Universidade de São Paulo
Instituto de Matemática e Estatística

Trabalho de Conclusão de Curso

Scrum no Desenvolvimento de Jogos Eletrônicos

Disciplina: *MAC499*
Trabalho de Formatura Supervisionado
Prof. Carlos Eduardo Ferreira

Orientador:
Prof. Flávio Soares Corrêa da Silva

Vinícius Kiwi Daros
vinicius@uspgamedev.org
Nº USP: 5893980

1 de dezembro de 2011

Conteúdo

Parte Objetiva

Introdução

Assim como qualquer atividade realizada em grupo, desenvolver jogos eletrônicos é uma tarefa que pode ser abordada de inúmeras formas. Acompanhando o trabalho de um grupo de estudos formado por alunos focados na criação de jogos, chamado *USPGameDev*, durante o ano de 2010, foi possível ver como uma abordagem ingênua, sem uma metodologia clara de trabalho, pode ser ineficiente, apesar de mesmo assim ter levado a um resultado satisfatório. Mas introduzir uma metodologia de desenvolvimento realmente aumenta a produtividade?

Este estudo visa responder essa questão. Para tanto, será mostrada uma comparação entre duas fases do *USPGameDev*. Na primeira, os desenvolvedores não aplicavam qualquer tipo formal de metodologia, trabalhando de forma relativamente livre. Já na segunda, os conceitos de ***Scrum*** foram introduzidos e o grupo trabalhou seguindo essa metodologia.

O que é *Scrum*

Scrum é um *framework* de gerenciamento de projetos usado principalmente no desenvolvimento ágil de *software*, apesar de poder ser aplicado em projetos de diferentes outras áreas. O processo de trabalho usando ***Scrum*** é iterativo e incremental, se baseando em ciclos e objetivando a entrega constante e rápida de versões de *software* que cada vez atendam melhor às necessidades do cliente.

Nesta seção, será apresentada uma visão geral dos principais conceitos do ***Scrum***, assim como suas práticas e fluxo de trabalho. Apesar desta seção não se restringir ao escopo de jogos eletrônicos, o foco das explicações será projetos de desenvolvimento de *software* dessa categoria.

História

Em 1986, Hirotaka Takeuchi e Ikujiro Nonaka publicaram um artigo intitulado *The New New Product Development Game*[1], no qual são descritas as características de gerenciamento do processo de desenvolvimento adotadas por algumas empresas que se destacavam por lançar produtos inovadores de forma rápida e com grande sucesso. A estratégia dessas empresas se baseava em mover todo o time como uma unidade em direção ao objetivo, chamada então de *abordagem holística* ou *rugby*.

Em 1990, Peter DeGrace e Leslie Hulet Stahl publicaram o livro *Wicked problems, righteous solutions*[2], onde denominaram essa abordagem por ***Scrum*** devido à formação do rugby de mesmo nome, representada na figura 1. Essa foi a primeira vez que ***Scrum*** foi proposto como modelo de desenvolvimento de *software*.

Finalmente, em 2002, Ken Schwaber e Mike Beedle escreveram o livro *Agile Sofware Develpment with Scrum*[3] baseando-se nas experiências profissionais que tiveram ao aplicar as propostas de DeGrace e Stahl na prática. Esse foi o título que popularizou o ***Scrum***.

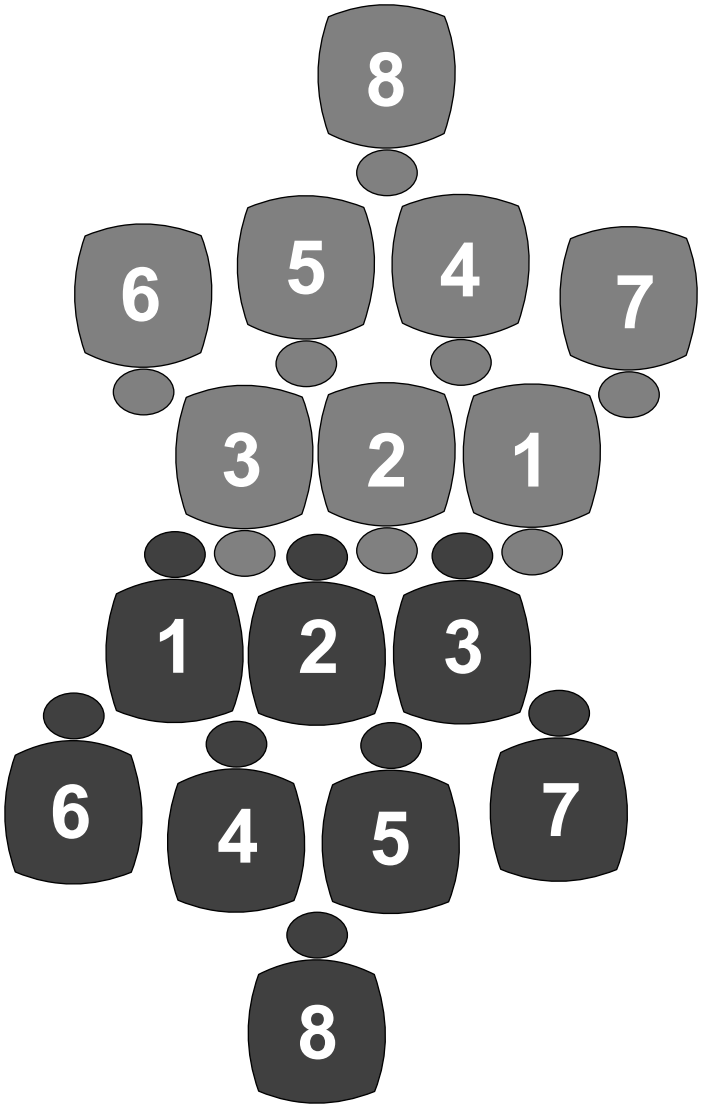

Figura 1: Formação Scrum

Visão geral

Antes de detalhar os elementos do ***Scrum***, será dada uma breve descrição geral do fluxo de atividades envolvidas no processo. Isso se justifica, pois ter em mente a ideia do ciclo de trabalho como um todo ajuda na compreensão da influência de cada prática e como elas

interagem entre si. A princípio, pode-se ter a impressão de que vários conceitos estarão sendo deixados soltos, sem explicação. Entretanto, todos os conceitos citados receberão atenção individual posteriormente.

Como já dito no início da seção, o ***Scrum*** é interativo, o que implica na existência de ciclos de trabalho, denominados *sprints* e que duram de duas a quatro semanas. Em cada *sprint*, o *product owner* verifica na lista de funcionalidades a serem implementadas, chamada de *product backlog*, aquelas que agregam maior valor ao produto e, consequentemente, tem maior prioridade. Para cada item selecionado, avalia-se quais tarefas menores precisam ser cumpridas para que a funcionalidade seja implementada. Em cada dia durante o *sprint*, o time de desenvolvedores se reúne no *encontro diário* (*daily meeting*) para discutir sobre as tarefas e acompanhar o andamento global do time. Em seguida, trabalha-se com a meta de cumprir os itens do *sprint backlog*, que é a lista das tarefas selecionadas para serem feitas durante a iteração. Ao fim do *sprint*, se espera que a nova versão do produto tenha as funcionalidades propostas e que seja utilizável. Nesse momento, o *product owner* verifica se a nova versão atende ao que foi requisitado, reordena o *product backlog* e dá início a uma nova iteração.

Essa sequência de eventos de um *sprint* é ilustrada na figura 2.

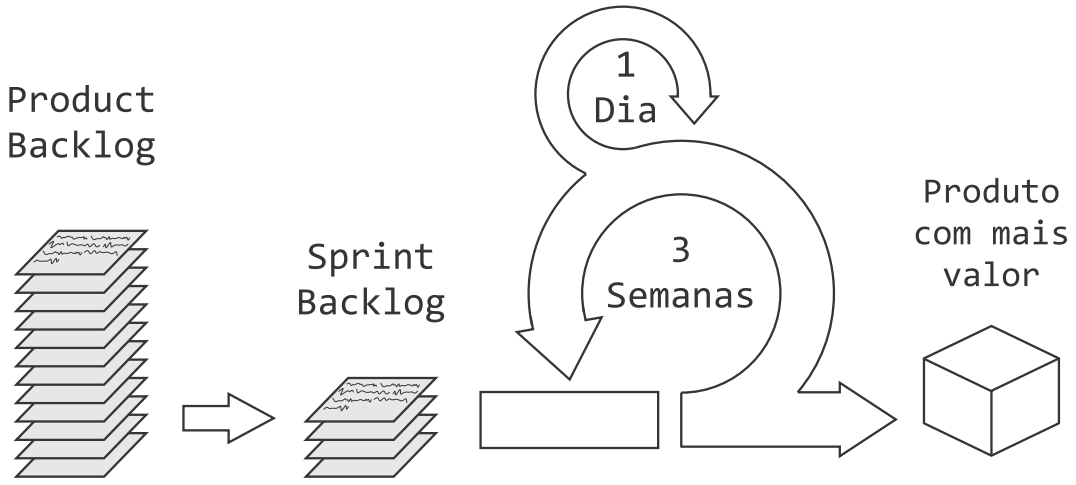

Figura 2: Processo de Scrum

Princípios do *Scrum*

O ***Scrum*** é um *framework* para gerência de projetos. Assim sendo, é natural presumir que suas práticas possam ser alteradas ou até mesmo não usadas de acordo com as necessidades e experiências de cada time. Entretanto, existem alguns princípios fundamentais que devem ser preservados. São eles:

Inspeção e adaptação: Deve-se constantemente inspecionar os artefatos e processos em busca de variáveis desfavoráveis, assim como mudanças no contexto do desenvolvimento, conhecimento e necessidades do cliente. Uma vez encontrada alguma dessas

situações, ajustes devem ser feitos o quanto antes, de modo a minimizar desvios de rota e garantir que o projeto sempre esteja progredindo em direção a um produto de valor crescente para o usuário;

Timeboxing: As atividades realizadas usando-se ***Scrum*** sempre possuem prazos, seja em semanas, dias ou minutos. Esses prazos são essenciais para que tanto desenvolvedores quanto os demais envolvidos no projeto possam se sincronizar;

Priorização: Nem todas as funcionalidades a serem implementadas tem o mesmo nível de importância e algumas delas nem mesmo geram valor real ao usuário do produto. Por essa razão, as tarefas a serem cumpridas devem ser classificadas por níveis de prioridade;

Auto-organização: Os times de desenvolvimento devem ter o poder e a responsabilidade de se auto-organizarem para que consigam cumprir dentro do prazo as tarefas com as quais se comprometeram. Para tanto, eles tem autonomia para modificar, adotar ou abolir quaisquer atividades e processos.

Papéis

O processo de desenvolvimento envolve pessoas com diferentes habilidades e que desempenham diferentes funções. A fim de maximizar a produtividade da equipe, o ***Scrum*** promove uma distinção bem definida das responsabilidades que cada pessoa deve ter. Essa distinção é feita através de papéis a serem desempenhados. Os três papéis adotados no ***Scrum*** são *product owner*, *scrum master*, time de desenvolvimento. Abaixo, cada um deles é descrito com mais detalhes.

Product owner

O *product owner* é, dentro do time, o representante das necessidades dos clientes. Ele serve de interface de comunicação, passando de forma clara para os desenvolvedores o que precisa ser feito e apresentando para os clientes as questões técnicas envolvidas no processo de produção.

Por esse motivo, o *product owner* é o responsável por lidar com o *product backlog*. É ele quem define quais tarefas devem entrar na fila, assim como avaliar a prioridade de cada uma delas antes de cada *sprint*. Ninguém mais deve realizar essas atividades. Com isso, tem-se uma situação na qual ninguém faz requisições de funcionalidades diretamente aos desenvolvedores e nem o time gasta esforços implementando itens que não foram previamente discutidos e cuja necessidade não tenha sido verificada. Ou seja, para que um item seja incluído no *product backlog*, é preciso convencer o *product owner* de que será agregado valor ao produto.

