GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA PROPOSTA DE MELHORIA NO PROCESSO DE ESTIMATIVA DE TAMANHO DE SOFTWARE PARA PROJETOS GERENCIADOS POR SCRUM

POR

FLÁVIO ALMEIDA ARAÚJO SOBRINHO
(FAAS@CIN.UFPE.BR)

TRABALHO DE GRADUAÇÃO

Universidade Federal de Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~seccgrad

Recife-PE
2009

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA PROPOSTA DE MELHORIA NO PROCESSO DE ESTIMATIVA DE TAMANHO DE SOFTWARE PARA PROJETOS GERENCIADOS POR SCRUM

POR

FLÁVIO ALMEIDA ARAÚJO SOBRINHO

Monografia apresentada junto ao curso de Ciência da Computação da Universidade Federal de Pernambuco, na área de gerenciamento de projeto de software, como requisito obrigatório à obtenção do título de Bacharel.

Orientador: Prof. Dr. Hermano Perrelli de Moura

Co-orientadora: Luciana de Queiroz Leal

Recife-PE

2009

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA PROPOSTA DE MELHORIA NO PROCESSO DE ESTIMATIVA DE TAMANHO DE SOFTWARE PARA PROJETOS GERENCIADOS POR SCRUM

POR

FLÁVIO ALMEIDA ARAÚJO SOBRINHO

Monografia apresentada junto ao curso de Ciência da Computação da Universidade Federal de Pernambuco, na área de gerenciamento de projeto de software, como requisito obrigatório à obtenção do título de Bacharel.

Orientador: Prof. Dr. Hermano Perrelli de Moura
Co-orientadora: Luciana de Queiroz Leal

COMISSÃO EXAMINADORA

Prof. Dr. Hermano Perrelli de Moura
(Universidade Federal de Pernambuco)

Prof. Dr. Alexandre Marcos Lins de Vasconcelos
(Universidade Federal de Pernambuco)

Recife-PE, 01 de dezembro de 2009.

Recife-PE
2009

“…estimation is an art at best…”

K. Beck e M. Fowler

Dedico este trabalho à minha querida mãe Evânia.

Agradecimentos

Agradeço, primeiramente, à minha mãe Evânia, pessoa mais importante para mim, que torceu pelo meu sucesso em todos os momentos da minha vida e fez tudo que esteve ao seu alcance para que eu pudesse estar nesta universidade, sem ela nada teria acontecido. Agradeço pela paciência, carinho, compreensão e amor que ela me ofereceu.

Agradeço a meu pai, Eduardo, “Dudão – um coroa muito doido” (como prefere ser chamado) que serviu de inspiração e motivação pra mim por toda sua garra, bravura e inteligência.

Agradeço à minha tia, Niege, alguém que esteve Presente (com “P” maiúsculo) desde o meu nascimento e nunca mediu esforços para fazer qualquer coisa por mim. O mesmo vale para minha avó, Sebastiana, com toda sua dedicação comigo em meus primeiros anos de vida e por sua torcida, carinho e força inesgotáveis – e por sua grande orelha que adoro esticar ☺

Obrigado ao meu orientador, Prof. Hermano, e a minha co-orientadora, Luciana, pela atenção e paciência! Obrigado a todos os professores do Centro de Informática com os quais tive contato e foram importantes para mais esse passo em minha vida, em especial, ao Prof. André, que me ajudou bastante nesse fim de curso, sem ele eu estaria frito!

Obrigado a Vinícius, meu gerente na Pitang que apesar da complicação no projeto, me liberou esses dias para fazer este estimado trabalho de graduação, obrigado também a toda equipe desse mesmo projeto que segurou uma barra e me agüentou enquanto me dividia entre CIn/Pitang/Proativa/...!

Um agradecimento muito especial a todos os Bel’s deste país e, em particular alguns dos mais importantes: Matheus, João Gabriel, Adelmo, Ciro, Styve, Capixaba e Calegário.

E, como não poderia faltar aqui, um agradecimento pra lá de especial aos Bel’s que compõem a Proativa: Amirton, João Paulo e Lucas. Esses são os caras!

Agradeço aos meus grandes amigos Bruno, Paulo, Hugo, Flavinha, Clarissa, Zandra, Marcelo e Marcela.

Por fim, agradeço a Biu José.

Resumo

Um dos mais intensos desafios da Engenharia de Software é a consolidação de modelos adequados para estimativa de tamanho de software. Essas estimativas fornecem dados que permitem derivar outras medidas como esforço, tempo, custos e, ainda, prever a quantidade de pessoas que serão necessárias na execução do projeto. No entanto, atingir um alto grau de previsibilidade nos projetos de software é um desafio que há muito aguça os pesquisadores.

Os métodos desenvolvidos para estimar tamanho são os mais diversos, indo desde simples analogias entre projetos, até estudos sistemáticos utilizando abordagens matemáticas. O objetivo deste trabalho é propor uma melhoria no processo de estimativa de tamanho de software para projetos que seguem a metodologia Scrum de gerenciamento de projetos.

A partir da experiência do autor com *Scrum* e desenvolvimento de software numa empresa de Tecnologia da Informação situada em Recife; e da realização de um estudo bibliográfico na literatura sobre o estado da arte no que se refere à estimativa de tamanho de software, será apresentado este trabalho.

Abstract

One of the most intense challenges of software engineering is the consolidation of appropriate models to estimate the size of software. These estimates provide data to derive other measures such as effort, time, cost, and also predict the amount of people that will be needed to run the project. However, achieving a high degree of predictability in software projects is a challenge that has long sharpens the researchers.

The methods developed to estimate size are the most diverse, ranging from simple analogies between projects by systematic studies using mathematical approaches. This work is composed of an improvement in the process of estimating size of software for projects that follow the Scrum methodology of management.

From the author's experience with Scrum and software development company in the Information Technology located in Recife, and carrying a bibliographic literature on the state of the art regarding the estimated size of software will be presented this work.

Sumário

1 Introdução 13
1.1 Objetivo 13
1.2 Metodologia 13
1.3 Estrutura do documento 14
2 Estimativas de software 15
2.1 Análise de Pontos de Função (APF) 15
2.1.1 Determinar o tipo de contagem 16
2.1.2 Identificar o escopo da contagem e a fronteira da aplicação 17
2.1.3 Contar as funções de dados 17
2.1.4 Contar as funções transacionais 18
2.1.5 Determinar o PF não ajustado 21
2.1.6 Determinar o fator de ajuste 21
2.1.7 Calcular os Pontos de Função Ajustados 22
2.2 Pontos de Caso de Uso (PCU) 22
2.2.1 Avaliação dos atores e atribuição do grau de complexidade 23
2.2.2 Avaliação dos casos de uso e atribuição do grau de complexidade 23
2.2.3 Cálculo dos Pontos de Caso de Uso não ajustados 23
2.2.4 Determinação do fator de complexidade técnica 23
2.2.5 Determinação do fator de complexidade ambiental 24
2.2.6 Cálculo dos Pontos de Caso de Uso ajustados 25
2.3 Dia ideal 25
2.4 Pontos de estória do usuário 25
2.5 Planning poker 26
3 Scrum 28
3.1 Sprints 29
3.2 Papéis 29
3.3 Reuniões 30
3.4 Artefatos 31
3.5 Processo de estimativas em Scrum 34
4 Melhoria no processo de estimativa de tamanho em Scrum 35
4.1 Metodologia 35

4.2 Organização e projetos observados 35
4.3 Melhoria 1: Pontos de estória e planning poker 35
4.4 Melhoria 2: Pontos de estória com uma unidade de medida 36
4.5 Melhoria 3: Apenas uma rodada de planning poker 37
4.6 Melhoria 4: Velocidade do time e Fator de Ajuste da Produtividade 37
4.7 Discussão 38
5 Considerações finais 39
5.1 Contribuições 39
5.2 Trabalhos futuros 40
Referências Bibliográficas 41

Lista de ilustrações

Figura 2.1 - Planning poker. Retirado de: http://en.wikipedia.org/wiki/Planning_poker. 26
Figura 3.1 - Uma introdução visual ao Scrum, Mountain Goat Software. 29
Figura 3.2 – Cartão de estória do usuário (Beck K. , 1999). 31
Figura 3.3- Exemplo de Burndown Chart, indicando a quantidade de Pontos de Estória restante em cada dia do Sprint (Kniberg, 2007). 33
Figura 3.4 - As tarefas se movem no quadro da esquerda para direita até chegarem em "Pronto", o que significa que foram concluídas (Kniberg, 2007). 33

Lista de tabelas

Tabela 2.1 - Parâmetros para definição de complexidade de ALI e AIE. 18
Tabela 2.2 - Parâmetros para definição de complexidade de EE. 19
Tabela 2.3 - Parâmetros para definição de complexidade de SE. 20
Tabela 2.4 - Parâmetros para definição de complexidade de CE. 20
Tabela 2.5 - Características gerais do sistema. 21
Tabela 2.6 - Atividades do processo de contagem dos Pontos de Caso de Uso (Karner, 1993). 22
Tabela 2.7 - Pesos dos atores na PCU (Karner, 1993). 23
Tabela 2.8 - Pesos dos casos de uso na PCU (Karner, 1993). 23
Tabela 2.9 - Fatores e pesos de complexidade técnica na PCU (Karner, 1993). 24
Tabela 2.10 - Fatores e pesos de complexidade ambiental na PCU (Karner, 1993). 25
Tabela 3.1 - Exemplo de Product Backlog (Kniberg, 2007). 32

1 Introdução

Atingir um alto grau de previsibilidade de projetos de software é um desafio que há muito aguça os pesquisadores. Em algumas situações se tem a impressão de que não há como estabelecer uma métrica baseada nas ciências exatas. O Standish Group (2001) apresentou que apenas 27% de projetos avaliados foram finalizados no tempo e custo previstos e que 50% dos projetos custaram em média 180% a mais sobre a estimativa original. Ocorreu uma melhoria nas estatísticas de projetos bem sucedidos em relação a 1994, fixado em 16%, para 2001 calculado em 27%. A melhora é conseqüência direta dos investimentos na área de processos de engenharia de software e do amadurecimento dos profissionais.

