PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

CURSO DE ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

LUIZ RICARDO GODOI DE CASTRO SILVA

UMA ANÁLISE DAS TÉCNICAS DE ESTIMAÇÃO DE SOFTWARE USADAS NOS PROCESSOS TRADICIONAIS E ÁGEIS

São Paulo, abril de 2013

LUIZ RICARDO GODOI DE CASTRO SILVA

UMA ANÁLISE DAS TÉCNICAS DE ESTIMAÇÃO DE SOFTWARE USADAS NOS PROCESSOS TRADICIONAIS E ÁGEIS

Monografia apresentada ao Curso de Especialização em Engenharia de Software da Pontifícia Universidade Católica de São Paulo, como requisito parcial para obtenção do título de Especialista em Engenharia de Software, orientado pelo Prof. Dr. Ítalo Santiago Vega.

São Paulo, abril de 2013

UMA ANÁLISE DAS TÉCNICAS DE ESTIMAÇÃO DE SOFTWARE USADAS NOS PROCESSOS TRADICIONAIS E ÁGEIS

LUIZ RICARDO GODOI DE CASTRO SILVA

Monografia aprovada, em 10 de abril de 2013, pelo orientador:

Professor Doutor Ítalo S. Vega

Orientador

Dedico este trabalho a Jesus Cristo, o Autor da Vida, de quem provém toda a Sabedoria verdadeira.

Agradeço primeiramente ao Filho de Deus, Jesus, que, desde que confiei minha vida a Ele, tem transformado meu entendimento sobre todas as coisas.

Também à minha esposa, Fernanda, por suportar o tempo em que estive dedicado aos estudos.

Ao orientador deste trabalho, Dr. Ítalo, pelas inestimáveis explanações que transformaram completamente minha abordagem sobre a Engenharia de Software.

Por fim, aos meus colegas de turma, os quais compartilharam experiências importantes.

“Assim, quer vocês comam, bebam ou façam qualquer outra coisa, façam tudo para a glória de Deus.”

Paulo de Tarso (I Coríntios 15:31)

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A estimação é uma atividade importante e desafiadora do desenvolvimento de software. A gerência e os clientes precisam ter um horizonte do projeto e, dessa forma, tomar boas decisões de negócio. Estimar é fornecer uma visão clara o suficiente para que essas decisões sejam tomadas e os objetivos do projeto atingidos. Isto é feito tentando prever o esforço necessário para desenvolver o software e seus componentes.

Estimativas não são valores matemáticos absolutos, são representações da ordem de grandeza dos componentes de um software. Elas devem ser ajustadas de acordo com diversas variáveis. Os requisitos, os processos de gerenciamento e desenvolvimento de software, a arquitetura, as características e restrições do projeto, os fatores humanos, políticos e ambientais estão entre os itens que devem ser considerados na estimação.

Existem diversas técnicas e abordagens de estimação, mas não há uma relação objetiva de superioridade entre elas, cada uma possui características que as tornam mais ou menos adequadas para diferentes contextos.

Cabe ao gerente do projeto adotar uma técnica, seja ela formal ou empírica, de acordo com as necessidades do projeto. Porém, a qualidade da estimação dependerá da habilidade, do conhecimento e da experiência do estimador mais do que da técnica utilizada.

Palavras-chave: estimação de software, técnicas de estimação, engenharia de software.

ABSTRACT

Estimation is an important and challenging software development activity. The managers and customers must have a horizon of the project and thus make good business decisions. Estimating is to provide a view clear enough that these decisions are taken and the project objectives achieved. This is done trying to predict the effort required to develop the software and its components.

Estimates are not absolute mathematical values, but they are order of magnitude representations of software components. They should be adjusted according to several variables. Requirements, software management and development processes, architecture, characteristics and constraints of the project, human, environmental and political factors have to be considered in the estimation.

There are several estimation techniques and approaches, but there is no objective relation of superiority among them, each one has its characteristics that make them more or less suitable for different contexts.

Rests with the project manager the responsibility to adopt a technique, whether formal or empirical, according to project requirements. However, the quality of the estimation depends on the skill, knowledge and experience of the estimator more than the technique used.