Como responsável pelo *product backlog*, o *product owner* deve garantir que todas as histórias sejam claras e que o time entenda exatamente o que se espera de cada uma delas. Além disso, outra obrigação dessa pessoa é participar do planejamento dos *sprints* e *releases*, sempre direcionando o andamento do projeto rumo a versões que atendam às necessidades mais críticas dos clientes.

Também é de responsabilidade do *product owner* garantir que o time como um todo compartilhe a mesma visão sobre o projeto. Isto é, tanto programadores quanto designers e artistas precisam conseguir visualizar o mesmo resultado esperado, ou de forma mais próxima possível. Assim, cabe ao *product owner* certificar-se que as histórias representam a mesma coisa para todos os integrantes da equipe.

Dadas essas responsabilidades, a pessoa que for desempenhar esse papel precisa ter uma visão ampla das necessidades do projeto, além de ser capaz de traduzir as necessidades dos clientes em tarefas e conseguir tirar as dúvidas dos desenvolvedores em relação a questões de negócio.

Scrum master

O *scrum master* é a pessoa responsável por introduzir e colocar em prática o ***Scrum***. Primeiramente, ele deve mostrar aos envolvidos no projeto como o ***Scrum*** funciona, quais os principais conceitos, quais práticas e atividades precisam ser realizadas e a motivação para a adoção desse método de trabalho. Ao longo do desenvolvimento, o *scrum master* será a referência para as outras pessoas e deve tirar todas as dúvidas de modo a compartilhar o máximo possível seu conhecimento.

Atuando como um verdadeiro instrutor, o *scrum master* ajuda cada um a entender o papel que tem no desenvolvimento do produto e de quais formas pode contribuir melhor, ou até mesmo como não atrapalhar os demais. À medida que essa mentalidade se distribuí e as pessoas passam a ser mais conscientes das maneiras como podem atuar melhor, maximiza-se o valor que o time consegue agregar ao produto a cada *sprint* e a velocidade do desenvolvimento aumenta, assim como a qualidade dos resultados parciais.

Outra incumbência de grande importância, senão a mais importante, atribuída ao *scrum master* é garantir que o time esteja sempre no ambiente mais favorável à realização de suas atividades e ao cumprimento de suas tarefas. Esse é um trabalho que envolve desde minimizar interferências externas, tais como gerentes exigindo que funcionalidades não planejadas para o *sprint* sejam implementadas para o dia seguinte, garantir que o time tenha material de escritório como cartões para escrever as histórias, lousas onde colar os cartões, *softwares* instalados nas máquinas de trabalho, dentre muitos outros.

Além disso, o *scrum master* deve se esforçar para rapidamente remover quaisquer impedimentos que eventualmente surjam. Entende-se por impedimento qualquer dificuldade que bloqueie o progresso do time no desenvolvimento do projeto ou que não permita a realização de alguma prática do ***Scrum***. Por exemplo, não ter acesso à internet no local de trabalho é um grande impedimento.

O *scrum master* também deve monitorar o progresso do grupo e, com base nesse acompanhamento, sugerir a implantação, modificação ou até mesmo abolição de algumas práticas. Com esse embasamento, é possível questionar de forma consciente algumas regras sugeridas pelo ***Scrum*** e fazer as adaptações e flexibilizações das práticas de modo a elas se moldarem melhor à equipe e cumprirem seu objetivo primário, aumentar a produtividade e não atrapalhar.

Em suma, a função do *scrum master* é guiar o time em direção ao aprendizado e mecanismos do ***Scrum***. Com isso, a necessidade de uma pessoa desempenhando esse papel tende a diminuir conforme a equipe se torna mais experiente.

Time

Idealmente, o time de desenvolvimento é formado por pessoas de diversas competências, cuja soma de habilidades seja capaz de cobrir todas as demandas que surjam durante o decorrer do projeto. Assim sendo, o time é basicamente composto pelas pessoas que trabalharão para transformar os requisitos em produto.

O time é responsável por se autogerir e se organizar de modo a saber com quanto trabalho pode se comprometer por *sprint* e conseguir entregá-lo ao fim do prazo. A principal obrigação do time é, portanto, entregar os conjuntos de funcionalidades descritos no *product backlog*.

Product Backlog

O *backlog* do produto consiste em uma fila de todas as funcionalidades, requisitos e quaisquer outras atividades que representam um trabalho a ser feito ou a ser implementado dentro do sistema. Essa fila é ordenada por prioridade, sendo que os itens mais prioritários ficam no início. Quanto mais próximo do início está um item, maior sua urgência. Mas também, mais discussão houve a seu respeito e maior é seu nível de detalhamento.

Os itens do *backlog*, também chamados de *histórias*, devem ser priorizados seguindo os critérios abaixo:

1. **Valor:**
   O principal critério para escolher qual história tem maior prioridade é o valor que ela traz para o usuário final. Além disso, algumas histórias não trazem valor em si, mas aumentam a produtividade do time. Isso também é considerado como uma forma de valor;

2. **Custo:**
   O custo é medido basicamente pelo esforço em se desenvolver uma funcionalidade. Assim, pode haver situações nas quais alguma história pode gerar grande valor, porém seu alto custo a inviabiliza. Em geral, quando duas funcionalidades tem o mesmo valor, dá-se maior prioridade àquela cujo custo é menor;

3. **Risco:**
   Risco implica em incerteza com relação ao valor e ao custo. Por esse motivo, tarefas mais arriscadas ganham maior prioridade, de modo que as incertezas sejam sanadas;

4. **Conhecimento:**
   Em alguns momentos, não se tem informação suficiente para estimar uma história. Nesses casos, pode-se fazer um *spike* - tarefa de curto prazo - para obter mais conhecimento sobre os custos e riscos. Por exemplo, fazer um protótipo para verificar a dificuldade em implementar troca de pacotes por rede é um *spike* que pode ajudar muito a estimar o esforço necessário para desenvolver um jogo multiplayer.

As histórias no *backlog* do produto são de altíssimo nível, não carregam muitos detalhes, apenas descrevem o resultado desejado após sua conclusão e são grandes, ou seja, sua complexidade é alta e é preciso implementar muitas coisas até que seja possível obter

o resultado descrito. Por conta dessas características, essas histórias são denominadas por *histórias épicas*.

Uma forma muito conveniente de trabalhar com as histórias é escrevê-las em cartões, com será melhor explicado adiante. Para padronizar a escrita e garantir que o texto passe claramente a mensagem do que deve ser feito e com qual finalidade, é recomendável a seguinte formatação:

> *Como [PAPEL], quero [OBJETIVO], pois com isso [MOTIVO].*

Papel: Quem será beneficiado diretamente com o cumprimento da história. Pode ser o jogador final, os designers, uma classe do próprio jogo, entre outros.

Objetivo: Descrição sucinta da funcionalidade desejada. Não necessariamente é uma atividade relacionada ao jogo em si, pois pode ser a construção de uma ferramenta que auxilie os artistas a testarem os mapas nos quais estão trabalhando, por exemplo.

Motivo: Qual é o benefício que se deseja conseguir com a realização da tarefa. Esse campo não é obrigatório, já que não é raro, principalmente em tarefas pequenas, que o benefício esteja implícito. Por exemplo, em *Como jogador, quero que exista um indicador dos meus pontos de vida* é perfeitamente razoável omitir o campo motivo. Apesar disso, é recomendável completá-lo pois a motivação geralmente ajuda a completar a ideia do objetivo.

A título de ilustração, segue abaixo um exemplo de história que poderia estar no *backlog* do produto:

> *Como jogador, quero que o cenário da cidade tenha várias pessoas andando de um lado para o outro, pois com isso passa-se melhor a sensação de se estar em uma metrópole.*

Adiante veremos que as histórias épicas devem ser quebradas em histórias menores, de menor complexidade e mais simples de serem resolvidas. Mesmo quando se tornam menores, as histórias devem continuar seguindo o mesmo formato. Por exemplo, a história acima poderia ser quebrada em outras três menores:

> *Como pessoa do cenário, quero andar sempre em um mesmo sentido, pois com isso passo a impressão que estou indo à algum lugar.*
>
> *Como pessoa do cenário, quero desviar das outras pessoas mas voltar a andar no meu sentido original depois disso.*
>
> *Como pessoa do cenário, quero andar apenas sobre as calçadas, pois com isso eu não sou atropelada.*

Há situações nas quais não vale a pena quebrar ainda mais uma história, mas é preciso adicionar mais detalhes ou restrições sobre o que deve ser feito. Nesses casos, registra-se junto com a história suas *Condições de Satisfação*, ou *CoS*, do inglês *Conditions of Satisfaction*. Segue um exemplo de história que se encaixa nesse quadro:

Como jogador, quero ver alguma reação dos inimigos quando eles forem atingidos, pois com isso consigo saber se acertei o tiro.

CoS:

- *Quando atingido na cabeça, o inimigo deve cair de costas;*
- *Quando atingido do lado esquerdo, o inimigo deve cair girando para esquerda;*
- *Quando atingido do lado direito, o inimigo deve cair girando para direita;*

Sprint Backlog

Assim como o *product backlog*, o *sprint backlog* é uma fila de tarefas a serem cumpridas. Entretanto, existem algumas particularidades.

Primeiramente, as tarefas no *sprint backlog* são menores e mais simples, pois são partes de uma história épica que foi quebrada. Além disso, essas tarefas possuem uma estimativa de quanto tempo será necessário para que cada uma delas seja cumprida, sendo que soma dessas estimativas não deve superar a capacidade de trabalho do time dentro de um *sprint*.

Outra característica particular do *sprint backlog* reside no fato de que nenhuma tarefa pode ser adicionada à fila durante o andamento do *sprint*. Desse modo, uma vez que o time tenha se comprometido a entregar certas funcionalidades em determinado prazo, os desenvolvedores ficam protegidos de receber exigências não planejadas e que atrapalhem o andamento da iteração. Com isso, tem-se a certeza de que o único momento no qual novas tarefas são adicionadas ao *sprint backlog* é durante a reunião de planejamento do *sprint*.

Sprint

Como já dito anteriormente, o *sprint* é a iteração dentro do ***Scrum***. Esse ciclo de trabalho deve ter uma duração fixa de duas a quatro semanas, isto é, o time deve decidir qual será o tamanho dos seus *sprints* e, uma vez feito isso, deve-se respeitar esse prazo e não prolongá-los ou encurtá-los.

A decisão do tamanho do *sprint* é particular para cada equipe e deve ser tomada de modo que o grupo se sinta hábil a cumprir tarefas dentro desse prazo, mas sem perder o senso de urgência proporcionado por não ter tempo sobrando. Em geral, se deve dar preferência a *sprints* mais curtos, pois, para equipes inexperientes, o ciclo rápido ajuda na familiarização do processo como um todo e, para equipes experientes, a alta frequência de contato com o cliente proporciona um direcionamento mais estreito para o andamento do projeto. A opção por *sprints* longos deve ser feita apenas quando há dificuldades em agendar reuniões constantes com os clientes e *product owners*, mas essa adaptação deve ser evitada sempre que possível.

Idealmente, um *sprint* pode ser particionado em cinco segmentos. A seguir, veremos as atividades feitas em cada uma dessas etapas.

Priorização

Todo *sprint* começa com uma reunião de priorização. Nessa reunião, o *product owner* ordena o *product backlog*, encontra a história de maior prioridade e a apresenta para o time de desenvolvimento. Nesse momento, o *product owner* explica exatamente o que espera que seja feito naquela história em particular e o time tem a oportunidade de tirar suas dúvidas quanto a questões de design e quais os comportamentos esperados da funcionalidade em diferentes situações de uso.