Os dados mostram que é possível e necessário aumentar a eficiência do processo de software. Foi apoiado nesse ponto e numa necessidade real do autor em atuar na melhoria do processo de estimativa de um projeto comercial do qual o mesmo participa, que surgiu a motivação para este trabalho.

As técnicas de estimativa de tamanho de software mais tradicionais, nem sempre, condizem com os princípios do Scrum, elas são algumas vezes pouco coerentes com os tipos de projetos que se demanda hoje e outras vezes requerem muita energia gerencial. Além do fato, de que para a maioria delas é necessário um bom amadurecimento, ou total consolidação, dos requisitos do sistema, fato que nem sempre ocorre com projetos gerenciados por Scrum.

Nas seções seguintes deste capítulo, serão definidos: o objetivo do trabalho, a metodologia aplicada e, na última seção, a estrutura do documento.

1.1 Objetivo

O objetivo deste trabalho é propor uma melhoria no processo de estimativa de tamanho de software para projetos que seguem a metodologia Scrum de gerenciamento de projetos.

1.2 Metodologia

Para a realização deste trabalho, foi feito um estudo bibliográfico na literatura, buscando o estado da arte no que se refere à estimativa de tamanho de software. Além da observação e estudo da metodologia Scrum sendo posta em prática ao longo de cerca de doze *Sprints*.

A proposta é um resultado empírico envolvendo a vivência do autor com gestão de projetos, desenvolvimento de software e gerenciamento com Scrum. O estudo prático que embasou esse trabalho teve seu foco numa empresa de desenvolvimento de software

situada em Recife da qual o autor é colaborador. Mais detalhes sobre a organização e o projeto em questão serão vistos no Capítulo 4.

1.3 Estrutura do documento

Este trabalho está organizado com a seguinte estrutura:

No Capítulo 2, **Estimativas de software**, são apresentadas as definições para estimativa de tamanho de software e a relevância de se ter um processo de estimativa adequado para o sucesso de um projeto de desenvolvimento de software. Ainda neste capítulo, são expostas algumas das técnicas de estimativa de tamanho de software mais empregadas.

No Capítulo 3, **Scrum**, discorre-se sobre os princípios do Manifesto Ágil e define-se a metodologia Scrum para desenvolvimento ágil de software.

No Capítulo 4, **Melhoria no processo de estimativa de tamanho em Scrum**, o leitor verá uma proposta de melhoria no processo de estimativa de tamanho de software para projetos gerenciados por Scrum.

No Capítulo 5, **Considerações finais**, encontram-se as principais conclusões deste trabalho e perspectivas futuras.

2 Estimativas de software

A tríade escopo, prazo e qualidade é há bastante tempo familiar às organizações de desenvolvimento de software. Para manter o equilíbrio deste tripé e concluir o projeto com sucesso, é necessário lançar mão de técnicas de planejamento e gestão: dos projetos, processos e produtos de software. Nesse contexto, são criados mecanismos baseados em informações quantitativas e qualitativas para o acompanhamento do projeto. Sob a perspectiva do gerente, essas informações são as métricas que orientam a condução do projeto.

De acordo com Monteiro (2005): "A medição é fundamental para qualquer atividade de engenharia; a engenharia de software não é a exceção." Muitas vezes, na engenharia de software, o planejamento de um projeto pode ter uma variação nas estimativas devido à falta de dados históricos, e à disponibilização de reduzido número de técnicas maduras (Pressman, 2002).

As estimativas de custo e esforço em projetos de software são, comumente, baseadas na predição do tamanho do sistema que será desenvolvido (Abrahao & Insfran, 2008). A atividade de estimar o tamanho do software é uma das mais críticas dentro do ciclo de vida de desenvolvimento, pois será a partir dela que o projeto será contratualmente regido.

Contudo, o desempenho das técnicas aplicadas nesse propósito, por muitas vezes, é bastante incipiente (Haugen, 2006). Ainda, segundo Haugen (2006), a pouca precisão das estimativas causam estouro no orçamento, atrasos, perda de contratos e levam a baixa qualidade do software.

A literatura aponta diversas métricas de estimativa de tamanho de projetos de software. Contudo, escolher a mais adequada não é uma decisão trivial. Alguns exemplos de técnicas para esse tipo de métrica são: Análise de Pontos de Função (APF); Pontos de Caso de Uso (PCU), Mk II Function Point Analysis, Features Points, LOC (Lines of Code), COSMIC Full Function Point, Dia ideal, Pontos de estória, Planning poker. Neste capítulo, serão discutidas algumas dessas técnicas.

2.1 Análise de Pontos de Função (APF)

No início da década de setenta, pesquisadores da IBM perceberam que um sistema seria mais bem avaliado através das funções executadas pelos programas, em oposição ao volume ou complexidade do código gerado (técnica bastante usada na época).

Em 1979, Albrecht, trabalhando para IBM, definiu a Análise de Pontos de Função (Function Point Analysis), já em 1984, ele refinou as definições sobre a APF no IBM CIS Guideline 313, *AD/M Productivity Measurement and Estimate Validation*. E, em 1986, um

grupo de usuários da APF formou o *International Function Point User Group* (IFPUG), o qual é responsável por manter seus associados informados a respeito das novas atualizações da técnica.

A APF tornou-se um padrão internacional em 2002 através da norma ISO/IEC 20.926[1] e é considerada uma das primeiras métricas a medir o tamanho do software com alguma precisão. Ainda hoje é uma técnica bastante difundida.

Segundo Pressman (2002) a APF é uma métrica orientada à função, derivada a partir de medidas diretas, que dimensionam o software considerando a funcionalidade entregue ao usuário final. A técnica estabelece uma medida, Pontos de Função (PF), de tamanho para o software que é calculado através da quantificação das funções desenvolvidas sob perspectiva do usuário. Os objetivos da APF, de acordo com a IFPUG, são:

- Medir o que foi requisitado e recebido pelo usuário;
- Prover uma métrica para a qualidade e análise de produtividade;
- Fornecer um método para estimar o tamanho do software, independentemente da tecnologia utilizada para implementação e sob a perspectiva do usuário;
- E prover um fator de normalização para comparação de software.

A métrica APF tem como princípio básico focar na especificação de requisitos para obter estimativas de tempo, custo, esforço e recursos na fase inicial do projeto ou na manutenção de software, independentemente da tecnologia utilizada para implementação.

O processo de contagem dos Pontos de Função deve ser composto pelas seguintes etapas (Albrecht, 1979):

1. Determinar o tipo de contagem
2. Identificar o escopo da contagem e a fronteira da aplicação
3. Contar as funções de dados
4. Contar as funções transacionais
5. Determinar o PF não ajustado
6. Determinar o fator de ajuste
7. Calcular os Pontos de Função Ajustados

2.1.1 Determinar o tipo de contagem

A APF se propõe a estimar o tamanho de projetos de desenvolvimento; aplicações em uso ou projetos de manutenção.

[1] ISO/IEC 20926: http://www.iso.org/iso/catalogue_detail.htm?csnumber=35582

2.1.2 Identificar o escopo da contagem e a fronteira da aplicação

O escopo define as funções que serão contadas de acordo com a visão do usuário, e a fronteira da aplicação separa o projeto a ser contado dos usuários das aplicações externas ao domínio do projeto.

2.1.3 Contar as funções de dados

As funções de dados consistem de: Arquivos Lógicos Internos, grupos de dados logicamente relacionados, cujo processo de alteração é feito em outra aplicação, (ALIs); e Arquivos de Interface Externa, grupos de dados logicamente relacionados ou informações de controle, que são alterados na própria aplicação, (AIEs). Um AIE de uma aplicação sempre será contado como um ALI na aplicação de origem. Para identificar um ALI, devem ser observadas as seguintes regras:

- Se o grupo de dados ou informações de controle é lógico e identificável pelo usuário;
- Se o grupo de dados é mantido através de um processo elementar dentro da fronteira da aplicação a ser contada.