Keywords: software estimation, estimation techniques, software engineering.

SUMÁRIO

INTRODUÇÃO

Estimação é uma atividade essencial de qualquer projeto de desenvolvimento de software. As estimativas podem definir a viabilidade do projeto e a forma como são redigidos contratos entre as partes numa eventual terceirização do desenvolvimento.

Entretanto, estimativas são apenas previsões. Elas contêm suposições sobre vários elementos utilizados no processo de estimação e mudam conforme os requisitos e muitos fatores que envolvem o desenvolvimento de software. Estimar é prever o futuro, portanto não existe um método científico para gerar estimativas comprovadamente verdadeiras.

Estimativas nunca refletem perfeitamente a realidade, mas é necessário que o profissional busque resultados suficientemente bons. Pressman (2006, p. 524) afirma que “um erro de estimativa pode fazer a diferença entre lucro e prejuízo”. O autor vai mais além e diz que, em uma era de terceirização e concorrência acirrada, a capacidade de estimar com mais acurácia tornou-se um fator crítico de sobrevivência para muitas organizações de tecnologia da informação.

Por outro lado, a atividade de estimação é importante para uma maior compreensão do projeto, não apenas para obter uma data de entrega. Quando sistematicamente executada, ela ajuda a fornecer uma visão abrangente sobre o ambiente no qual o sistema está inserido, pois as características da organização, da equipe, do problema e muitas outras devem ser levadas em consideração. As estimativas iniciais são ajustadas com o tempo, mas cada ajuste aprofunda o conhecimento sobre variáveis que influenciam o projeto.

Os diferentes modelos e técnicas utilizados na atividade de estimação fornecem métodos organizados e coerentes de executar esta atividade, mas a escolha de um dentre os meios disponíveis não é um fator primário para a qualidade das estimativas. A habilidade e a experiência do profissional são fatores primordiais, já que esta é uma atividade que envolve mais intuição do que cálculos. Porém, um método eficaz e que atenda as necessidades

específicas do projeto e da organização é uma ferramenta valiosa nas mãos de um estimador qualificado.

As estimativas são feitas com base no que se planeja construir, portanto a correta elicitação dos requisitos é uma tarefa fundamental para a obtenção de estimativas adequadas. A qualidade dos requisitos é diretamente proporcional à qualidade das estimativas. Toda estimativa é inválida quando não de define corretamente o que deve ser feito, pois ela não reflete a realidade do software.

A execução do processo de desenvolvimento de software deve ser gerenciada de forma a verificar se as estimativas anteriormente geradas estão compatíveis com a realidade. As estimativas devem ser calibradas no decorrer do projeto nos momentos adequados. O processo de desenvolvimento possui um papel importante, devendo fornecer a transparência e a estrutura necessárias para medição e controle.

O presente estudo fornece uma visão geral sobre a atividade de estimação no contexto de um projeto de software, revisando pontos importantes relacionados à atividade de estimação, discorrendo sobre engenharia de requisitos, modelos de processos de desenvolvimento e gerenciamento, modelos e técnicas de estimação e expondo um estudo de caso para ilustrar a aplicação do conteúdo teórico.

Através de revisão bibliográfica, serão apresentadas as bases teóricas relevantes da Engenharia de Software para o tema proposto, incluindo abordagens consideradas “tradicionais” e “ágeis”, ou seja, com maior ou menor rigor e grau de burocracia.

Organização do Trabalho

O trabalho está dividido em seis capítulos, onde os primeiros quatro contêm o referencial teórico necessário para o tema, o quinto um estudo de caso prático e o último conclusões e o desfecho. A estrutura dos capítulos da monografia está detalhada a seguir:

No capítulo I, são apresentados conceitos e definições da Engenharia de Software relevantes para a estimação. Este capítulo também apresenta conceitos importantes sobre a natureza do software que influenciam diretamente as estimativas.

No capítulo II, são apresentadas noções de engenharia de requisitos, as quais fornecem as entradas necessárias para que a atividade estimação possa ocorrer.