Juntamente com isso, o *product owner* define o *sprint goal*, que é o objetivo geral da iteração. O *sprint goal* geralmente acaba determinando o tema das tarefas e ajuda a equipe a manter o foco dos esforços sem se dispersar em tarefas não tão relevantes no momento. Por exemplo, se o objetivo do *sprint* é *"Ter inimigos que persigam o personagem principal"*, os desenvolvedores que estiverem implementando os mecanismos de tomada de decisão dos inimigos se sentirão menos tentados a adicionar no algoritmo de busca pelo personagem do jogador uma condição que faz o vilão alterar sua rota caso veja um item que pode ser pego. Mesmo que essa tentação exista, tendo um *sprint goal*, o pensamento de *"Mas isso não ajuda em nada os inimigos a se movimentarem em direção ao jogador. É melhor deixar isso para outra hora."* fica muito mais forte, o que contribui para se evitar o gasto de tempo em tarefas digressivas.

Feita essa etapa, o time julga superficialmente o tamanho da história e, caso ela tenha um tamanho adequado à capacidade de trabalho da equipe em um *sprint*, se encerra a reunião. Se a história for grande demais para ser implementada em apenas uma iteração, deve-se dividir a funcionalidade em duas ou mais histórias menores e, dentre essas, se escolhe aquela que o *product owner* acredita ter maior valor. Por fim, caso a história de maior prioridade seja considerada pequena ou fácil demais, se repete esse processo para a história seguinte no *product backlog*, de modo que o time tenha tarefas suficientes para não ficar ocioso durante o *sprint*.

Planejamento

Uma vez escolhidas quais histórias serão implementadas, a equipe deve organizar como será seu trabalho durante o *sprint*. Para isso, primeiramente deve-se fazer o levantamento dos possíveis obstáculos que podem surgir durante o iteração, tais como feriados, necessidade de aprendizado de alguma tecnologia específica, indisponibilidade do ambiente de trabalho, entre outros.

Na sequência, é feita uma discussão detalhada sobre cada uma das histórias do *sprint backlog*. O objetivo dessa conversa é definir o mais precisamente possível o que deve ser feito e como será feito. É nesse momento que finalmente os desenvolvedores deixam de falar em alto nível e discutem detalhes de implementação, por exemplo. Tendo ficado claro quais serão as atividades necessárias para se cumprir as exigências das histórias, se escrevem essas atividades no formato de tarefas e, para cada uma delas, é feita uma estimativa do tempo requerido para sua conclusão. Essa estimativa indica uma quantidade de "tempo ideal" necessário para se concluir a tarefa e é representada em pontos, não exatamente em horas e minutos. Por se tratar de um tópico ligeiramente mais complexo, uma explicação mais elaborada sobre estimativas será dada em uma seção dedicada ao assunto. Por hora, é suficiente considerar que a estimativa de cada tarefa dá uma medida

do quanto ela é trabalhosa.

Eventualmente, assim como acontece com as histórias, uma tarefa pode acabar ficando grande demais. Nessa situação, provavelmente é possível dividi-la em outras de menor complexidade e, consequentemente, mais rápidas de serem cumpridas.

Quando não se sabe o suficiente sobre uma tarefa para conseguir estimá-la ou mesmo dividi-la, deve-se considerá-la trabalhosa, dando-lhe uma estimativa alta. Uma vez que não se tem conhecimento suficiente nem para estimar a tarefa, provavelmente também não se sabe exatamente como cumpri-la. Por essa razão, é recomendável buscar mais informações sobre a questão e até mesmo criar um *spike*.

Assim que se tem todas as tarefas discutidas e estimadas, é hora de usar essa métrica. Soma-se o valor de todas as estimativas e confronta-se o resultado com a quantidade de pontos cumpridos nos últimos *sprints*. Se o resultado da soma estiver próximo ao valor médio dos *sprints* anteriores, ou ligeiramente menor, então há um forte indício de que o time conseguirá cumprir o que se propôs a entregar ao fim da iteração. Caso a soma tenha ficado maior do que a média anterior, é preciso avaliar a situação com cautela: se a equipe estiver aumentando gradativamente sua capacidade de trabalho, também chamada de velocidade, então não há problema em assumir um compromisso maior do que os já entregues; porém, em uma situação na qual a velocidade da equipe já está estabilizada, o que se sugere é que o *product owner* reveja as histórias e tarefas do *sprint* e tente quebrá-las mais ainda. Feito isso, avalia-se quais tarefas tem o menor impacto sobre o objetivo do *sprint* e opta-se por não inclui-las na iteração.

Acompanhamento

Depois dessas fases preliminares, o *sprint* já está planejado, as tarefas estão separadas, todos tem em mente o que precisa ser feito e a forma como fazê-lo. Com isso, o ambiente está completamente favorável e é hora da equipe começar realmente a produzir. Entretanto, nesse período entre o início e fim do *sprint*, não se abandona o ***Scrum*** deixando cada integrante do time fazer suas tarefas isoladamente e só voltar a falar na metodologia após três semanas. Ao longo da iteração, ainda há elementos que ajudam o trabalho a caminhar bem, a acompanhar o andamento do *sprint* e a medir de forma menos subjetiva a velocidade da equipe. Abaixo, veremos alguns itens que auxiliam a fazer esse acompanhamento.

- **Cartões de tarefas**
  Já foi dito que é preciso criar histórias e tarefas, contudo ainda não foi especificado onde se deve escrevê-las. A resposta a essa questão são os cartões, mais precisamente, fichas pautadas de aproximadamente 7x12 cm. A vantagem desse tipo de cartão é, além do seu tamanho que é suficiente para uma boa descrição e ao mesmo tempo pequeno demais para um texto longo, sua resistência. Essa característica é particularmente interessante para as histórias, que são constantemente manuseadas, ordenadas nos *backlogs* e não devem rasgar facilmente como acontece com as folhas de papel sulfite. Além disso, quando se usa esses cartões, é possível escrever as condições de satisfação da história no verso, o que é bastante cômodo.

  Para as tarefas, que tem um tempo de vida relativamente mais curto, a necessidade de um papel cartão é menor. Assim a opção mais vantajosa é o uso de papéis de

anotações com o verso autoadesivo. Essa característica facilita bastante a colocação dessas anotações no quadro de tarefas.

- **Gráfico *burndown* e *burnup***
  Para acompanhar a velocidade da equipe, usa-se os pontos das estimativas. Ao final de cada dia de trabalho, basta somar os pontos de estimativa de todas as tarefas já cumpridas e com isso tem-se uma métrica objetiva do volume de trabalho completado. É possível colocar esses resultados diários em um gráfico e isso ajuda o time a ter noção de como está sendo seu progresso ao longo do *sprint*.

  Existem basicamente duas principais opções quanto ao formato desse gráfico: o *burndown*, exemplificado na figura 3, e o *burnup*, mostrado na figura 4. Em ambos, o eixo das abscissas indica o os dias nos quais houve trabalho e o eixo das ordenadas está graduado com os pontos de estimativa. A diferença reside em como se interpreta o uso dos pontos: no *burndown*, se considera que o time inicia a iteração tendo por fazer todos os pontos planejados e, à medida que as tarefas vão sendo cumpridas, os pontos pendentes vão diminuindo, o que consequentemente faz o gráfico decrescer; já no *burnup*, a equipe marca o acumulado de pontos já cumpridos por dia e por isso o gráfico é crescente.

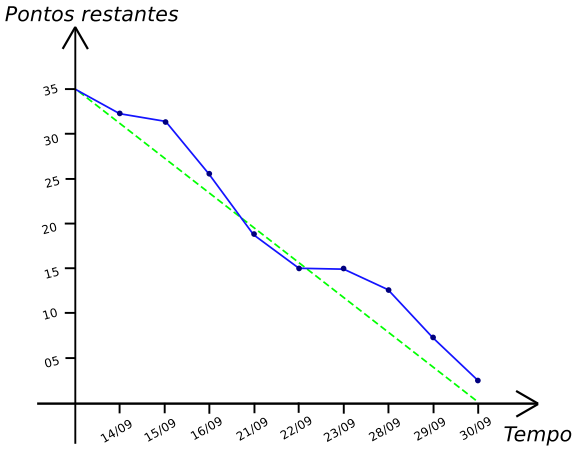

Figura 3: Gráfico *Burndown*

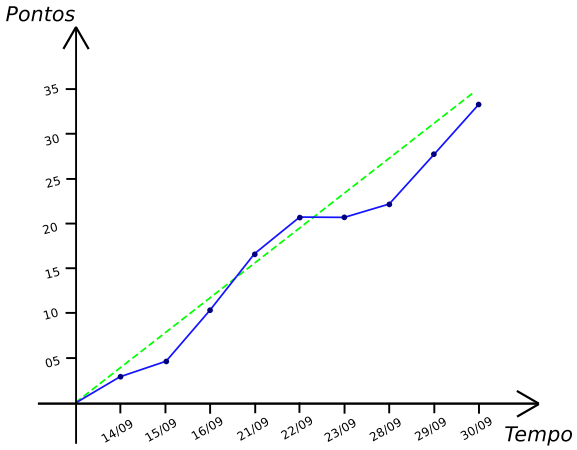

Figura 4: Gráfico *Burnup*

Independentemente da representação escolhida, é possível traçar uma linha (verde pontilhada, nas figuras) que indica a velocidade ideal com a qual o time conseguiria atingir perfeitamente a meta no último dia do *sprint*. Porém, vale ressaltar que o objetivo dos gráficos não é fazer com que a equipe tente acompanhar essa linha ideal, mas sim fornecer uma visualização rápida de como está sendo o progresso e, principalmente, mostrar qual é a tendência do ritmo de trabalho. Por exemplo, ambos os gráficos mostram claramente a tendência desfavorável no final do segundo dia do *sprint*. Isto é, a projeção da curva sugere que o time não conseguiria alcançar a meta de pontos caso o ritmo de trabalho não fosse acelerado.

Ter esse tipo de informação já no fim do segundo dia de iteração é de grande valia, pois com isso é possível tomar rapidamente medidas que podem "salvar" o *sprint*

antes que ele falhe. Inclusive, a análise dos gráficos pode auxiliar no planejamento de quantos pontos a equipe pode prometer cumprir para os *sprints* seguintes.

Portanto, tanto o *burndown* quanto o *burnup* servem para ajudar o time a se auto regular durante o *sprint*. Assim, a escolha entre qual dos dois tipos de gráfico será usado é apenas uma questão de preferência de cada grupo. O importante é usar um dos dois, pois são ferramentas de acompanhamento muito poderosas.

- **Quadro de tarefas**
  Os cartões de histórias e de tarefas precisam estar visíveis aos desenvolvedores e, portanto, devem ser fixados em um quadro colocado em algum local de fácil acesso dentro do ambiente de trabalho. O posicionamento do quadro é importante, pois muitas vezes as reuniões diárias são feitas em frente a ele. Além disso, o mais indicado para essa finalidade é se utilizar um quadro branco, que possui uma superfície lisa muito favorável ao uso das notas autoadesivas e que também permite se escrever no próprio quadro usando canetas especiais para isso.