A complexidade dos ALIs é identificada através do número de itens de dados (campos da tabela e deve-se contabilizar um item de dado para cada campo) e de registros lógicos (subgrupos de dados reconhecidos pelo usuário dentro ALI). Para identificar os itens de dados, seguir as regras:

- Contar um item de dados para cada campo reconhecido pelo usuário como único, não repetido, mantido por um ALI ou recuperado de um AIE ou de um ALI;
- Quando duas aplicações mantêm e ou referenciam o mesmo ALI ou AIE, mas cada uma mantém ou referencia itens de dados separados, conte apenas os itens de dados utilizados em cada aplicação para avaliar a complexidade dos ALI e AIE;
- Contar um item de dado para cada campo requerido pelo usuário para estabelecer um relacionamento com outro ALI ou AIE que, no caso, é considerada chave estrangeira.

Para identificar um AIE, verificar grupos de dados ou informação de controle que satisfaçam a definição de um AIE. As seguintes regras devem ser verificadas:

- O grupo de dados ou informação de controle é lógico e identificável pelo usuário;
- O grupo de dados é referenciado pela aplicação a ser contada mas pertence à outra aplicação fora da fronteira da aplicação;
- O grupo de dados não mantido pela aplicação a ser contada;

- O grupo de dados é mantido em um Arquivo Lógico Interno de outra aplicação.

De acordo com o número de itens de dados e de registros lógicos identificados, classifica-se o ALI e AIE conforme a **Tabela 2.1** e atribui os pesos de 7, 10 ou 15 PFs para os ALIs e 5, 7 ou 10 PFs para os AIE respectivamente à complexidade simples, média ou alta.

	Itens de Dados		
Registros Lógicos	*1 a 19*	*20 a 50*	*51 ou mais*
1	Simples	Simples	Média
2 a 5	Simples	Média	Alta
6 ou mais	Média	Alta	Alta

Tabela 2.1 - Parâmetros para definição de complexidade de ALI e AIE.

2.1.4 Contar as funções transacionais

Estas funções representam as funções de processamento dos dados fornecidos pela aplicação ao usuário. As funções transacionais podem ser Entradas Externas, recebidas de fora da fronteira da aplicação, (EE); Saídas Externas, enviadas para fora da fronteira da aplicação, (SE); e Consultas Externas, entrada e saída de dados, (CE). Compõem-se de itens referenciados (parâmetros que estão sendo representados pelos seus tipos) e de arquivos referenciados.

Para identificar as EEs, devem ser analisados todos os processos elementares que recebem dados de fora da aplicação e que atualizam um ou mais ALIs de acordo com as seguintes regras:

- Os dados ou informação de controle devem ser recebidos de fora da fronteira da aplicação;
- No mínimo, um AIE é mantido, se a entrada de dados pela fronteira não for informação de controle que altera o comportamento do sistema.

Para contabilizar o item de dado referenciado na transação de EE, considerar as seguintes regras:

- Os itens de dados de acordo com a visão do usuário;
- Contar os itens de dados, que aparecem mais de uma vez em um arquivo por causa da tecnologia ou técnica de implementação apenas uma vez;
- Considerar um item de dado adicional para as teclas de função ou linha(s) de comando que especificam a ação a ser tomada pela EE;
- Contabilizar os campos não informados pelo usuário, mas que são atualizados em um ALI por uma EE;

- As mensagens de erros, confirmação ou itens de dados adicionais, devem ser consideradas, caso sejam requeridas pelo usuário.

De acordo com o número de itens de dados, ALI e AIE referenciados, classifica-se a EE em baixa, média ou alta, conforme a **Tabela 2.2** com os respectivos pesos 3, 4 ou 6 PF.

	Itens de Dados		
Arquivos Referenciados	*1 a 4*	*5 a 15*	*16 ou mais*
0 a 1	Baixa	Baixa	Média
2	Baixa	Média	Alta
3 ou mais	Média	Alta	Alta

Tabela 2.2 - Parâmetros para definição de complexidade de EE.

Para identificar as SE, verificar o processamento lógico do processo elementar de acordo com as seguintes regras:

- Se contém no mínimo, uma fórmula matemática ou cálculo;
- Se cria dados derivados;
- Se mantém no mínimo, um ALI;
- Se altera o comportamento do sistema.

Já a complexidade da SE é verificada através do número de arquivos e elementos de dados referenciados. Deve-se contar um arquivo referenciado para cada ALI mantido ou lido ou um AIE lido durante o processamento da SE.

Para identificar os itens de dados, observar as regras:

- Contar um item de dado para cada campo reconhecido pelo usuário, não repetido, que entre pela fronteira da aplicação e seja requerido para especificar quando, qual e/ou como o dado será recuperado ou gerado pelo processo elementar ou que saia da fronteira da aplicação. Se um item de dado entra e sai pela fronteira da aplicação, contá-lo apenas uma vez;
- Contar um item de dado quando a aplicação enviar mensagens de resposta para fora da fronteira, indicar um erro ocorrido durante o processamento, confirmar o processamento completo ou verificar se o processamento deva continuar;
- Contar um item de dado para a habilidade de especificar uma ação a ser tomada, quando existem múltiplos métodos para chamar o mesmo processo lógico;

- Não contar os campos que são recuperados ou derivados pelo sistema e armazenados em um ALI durante o processo elementar se esses campos não cruzam a fronteira da aplicação;
- Não contar variáveis, números de páginas, data/hora ou selos gerados pelo sistema.

Após identificar os arquivos e os itens de dados referenciados, classificar a SE conforme a **Tabela 2.3** abaixo e atribuir o peso 4, 5 ou 7 PF respectivo à complexidade baixa, média ou alta.

	Itens de Dados		
Arquivos Referenciados	*1 a 5*	*6 a 19*	*20 ou mais*
0 a 1	Baixa	Baixa	Média
2 a 3	Baixa	Média	Alta
4 ou mais	Média	Alta	Alta

Tabela 2.3 - Parâmetros para definição de complexidade de SE.

A CE visa apresentar informação para o usuário através da recuperação de dados ou informação de controle. Verificar se o processamento lógico é elementar, segundo as regras abaixo:

- Recuperar dados ou informações de controle de um ALI ou AIE;
- Não contém fórmulas matemáticas ou cálculos;
- Não cria dados derivados;
- Não mantém ALI;
- Não altera o comportamento do sistema.

De acordo com o número de itens de dados e arquivos referenciados, classificam-se as CEs conforme a **Tabela 2.4** e atribui-se o peso 3, 4 ou 6 PF conforme a complexidade baixa, média ou alta.

	Itens de Dados		
Arquivos Referenciados	*1 a 5*	*6 a 19*	*20 ou mais*
0 a 1	Baixa	Baixa	Média
2 a 3	Baixa	Média	Alta
4 ou mais	Média	Alta	Alta

Tabela 2.4 - Parâmetros para definição de complexidade de CE.

2.1.5 Determinar o PF não ajustado

Após identificar as funções de dados e transacionais, multiplicar o total de ALI, AIE, EE, SE e CE pela respectiva complexidade para determinar o valor de PF não ajustado. De acordo com a complexidade, cada uma das funções de dados e transacionais contribui com um determinado número de PF. O total desses pontos de função computados é chamado de pontos de função não-ajustados.

2.1.6 Determinar o fator de ajuste

O nível de influência dos fatores de ajustes técnicos é determinado com base nas 14 características gerais de sistemas, conforme mostra a **Tabela 2.5**. É atribuída uma nota de 0 a 5 a cada uma das Características Gerais do Sistema correspondendo ao seguinte critério: nenhuma influência, influência incidental, moderada, média, significante, essencial. Em seguida devem-se somar todas as notas atribuídas e aplicar a fórmula:

$$\textbf{Fator de ajuste} = 0{,}65 + (0{,}01 \times \sum \text{notas})$$

Equação 1

Características Gerais do Sistema
Comunicação de Dados
Processamento de Dados Distribuído (Funções Distribuídas)
Performance
Configuração do equipamento
Volume de Transações
Entrada de Dados On-Line
Interface com o usuário
Atualização On-Line
Processamento Complexo
Reusabilidade
Facilidade de Implantação
Facilidade Operacional
Múltiplos Locais
Facilidade de mudanças

Tabela 2.5 - Características gerais do sistema.

Estas características influenciarão a complexidade do software e conseqüentemente seu tamanho. As características gerais do software podem variar o tamanho do software em 35% mais ou menos.

2.1.7 Calcular os Pontos de Função Ajustados

Os Pontos de Função ajustados são calculados a partir dos **PFs não ajustados** e do **fator de ajuste** de acordo com a seguinte fórmula:

APF = PF não ajustado x Fator de ajuste

Equação 2

2.2 Pontos de Caso de Uso (PCU)

A métrica Pontos de Caso de Uso (PCU) foi apresentada no início da década de 90, como resultado de uma tese de doutorado de Karner (1993), utilizando como base a APF. Sua motivação foi a disseminação de sistemas orientados a objetos e a suspeita de que a métrica APF não seria adequada para avaliar sistemas modelados nesse contexto. Empresas como a Rational, posteriormente adquirida pela IBM, e a SUN, utilizam a métrica PCU.