No capítulo III, os modelos de processos de desenvolvimento e gerenciamento são brevemente revisados. A estimação é uma atividade do processo executado em um

determinado projeto. Os processos de desenvolvimento e gerenciamento devem fornecer as ferramentas adequadas para adaptações e transparência para medições

No capítulo IV, são apresentados os conceitos gerais de estimação de software, além de um conjunto de diferentes técnicas utilizadas.

No capítulo V, um estudo de caso ilustra a atividade de estimação em um projeto de modificação de software.

O capítulo VI contém as conclusões do presente estudo. São ainda propostos trabalhos futuros cujos desenvolvimentos dariam prosseguimento e maior profundidade à discussão do assunto.

CAPITULO I

NOÇÕES DE ENGENHARIA DE SOFTWARE

1.1 Introdução

Neste capítulo são apresentadas definições de termos utilizados neste estudo com o intuito de nivelar o entendimento sobre os mesmos. Embora tais conceitos possam ser considerados de senso comum, é certo que existem visões distintas e conflitantes.

Os próximos tópicos revisarão alguns conceitos da Engenharia de Software pertinentes ao tema.

1.2 Definição de aplicativo de software

Aplicativos de software são “programas isolados que resolvem uma necessidade específica do negócio” (Pressman, 2006, p. 4). Exemplos disso são softwares que processam dados comerciais ou técnicos que facilitam as operações e a gestão de um negócio. Além do

código-fonte, o software inclui toda a documentação e os dados necessários para que o programa funcione corretamente (Sommerville, 2003, p. 5).

1.3 Definição de Engenharia de Software

A Engenharia de Software é a disciplina que trata de todos os aspectos do desenvolvimento de um software, incluindo as atividades de engenharia de requisitos, os modelos de processos e os modelos e técnicas de estimação (Sommerville, 2003, p. 6-7).

Um aplicativo de software é desenvolvido através de um processo. Não é algo que se fabrica a partir de matéria prima, nem é montado a partir de partes menores. Segundo Pressman (2006, p. 4), o software apresenta esta característica especial em relação a outros tipos de produtos, ou seja: ele não é fabricado no sentido clássico, mas desenvolvido, passando por um processo de engenharia.

A Engenharia de Software fornece abordagens sólidas para aumentar as chances de que os objetivos de negócio sejam atingidos em termos de prazo, qualidade e funcionalidades. Segundo Campos (2009, p. 2), as organizações enfrentam hoje o desafio de desempenhar suas atividades de forma produtiva, com qualidade e cumprindo o planejamento estratégico. Portanto, o uso de uma abordagem adequada no desenvolvimento de um software para elicitação de requisitos, estimação, desenvolvimento e controle é fundamental para as organizações.

1.4 Definição de projeto de software

Um projeto consiste num empreendimento temporário, cujo objetivo é a criação de um produto ou prestação de um serviço. Um projeto de software consiste no desenvolvimento de um software, incluindo os artefatos relacionados.

Independente do modelo de processo adotado, o empreendimento de construção de um software envolve atividades de diversas áreas de conhecimento utilizadas em maior ou menor grau durante as fases do projeto, nos âmbitos de gerenciamento e desenvolvimento.

