de desenvolvimento de jogos adotado pela fábrica MOSAIC. Apesar de ser baseada no modelo cascata, a abordagem é distinta quando aplicada a metodologia *Scrum* para desenvolvimento de jogos, pois esse ciclo se repete a cada *Sprint*, já no modelo tradicional executa-se uma única vez.

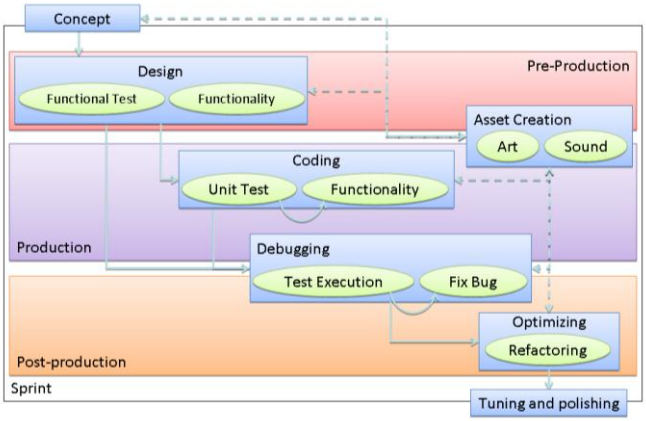

Fig. 2 - Atividades que compõem cada *sprint*.

4.4 Times

Além dos papéis inerentes ao processo *Scrum*, foram incorporados papéis específicos ao processo de desenvolvimento de jogos. Estes papéis compõem a equipe de desenvolvimento do grupo de game design, Web-Móvel, podendo haver casos de intersecção:

Arte Designer: responsável por desenhar personagens, cenários, instrumentos do jogo, efeitos resultantes das ações do jogador, ícones e menus do jogo.

Áudio Designer: será responsável por toda sonorização do jogo, desde sons emitidos pelos personagens, aos sons das armas e demais efeitos pertencentes ao jogo.

Animador: responsável por criar e manter as animações dos elementos existentes no jogo.

A equipe da fábrica encontra-se organizada em subgrupos com times constituídos de 2 a 8 membros. Os times selecionam o objetivo da *sprint* e especificam os produtos de trabalho em conjunto.

Alguns times em particular atuam no apoio ao desenvolvimento. Esses times definem, compartilham e melhoram continuamente processos, técnicas, padrões e ferramentas que apoiam os times, promovendo a capacitação dos mesmos. É apresentado na Fig. 3 uma visão geral da organização da fábrica, seus times e papéis.

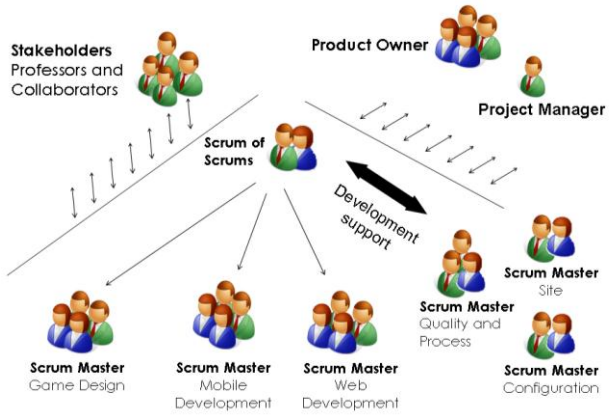

Fig. 3. Organização da Fábrica MOSAIC.

Ressaltando-se, ainda, que durante o desenvolvimento do projeto, pode-se contar com a participação de colaboradores externos. Estes colaboradores foram avaliados pelo líder do time em que o mesmo demonstrou interesse.

5. Problemas Enfrentados e Lições Aprendidas

Ao longo do desenvolvimento do projeto foi percebido que nem todas as práticas do *Scrum* eram diretamente aplicadas ao contexto de desenvolvimento distribuído de software. A seguir apresentam-se os maiores desafios para a utilização de *Scrum* no desenvolvimento de jogos sociais *open source* com times distribuídos e como foram solucionados:

- O *Scrum* defende a unidade da equipe de desenvolvimento. Isso está fortemente relacionado com a presença física da equipe e com iterações diárias. As equipes conseguiram superar o desafio de estar geograficamente distribuídas com o uso sistemático de fóruns, listas de discussões e ferramentas de comunicação (videoconferências, chats, entre outros).
- As reuniões diárias previstas no *Scrum* foram substituídas por 2 (duas) reuniões semanais de cada time, conforme o processo definido para a fábrica de software;
- O *Scrum* é focado em equipes auto-organizadas e auto-gerenciáveis, além de prever a questão motivacional como principal aspecto de sucesso do projeto. Esses mesmos aspectos puderam ser percebidos no desenvolvimento distribuído, onde a equipe poderia integrar um membro externo.
- A preocupação com os requisitos não funcionais deverá ser constante por parte de todos os times envolvidos, mas será mais observado pelo time de processo e game design.