Até o momento, apesar da propriedade da IBM, não houve disponibilização de versões posteriores à publicação original, nem criação de programas de certificação de profissionais usuários da métrica, ou reconhecimento por normas internacionais. Como ponto de destaque, os projetos orientados a objetos podem ser medidos diretamente através dos diagramas de casos de usos modelados, uma vez que a pontuação é baseada em casos de uso.

A contagem dos Pontos de Caso de Uso (PCU) é executada realizando-se seis atividades, conformo disposto na **Tabela 2.6**. A identificação de atores e casos de uso deverá ser realizada a partir dos diagramas de caso de uso e de classes, presentes em projetos Orientados a Objetos (OO).

#	**Atividade**
1	Avaliação dos atores e atribuição do grau de complexidade
2	Avaliação dos casos de uso e atribuição do grau de complexidade
3	Cálculo dos Pontos de Caso de Uso não Ajustados (PCUNA)
4	Determinação do fator de complexidade técnica
5	Determinação do fator de complexidade ambiental
6	Cálculo dos Pontos de Caso de Uso Ajustados (PCUA)

Tabela 2.6 - Atividades do processo de contagem dos Pontos de Caso de Uso (Karner, 1993).

2.2.1 Avaliação dos atores e atribuição do grau de complexidade

Os atores identificados na aplicação a ser medida deverão ser classificados em Simples, Médio ou Complexo. As regras de classificação e os pesos associados estão dispostos na **Tabela 2.7**.

Complexidade	Regra de definição	Peso
Simples	Ator que representa outro sistema, com uma interface de integração definida	1
Médio	Atores que representam um dos seguintes cenários: - Interação com outros sistemas através de protocolo de comunicação; - Interação humana via interfaces texto	2
Complexo	Interação através de interface gráfica de usuários	3

Tabela 2.7 - Pesos dos atores na PCU (Karner, 1993).

2.2.2 Avaliação dos casos de uso e atribuição do grau de complexidade

Assim como ocorre com os atores, os casos de uso da aplicação em avaliação deverão ser classificados em Simples, Médio ou Complexo. As regras de classificação e os pesos associados estão dispostos na **Tabela 2.8**.

Complexidade	Regra de definição	Peso
Simples	Envolve no máximo três transações e é realizado com até cinco classes de objetos	5
Médio	Envolve entre quatro e sete transações e é realizado com cinco a dez classes de objetos	10
Complexo	Envolve mais que sete transações e é realizado com no mínimo dez classes de objetos	15

Tabela 2.8 - Pesos dos casos de uso na PCU (Karner, 1993).

2.2.3 Cálculo dos Pontos de Caso de Uso não ajustados

O cálculo dos Pontos de Caso de Uso não Ajustados (PCUNA) é determinado pela soma dos pontos obtidos por todos os atores e casos de usos presentes na aplicação em avaliação, de acordo com a fórmula:

$$\textbf{PCUNA} = \sum \text{pesos dos atores} + \sum \text{pesos dos casos de uso.}$$

Equação 3

2.2.4 Determinação do fator de complexidade técnica

Para melhor precisão na estimativa, a PCU prevê a avaliação de dificuldade da construção da aplicação. Para tanto, treze fatores devem ser classificados em uma escala

de 0 (zero), indicando ser irrelevante, a 5 (cinco), representando ser essencial. A **Tabela 2.9** apresenta os fatores e seus respectivos pesos.

Fator	Descrição	Peso
T1	Sistema distribuído	2
T2	Tempo de resposta ou desempenho	1
T3	Eficiência do usuário final	1
T4	Complexidade do processamento interno	1
T5	Reusabilidade de código	1
T6	Fácil instalação	0.5
T7	Facilidade de uso	0.5
T8	Portabilidade	2
T9	Facilidade de mudanças	1
T10	Concorrência	1
T11	Segurança	1
T12	Acesso por terceiros	1
T13	Treinamento de usuários	1

Tabela 2.9 - Fatores e pesos de complexidade técnica na PCU (Karner, 1993).

O Fator de Complexidade Técnica (FCT) é obtido através da fórmula:

$$\textbf{FCT} = 0{,}6 + (0{,}01 \times \sum \text{Fatores técnicos}).$$

Equação 4

2.2.5 Determinação do fator de complexidade ambiental

Assim como a avaliação de desafios técnicos, a PCU também prevê a avaliação da eficiência do projeto através do mapeamento de fatores ambientais. A escala de medida também classifica cada fator em 0 (zero), indicando ser irrelevante a 5 (cinco), representando ser essencial. A **Tabela 2.10** apresenta os fatores e seus respectivos pesos.

#	Regra de definição	Peso
1	Familiaridade com o processo de desenvolvimento do software	1,5
2	Trabalhadores com dedicação parcial	-1
3	Capacidade do analista Líder do Projeto	0,5
4	Experiência na aplicação	0,5
5	Experiência com Orientação a Objetos, na linguagem e na técnica de desenvolvimento	1

6	Motivação	1
7	Dificuldade na linguagem de programação	-1
8	Requisitos estáveis	2

Tabela 2.10 - Fatores e pesos de complexidade ambiental na PCU (Karner, 1993).

O Fator de Complexidade Ambiental (FCA) é obtido através da fórmula:

$$\mathbf{FCA} = 1{,}4 - (0{,}03 \times \sum \text{Fatores ambientais})$$

Equação 5

2.2.6 Cálculo dos Pontos de Caso de Uso ajustados

Finalmente, os Pontos de Caso de Uso Ajustados (PCUA), são calculados envolvendo os Pontos de Caso de Uso não Ajustados (PCUNA), o Fator de Complexidade Técnica (FCT) e o Fator de Complexidade Ambiental (FCA), de acordo com fórmula:

$$\mathbf{PCUA} = \text{PCUNA} \times \text{FCT} \times \text{FCA}$$

Equação 6

2.3 Dia ideal

O “dia ideal” é uma livre-tradução do termo definido por Kent Beck para referenciar um dia totalmente livre de impedimentos para o engenheiro de software – programador – que ele chamou de “Ideal Engineering Time”.

Em seu livro, *Extreme Programming Explained*, Beck descreveu o dia ideal, como o tempo necessário para concluir uma estória do usuário (ver seção 3.4) “sem interrupções ou reuniões" (Beck K. , 1999). Esta idéia ressalta que os desenvolvedores de software eventualmente executam outras atividades durante o dia, além de programar. Eles vão para as reuniões, freqüentam seminários, conversam com colegas, limpam suas mesas, etc. É difícil para um desenvolvedor, de forma realista, incorporar todas essas outras tarefas diárias em uma estimativa. Assim, os desenvolvedores estimam em tempo ideal (Hohman, 2005).

2.4 Pontos de estória do usuário

Numa tentativa de sanar as lacunas deixadas pela estimativa do dia ideal, Cohn (2004) apresentou a técnica de pontos por estória do usuário, também conhecida como *story points*. Os pontos são uma unidade arbitrária e indivisível relativa à complexidade da estória que visa a um mapeamento de tempo de calendário necessário para a conclusão da atividade (Hohman, 2005). Ou seja, uma estória com dois pontos de tamanho deve levar, aproximadamente, duas vezes o tempo (de calendário) necessário para implementar uma estória com um ponto.

Os desenvolvedores estimam cada estória, atribuindo um número de pontos, tentando garantir que as estórias de complexidade semelhante recebam o mesmo número de pontos. Para evitar falsa precisão, é comum, se utilizar de faixas de números esparsas (como a seqüência de Fibonacci), desse modo se evita a alta granularidade nas estimativas e discussões pouco relevantes como: “a estimativa é 2 ou 2,5?”. Isso estimula quebrar estórias grandes, em menores, para tornar a estimativa mais confiável.

A velocidade da equipe pode ser medida em número de pontos completado por iteração. Como os pontos da estória não são unidades de tempo, se evita a confusão entre o ideal e o real, pois para o cliente pode não ser claro que dez dias ideais vão levar mais que dez dias de cronograma. É mais fácil explicar aos clientes que o orçamento para a próxima iteração é de vinte pontos, em oposição a vinte dias ideais.

Para ambos, as estimativas de tempo ideal e de ponto por estória são indícios de complexidade, e não de tempo até a conclusão. Usar uma unidade de estimativa abstrata pode tornar o acompanhamento do cronograma mais claro para cliente e equipe.

2.5 Planning poker

A idéia que envolve o planning poker é bem simples. As estórias são individualmente apresentadas para serem estimadas. Após um período de discussão, cada participante (munido com uma pilha de cartas com valores numéricos) escolhe do seu monte uma carta com o valor que representa sua estimativa de tamanho para a estória em discussão. Todas as estimativas são mantidas em sigilo até que todos os participantes escolham suas cartas. Em seguida, as estimativas são reveladas ao mesmo tempo e a discussão pode começar de novo com os participantes que atribuíram o menor e o maior peso, respectivamente, expondo seus pontos de vista para tentar convencer os demais. A **Figura 2.1** mostra um jogo de cartas para o planning poker.