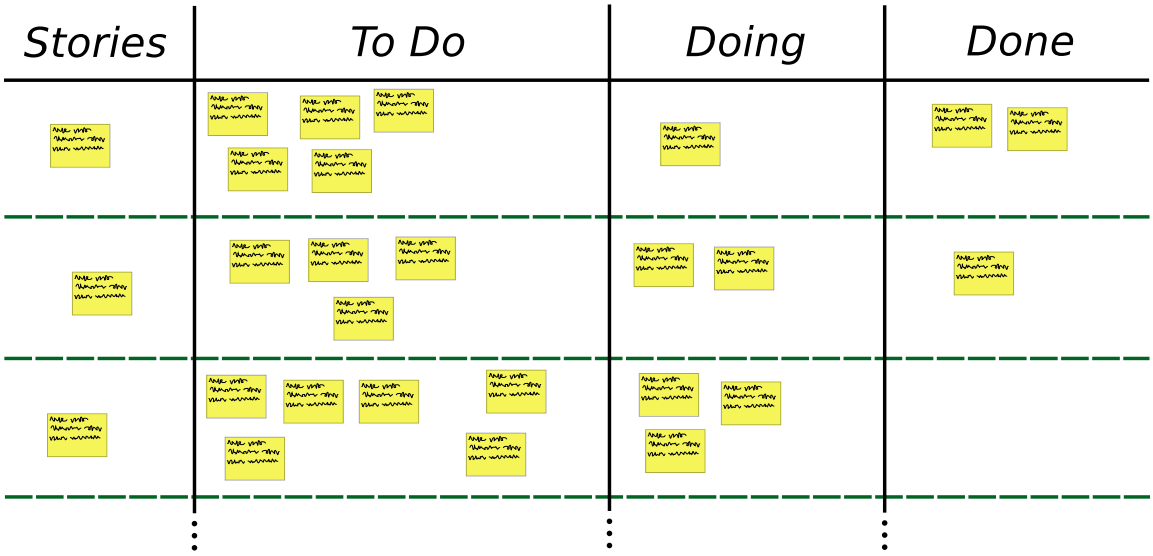

Figura 5: Quadro de tarefas

Nesse quadro deve haver ao menos quatro colunas, nas quais serão distribuídos os cartões. A figura 5 mostra com é o normalmente montado o quadro de tarefas. Na primeira coluna, são colocados os cartões das histórias do *sprint*. Na segunda coluna, são fixadas todas as tarefas que ainda não começaram a ser trabalhadas. Normalmente, por questão de organização, se coloca as tarefas alinhadas horizontalmente com as histórias às quais pertencem. Já na terceira coluna, ficam as tarefas que estão sendo feitas por alguém. É comum que nessa coluna deixe-se anotado ao lado de cada tarefa os nomes das pessoas que estão trabalhando nelas. Por fim, na quarta coluna são colocadas as tarefas que já foram concluídas.

No início do *sprint*, as histórias do *sprint backlog* são colocadas nas primeiras colunas e todas as tarefas ficam na segunda coluna. À medida que as pessoas começam

a implementar as tarefas e as terminam, os cartões vão passando para as colunas seguintes. Dessa forma, também é possível ter uma ideia de como está o andamento da iteração e de cada história, ganhando assim uma visualização complementar aos gráficos de velocidade.

Dependendo de como cada grupo organiza seu processo de trabalho, eventualmente há a necessidade de se colocar outras colunas além dessas iniciais, o que é totalmente válido. Por exemplo, é possível que algumas tarefas precisem passar por algum tipo de validação ou aprovação antes de serem consideradas prontas. Nessa situação, o quadro poderia ter uma coluna intitulada *"To be approved"* à esquerda de *"Done"*.

- **Daily meetings**
  A primeira atividade da equipe no início de cada dia de trabalho deve ser uma curta reunião de acompanhamento. Para manter a conversa dinâmica e rápida, não se deve gastar mais do que quinze minutos nessa atividade e todos devem permanecer em pé, o que é um fator que por si só já contribui para que não haja prolongamentos desnecessários.

  Os objetivos dessa reunião são relembrar o *sprint goal*, verificar se o cumprimento das tarefas está direcionado a atingir essa meta, fazer com que cada integrante saiba como está o progresso dos demais no grupo e identificar possíveis impedimentos. Para conduzir a conversa nesse sentido, todos devem responder à três perguntas:

  1. O que eu fiz desde a última reunião?
  2. O que me comprometo a fazer até a próxima?
  3. Quais são as questões ou problemas que estão me atrasando?

  Ao responder essas perguntas, cada um reflete sobre seu desempenho no grupo e cria-se um senso de comprometimento maior, pois ninguém ficará confortável em dizer que não fez nada no dia anterior ou que pretende apenas ficar matando tempo no dia atual. Além disso, compartilhar as dificuldades enfrentadas é uma forma de se encontrar soluções mais rapidamente e também evita-se que várias pessoas percam tempo com o mesmo problema em momentos diferentes.

  Por fim, caso o time ache interessante para organizar melhor os temas durante a conversa, é possível agrupar as pessoas de modo que todos que estiverem trabalhando numa mesma história falem em sequência. Isso se justifica pois normalmente tarefas da mesma história tem questões em comum e, dessa forma, evita-se que a conversa fique indo e voltando para um tema que poderia ser discutido todo de uma vez.

Revisão

Quando um *sprint* termina, é hora de mostrar ao cliente o resultado do trabalho. Isso acontece em uma reunião entre o cliente e o time de desenvolvimento, a qual também deve ter uma duração predefinida e não deve exceder uma ou duas horas. Basicamente, o roteiro dessa reunião consiste em relembrar para o cliente qual era o objetivo do *sprint*, relatar se não foi possível cumprir alguma história e fazer uma demonstração do protótipo

da iteração, que espera-se estar mais próximo do produto final. Em seguida, passa-se o comando para o cliente, que deve jogar a versão da demonstração e dar seus comentários.

Feito isso, o cliente julga se o objetivo do *sprint* foi atingido. Caso a resposta seja negativa ou se o cliente não tenha gostado do resultado apresentado, as histórias que falharam deverão ser descartadas ou voltar para o *product backlog* com indicações de como devem ser modificadas, sendo que essa decisão fica a cargo do próprio cliente. O mesmo deve ser feito com as histórias que eventualmente não tenham sido completadas.

Essa reunião traz uma série de benefícios. O mais imediato é justamente ter um contato regular com o cliente, o que é fundamental para garantir que o projeto esteja rumando em direção a atender às expectativas. Além disso, essa interação entre cliente e time de desenvolvimento é uma ótima oportunidade para ambas as partes tirarem dúvidas e reforçarem o compartilhamento de uma mesma visão sobre o projeto. Isso garante que a equipe está entendendo o que é pedido e que o cliente está esperando como resultado o que o time de fato está desenvolvendo. Ademais, com esse tipo de acompanhamento, o cliente passa a ter um papel mais ativo no desenvolvimento, o que lhe dá uma sensação maior de também ter responsabilidade na qualidade do produto e inclusive evita alegações como "Mas não era bem assim que eu tinha pensado que ficaria" ou "O jogo não ficou bom porque vocês não fizeram o que eu pedi" ao fim do projeto.

Outro fator positivo dessa apresentação é que o jogo é mostrado como um produto único e não um conjunto de independentes trabalhos das diferentes disciplinas de cada desenvolvedor. Por isso, essa reunião reforça a ideia de que o grupo está trabalhando interdisciplinarmente na construção de uma só obra e que é a qualidade do resultado depende tanto do bom desempenho de cada área quanto da boa integração e colaboração entre elas.

Retrospectiva

Depois da reunião de revisão, não necessariamente no mesmo dia, mas antes do início do *sprint* seguinte, é feita uma reunião de retrospectiva com os membros da equipe de desenvolvimento. Novamente, antes de tudo é preciso estipular a duração do encontro, que normalmente leva em torno de três horas para *sprints* de quatro semanas. Certamente, se os *sprints* forem menores ou assim for a vontade do time, essa reunião pode ser mais curta. O objetivo dessa reunião é refletir sobre como foi o *sprint* e quais melhoras tanto nos processos quanto na remoção de impedimentos podem ser feitos. Para guiar a reunião a levantar esses fatos, existem algumas perguntas chave que podem ser feitas:

1. O que devemos parar de fazer?
2. O que está funcionando e devemos continuar fazendo?
3. O que deveríamos tentar fazer?

Enquanto essas questões são abordadas, os itens levantados devem ser de alguma forma registrados, pois serão usados adiante. Uma maneira efetiva de se conseguir isso e que ao mesmo tempo favorece a participação de todos, não apenas daqueles que gostam mais de falar, é escrever essas três perguntas em três colunas largas em um quadro branco, pedir para os integrantes do time escreverem em notas autoadesivas tudo o que pensam sobre cada pergunta e colá-las na coluna correspondente. Também é preciso estabelecer um

tempo limite para essa subatividade e, ao seu término, o *scrum master* deve ler em voz alta todas as anotações e agrupar aquelas que tratam do mesmo assunto.

Todos os tópicos no quadro são candidatos a serem discutidos, porém na maioria das vezes há mais assuntos do que o compatível com o tempo de reunião. Por isso, é preciso estabelecer um consenso sobre quais temas serão abordados. Preferencialmente, ao menos um tópico referente a cada pergunta deve ser escolhido. Uma vez que se tenha definido os assuntos mais relevantes, o grupo deve conversar sobre eles.

Com isso, espera-se que a equipe consiga identificar seus próprios erros e acertos e, se baseando nesse conhecimento, seja capaz de criar um plano de melhoras para o próximo *sprint*. Esse plano consiste basicamente em medidas para resolver ou evitar os problemas levantados e podem incluir também a adaptação de alguma das práticas em uso ou a adoção de algo novo ao processo. Da forma que foi dada essa explicação, talvez o leitor possa imaginar que essas ações tratam exclusivamente do comportamento coletivo, entretanto eventualmente algumas medidas precisam ser tomadas sobre os indivíduos. Por exemplo, se alguém tem chegado atrasado várias vezes ou tem trabalhado em funcionalidades que não eram tarefas do *sprint*, então pode-se colocar como parte do plano de melhorias o comprometimento dessa pessoa em mudar sua atitude.

Uma vez que os pontos escolhidos tenham sido discutidos, as ações a serem tomadas devem ficar anotadas e, se possível, visíveis próximo ao quadro de tarefas e ao gráfico. Isso serve para a equipe como lembrete de quais mudanças foram propostas e contribui para que as pessoas de fato mudem conforme o sugerido. Além disso, também é válido rever essas anotações na retrospectiva seguinte para que se avalie quais medidas foram realmente postas em prática, o que deu certo, o que não foi feito e a razão para isso.

Em alguns casos, não se consegue colocar em prática tudo o que foi pensado ou exatamente da forma como se planejou. Por isso, caso a equipe ainda deseje insistir em alguma melhoria sugestão em uma retrospectiva anterior, pode-se manter a proposta por quantos *sprints* forem precisos. Entretanto, ser uma mesma ideia permaneceu no plano de melhoras por várias iterações mas inda não tiver sido implementada, então é preciso reconsiderar sua viabilidade ou ao menos estipular uma maneira alternativa de fazê-la sair do papel.

Em muitas vezes, um tópico de discussão que acaba surgindo naturalmente na retrospectiva é o nível de acerto das estimativas. Mas vale ressaltar que em equipes que estão começando a adotar o ***Scrum*** ou que tenham sido formadas recentemente é importante cobrir esse assunto, mesmo que ele não seja levantado pela equipe. É comum que as estimativas comecem bem erradas nos primeiros *sprints*, assim, conversar sobre isso ajuda o grupo a identificar quais foram as origens dos principais erros e consegue-se calibrar mais rapidamente essa habilidade coletiva.

Ao término da reunião, é realmente importante que todos da equipe joguem a versão do jogo que foi apresentada. Apesar de parecer uma preocupação boba colocar essa atividade como obrigatória, essa ação é muito válida já que não são raras as vezes em que uma parcela do time acaba não jogando o jogo durante o desenvolvimento. O ato de brincar com o protótipo ajuda o próprio time a encontrar elementos que não estão muito bons e áreas que podem ser melhor exploradas. Inclusive, os conceitos de que todos estão trabalhando para um mesmo objetivo e que a contribuição individual está construindo algo maior são reforçados por essa experiência. Além disso, depois de um *sprint* trabalhando e das reuniões finais, ter esse momento de descontração ajuda a renovar o ânimo das pessoas

para continuar com o projeto.