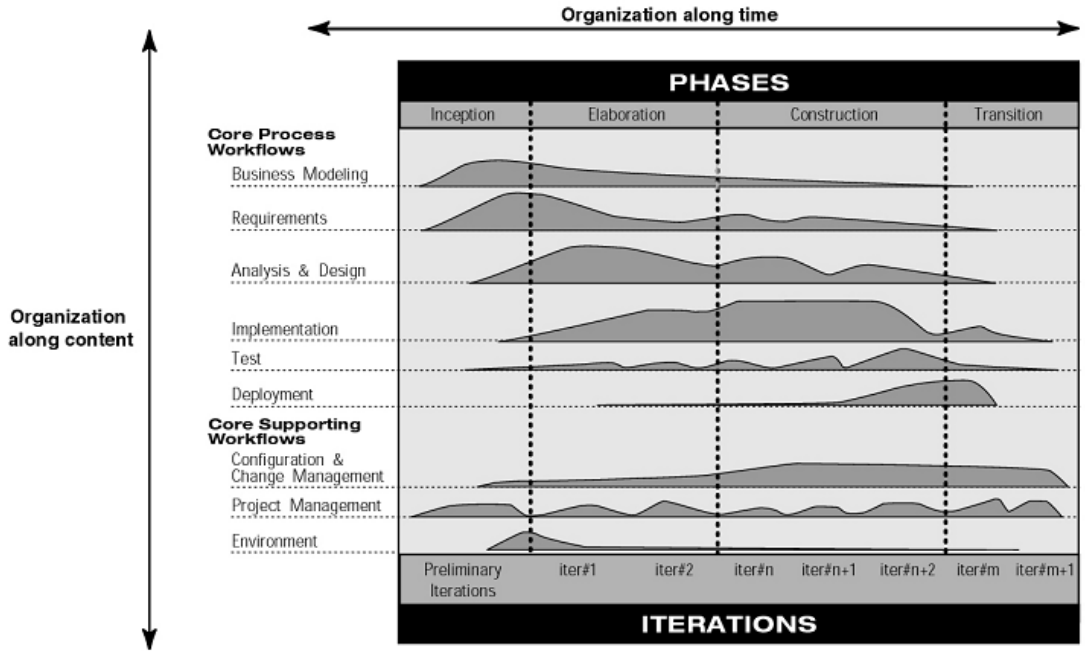

Figura 1 - Visão geral de um modelo iterativo (Rational, 2001)

O Processo Unificado (Rational, 2001), por exemplo, classifica as atividades da Engenharia de Software em nove disciplinas, dentre as quais cinco são relacionadas diretamente ao produto de software e três ao controle e gerenciamento, isto é, atividades de suporte ao desenvolvimento (Figura 1). No decorrer do projeto, as disciplinas demonstram um maior ou menor grau de atividade e nota-se que o gerenciamento do projeto é a única disciplina utilizada com certa regularidade no decorrer do tempo.

As estimativas de software são o fundamento para o planejamento do projeto ao permitirem uma visão geral do esforço necessário para o desenvolvimento e de variáveis que influenciam o projeto positivamente ou negativamente, tais como a produtividade da equipe e a complexidade do domínio.

1.5 Noções sobre arquitetura de software

Antes do desenvolvimento de um software é necessário definir sua arquitetura. O projeto da arquitetura é realizado através da decomposição do software em componentes. A arquitetura descreve o papel dos componentes que compõem o software e o relacionamento eles (Sommerville, 2003, p. 182).

A arquitetura do software fornece um framework estrutural básico para o desenvolvimento do software. Os diversos componentes do sistema agrupam elementos similares, tal como objetos com comportamento semelhante, e facilitam a estruturação do

software. Dessa forma, é possível determinar com facilidade quais partes do sistema são afetadas pela implementação ou alteração de uma funcionalidade, pois os tipos de componentes e objetos estarão previamente definidos.

A decomposição arquitetural fornece um ponto de partida para técnicas de estimação baseadas em objetos ou elementos do software. A estimação é possível ao contar os elementos de um componente e estimar cada componente dessa forma. A estimativa resultante tende a ter mais qualidade devido ao conhecimento do tipo de elemento do componente em relação a estimativas geradas sem uma arquitetura definida.

Além disso, a arquitetura influencia diretamente na complexidade do software (Pressman, 2006, p. 223). Quanto mais dependências compartilhadas de recursos, tal como banco de dados ou arquivos, e dependências entre os componentes que compõem o software, maior será a sua complexidade. Um grande número de interdependências faz com que qualquer alteração cause maiores impactos em outros componentes.

Através da arquitetura é possível analisar o impacto de modificações no software. Isso é feito considerando as dependências entre os componentes. Além disso, ao desenvolver uma nova funcionalidade, é mais fácil identificar com antecedência os elementos dos componentes que serão necessários desenvolver, modificar ou reusar.

A análise de impacto de alterações é importante para o ajuste das estimativas dos componentes afetados quando se deseja obter uma posição atualizada do projeto.

1.6 Barreiras do desenvolvimento e estimação de software

A atividade de estimação consiste em tentar antecipar o tamanho ou esforço de desenvolvimento de um produto abstrato. Existe uma série de desafios que fazem com que o software seja difícil de estimar e, consequentemente, o projeto difícil de planejar.