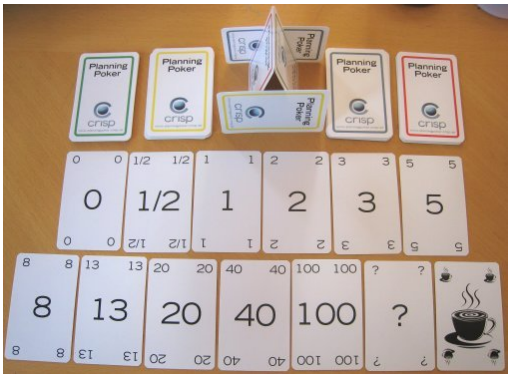

Figura 2.1 - Planning poker. Retirado de: http://en.wikipedia.org/wiki/Planning_poker.

Estudos de Haugen (2006) mostram uma melhora na eficiência das estimativas com pontos por estórias quando utilizada esta técnica para atribuição dos pontos. De acordo com o autor, o maior problema com as técnicas não-estruturadas é que pessoas com personalidades mais fortes e indivíduos com, supostamente, melhor compreensão da estória, podem influenciar os demais participantes. O sujeito mais disposto a falar, assim que tiver a oportunidade, vai tomar a palavra e dar sua estimativa, deixando os outros membros estimulados a seguir aquela linha de pensamento apresentada, ou com receio de expor seu ponto de vista, se manterão caladas.

3 Scrum

Após o lançamento do Manifesto Ágil em 2001, um novo termo passou a coexistir no universo de desenvolvimento de software, *processo ágil*. Os princípios do Manifesto Ágil, são:

- Indivíduos e iterações ao invés de processos e ferramentas;
- Software funcionando ao invés de documentação abrangente;
- Colaboração com o cliente ao invés de negociação de contratos e
- Responder a mudanças ao invés de seguir um plano.

Embora exista valor nos itens à direita, os processos ágeis valorizam mais os itens que estão à esquerda (Beck, et al., 2001).

Processos ágeis são baseados no ciclo de vida iterativo e incremental afim de permitir que os inevitáveis erros das pessoas sejam descobertos relativamente cedo e reparados de forma metódica (Cockburn, 2002). Além disso, eles estimulam refatoração, auto-organização e, especialmente, adaptação.

Entre as metodologias que se apóiam nos princípios do Manifesto Ágil, está o Scrum que, segundo Schwaber (2004) - principal formalizador da metodologia - Scrum é:

> "... o mais perplexo e paradoxal processo para gerenciamento de projetos complexos. Porém, Scrum é incrivelmente simples. [...] Scrum não é um processo prescritivo; ele não descreve o que fazer em cada circunstância. Ele é usado para trabalhos complexos nos quais é impossível predizer tudo que irá acontecer." .

Ainda de acordo com Schwaber (2004) Scrum é algo sobre o "senso comum", indicando que não há uma forma padrão para o processo, mas, sim, uma filosofia para a gestão ágil do projeto.

Indo mais além, Cohn (2002) diz que Scrum é um processo ágil que permite manter o foco na entrega do maior valor de negócio, no menor tempo possível:

- Isto permite a rápida e contínua inspeção do software em produção (em intervalos de duas a quatro semanas);
- As necessidades do negócio é que determinam as prioridades do desenvolvimento de um sistema;
- As equipes se auto-organizam para definir a melhor maneira de entregar as funcionalidades de maior prioridade;
- Num intervalo de duas a quatro semanas por iteração, todos podem ver o real software em produção, decidindo se o mesmo deve ser liberado ou continuar a ser aprimorado por mais um *Sprint*.

3.1 Sprints

Projetos Scrum progridem em uma série de iterações chamadas *Sprints*, que ocorrem num período de de duas a quatro semanas. É recomendado que esse período se mantenha constante ao longo do desenvolvimento do projeto, pois um período constante leva a um melhor "ritmo". O produto é projetado, codificado e testado durante o *Sprint.* A **Figura 3.1** apresenta uma visão geral do Scrum em execução.

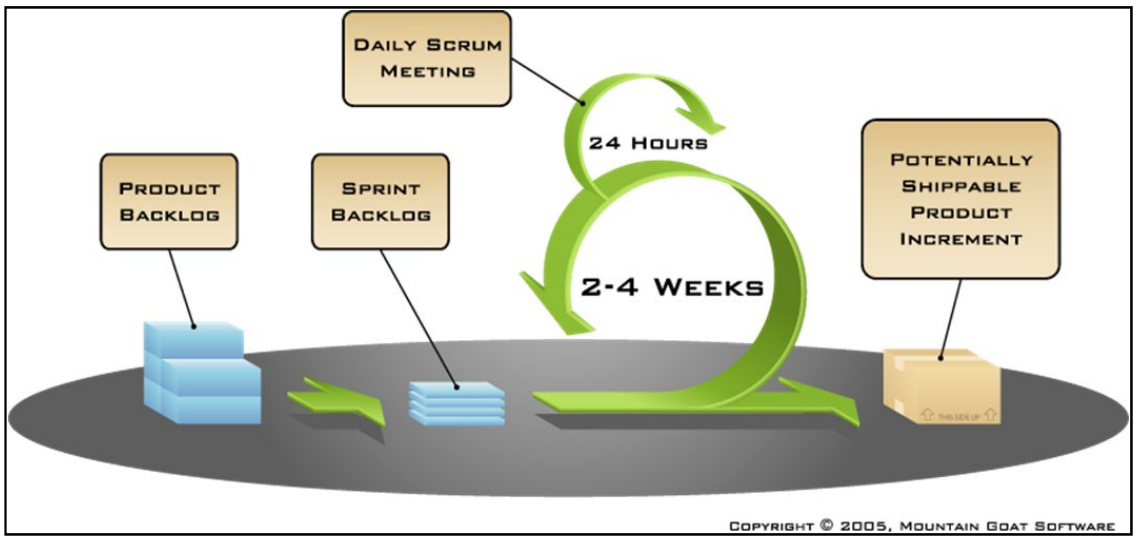

Figura 3.1 - Uma introdução visual ao Scrum, Mountain Goat Software.

3.2 Papéis

De acordo com Schwaber (2004), o Scrum é dividido em 3 papéis: Product Owner, ScrumMaster e o Time. Sendo as responsabilidades divididas da seguinte forma:

Product Owner (PO) – define as funcionalidades do produto; decide datas de lançamento e conteúdo; é o responsável pelo controle do Retorno de Investimento (ROI - Return On Investment) do projeto; é ele quem prioriza funcionalidades de acordo com o valor de negócio (importância) do requisito; aceita ou rejeita o resultado dos trabalhos desenvolvidos pelo Time.

ScrumMaster – é responsável pelo processo, para treinar os envolvidos no projeto e aplicar os valores do Scrum. Ele representa a gerência para o projeto, removendo obstáculos e garantindo a plena funcionalidade e produtividade da equipe. Também, garante a colaboração entre os diversos papéis e funções.

Time – é a equipe responsável por desenvolver as funcionalidades. Os times para o Scrum são auto-organizados e multi-funcionais. Os membros do time são coletivamente responsáveis pelo sucesso de cada iteração e do projeto como um todo. É recomendado um Time com no máximo 9 pessoas.

3.3 Reuniões

O processo de desenvolvimento se inicia com uma reunião de planejamento, chamada *Sprint Planning Meeting*, na qual o *Product Owner* e o Time decidem em conjunto o que deverá ser implementado do *Product Backlog*. Esta reunião é dividida em duas partes:

- *Sprint Planning 1*, o Product Owner apresenta os requisitos de maior valor e prioriza aqueles que devem ser implementados. O Time então define colaborativamente o que poderá entrar no desenvolvimento do próximo Sprint, considerando sua capacidade de produção.
- *Sprint Planning 2*, o Time planeja seu trabalho, definindo o *Sprint Backlog* (mais detalhes na próxima seção), que são as tarefas necessárias para implementar as funcionalidades selecionada no *Product Backlog*. A lista de tarefas pode ser modificada ao longo da Sprint pelo Time e recomenda-se que as tarefas sejam divididas de forma a levar no máximo 16 horas para a sua conclusão.

Depois da reunião de planejamento, o Time realiza o desenvolvimento do produto durante o *Sprint*. Durante esse período alterações no *Product Backlog* não são permitidas.

Durante o Sprint, diariamente o Time se reúne no *Daily Scrum Meeting* ou Reunião Diária que deve durar, no máximo, 15 minutos. Nessas reuniões são respondidas por cada integrante do time as seguintes perguntas:

- que eu fiz desde a última reunião?
- que eu irei fazer até a próxima reunião?
- Quais os impedimentos bloqueando minhas tarefas?

Essa reunião deve ser realizada no formato *Stand up Meeting,* o que facilita ao cumprimento do tempo por todos estarem em pé e um valor que é abordado nesse momento é o da comunicação. Essa prática de falar os problemas e o que ocorreu resulta em um momento rico de troca de conhecimento da equipe e de comunicação resultando uma importante a sincronização do trabalho do time.