Estimativas

Como as habilidades de um ser humano comum em prever o futuro são consideravelmente limitadas, a etapa de estimar o esforço a ser empenhado em cada tarefa é muito subjetiva. Não há uma regra que defina como fazer estimativas corretamente e essa capacidade varia bastante de time para time e de pessoa para pessoa. Conforme cada indivíduo vai ganhando mais experiência e a equipe vai amadurecendo, a tendência é que as estimativas comecem a convergir para valores cada vez mais corretos. Porém, existem algumas técnicas que podem ajudar essas previsões serem próximas do que deveriam, mesmo quando ainda não se tem prática com essa atividade.

Pontos

O mecanismo mais básico das estimativas consiste na unidade de medida usada para indicar o quanto uma história ou tarefa é complexa. Essa medida é feita em pontos. Assim como o conceito de dificuldade, a ideia de pontos é abstrata, mas, quando incorporada, se torna muito simples de entender e usar.

Como os pontos representam dificuldade, ambos tem características muito similares. Por exemplo, não é trivial dizer se algo é difícil ou não, mas é relativamente simples afirmar que uma coisa é mais difícil que outra. Isto é, dar um valor para dificuldade é complicado, porém usá-la comparativamente é simples. O mesmo se aplica aos pontos. Não é fácil dizer quantos pontos uma tarefa deve ter, mas quando já se tem definida a pontuação de determinada tarefa, é fácil usá-la como referência para dizer que outra atividade é três vezes mais complexa ou duas vezes mais fácil. Dada essa abordagem, o primeiro passo para conseguir usar os pontos é portanto definir uma, ou algumas, tarefas como as unidades, isto é, classificadas como tendo um ponto de dificuldade. Com essa definição em mente, na hora de estimar outras tarefas, basta compará-las com a unidade e verificar aproximadamente quantas vezes mais ou menos trabalho será preciso para cumpri-las.

Para algumas equipes, pode ser um pouco difícil se acostumar com a ideia de pontos devido a falta de parâmetros que a aproxime da realidade, ou então as pessoas do grupo podem já estar acostumadas a usar horas de trabalho como medida e não se sentem confortáveis com a abstração dos pontos. Nesses casos, é possível encarar os pontos como "horas ideais", isto é, um ponto representaria o trabalho realizado em uma hora de total dedicação ao cumprimento da tarefa, sem pausas para ir ao banheiro, sem atender ao telefone, sem parar para conversar com os colegas e sem quaisquer outras distrações.

Poker de planejamento

Tendo a ideia de pontos como medida de dificuldade clara para o grupo, falta definir uma maneira para que se chegue em um acordo sobre a estimativa de cada tarefa. A primeira ideia que normalmente se tem é deixar cada um dar sua opinião e tentar se chegar em um consenso ou fazer uma votação. Entretanto essa não é uma boa abordagem, pois um bom argumentador na equipe tende a influenciar muito a opinião dos demais, antes até deles

chegarem a uma conclusão de quanto eles mesmos acham que vale a tarefa. Isso tornaria a estimativa muito tendenciosa e não refletiria a opinião real da maioria.

Para evitar esse tipo de problema, é possível fazer as rodadas de estimativas usando um jogo com cartas, que na verdade não se aproxima nem um pouco do poker. O jogo consiste no seguinte: cada pessoa fica com um conjunto de cartas numeradas; o *scrum master* pega a ficha de uma tarefa e a lê em voz alta; se alguém tiver alguma dúvida com relação à tarefa, pode-se fazer perguntas até que todos tenham compreendido exatamente o que precisa ser feito; cada pessoa escolhe uma carta, sem mostrá-la, com o número que acha ser a quantidade de pontos que representa mais adequadamente a dificuldade da tarefa; quando todos já tiverem tomado sua decisão, todas as cartas escolhidas são mostradas simultaneamente; as pessoas que mostraram o maior e menor valor explicam porque acharam que a tarefa é mais difícil ou mais fácil do que o valor médio sugerido pelos demais; terminada a argumentação, todos recolhem suas cartas e fazem a escolha novamente, levando em conta o que foi discutido; repetem-se esses passos até que o grupo chegue em um acordo.

Sequência de Fibonacci

Depois de algumas rodadas jogando o poker de planejamento, é provável que se perceba dificuldade em resolver impasses para diferenças de estimativa próximas. Uma forma de minimizar esse problema é usar cartas apenas com os números pertencentes à sequência de Fibonacci: 1, 2, 3, 5, 8, 13 ...

Na sequência de Fibonacci, um valor acaba tendo uma diferença considerável para o seguinte, o que facilita a escolha da estimativa, principalmente para tarefas mais difíceis. Para valores baixos, não há muita necessidade de discussão, pois tanto tarefas de um quanto de dois pontos são relativamente fáceis. Então esse pequeno erro na estimativa não tem grande impacto no planejamento do *sprint*. Já para valores altos, as diferenças rapidamente começam a se tornar gritantes. Com isso, decidir se uma tarefa tem dificuldade oito ou treze é consideravelmente mais fácil do que tomar a decisão quando as opções são todos os valores inteiros contidos nesse intervalo.

Inclusive, quando as tarefas são mais difíceis, há uma tendência de que o grau de incerteza que as envolve seja maior. Assim, se o desejo inicial era que determinada tarefa fosse estimada em dez pontos, lhe atribuir treze pontos não é uma estimativa ruim, já que é grande a chance de que a tarefa seja mais difícil do que a princípio parecia.

Isso mostra como o uso da sequência de Fibonacci favorece que sejam feitos arredondamentos para cima nas estimativas. Essa prática é muito saudável, pois há uma tendência coletiva em subestimar a dificuldade das tarefas. Dessa forma, ganha-se uma margem de segurança que evita, mesmo em times com a habilidade de estimar bem calibrada, uma soma total muito justa e sem folgas para imprevistos.

USPGameDev

Antes de prosseguir para as seções relacionadas a análise do uso e não uso de ***Scrum***, é importante conhecer um pouco sobre o contexto onde essas observações foram feitas. Por esse motivo, os próximos parágrafos darão uma breve descrição do que é o *USPGameDev*.

História

Em 2009, o orientador pedagógico da Escola Politécnica da USP (Poli), Giuliano Salcas Olguin, iniciou um trabalho de acompanhamento e avaliação do curso de Bacharelado em Ciência da Computação (BCC) do Instituto de Matemática e Estatística da USP (IME). Um dos objetivos dessa atividade era fomentar a iniciativa de alunos em trabalhos de extensão. Nesse cenário, em novembro daquele ano, surgiu a proposta de reunir alunos do BCC e da Engenharia de Computação da Poli interessados em estudar e desenvolver jogos eletrônicos e constituir um grupo de estudos dessa área. Assim, foi criado o *USPGameDev*, cuja composição inicial era formada por vinte e uma pessoas, sendo apenas um aluno de engenharia e os demais alunos do BCC.

Durante o ano de 2010, o grupo se dedicou exclusivamente a duas atividades: Construir uma biblioteca para o desenvolvimento de jogos em duas dimensões para computador, posteriormente batizada de *UGDK*, e implementar um de jogo, intitulado *Horus Eye*, utilizando-se essa biblioteca. A *UGDK* levou por volta de sete meses para ficar pronta para o uso e, após três meses a partir daí, o *Horus Eye* teve sua primeira *release*.

Entre dezembro de 2010 e o início de abril de 2011, o grupo passou por um período de seleção de novos membros e uma fase de baixa atividade. Apesar desse intervalo de pouca produção, a chegada de novos integrantes trouxe ao *USPGameDev* uma maior diversificação de talentos. Com isso, o time passou a cobrir competências outrora deficitárias, como roteiro.

Local de trabalho

No início, o grupo não tinha um local fixo para se encontrar e fazer as reuniões de trabalho. Assim, sendo a maioria dos integrantes alunos do BCC, usava-se de improviso salas de aula do IME que estivessem desocupadas. Apesar do espaço físico satisfatório, as salas de aula só eram lugares apropriados para reuniões voltadas a definição de propostas, debates de ideias e planejamento. Para trabalhar no desenvolvimento de fato, as principais desvantagens das salas eram: falta de computadores e tomadas suficientes para ligar os notebooks pessoais dos participantes; cadeiras universitárias não acomodavam adequadamente os notebooks; falta de acesso à internet.

Depois de cerca de um mês, foi autorizado o uso da sala de reuniões da Orientação Pedagógica (OP) da Poli. Nessa sala, as necessidades da equipe eram muito melhor atendidas. Havia mesas grandes o suficiente, tomadas e era possível acessar a internet através de uma rede sem fio. Entretanto, eventualmente a sala não estava disponível, pois sua finalidade não era atender ao grupo, mas sim à OP.

No final de outubro de 2010, depois de grande empenho do Giuliano em resolver essa questão, o *USPGameDev* recebeu uma sala fixa e exclusiva no prédio do Biênio da Poli. Essa nova sala acomodava muito bem o grupo, porém o sinal da rede sem fio era intermitente e por várias vezes completamente nulo. No início de 2011, com ajuda do Prof. Carlos Eduardo Ferreira do IME, novos equipamentos de rede foram comprados e puderam ser instalados na sala de modo a resolver esse problema.

Não se enfrentou mais questões referentes ao local de trabalho até junho de 2011, quando começou uma reforma na sala a fim de transforma-la em um laboratório. Até o fim das obras, que ocorreu no começo de agosto, as reuniões voltaram à situação nômade

da época de criação do grupo. Com tudo, ao fim da reforma, a sala do *USPGameDev* passou a conter uma infraestrutura que atendia plenamente às suas necessidades: acesso à internet, tomadas, mesas adequadas, computadores e quadros brancos nas paredes.

Sem *Scrum*

Como já comentado na introdução, durante o ano de 2010 o *USPGameDev* viveu sua primeira fase, na qual o ***Scrum*** ainda não tinha sido adotado. Na verdade, nenhuma metodologia formal de gerenciamento de projetos estava em prática. Nessa seção, serão dados alguns detalhes de como foi a experiência de desenvolver o projeto nesse cenário, quais foram os pontos positivos e negativos.

Características gerais

Organização

Apesar de não haver nenhuma metodologia em vigor nesse período, vários integrantes do grupo já tinham algum conhecimento sobre metodologias ágeis e isso influenciou de certa forma o modo de trabalho da equipe. Com isso, o desenvolvimento do projeto não ficou completamente desorganizado.

Em virtude dessa familiaridade com métodos ágeis, desde o início foi sugerida a separação de tarefas em itens relativamente pequenos e que isso ficasse listado em algum lugar visível para que todos pudessem ver o que precisava ser feito. Por esse motivo, sempre que era identificada a necessidade de se implementar alguma funcionalidade, escrevia-se a tarefa na lousa da sala, formando assim uma listagem do que esperava-se que ficasse pronto. Quando uma tarefa estava sendo trabalhada por alguém, ao lado do item na lousa era escrito os nomes dos responsáveis. Além disso, quando uma tarefa era terminada, a apagava-se da lousa. Esse era um processo contínuo, onde tarefas iam sendo colocadas e tiradas da lista constantemente, sem haver a ideia de conclusão de etapas ou iterações.

Reuniões

O descrito na seção anterior era o retrato das reuniões de trabalho, quando o grupo se dedicava a implementação. Porém em vários momentos era preciso tomar decisões sobre diferentes aspectos tanto da *UGDK* quanto do *Horus Eye*. Por exemplo, foi necessário definir a arquitetura base da biblioteca e também do jogo, precisava-se escolher qual seria o estilo do jogo, quais os tipos de movimentos e ações o personagem poderia realizar, além de muitas outras caraterísticas importantes para se moldar o resultado. Tendo em vista que o jogo não era um produto encomendado por um cliente, nem um projeto de alguma disciplina, não havia qualquer especificação. Ou seja, o próprio grupo precisava definir tudo e, para tal, era necessário que todos dessem sua opinião e que as escolhas fossem sempre coletivas. Isso era feito em reuniões de discussão.