Ao final do projeto teve-se como lições aprendidas as ferramentas, métodos e processos para adaptação do *Scrum* durante o desenvolvimento de um jogo social com times distribuídos. Proporcionando um gerenciamento descentralizado dos times envolvidos durante todo o desenvolvimento do projeto.

6. Conclusões

Este trabalho apresentou um estudo de caso sobre a aplicação de métodos ágeis, baseado na abordagem proposta pelo *Scrum*, em um processo para desenvolvimento de jogos socais *open source* com times distribuídos. Relatou-se a experiência adquirida durante todo o processo de adaptação da metodologia *Scrum* para o processo de desenvolvimento distribuído.

O projeto teve duração de 5 (cinco) meses (entre fevereiro e julho de 2011), durante este período foram criados 36 (trinta e seis) itens de *backlog* e realizadas 303 (trezentos e três) tarefas, dividas entre os times da fábrica MOSAIC.

Apesar do *Scrum* não cobrir todas as características especificadas para equipes distribuídas, foi possível fazer uso de diversos aspectos de desenvolvimento ágil sem comprometer os requisitos exigidos no projeto.

Referências

[1] Kontio, J., Höglund, M., Rydén, J. and Abrahamsson, P. (2004) "Managing Commitments and Risks: Challenges in Distributed Agile Development,". In Proceedings of the 26th International Conference on Software Engineering, pp. 732-733.

[2] RAYMOND, E. S. (1998) "The Cathedral and the Bazaar". Disponível em: http://www.firstmonday.org/issues/issue3_3/raymond/. Acesso em 15 Mai 2007.

[3] LOVELL, Nicholas, WHAT IS A SOCIAL GAME?, 2011. Disponível em: http://www.gamesbrief.com/2011/01/what-is-a-social-game/. Acesso em: 18 jun 2011.

[4] POSEA, Vlad; BALINT, Mihaela; DIMITRIU Alexandru; IOSUP, Alexandru. An Analysis of the BBO Fans Online Social Gaming Community, 2010.

[5] Gonzáles, J. e Robles, G.(2003) "Free Software Engineering : A Field to Explore", Upgrade - Software Engineering State of Art, Novática, volume IV, N. 4.

[6] KEITH, Clinton. Agile Game Development with Scrum (Addison-Wesley Signature Series. Editora: Addison-Wesley Professional; 1º ed, 2010. 384 pg.

[7] SCHWABER, KEN e SUTHERLAND,JEFF. Scrum, fevereiro 2010. Disponível em: <http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%20PTBR.pdf>. Acesso em: 17 mai. 2011.

[8] ZANONI, R. Modelo de gerência de Projeto Baseado no PMI para ambientes de Desenvolvimento de Software Distribuído. Dissertação de Mestrado, PUC/RS, Porto Alegre, RS, Brasil, 2002.

[9] AUDY, J.; PRIKLADNICKI, R. (2007). Desenvolvimento Distribuído de Software: Desenvolvimento de software com equipes distribuídas. Rio de Janeiro: Elsevier.

[10] KOMI-Sirvo, S; TIHINEN M. (2005). Lessons Learned by Participants of Distributed Software Development. Journal Knowledge and Process Management, vol. 12 n 2 p. 108 – 122.

[11] CARMEL, E. Global Software Teams – Collaborating Across Borders and Time Zones. Prentice Hall, EUA, 1999.

[12] PRIKLADNICKI, R. MuNDDoS Um Modelo de Referência para Desenvolvimento Distribuído de Software. Dissertação de Mestrado, PUC/RS, Porto Alegre, RS, Brasil, 2003.

[13]FREITAS, A. V. APSEE Global: a Model of Processes Manegement of Distributed Software Processes. Faculdade de Informática – UFRS – RS – BRAZIL, 2005.

[14] MOSAIC (2011). "Mobile Social Applications in the Cloud", Maio, 2011, Disponível em: <http://www.mosaic.eng.br>. Acesso em: 15 jun 2011.