Ao final do *Sprint* é realizada a reunião de revisão, *Sprint Review Meeting*, na qual o Time apresenta o resultado alcançado na iteração ao PO. Neste momento as funcionalidades são inspecionadas e adaptações do projeto podem ser realizadas.

Após essa prática é feito o *Sprint Retrospective,* que tem por objetivo alinhar toda a equipe sobre o que está e não está funcionando bem dentro do processo. A equipe discute o que gostaria de:

- iniciar a fazer;
- parar de fazer;
- continuar fazendo.

Essa prática é importante para a equipe se conhecer melhor e aumentar a eficiência nos próximos *Sprints*.

3.4 Artefatos

Os artefatos envolvidos no Scrum são mínimos e podem ser geridos da maneira mais conveniente à organização. Seus itens são compostos por:

Estórias do usuário – servem a mesma finalidade dos casos de uso, mas não são a mesma coisa. São usadas em substituição a grandes documentos de requisitos, escritas pelo usuário, sem expressões técnicas, mas com a descrição daquilo que é necessário ao sistema. As estórias são semelhantes aos cenários de uso, exceto que elas não se limitam a descrever uma interface com usuário. As estórias não devem ser confundidas com a especificação tradicional de requisitos, elas devem conter apenas detalhe suficiente para os desenvolvedores conseguirem estimar seu tamanho aproximado e efetuarem o planejamento do *release*. No momento da implementação da estória os detalhes devem ser levantados com o cliente, de preferência, pessoalmente. A **Figura 3.2** mostra um exemplo de estória do usuário.

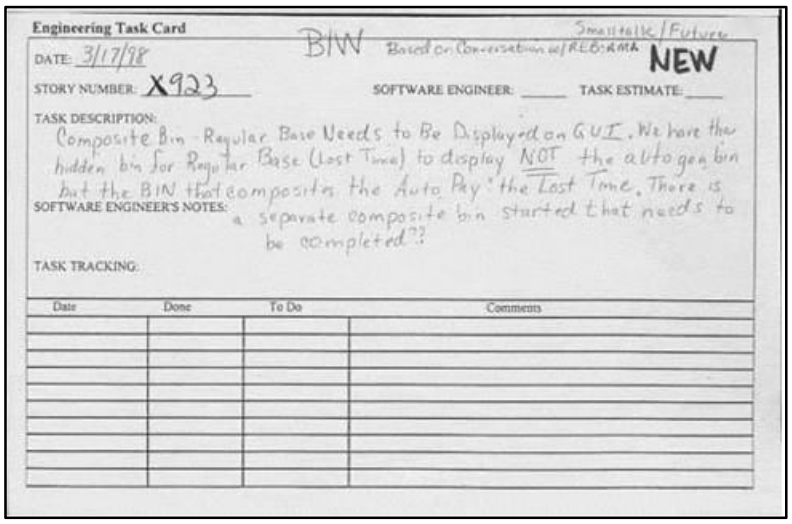

Figura 3.2 – Cartão de estória do usuário (Beck K. , 1999).

Product Backlog – é a lista de todos os requisitos que devem ser desenvolvidos no projeto. Cada item desta lista é chamado *Backlog Item* o qual deve estar priorizado com um peso de importância atribuído pelo PO. O *Product Backlog* priorizado é o ponto de partida para o início do *Sprint*. Na **Tabela 3.1**, é apresentado um exemplo de *Product Backlog* priorizado e com uma estimativa inicial.

PRODUCT BACKLOG (exemplo)					
ID	**Nome**	**Imp**	**Est**	**Como demonstrar**	**Notas**
1	Depósito	30	5	Logar-se, abrir a página de depósito, depositar R$ 10,00, ir para a página do meu saldo e verificar que este aumentou em R$ 10,00.	Precisa de uma diagrama UML de sequência. Não é necessário se preocupar com criptografia por enquanto.
2	Verificar seu próprio histórico de transações	10	8	Logar-se, clicar em "transações". Fazer um depósito. Voltar para transações, verificar se o novo depósito é listado.	Usar paginação para evitar consultas muito grandes ao banco de dados. Projetar de forma similar à página de visualização de usuários.

Tabela 3.1 - Exemplo de Product Backlog (Kniberg, 2007).

Sprint Goal – uma descrição breve e direta sobre o(s) objetivo(s) do Sprint que será iniciado. Cada objetivo deve ser descrito em termos de negócio, de modo que pessoas de fora da equipe entendam seu significado.

Sprint Backlog – é a lista de tarefas necessárias para implementar as funcionalidades selecionadas no Product Backlog para o *Sprint*. Cada indivíduo escolhe a atividade que lhe for mais conveniente dentre aquelas apontadas no *Sprint Backlog*. Deve haver uma atualização diária de quanto tempo resta em cada atividade da lista.

Burndown Chart – é um gráfico bastante simples, porém umas das ferramentas mais poderosas para o acompanhamento do *Sprint* e apontado por alguns como a única necessária. Este gráfico mostra ao longo do tempo a quantidade de trabalho que ainda resta para se realizar durante o *Sprint.*

A **Figura 3.3** apresenta um Burndown de Pontos de Estórias X Tempo Transcorrido no Sprint. Os Pontos de Estória são uma estimativa de tamanho atribuída pela equipe a

cada Estória (ou Item de *Backlog*) dentro do Sprint. No primeiro dia do sprint, 1 de agosto, a equipe estimou que havia aproximadamente 70 pontos de estória; em 16 de agosto a equipe estimou que havia aproximadamente 15 pontos de estória de trabalho a serem feitos. A linha de tendência tracejada mostra que eles estão aproximadamente dentro do prazo para completar todas as atividades do *Sprint* (Kniberg, 2007).

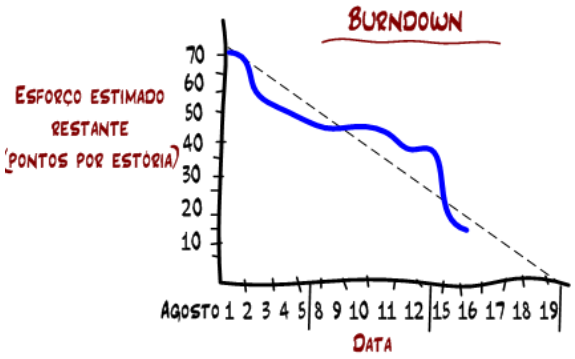

Figura 3.3- Exemplo de Burndown Chart, indicando a quantidade de Pontos de Estória restante em cada dia do Sprint (Kniberg, 2007).

Quadro de tarefas – também conhecido como *dashboard:* trata-se de um quadro onde todas as estórias (requisitos ou itens de *backlog*) que compõem o *Sprint Backlog* são expostas para que todos possam ver o andamento do *Sprint*. Neste quadro existem vários status pelos quais a tarefa se move ao longo do *Sprint*. A **Figura 3.4** mostra um exemplo desse quadro. No quadro também constam: o objetivo do *Sprint*, o gráfico de *Burndown*, a lista de tarefas que não foram planejadas, mas criadas dentro do *Sprint*, e uma lista das prováveis próximas estórias a serem desenvolvidas.

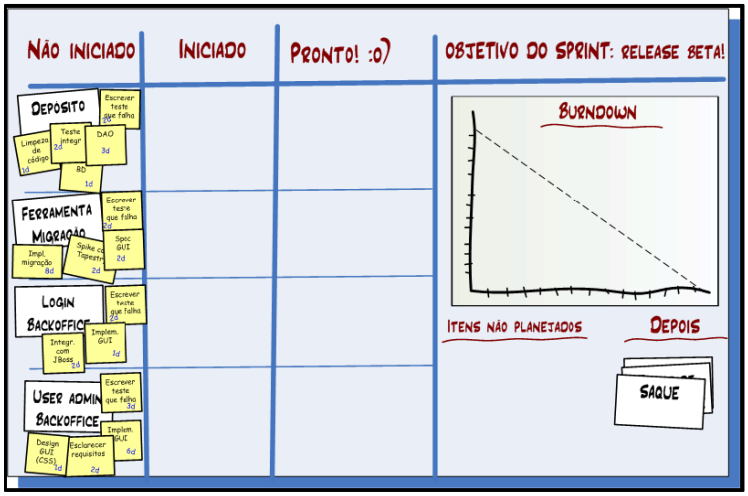

Figura 3.4 - As tarefas se movem no quadro da esquerda para direita até chegarem em "Pronto", o que significa que foram concluídas (Kniberg, 2007).

3.5 Processo de estimativas em Scrum

Cohn (2004) sugere o uso de pontos por estória e do *planning poker* como um processo de estimativa. Com o uso de bases históricas para cada time é possível realizar uma estimativa de quantos pontos poderão ser desenvolvidos num *Sprint*.

Para utilizar a técnica de estimativa de pontos por estória em Scrum, inicialmente o *Product Owner* deve atribuir valores de negócio para cada um dos itens do *Product Backlog*, priorizando quantitativamente todos os itens. Estes valores podem ser números arbitrários, de 1 a 1000, por exemplo, ou o retorno esperado, em reais, de cada item.