Essas reuniões de discussão eram focadas exclusivamente em tomar decisões. Normalmente, todos os presentes se sentavam formando uma grande roda e, um a um, os assuntos

da pauta eram discutidos. Mas antes de chegar em qualquer tipo de escolha, era preciso levantar as opções entre as quais escolher. Por isso, fazia-se uso constante de *brainstorming*, onde todos dão várias ideias relacionadas a resolução da questão, todas são anotadas e, quando não há mais novas ideias, verifica-se as propostas e então discute-se os problemas e viabilidade de cada uma.

O grupo fazia essas reuniões para trabalhar no projeto duas vezes por semana, em sessões de três horas cada.

Cronograma

Apesar do caráter contínuo da realização das tarefas e a ausência de pontos de verificação de acompanhamento do projeto, os chamados *milestones*, ainda sim questões burocráticas exigiram a confecção de um cronograma para o lançamento do jogo. Dada essa obrigação e a falta de parâmetros para definir o que seria entregue e em qual momento, o cronograma foi feito da seguinte forma: em uma reunião de discussão, levantou-se as principais características que se desejava na versão de lançamento do jogo e a data desse evento; dividiu-se a quantidade de funcionalidades pelo número de meses até o fim do prazo; com esse resultado, um número igual de funcionalidades foi distribuído em cada mês e assim definiu-se o cronograma.

Pessoas

Na seção de história, foi mencionado que praticamente todos os integrantes do grupo estavam cursando Ciência da Computação. Entretanto, apesar da unanimidade do curso, o time não era completamente uniforme. Havia alunos do segundo ao quarto ano da graduação, além de dois que cursavam o mestrado. Apesar desse perfil se manter relativamente constante ao longo do ano, o quadro de integrantes sofreu muitas variações, sendo que algumas pessoas foram saindo do grupo e outras foram entrando.

Fórum

Para estabelecer um canal de comunicação eficiente entre essas pessoas de diferentes turmas, criou-se um fórum na internet. Essa ferramenta se tornou o meio de comunicação oficial do grupo e muitas questões foram discutidas ou publicadas lá. O fórum também serviu muito bem como ferramenta para a divulgação de avisos, principalmente relacionados a questões de agendamento de reuniões e divulgação dos locais onde seriam feitas.

Tarefas

Mesmo com a escassez de planejamento a longo prazo e a superficialidade do cronograma, alguns objetivos já estavam claros e a maioria das tarefas girava em torno deles. Porém, à medida que ideias iam surgindo, tarefas que não necessariamente eram essenciais ao cumprimento desses objetivos também acabavam sendo colocadas na lista. Em geral, a implementação dessas tarefas era uma atividade interessante ou desafiadora e, uma vez que o grupo era formado basicamente por programadores, por isso muitas funcionalidades acabavam entrando na lista, mesmo sem ter relação com o objetivo.

Por outro lado, algumas tarefas que eram necessáriasi, porém consideradas chatas, acabavam sendo negligenciadas e passavam longos períodos na listagem sem serem cumpridas.

Mesmo assim, quando o foco se tornou a finalização do *Horus Eye* para o lançamento, a maioria das tarefas relacionadas à construção da *UGDK* e da base do jogo já tinham sido terminadas. Com isso, passou-se a implementar funcionalidades que apenas usavam toda essa infraestrutura construída. Nesse momento, a velocidade do projeto aumentou muito. Isto é, em pouco mais de um mês passou-se de uma versão onde só havia uma fase, um tipo de inimigo e o personagem principal só se movimentava e disparava tiros simples para uma versão com várias fases, inimigos de comportamentos e habilidades diferentes, além de mais dois tipos de magias para o personagem principal.

O que deu certo

Segundo os próprios desenvolvedores, uma das melhores práticas durante essa fase foi usar a lousa para listar as tarefas a serem feitas. Ao chegar no ambiente de trabalho, era possível ver o que ainda precisava ser implementado e, como cada um podia escolher qual problema iria atacar, não havia o desconforto de receber a atribuição de uma tarefa que não agradava.

Outra medida essencial para a organização do grupo e do projeto foi o uso do fórum. Esse foi um dos primeiros elementos de infraestrutura conseguidos e, apesar disso, durante alguns períodos de problemas com o servidor, foi possível notar que a comunicação do grupo foi fortemente prejudicada.

Além disso, a heterogeneidade de experiência entre os membros do *USPGameDev* não foi um fator de impacto negativo. Na verdade, foi o oposto. O convívio com alunos mais próximos de concluir o curso trouxe para os novatos uma rica troca de conhecimentos. Além de serem apresentados a várias ferramentas de desenvolvimento, os novatos também tiveram contato com conceitos relativamente avançados sobre arquitetura, técnicas de padrões de projeto, orientação a objetos e muitos outros. Essa troca de experiências, juntamente com a boa construção da biblioteca e da base do jogo, permitiu o ganho de velocidade de produção citado na seção Tarefas anteriormente.

Problemas enfrentados

Os problemas mais imediatos que surgiram ao tentar reunir um grupo de aproximadamente vinte alunos em reuniões periódicas dentro do espaço da universidade foram relacionados a infraestrutura. O grupo passou por longos períodos sem um local de trabalho fixo e isso realmente afetou o andamento do projeto. Mas esse era um fator externo e não havia como a equipe se prevenir ou gerenciar essa questão.

Em relação às questões internas, a primeira coisa que se notava era que as reuniões de discussão eram sempre demoradas e era difícil chegar em um consenso. No geral, evitava-se fazer votações, pois eventualmente uma parcela considerável do grupo ficaria descontente com o resultado, mesmo sendo a minoria. Por esse motivo, tentava-se ao máximo encontrar soluções e tomar escolhas que fossem bem aceitas quase que unanimemente. Como é de se esperar, essa era uma tarefa especialmente difícil dada a quantidade de pessoas no grupo e a impressão que todos tinham era a de que essas reuniões duravam “para

sempre".

Como as decisões tomadas eram sempre bem gerais e não se restringiam a uma categoria de funcionalidades, as tarefas criadas após as reuniões de discussão eram das mais variadas. Assim, a lista na lousa costumava ficar longa e as tarefas muitas vezes tinham pouca relação umas com as outras. Somando a isso o fato de que novas tarefas eram adicionadas à lista constantemente, havia um problema de que as tarefas consideradas menos interessantes acabavam permanecendo em aberto por muito mais tempo que as outras.

Apesar do número constantemente alto de tarefas na lousa, havia pouca exigência de comprometimento individual. Com isso, era corriqueiro ver pessoas assíduas às reuniões mas que produziam pouquíssimo.

Outro fator problemático foi a falta de diversidade nas habilidades da equipe. Sem game designers, artistas e roteiristas, praticamente todas as ideias e tarefas eram relacionadas a questões de implementação. Dessa forma, ao se analisar o resultado final do *Horus Eye*, é possível listar uma grande quantidade de funcionalidades, inclusive algumas supérfluas, mas nota-se uma deficiência em relação a roteiro, arte e experiência de jogo. Por exemplo, foi feito um menu de ajuda com as instruções básicas de como jogar, mas uma opção mais adequada teria sido criar uma fase introdutória que fosse um tutorial interativo sobre essas instruções. Uma frase dita por um dos próprios desenvolvedores exemplifica bem qual foi a impressão que se teve sobre o resultado do projeto: "O código ficou mais bonito que o jogo".

Além disso, como parte da proposta da criação da *UGDK* era que a biblioteca pudesse ser usada na criação de vários jogos, inclusive por pessoas de fora do *USPGameDev*, e não exclusivamente no *Horus Eye*, um pré-requisito para tal era a existência de uma documentação. Isso porque uma biblioteca sem documentação é praticamente inutilizável por pessoas que não participaram de sua criação e não estão familiarizadas com seu funcionamento. Entretanto, como a tarefa de documentar era considerada uma das mais chatas de todo o projeto, ela não foi cumprida.

Por fim, dada a forma como o cronograma foi elaborado, é desnecessário dizer que ele foi praticamente ignorado. De fato, o único ponto que impediu que o documento fosse completamente esquecido era a data do lançamento, pois já tinha sido marcada e, independentemente do estado do jogo, esse evento iria acontecer no dia agendado.

Com *Scrum*

Já em 2011, mais precisamente, durante o segundo semestre, o *USPGameDev* começou a seguir os princípios do ***Scrum***. A seguir será descrito como foi essa experiência, as práticas que foram possíveis de se adotar, quais delas se mostraram inviáveis e a razão para isso.

Iniciação no *Scrum*

Para colocar o ***Scrum*** em prática dentro do *USPGameDev*, era preciso que os integrantes do grupo tomassem conhecimento dos principais conceitos da metodologia. Entretanto, todos concordaram que dificilmente as pessoas leriam qualquer tipo de referência e que uma palestra seria melhor aceita. Assim, essa apresentação deu início a implantação da

metodologia no grupo, com o Prof. Flávio Soares e Giuliano Olguin nos papéis de *product owners* e o autor deste texto como *scrum master*.

Definiu-se que os *sprints* teriam duração de três semanas, um dos quadros brancos da sala foi reservado como quadro de tarefas e, próximo a ele, foi montada a estrutura do gráfico *burnup*. Com o ambiente organizado, três reuniões semanais de quatro horas cada e uma nova proposta de jogo, o *USPGameDev* iniciou sua experiência de pouco mais que quatro meses seguindo ***Scrum***.

Contudo, de imediato já foi preciso fazer uma certa adaptação com relação ao ciclo dos *sprints*. Em virtude da limitada disponibilidade tanto dos integrantes da equipe quanto dos *product owners*, não era possível agendar as reuniões de priorização em um horário compatível a todos. Com isso, o próprio grupo definia o objetivo do *sprint*, escrevia as histórias e o *scrum master* as apresentava aos *product owners* em uma reunião rápida. Nessa reunião, os *product owners* davam seus comentários sobre as histórias e as ordenavam por prioridade. Além disso, também era apresentado o resultado do *sprint* anterior. Dessa forma, esses encontros se tornaram a união compacta das reuniões de priorização e de revisão. Com as histórias ordenadas, o time fazia o poker do planejamento para estimá-las e as colocava no quadro de tarefas.

O que deu certo

A primeira mudança que agradou a equipe de desenvolvimento foi adotar um modelo iterativo. Ter definido que depois de um certo período o *sprint* terminaria e que, nesse momento, obrigatoriamente deveria ser mostrado um progresso no projeto já acrescentou um fator de urgência inexistente anteriormente. Além disso, deixar explícito qual era o objetivo do *sprint* também incluiu um fator de motivação nas iterações, pois deixava mais evidente qual era a razão das tarefas e o que estava sendo esperado a curto prazo.

Ainda sobre o uso de *sprints*, o time também gostou de ver que, ao término da iteração, estava pronto um bloco de funcionalidades relacionadas entre si e que davam uma nova característica ao jogo. Ou seja, ter objetivos por *sprint* ajudava não só a direcionar as tarefas, mas também identificar em quais aspectos houve melhorias. Apesar dessa forma, que restringe a áreas das tarefas, ser menos flexível que a anterior, a sensação de progresso em etapas é maior.

Outro ponto que contribuiu para motivar tanto os desenvolvedores quanto os *product owners* foi, desde o primeiro *sprint*, ter um protótipo funcional. Pode-se dizer que isso realmente foi contrastante com a fase anterior, pois no primeiro projeto demorou aproximadamente quatro meses até que houvesse uma janela mostrando uma animação na tela.

Foi possível confirmar o valor desses elementos motivacionais ao notar que um do integrantes que quase não produzia passou a pegar algumas tarefas, pois estava claro e bem definido o que precisava ser feito. Além disso, sabia-se exatamente como o cumprimento da tarefa ajudaria a chegar mais próximo ao objetivo do *sprint*.