Por um lado, isso se deve a características intrínsecas ao software. Brooks (1987) afirma que, por definição, o software é complexo e irredutível, ou seja, não é possível simplificá-lo sem que haja perda de informação. Ele apresenta quatro características importantes que contribuem para isso, tornando o software essencialmente difícil de construir:

- **Complexidade**: entidades de software são extremamente complexas para seu tamanho e não existem duas partes iguais no nível de algoritmo. Isso faz parte da natureza do software, não ocorrendo por acaso. A escalabilidade de um software, por exemplo, não consiste simplesmente em fazer suas partes maiores ou menores, diferentemente de construções físicas realizadas pelo homem.

- **Conformidade**: apesar de existirem áreas onde pessoas lidam com alta complexidade, o software possui algumas complicações adicionais. Por exemplo, um físico lida com a complexidade do átomo, mas este possui uma interface "bem definida" pela natureza, ou seja, os átomos possuem uma conformidade. O software tem diversas interfaces diferentes definidas por diversas pessoas e geralmente há pouca ou nenhuma conformidade nestas definições.
- **Mutabilidade:** o software parece estar sempre pressionado a mudar. Produtos fabricados como pontes, carros e máquinas também estão, mas geralmente eles são substituídos por novos modelos, não sendo modificados com a mesma facilidade de um software. Eles são produtos tangíveis, com escopo bem definido e o custo de modificações maiores é impeditivo. Um software, por ser abstrato e com baixos custos diretos para se mexer, muitas vezes sem aparentes consequências imediatas, sofre constantes mudanças. Além disso, a maioria das mudanças tem como objetivo fazer com que o software vá além de seus limites iniciais.
- **Invisibilidade ou intangibilidade**: apesar de existirem interfaces e linguagens amigáveis, o software não pode ser visualizado através de imagens. Em geral, desenvolvedor e usuário não conseguem enxergar o software como um todo. Isso ocorre porque os usuários geralmente não compreendem as questões tecnológicas e os desenvolvedores não entendem todas as regras do negócio, além do que um software pode ser desenvolvido por diversos indivíduos especialistas em diferentes áreas. Portanto, como cliente e desenvolvedor não conseguem ter em mente o software por completo, as chances de erros aumentam.

A estimação é realizada com base numa abstração em forma de requisitos. Por definição, ela não pode ser exata ao considerar as características apresentadas acima, pois a estimativa é, na verdade, baseada numa ideia pré-concebida do software.

Mesmo medidas do produto final de software são subjetivas. Não existem métricas objetivas de tamanho, qualidade, eficiência, robustez, usabilidade e tantos outros aspectos. Mesmo o número de linhas de código ou caracteres do código fonte não possuem uma relação direta com as funcionalidades e características do software, embora muito esforço tenha sido empreendido em conseguir uma aproximação razoável. Isso implica em que qualquer

comparação entre softwares, processos de desenvolvimento, técnicas estimação e de engenharia de requisitos somente podem ser realizadas através de critérios específicos e objetivos. Portanto, não é possível afirmar categoricamente que uma técnica, método ou modelo seja superior a qualquer outro.

Por outro lado, o fator humano acrescenta ainda mais dificuldades no desenvolvimento e na estimação de um software. Enquanto desenvolvedores e arquitetos pensam em nível técnico e funcional, os gerentes observam o cronograma e os custos. Não é tarefa fácil conciliar ambas as perspectivas, sendo um desafio alinhar os objetivos estratégicos de uma organização com os aspectos técnicos e abstratos de um produto de software.

Do ponto de vista de quem está gerenciando o projeto, prazo e cronograma são fundamentais para que decisões sejam tomadas e estratégias de negócio estabelecidas, portanto boas estimativas são necessárias. Isso pode ser um problema do ponto de vista da equipe de desenvolvimento, por exemplo, quando prazos arbitrários são definidos ou tempo e recursos disponíveis são tecnicamente insuficientes.