Em seguida, o time deve identificar o item do *Product Backlog* mais simples de implementar e atribuir para este item um número de pontos com baixo valor, normalmente 1 ou 2. A seguir, o time deve estimar todos os itens do *Product Backlog* tomando este item como referencial e responder perguntas como “Se A possui tamanho 2, qual o tamanho de B?”.

É recomendado o uso da Seqüencia de Fibonnaci (1, 2, 3, 5, 8, 13, 21, 34...) para atribuição dos pontos. A fim de que, o time convirja mais rápido para um resultado (e por motivos já mencionados na seção 2.5).

O *Product Owner* deve estar presente nesta sessão (*Sprint Planning Meeting)* de estimativa para tirar dúvidas e esclarecer melhor as funcionalidades que vão ser implementadas. Com o *Product Backlog* estimado, no primeiro *Sprint* utiliza-se de bases históricas (ou da experiência do time, quando dados não estão disponíveis) para definir quantos pontos o time desenvolverá. A partir do segundo *Sprint*, usa-se o mesmo número que foi efetivamente desenvolvido no Sprint anterior. Apenas estórias completamente desenvolvidas, testadas e aceitas pelo cliente são consideradas para a velocidade de desenvolvimento do time.

4 Melhoria no processo de estimativa de tamanho em Scrum

Como foram observados no Capítulo 2, os tipos e variações de técnicas para estimar o tamanho de um software são diversas. Neste trabalho, interessam as técnicas aplicadas no planejamento de *releases* e iterações de projetos gerenciados por metodologia ágil, mais especificamente, aqueles que utilizam Scrum.

Neste capítulo, será apresentada uma proposta de melhoria no processo de estimativa do Scrum.

4.1 Metodologia

O conjunto de melhorias aqui apresentadas é fruto da observação e estudo da metodologia Scrum sendo posta em prática ao longo de cerca de doze *Sprints*. A proposta é um resultado empírico envolvendo a vivência do autor com gestão de projetos, desenvolvimento de software e gerenciamento com Scrum.

Foi imprescindível a esse trabalho um amplo estudo da literatura sobre Scrum e análise de casos de sucesso. Entretanto, é importante mencionar que, muitas vezes, práticas aplicadas com grande sucesso em certas organizações e projetos, podem não ser tão satisfatórias em outros cenários. Por tanto, os resultados aqui observados não são propostos como um padrão a ser empregado em qualquer projeto, mas eles refletem uma situação própria que eventualmente podem ser utilizados em outro contexto.

4.2 Organização e projetos observados

Foram observadas duas organizações situadas em Recife para se chegar às melhorias aqui apontadas: a Proativa e a “PCS” (optou-se por preservar a identidade da organização, sendo este apenas um codinome). Da primeira organização, o autor é um dos sócios e na segunda, presta serviço como colaborador.

Para este trabalho a segunda organização teve uma relevância significativamente maior e o foco deste trabalho foi em um dos projetos desta empresa. O projeto analisado da “PCS” trata-se de um grande sistema previdenciário cujo cliente é um banco nacional. Este projeto está em andamento, seu tempo de desenvolvimento foi estimado em cerca de dois anos, seu orçamento próximo a R$ 2 milhões e possui 11 participantes entre desenvolvedores, gerente e analista trabalhando *full-time*, além destes há gerente de configuração e gerente de qualidade.

4.3 Melhoria 1: Pontos de estória e planning poker

De acordo com a Seção 3.5, o Scrum, por padrão, adota o uso de pontos de estória para o dimensionamento de tamanho do software e para o cálculo de velocidade do time em cada *Sprint*. A literatura específica, também, afirma que é uma técnica bastante adequada.

Existem sobre essa questão duas linhas divergentes, para Cohn (2007) os pontos de estória são uma boa métrica quando aplicada no longo prazo, planejamento de *release*, por exemplo; já Sutherland (2009) é enfático em utilizar a técnica num contexto de curto prazo, como no planejamento de *Sprint*. Ambos os autores são reconhecidos ícones do movimento Ágil e indivíduos responsáveis pela formalização do Scrum ao lado de outros especialistas.

Dessa forma, neste trabalho será considerada a adoção da prática de pontos de estória para planejamento de *Sprint* e *Realease*, com o auxílio do planning poker (devido às razões já discutidas na Seção 2.5). Contudo, os pontos terão uma unidade e poker sofrerá uma ligeira variação, tais mudanças serão vistas nas seções seguintes.

4.4 Melhoria 2: Pontos de estória com uma unidade de medida

A técnica original propõe que os pontos de estória sejam adimensionais, ou seja, sem qualquer unidade de medida, ele representa uma unidade indivisível e neutra.

Aqui, sugere-se que os pontos de estória possuam, agora, uma grandeza. A idéia é utilizar a noção do "dia ideal" no momento de pontuar a estória.

Esta necessidade foi verificada empiricamente pelo autor e sem dados levantados formalmente através de sua própria experiência no desenvolvimento de software e da sua vivência numa empresa de Tecnologia da Informação (TI) situada em Recife.

Na hora de dar uma estimativa acerca do tamanho de uma estória, os desenvolvedores sentem falta de algum parâmetro - "essa estória tem peso dois o quê?" – é um dos questionamentos. Outro fator, num projeto que se estende por muito tempo, fatalmente, haverá certa rotatividade de desenvolvedores, de modo que cada vez que entra/sai alguém a configuração do time muda.

O impacto disso é na depreciação do fator histórico de *Sprints* passados. Ou seja, a velocidade do time calculado com base num quesito abstrato (pontos de estória) começa a perder um pouco o sentido, pois com o time modificado a linha de pensamento e de parâmetros, vai com aqueles que saíram. Enquanto que para os que chegam, uma estória que possuía "peso" 3, pode ser vista como 13, pois o ponto de vista mudou completamente. Dessa maneira, é possível que ocorra uma nova "onda" de estimativas com maior fator de erro.

Para dirimir esse fenômeno, é proposto que a pontuação da estória seja, exatamente, o valor do "dia ideal" para completar a estória, ou seja, pergunta-se a equipe "quanto dias ideais de um homem seriam necessários para concluir esta estória?". A resposta, que virá através do *poker*, será a pontuação da estória em homens-dia.

4.5 Melhoria 3: Apenas uma rodada de planning poker

No seu trabalho, Hohman (2005) sugere adotar três ordens de grandeza de tempos para as estimativas, “muito provavelmente”, “no mínimo” e “no máximo”. Dessa forma, ele consegue um acompanhamento mais real do dia-a-dia do programador. No entanto, essa abordagem consome bastante energia de gerenciamento.

Aqui se propõe que haja apenas uma única rodada de *planning poker*, essa iniciativa possui dois objetivos específicos:

1. Aumentar a performance das reuniões de planejamento, que por várias vezes, recaem em 3 ou 4 rodadas de *poker*;
2. Ter acesso à margem que Hohman (2005) descreve em seu trabalho, mas de uma maneira que demande menos gerenciamento.

Então, logo na primeira rodada do *poker*, serão contabilizados o maior e o menor valor. A média aritmética dessas grandezas será a pontuação da estória.

4.6 Melhoria 4: Velocidade do time e Fator de Ajuste da Produtividade

Sabe-se que um desenvolvedor não será produtivo durante suas 8 (oito) horas de trabalho, ou seja, haverá reuniões, leitura de email, conversa com colegas, problemas de infra-estrutura (energia, internet, ar-condicionado). Como a estimativa está calculada em “dia ideal” é necessário um fator que estabeleça uma aproximação da estimativa com o “dia real” do programador para que seja calculado quanto de trabalho o time pode, efetivamente, concluir. Ou seja, é necessário descobrir quanto tempo o time está sendo produtivo, em média.

Chamaremos de Fator de Ajuste da Produtividade (FAP) o percentual de tempo útil (com foco total na atividade) que a equipe expressa durante um dado momento no projeto. Este fator pode variar bastante entre *Sprints* e irá depender do comprometimento do time e da quantidade de impedimentos que está perturbando o andamento do *Sprint*.

O FAP varia no intervalo [0, 1] quanto mais próximo de 1, mais livre de impedimentos e focado na atividade estará o time. Dessa forma, a velocidade do *Sprint,* ou a quantidade de pontos que a equipe pode resolver, será:

$$\mathbf{VS} = \text{FAP} \times \sum \text{homens-dia disponíveis no Sprint}$$

Equação 7 - Velocidade do Sprint em pontos de estória.

Para o primeiro *Sprint*, pode-se iniciar com um FAP qualquer, “0,7”, por exemplo. Pois, o fator deverá ser ajustado após cada *Sprint* de acordo com o número de pontos de estória que foram efetivamente concluídos pela equipe e a soma dos recursos humanos,

que representam a equipe, disponibilizados no mesmo *Sprint* (em homens-dia). Uma fórmula possível é:

FAP = ∑pontos concluídos / ∑homens-dia disponíveis

Equação 8 – Atualização do Fator de Produtividade.