Uma mudança que não foi totalmente nova, mas que mesmo assim trouxe melhora na organização foi o uso mais estruturado do quadro de tarefas. Apesar da lousa já desempenhar um papel muito próximo, ter o quadro organizado em colunas referentes a cada possível estado das tarefas e agrupá-las por histórias realmente melhorou o acompanhamento do trabalho cotidiano.

Por fim, o gráfico *burnup* foi uma ferramenta de acompanhamento que surpreendeu por seu valor. Além de ser um indicador visual explícito do quanto o grupo havia trabalhado, ou deixado de trabalhar, nas últimas reuniões, ele também reforçava o senso de urgência. Essa urgência era ainda mais incitada à medida que a curva de pontos cumpridos rumava em direção à data de término do *sprint* mas não conseguia se aproximar do tracejado de velocidade ideal, ou formava patamares. Apesar desse aspecto quase opressor do gráfico, a atividade de cumprir tarefas para poder colocar os respectivos pontos no gráfico e atingir a meta tinha até um caráter inesperadamente lúdico, o que também agradou a equipe.

O que quase deu certo

Apesar da aparente simplicidade na definição de histórias e tarefas, houve dificuldade em diferenciar corretamente esses dois conceitos na prática. Principalmente no início do uso do ***Scrum***, as histórias ficaram muito pequenas e praticamente não as dividiam-nas em tarefas. Em seguida, na tentativa de remediar essa situação, acabou-se criando histórias muito grandes e que acabaram não sendo concluídas em um *sprint*. Ou seja, mesmo usando histórias e tarefas, o grupo ainda precisa adequar melhor a forma como lidar com esses artefatos.

Outra prática que foi usada mas ainda precisa ser melhor calibrada é a realização de estimativas. No geral, as pessoas tiveram uma grande tendência em subestimar a complexidade das tarefas e, em virtude disso, se comprometer com entregas maiores do que a real capacidade do grupo. Em contrapartida, algumas atividades, como escrever documentação sobre determinada classe do projeto, tiveram grande erro na estimativa por terem sido consideradas muito mais trabalhosas do que na verdade foram. Assim, é preciso que essa habilidade coletiva continue sendo posta em prática para se conseguir resultados melhores.

A participação dos *product owners* foi outra prática que também foi implementada, mas que não funcionou exatamente como proposto. Como explicado na seção que descreve o início do uso de ***Scrum*** no grupo, em virtude de incompatibilidade de horários, agendar reuniões longas entre os desenvolvedores e os *product owners* não era possível. Assim, tanto as reuniões de priorização quanto de revisão acabaram se resumindo a encontros rápidos de acompanhamento. Uma vez que o *scrum master* já levava à reunião o objetivo do *sprint* e quais histórias foram levantadas, faltando apenas que fossem dadas as prioridades, os *product owners* acabavam não criando histórias. Além disso, como a priorização precisava ser feita somente entre as funcionalidades já levantadas, as prioridades muitas vezes eram induzidas pelas próprias interdependências das histórias.

Ainda que essas reuniões não tenham funcionado exatamente como deviam, só o fato de haver esse acompanhamento e dos *product owners* estarem próximos à evolução do projeto, fornecendo comentários, críticas e sugestões, já foi de grande valia. Essa visão de quem está envolvido no projeto, mas que não vive o dia a dia do desenvolvimento é importante por conseguir levantar ideias baseadas em uma perspectiva mais global. Esse foi outro ponto de oposição ao que aconteceu na primeira fase do grupo, pois anteriormente houve pouquíssimo acompanhamento "externo", menos ainda de modo a influenciar o andamento ou decisões do projeto.

O que não deu certo

A primeira prática que se mostrou completamente inviável foi o *daily meeting*. Para que essa atividade acontecesse, seria preciso que todos os integrantes estivessem presentes e no início das reuniões. Mas essa situação simplesmente não acontecia. Como os integrantes do grupo são alunos de vários cursos e de anos diferentes, não há um horário livre comum a todos. Com isso, ter todos presentes em uma mesma reunião era raro e, no mesmo horário então, praticamente nunca aconteceu.

Essa questão da assiduidade também interferiu em outras técnicas. A atividade de quebrar as histórias em tarefas, jogar o poker de planejamento e fazer as estimativas não foi colocada em prática tantas vezes quanto se deveria. O principal problema foi que em várias ocasiões, nos dias das reuniões planejadas para acontecer essa atividade, havia poucas pessoas presentes e acabava-se deixando isso para outro momento. Esse mesmo problema acontecia com as reuniões de retrospectiva. Essas últimas, porém, praticamente não foram feitas.

Além disso, o *product backlog* foi relativamente negligenciado devido à relação com os *product owners*. Como era o próprio time que criava as histórias e só fazia isso entre o fim de um *sprint* e início de outro, novas propostas de funcionalidades não iam surgindo durante a iteração. Dessa forma, o *product backlog* estava sempre vazio, já que todas as histórias criadas iam direto para o *sprint backlog*. Essa foi inclusive uma das causas do pequeno impacto que a priorização tinha sobre o planejamento do *sprint*.

Porém, mesmo tendo as histórias e tarefas no quadro, houve momentos nos quais algumas pessoas acabavam tendo ideias outras além das programadas para o *sprint* e já começavam a implementá-las. Provavelmente esse reflexo é herança da antiga forma de trabalho e precisa ser evitado. Se não, os esforços em planejamento, organização e acompanhamento do *sprint* não terão utilidade.

Por fim, o último ponto que pode ter sido considerado um equívoco foi o dimensionamento do time. Como dito anteriormente, a equipe era composta por aproximadamente vinte pessoas e pensou-se que a recomendação de que o ***Scrum*** deveria ser usado em grupos pequenos poderia sofrer um relaxamento. Dessa forma, optou-se por manter o tamanho da equipe mas trabalhar simultaneamente no novo projeto e dar continuidade ao *Horus Eye*. Contudo, após vários *sprints*, chegou-se a conclusão de que essa, de fato, não é uma boa abordagem e que provavelmente teria sido melhor se o time tivesse sido dividido em dois: um para trabalhar no *Horus Eye* e *UGDK* e outro focado apenas no segundo jogo.

Problemas enfrentados

Além dos fatores relativos à inserção da metodologia no cenário do *USPGameDev*, alguns outros também trouxeram dificuldades para o grupo durante o segundo semestre de 2011. Os problemas relacionados a horários certamente tiveram grande impacto, como foi possível ver nas seções anteriores, mas ainda existiram alguns outros.

Uma mudança negativa que houve no grupo após o término do ano de 2010 foi a saída de muitos integrantes mais experientes, que cursavam os últimos anos da graduação. Não é possível dizer que o perfil do grupo passou a ser de novatos, pois a maioria dos membros remanescentes estava cursando o terceiro ano em 2011. Mas os seniores, além de maior vivência no desenvolvimento de projetos, tinham também boas capacidades técnicas

e grande conhecimento sobre a estrutura do *Horus Eye* e da *UGDK*, já que ambos foram construídos seguindo pesadamente as propostas dessas pessoas.

Outro fato relevante durante a virada de ano foi a realização do processo de seleção para o ingresso de novos membros. Mesmo tendo um planejamento bem discutido, essa atividade fez com que o grupo inteiro entrasse em um estado de inércia onde muito pouco foi produzido.

No âmbito individual, um ponto levantado pelos próprios desenvolvedores foi a pouca proatividade das pessoas. Independentemente da metodologia, o progresso dos trabalhos do *USPGameDev* depende exclusivamente do esforço voluntário de seus participantes. Assim, tendo passado a euforia e agitação da criação do grupo e levando em conta a saída de vários veteranos, a empolgação de muitas pessoas diminuiu. Consequentemente, a assiduidade geral também caiu e menos empenho estava sendo dado em cumprir as tarefas. Não se pode dizer que isso aconteceu com todos os participantes, mas essa diferença foi notória.

Para completar, fatores externos também foram obstáculos durante esse período. O primeiro foi a indisponibilidade da sala de trabalho devido a uma reforma no local. Em seguida, já com a sala pronta para se tornar um laboratório, precisou-se ajudar na instalação da mobília e dos computadores. Também ficou a cargo do grupo configurar todas as máquinas e montar uma rede com elas. Por fim, o grupo ministrou alguns cursos de tecnologias variadas durante aproximadamente um mês. Todo esse trabalho sem dúvida foi bom em muitos aspectos para o *USPGameDev* como grupo com pretensões de se oficializar dentro da universidade, porém essas atividades acabaram interferindo do andamento do projeto.

Conclusão

Com tudo o que foi levantado, foi possível aprender sobre as técnicas de gerenciamento de projeto utilizando o ***Scrum*** e aplicá-las na prática. Isso permitiu introduzir essa metodologia no *USPGameDev* e viabilizar uma comparação entre duas experiências de desenvolvimento: uma sem suporte de metodologia alguma e outra com práticas do ***Scrum***. Dessa análise, foi possível chegar a uma série de conclusões.

A primeira conclusão que se tem após esse trabalho é que o *USPGameDev* não está no formato mais adequado ao uso do ***Scrum***. O grupo é muito grande, pouco assíduo e não é pontual. Por esses motivos, algumas práticas não puderam ser adotadas durante e alguns problemas que a metodologia cobriria não foram atacados. Uma alternativa levantada que poderia resolver essa questão seria dividir o grupo em dois times de desenvolvimento independentes.

Quanto a questão de assiduidade e pontualidade, entendeu-se que esse é um problema inerente ao perfil dos membros do grupo. Como os desenvolvedores são alunos fazendo uma atividade sem qualquer tipo de retorno financeiro ou nota em qualquer disciplina, o nível de exigência quanto aos horários não pode ser o mesmo que o feito sobre um funcionário em uma empresa. Portanto, essa é uma dificuldade mais própria do contexto no qual o grupo está inserido do que à forma como se trabalha.

Outra conclusão que é possível tirar é que alguns problemas, como a confusão entre os

conceitos de história e tarefa, foram decorrentes da falta de experiência do *scrum master* em conduzir projetos usando a metodologia em questão. Seria interessante verificar também quais os resultados o grupo obteria caso fosse guiado por alguém mais experiente.

Além disso, foi possível observar que a falta de um cliente demandando funcionalidades se tornou uma deficiência no mecanismo da metodologia. Dada tal dificuldade, esse problema poderia ser superada se os *product owners* desempenhassem um papel muito mais ativo e com uma participação muito próxima ao time. Porém, como isso não aconteceu, viu-se que passar as responsabilidades para a própria equipe de desenvolvimento não é uma boa solução.

Em contra partida, mesmo com todas essas deficiências durante sua implantação, o ***Scrum*** conseguiu trazer benefícios ao grupo. Dentre eles, os mais significantes foram reduzir a ociosidade durante as reuniões e tornar esses encontros de trabalho mais objetivos e dinâmicos. Por conta disso, os integrantes do *USPGameDev* decidiram que seria vantajoso continuar usando as práticas do ***Scrum*** que foram implantadas com sucesso e inclusive já propuseram algumas adaptações que poderiam tornar a metodologia mais adequada ao contexto do grupo.

Em suma, o ***Scrum*** não se mostrou totalmente eficiente para um grupo como o *USPGameDev*. Entretanto, o uso da metodologia foi mais vantajoso do que um gerenciamento de projetos menos estruturado.

Parte Subjetiva

Desafios e frustrações no TCC

Para muitas pessoas, uma das partes mais difíceis do trabalho de formatura é a escolha do projeto. No meu caso, essa não foi uma parte problemática. A ideia de tentar usar na prática uma metodologia ágil no *USPGameDev* surgiu naturalmente depois do primeiro ano do grupo, pois os integrantes sentiram que a falta de uma estrutura para organizar o desenvolvimento do projeto estava trazendo grandes dificuldades em se atingir resultados que, a princípio, todos pensavam ser factíveis.