Portanto, tanto a natureza do software quanto os fatores humanos devem ser considerados na estimação. Da mesma forma, quando há alterações no contexto do projeto de software, tal como mudança nos requisitos, no domínio ou na equipe, as estimativas são afetadas e devem ser ajustadas.

1.7 Conclusão

Este capítulo definiu o tipo e a natureza da entidade de software, além de descrever brevemente alguns aspectos relacionados ao processo de desenvolvimento.

Sistemas de software são entidades complexas e difíceis de desenvolver. Por isso, a Engenharia de Software busca mitigar os riscos e diminuir as falhas em projetos de desenvolvimento software através de modelos de processos, técnicas e boas práticas.

Entretanto, a correta estimação de um software inclui compreender o que exatamente deve ser construído e como isso será realizado. Assim, os próximos capítulos apresentam abordagens da Engenharia de Software para a engenharia de requisitos e para o desenvolvimento e gerenciamento do projeto.

CAPITULO II

NOÇÕES DE ENGENHARIA DE REQUISITOS

2.1 Introdução

Este capítulo apresenta noções sobre a engenharia de requisitos. Os requisitos são a base para a atividade de estimação do software, pois fornecem a abstração necessária do problema para que os engenheiros software executem a devida análise do problema.

No contexto de uma organização, um requisito normalmente tem sua origem em uma necessidade de negócio. A ideia de se desenvolver um software surge quando há um conjunto de necessidades específicas a serem atendidas, as quais devem ser traduzidas em requisitos.

Este processo inclui uma série de dificuldades e desafios. Problemas de comunicação, entendimento e complexidade são alguns dos fatores que contribuem para isso.

Os próximos tópicos apresentam diferentes abordagens e conceitos sobre os requisitos que causam impacto direto na estimação de um software.

2.2 Definição de requisito

Sommerville (2003, p. 82) identifica duas visões distintas sobre o termo requisito. Este pode ser “uma visão abstrata, de alto nível, de uma função que o sistema deve fornecer ou de uma restrição do sistema” ou, em outro extremo, “uma definição detalhada, matematicamente formal, de uma função do sistema”.

O autor ainda destaca diferentes níveis de detalhamento dos requisitos:

- **Requisitos do usuário**: declarações em linguagem natural sobre as funções que o sistema deve fornecer.
- **Requisitos do sistema**: detalhamentos das funções e restrições do sistema, podendo ser usados no contrato de desenvolvimento de software.
- **Especificação de projeto de software**: descrição abstrata do projeto de software, acrescentando mais detalhes sobre a solução aos requisitos do sistema.

Os requisitos são a base para o desenvolvimento de uma solução para o problema. Com o entendimento dos requisitos num determinado momento do projeto se constrói o respectivo modelo de análise. A estimação do software pode ser feita com nesse modelo, que contém os elementos da solução do problema que serão desenvolvidos.

O nível de detalhamento e a maneira formal ou abstrata de especificação dos requisitos é uma escolha a ser feita de acordo com o projeto. Porém, cedo ou tarde, uma especificação matematicamente formal dos requisitos terá que ser criada como parte do programa de computador, seja através de documentação ou diretamente no código-fonte. A compreensão sobre o problema e a experiência da equipe no tipo de problema são critérios que podem definir essa questão. A necessidade de estimar em maior ou menor grau de detalhe também influencia na decisão.

2.3 Tipos de requisitos

Os requisitos podem ser classificados em funcionais e não funcionais (Sommerville, 2003, p. 83). Requisitos funcionais são aqueles ligados diretamente às funções ou reações que o sistema deve apresentar. Requisitos não funcionais são restrições impostas ao funcionamento do sistema, tais como restrições de tempo, de orçamento, do processo de desenvolvimento, políticas organizacionais, padrões que devem ser adotados, entre outros.

Existem ainda requisitos de domínio, ou seja, aqueles originados do domínio da aplicação, podendo estes ser funcionais ou não funcionais.

A distinção entre os tipos de requisitos é relevante no sentido de contribuir para o gerenciamento de prioridades. Se não há tempo ou recursos suficientes para desenvolver todos os requisitos, geralmente priorizam-se os requisitos funcionais.