4.7 Discussão

O intuito dessa proposta é preencher algumas necessidades que foram observadas no contexto apresentado. Dentre as várias técnicas discutidas no Capítulo 2, essa proposta buscou unir um pouco de algumas delas e, efetuando pequenas modificações, se chegou a um resultado plausível à realidade do projeto estudado. As técnicas tradicionais e mais antigas como APF não condizem com os princípios do Scrum: primeiro, porque nelas as estimativas são feitas por uma única pessoa (em geral o gerente) e não por quem de fato irá desenvolver o sistema (o time); segundo, que dentro da abordagem ágil os requisitos podem mudar constantemente e o objetivo é satisfazer ao cliente, com uma mais tradicional de estimativas fica muito custoso e complexo gerenciar com requisitos muito pouco definidos ou instáveis.

Os dados da aplicação das melhorias no projeto estudado não estão completamente consolidados, pois o segundo *Sprint* está em andamento durante a escrita deste trabalho. O primeiro *Sprint* foi bastante atípico tanto pela mudança no método de estimativas, quanto pelo fato do time ter sido quase que completamente renovado, mas pelo gráfico de Burndown, que se manteve muito próximo da linha planejada durante todo o *Sprint*, ficou explícito que este foi um dos *Sprints* com melhor andamento. Apenas isso foi suficiente para concluiu que a equipe estava, de fato, com problemas nas estimativas. E ainda, se percebeu o time muito mais empolgado em atingir a meta do *Sprint* e motivado em aumentar o fator de foco para o próximo *Sprint*!

Certamente, o conjunto de melhorias aqui propostas não é completo e nem suficiente por si só. Um ponto que está sendo analisado e também testado no *Sprint*, mas não foi citado aqui, é se vale apenas, ou não, as estimativas em hora associadas às tarefas de cada estória. A alternativa é se ter apenas o número de pontos da estória e o acompanhamento da baixa de pontos por dia de *Sprint* no gráfico de *Brundown*. Ainda é possível citar, pelo menos, outros dois fatores com influência direta no sucesso das estimativas em Scrum: (a) o time envolvido e motivado, de modo que o *Sprint Planning* seja bem executado, pois esta etapa é fundamental para o bom andamento do *Sprint*; (b) um Scrum Master com “bom senso” e percebendo as reais necessidades do time.

5 Considerações finais

Uma estimativa é apenas uma avaliação ou cálculo aproximado sobre algum fenômeno, no caso de software estimar o seu tamanho é um passo fundamental para a construção de um sistema. Para o projeto de desenvolvimento de software, seu sucesso está fortemente ligado ao poder de precisão das estimativas da equipe.

A maior lição observada com este trabalho é que Scrum é um processo, realmente, adaptativo e quem o faz, é quem o usa. Portanto, não cabe excesso de teoria e filosofia, e sim, melhoramentos em cada *Sprint* que reflitam a realidade da equipe e organização.

5.1 Contribuições

Nesse trabalho foi apresentada uma proposta de melhoria no processo de estimativa de tamanho de software para projetos que utilizam *Scrum* como metodologia de gerenciamento. E apesar de ter havido um projeto estudado particularmente, é esperado e bastante razoável que os pontos apontados neste trabalho se encaixem em diferentes projetos sendo gerenciados com Scrum. A seguir, uma síntese da proposta oferecida, os passos descritos acontecem durante a fase de *Sprint Planning* do *Scrum*:

1. O time estima o *Product Backlog* em pontos de estória
    a. Os pontos de estória equivalem à unidade “Homens-dia”;
    b. O número de pontos de uma estória representa quantos “dias ideais” um homem trabalhando sozinho levaria para completá-la;
    c. É utilizado o *planning poker* para esse processo – só deverá ocorrer uma única rodada; caso a equipe não seja unânime nesta rodada, será efetuada a média aritmética entre a maior e menor estimativa;
2. O *Scrum Master* instancia o Fator de Ajuste de Produtividade para o time do *Sprint*
    a. Caso seja o primeiro Sprint, pode-se usar qualquer valor. Por exemplo, 0,7;
    b. Caso contrário, o FAP é calculado de acordo com a velocidade do time no *Sprint* anterior;
3. O Scrum Master calcula a velocidade do time para conclusão de estórias com base no FAP e no tamanho do *Sprint*;
4. A equipe dá continuidade ao *Sprint Planning*, escolhendo as estórias que irão compor o *Sprint* com base na velocidade do time, ou seja, quantos pontos o time é capaz de concluir naquele *Sprint*.

5.2 Trabalhos futuros

Como trabalho futuro, sugere-se que a proposta aqui apresentada seja posta num estudo de caso controlado com projetos implementando as indicações deste trabalho, a fim de se alcançar um resultado comprovado do método. Através desse estudo, seria possível aperfeiçoar e sugerir um novo modelo para o processo de estimativa de tamanho de software em Scrum.

Também, como proposta para futuro, seriam um levantamento e categorização de projetos de software, visando à determinação da técnica de estimativa mais apropriada para cada categoria apresentada. O objetivo seria efetuar um mapeamento das metodologias de estimativa de tamanho para as categorias de projetos conhecidas. Desse resultado, seria possível sugerir um padrão para estimar software.

Referências Bibliográficas

Abrahao, S., & Insfran, E. (2008). A Metamodeling Approach to Estimate Software Size from Requirements Specifications. *Software Engineering and Advanced Applications. XXXIV*, pp. 465-475. Washington (DC): IEEE Computer Society.

Albrecht, A. J. (1979). Measuring Applications Development Productivity. *Proceedings of IBM Applications Development Join SHARE/GUIDE Symposium.* Monterey, CA.

Albrecht, A. J., & Gaffney, J. E. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering , VI* (9), pp. 639-648.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change* (1 ed.). Addison-Wesley.

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Agile Manifesto*. Acesso em 2009 de 11 de 26, disponível em http://agilemanifest.org Cockburn, A. (2002). *Agile software development.* Boston: Addison–Wesley.

Cohn, M. (2002). *An Overview of Scrum.* Acesso em 27 de novembro de 2009, disponível em Mountain Goat Software: http://www.mountaingoatsoftware.com/presentations/30-an-overview-of-scrum-january---in

Cohn, M. (2004). *User Stories Applied: For Agile Software Development.* Addison–Wesley.

Cohn, M. (2007). *Why I Don't Use Story Points for Sprint Planning*. Acesso em 28 de novembro de 2009, disponível em Mountain Goat Software: http://blog.mountaingoatsoftware.com/?p=15

Control Chaos. (s.d.). Acesso em 2009 de novembro de 27, disponível em Scrum - It's About Common Sense: http://www.controlchaos.com

Extreme Programming. (s.d.). *User Stories: A gentle introduction*. Acesso em 1 de dezembro de 2009, disponível em Extreme Programming: http://www.extremeprogramming.org/rules/userstories.html

Haugen, N. C. (2006). An Empirical Study of Using Planning Poker for User Story Estimation. *Proceedings of the Conference on AGILE 2006* (pp. 23-34). Washington (DC): IEEE Computer Society.

Hohman, M. M. (2005). Estimating in Actual Time. *Proceedings of the Agile Development Conference.* IEEE Computer Society.

Karner, G. (1993). *Use Case Points - Resource Estimation for Objectory Projects.* Thechnical Report, Objective Systems SF AB (Rational Software).

Kniberg, H. (2007). *Scrum e XP direto das Trincheiras - Como fazemos Scrum* (1 ed.). C4Media.

Monteiro, T. C. (2005). *Pontos de Caso de Uso Técnicos (TUCP): Uma Extensão da UCP.* Dissertação de Mestrado, Universidade de Fortaleza (UNIFOR), Mestrado em Informática Aplicada, Fortaleza.

Pádua, F. W. (2003). *Engenharia de software: fundamentos, métodos e padrões.* Rio de Janeiro: LTC.

Pereira, R., Tait, T. F., Trindade, D. F., Castro, C. Y., & Silva, J. V. (2007). Estimativas de Software: O Estudo de uma Aplicação Prática Utilizando a Técnica de Use Case Points. *Escola Regional de Informática*, *XIV*, pp. 24-35. Guarapuava-PR.

Pressman, R. S. (2002). *Engenharia de Software* (5 ed.). Rio de Janeiro: McGraw-Hill.

Schwaber, K. (2004). *Agile Project Management with Scrum* (1 ed.). Microsoft Press.

Standish Group. (1994). *Chaos Report 1994*. Acesso em 20 de novembro de 2009, disponível em The Standish Group International: http://www.standishgroup.com/sample_research/chaos_1994_1.php

Standish Group. (2001). *Chaos Report 2001*. Acesso em 20 de novembro de 2009, disponível em The Standish Group International : http://www.standishgroup.com/chaos/introduction.pdf

Sutherland, J. (2009). *Sprint Burndown: by hours or by story points?* Acesso em 28 de novembro de 2009, disponível em Scrum Log Jeff Sutherland: http://jeffsutherland.com/scrum/2009/04/sprint-burndown-by-hours-or-by-story.html

Wikipedia. (s.d.). *Cost estimation in software engineering*. Acesso em 18 de novembro de 2009, disponível em Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Software_estimation

Wikipedia. (s.d.). *Software development effort estimation*. Acesso em 18 de novembro de 2009, disponível em Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Software_development_effort_estimation