Entretanto, apenas estudar uma metodologia e fazer o grupo adotá-la dava a impressão de não ser um trabalho com valor acadêmico suficiente para um TCC. Assim, o primeiro desafio encontrado foi tentar dar mais relevância à proposta. Dada essa situação, em conversa com o Prof. Flávio, surgiu a idea de elaborar um estudo de caso baseado na comparação entre a falta e o uso de uma metodologia. Dessa forma, seria possível avaliar se realmente o uso de práticas de gerência de projeto cumpre o que promete.

Em seguida, foi preciso encontrar uma metodologia que fosse adequada ao contexto de produção de jogos e não apenas ao desenvolvimento de *software*. Essa sutil diferença era crucial, uma vez que criar um jogo envolve muitas outras atividades além de programar. Por esse motivo, não foi possível se ater a *eXtreme Programming*, disciplina a qual eu já havia cursado. Além disso, seria muito conveniente se fosse possível encontrar alguma metodologia que já fosse voltada para esse domínio. Foi com surpresa descobrir que praticamente não havia nada nesses moldes. A única referência encontrada foi o livro do Clinton Keith [4], que trata sobre ***Scrum*** e acabou se tornando a principal base deste trabalho. Uma vez definida a metodologia a ser usada, foi possível estudar sobre ela e tentar aplicá-la ao ambiente do *USPGameDev*.

Contudo, simplesmente montar um embasamento teórico era apenas uma fração do trabalho. Era preciso explicar para os integrantes do grupo as diferentes práticas, como elas funcionavam, a relação entre elas, o motivo para usar cada uma delas e, o mais difícil, fazer com que todos realmente as encorporassem à sua rotina de atividades. Este, sem dúvida, foi um dos maiores desafios e também um fator de frustração. Primeiro, porque não se consegue mudar o comportamento habitual de cada um tão facilmente quanto se troca o algoritmo de um programa. Segundo, porque a minha própria falta de experiência certamente contribuiu para que a motivação do grupo com relação aos novos métodos não tenha ficado tão alta. Não digo que as pessoas não tenham se esforçado em adotar o ***Scrum***, mas a impressão que tive foi que eu não consegui instigar nelas a noção de era preciso haver uma forte mudança de postura coletiva para tudo se encaixar e funcionar como um todo.

Apesar disso, a adoção da metodologia trouxe bons frutos e escutar dos próprios integrantes que houve uma melhora na condução do projeto foi muito gratificante.

Para concluir essa seção, não posso omitir que um outro fator de frustração foi o fato de, mesmo tendo um longo tempo onde diluir o trabalho de escrever esta monografia, houve a necessidade de desprender grande esforço por volta do final do prazo para que fosse possível concluir o texto. Por mais que eu tenha feito resumos durante a fase de estudos e várias anotações durante todo o período de acompanhamento do *USPGameDev*, poucas foram as vezes antes do final nas quais eu escrevi formalmente esses tópicos. Essa inabilidade em deixar as coisas prontas com antecedência foi realmente frustrante.

Relação entre disciplinas e o TCC

Apesar do pensamento imediatista de que várias disciplinas do curso não serviram para fazer o TCC, ou não serviram para nada, seria infantil não admitir que quase todas as disciplinas cursadas tiveram algum tipo de influência no trabalho desenvolvido, ou ao menos ajudaram a formar e consolidar o pensamento "já que ninguém vai ensinar isso direito, será preciso aprender sozinho mesmo".

Porém, algumas disciplinas tiveram influência direta ou nesse TCC ou no trabalho dentro do *USPGameDev*, o que também envolve esse TCC. Assim sendo, abaixo estão listadas essas disciplinas e o motivo de sua importância:

- **MAC 0122 - Princípios de Desenvolvimento de Algoritmos**
  Sem dúvida, foi a matéria que começou a mostrar a diferença entre um curso de Computação e um curso de programação. Essa disciplina mostrou o valor de se encontrar boas soluções para os problemas. Isto é, algoritmos e implementação claros, eficientes e elegantes. Além disso, nessa matéria também foi feito o primeiro contato com estruturas de dados básicas, mas que são usadas em praticamente todos os algoritmos.

- **MAT 0139 - Álgebra Linear para Computação**
  Dentre as disciplinas do departamento de matemática, foi a que mais se mostrou aplicável no desenvolvimento de jogos. O uso de vetores e operações sobre eles estão presentes desde a movimentação lógica dos personagens até às transformações necessárias para desenhar os elementos na posição correta da tela.

- **MAC 0211 - Laboratório de Programação I**
  Nessa disciplina foram apresentadas várias ferramentas de extrema importância para o desenvolvimento de projetos de porte maior do que os EPs costumavam ser. As mais relevantes foram: controles de versão (SVN), Make, CMake, LaTeX e Doxygen. Esse primeiro contato com essas ferramentas foi fundamental, pois elas foram utilizadas exaustivamente durante o curso. Além disso, o projeto proposto no semestre foi construir um jogo completo e essa experiência teve enorme valor.

- **MAC 0323 - Estruturas de Dados**
  Podendo ser vista como uma continuação de MAC 0122, essa disciplina apresenta

estruturas de dados mais complexas e os principais algoritmos usados para se trabalhar com elas. Além disso, os EPs dessa matérias são realmente trabalhosos e complicados, mas estimulam bastante a busca por boas soluções e incitam o gosto pela programação.

- **MAC 0242 - Laboratório de Programação II**
  Essa disciplina deu uma nova oportunidade para a criação de outro jogo durante o curso. Mas juntamente com isso, foi feito um primeiro contato com orientação a objetos, além do desenvolvimento de uma linguagem de programação simples e um compilador para ela. Essa atividade foi muito interessante para ajudar a entender como funciona a "mágica" que transforma texto em programas os quais o computador consegue executar.

- **MAC 0338 - Análise de Algoritmos**
  Apesar de raramente fazer o cálculo real de complexidade, em várias situações tanto no curso quanto no *USPGameDev*, escolhas entre diferentes algoritmos foram feitas com base na noção de eficiência que essa disciplina formou. Essa matéria também reforça formalmente alguns dos conceitos apresentados em MAC 0122 e MAC 0323.

- **MAC 0332 - Engenharia de Software**
  Embora essa disciplina tenha sido "conturbada" em decorrência de vários fatores, certamente teve grande influência nesse projeto. Não é possível dizer que esse TCC está fundamentado nos conceitos sugeridos em aula, mas as explicações sobre os métodos de gerenciamento de grandes projetos e os motivos pelos quais eles funcionam, ou não, ajudaram a perceber quais questões são realmente importantes dentro dessa área e contribuíram para a formação de uma visão crítica sobre cada uma das teorias.

- **MAC 0342 - Laboratório de Programação Extrema**
  Essa disciplina pode ser considerada como um contrapeso de MAC 0332, apresentando conceitos de metodologias ágeis de desenvolvimento de *software*. Como já dito anteriormente, as técnicas ensinadas nessa disciplina não eram suficientemente gerais para o contexto de criação de jogos, mas muitas das ideias apresentadas são semelhantes ao sugerido em ***Scrum***. Vale acrescentar que a vivência do projeto dessa matéria foi um ponto decisivo para a adoção de uma metodologia ágil no *USPGameDev* e nesse trabalho.

- **MAC 0441 - Programação Orientada a Objetos**
  Apesar dessa disciplina ser optativa, ela é muito valiosa para a boa estruturação de códigos de maior complexidade. A abordagem mais aprofundada sobre orientação a objetos complementa do que foi visto em MAC 00242 e os padrões de projeto apresentados muitas vezes solucionam perfeitamente várias questões recorrentes ou complicadas.

- **PCS 2530 - Design e Programação de Games**
  Sendo a única disciplina voltada para a criação de jogos na USP, ministrada na FAU, mas por um professor da Poli, apresentou uma visão bem mais afastada da

programação e muito direcionada ao design e a aspectos relacionados a experiência de jogar. Essa abordagem focando nas outras questões de um jogo além das funcionalidades e código fonte serviu para expor o quão viesado estava o pensamento do grupo e o quanto eram importantes vários elementos que foram negligenciados durante o desenvolvimento do *Horus Eye*.

- **MAC 0420 - Introdução a Computação Gráfica**
  Juntamente com MAC 0139, essa disciplina contribuiu bastante para solucionar questões relacionadas a projeções, posicionamento de câmera e mapeamento entre coordenadas da tela e coordenadas do mundo virtual. Além disso, nessa matéria foi feita uma introdução ao OpenGL, o que ajudou a entender muitos problemas que estavam acontecendo na *UGDK*.

Próximos passos

Apesar do término do TCC, há várias práticas de ***Scrum*** que ainda podem ser implantadas no *USPGameDev*. Também há uma ampla possibilidade de se elaborar práticas mais adaptadas ao perfil do grupo. Como meus planos envolvem iniciar o curso de mestrado no IME em 2012, pretendo continuar acompanhando o *USPGameDev* e ajudar nessas tarefas por pelo menos mais um ano.

Tratando especificamente sobre a adaptação das práticas e tendo em vista a falta de referências sobre o tema, uma das possibilidades de projeto de pós-graduação sugeridas pelo Prof. Flávio é o levantamento e elaboração de um conjunto de práticas especificamente direcionado para produção de jogos. Entretanto, acredito que ainda preciso de mais embasamento prático e vivência nesse meio para ter condições de propor uma nova forma de trabalho. Em virtude disso, caso esse rumo seja tomado, é possível que eu passe por um período como empregado em alguma(s) empresa(s) desse ramo.

Alternativamente, existe também a ideia de se afastar um pouco da área de jogos e buscar um conjunto genérico de práticas adequado a projetos de diferentes temáticas mas que fosse específico para grupos com as mesmas características do *USPGameDev*, ou seja, formado por estudantes, sem muita experiência, com conhecimentos variados, horários não totalmente compatíveis, entre outros. Esse trabalho poderia contribuir com a formação, manutenção e crescimento de vários grupos no ambiente da graduação.

Agradecimentos

Antes de encerrar, gostaria de dizer que este TCC não foi fruto apenas do meu esforço individual. Sem o *USPGameDev* e todas as pessoas envolvidas com o grupo, um projeto com essa proposta seria simplesmente inviável. Por isso, agradeço a colaboração de todos os alunos que se empenharam em criar e manter o grupo por esses dois anos e que contribuíram para essa experiência.

Além dos alunos, não posso deixar de agradecer ao Giuliano Olguin, por ter sido o responsável por acender a centelha que originou o *USPGameDev* e por todo o esforço que ele fez para conseguir condições e infraestrutura para o grupo realizar suas atividades. Também gostaria de expressar minha gratidão aos professores Flávio S. C. da Silva, Carlos Eduardo Ferreira e José Coelho de Pina por seu inestimável apoio e disposição em ajudar.

Para concluir esta seção e o trabalho, deixo dois "muito obrigado" especiais. Um aos meus colegas de turma, pois sem eles certamente eu não estaria me formando. Outro aos meus pais, pois sem o incentivo deles eu não teria nem mesmo entrado na graduação.

Bibliografia

[1] Hirotaka Takeuchi and Ikujito Nonaka. The new new product development game. *Harvard Business Review*, janeiro 1986.

[2] Peter DeGrace and Leslie Hulet Stahl. *Wicked Problems, Righteous Solutions: A Catolog of Modern Engineering Paradigms*. Prentice Hall, 1990.

[3] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.

[4] Clinton Keith. *Agile Game Development with Scrum*. Addison-Wesley Professional, 2010.

[5] Ken Schwaber and Jeff Sutherland. Scrum guide. Website, julho 2011. Disponível em http://www.scrum.org/scrumguides/.

[6] Scrum (development). Website, novembro 2011. Disponível em http://en.wikipedia.org/wiki/Scrum_(development).