Os requisitos funcionais são os componentes principais das estimativas. Em geral, eles são mapeados diretamente a funcionalidades ou elementos do sistema de software. Em decorrência disso, a qualidade dos requisitos funcionais tem grande efeito sobre a qualidade das estimativas.

Requisitos não funcionais afetam as estimativas em graus diferentes, dependendo da natureza de cada um. As técnicas e modelos de estimação possuem fatores de ajuste e calibragem que incluem restrições técnicas, complexidade do problema, ferramentas de desenvolvimento, etc. Portanto, esses tipos de requisitos também afetam a qualidade das estimativas.

2.4 Mutabilidade dos requisitos

Os requisitos de software não podem ser definidos estaticamente no decorrer do projeto. A definição de um requisito representa um instantâneo, mas não as mudanças que ele sofre no decorrer do tempo. Pressman (2006, p. 39) afirma que “hoje em dia, o trabalho de software é em ritmo rápido e sujeito a uma torrente sem fim de modificações (de características, funções e conteúdo da informação)”.

McConnell (1996) afirma que, em média, os requisitos mudam 25% desde a fase em que deveriam estar definidos até a primeira versão do software. Essa estimativa demonstra a natureza mutante dos requisitos e consiste num dos motivos pelos quais o software e as estimativas também não são estáticos.

2.5 Engenharia de requisitos

A engenharia de requisitos é uma atividade da Engenharia de Software que busca o entendimento do sistema a ser construído (Pressman, 2006, p. 117). Ela não é uma ciência exata, mas oferece uma abordagem sólida para enfrentar os desafios de comunicação e complexidade encontrados no início do projeto de software.

Além disso, os requisitos devem ser identificados e gerenciados no decorrer do projeto de forma que reflitam as reais necessidades dos clientes. O entendimento do sistema deve aumentar com o passar do tempo e os requisitos devem refletir este aprofundamento.

Pressman (2006, p. 117) afirma que é importante identificar corretamente os requisitos durante as fases de análise e modelagem. Ele contradiz os autores que defendem a prática de iniciar o desenvolvimento do software precipitadamente. Segundo esses autores, não valeria a pena gastar tempo com os requisitos iniciais devido às inevitáveis mudanças, de modo que as iterações iniciais tornariam as necessidades do cliente mais claras. Pressman afirma que a complexidade que surge no decorrer do projeto logo derruba esses argumentos.

Pressman (2006, p. 118) define as seguintes fases para a atividade de engenharia de requisitos:

- **Concepção**: fase inicial onde os engenheiros perguntam uma séria de questões livres de contexto para estabelecer um entendimento básico do problema e alguns fatores que influenciam no projeto.
- **Levantamento**: são feitas perguntas específicas sobre os objetivos do sistema, a meta a ser atingida, como o software será usado no dia-a-dia e assim por diante. Nesta fase, podem surgir problemas como:
    - **Escopo**: os limites do sistema não estão estabelecidos ou o cliente entra em detalhes técnicos desnecessários que dificultam o entendimento dos requisitos ao invés de esclarecê-los.
    - **Entendimento**: o cliente e os usuários não tem certeza do que é necessário, tem pouca compreensão do ambiente computacional, não tem entendimento do domínio do problema, omitem informações, apresentam necessidades conflitantes e assim por diante.
    - **Volatilidade (ou mutabilidade)**: os requisitos mudam ao longo do tempo.
- **Elaboração**: procuram-se eliminar os problemas das fases anteriores, tentando estabelecer um modelo refinado dos requisitos que consiste num modelo de análise e contém cenários de uso e a descrição de como o usuário irá interagir com o sistema.
- **Negociação**: os requisitos propostos pelo cliente ou pelos usuários que são conflitantes, desnecessários, de alto risco ou inatingíveis por características do

projeto, domínio ou por outro motivo precisam ser discutidos e negociados com o cliente desde o início do projeto, em cada iteração.

- **Validação**: uma análise da qualidade dos requisitos é realizada para garantir que não existem mais ambiguidades, inconsistências, omissões e erros através de uma revisão técnicas pelos engenheiros, clientes, usuários e outros interessados.

A estimação relaciona-se diretamente em como as atividades de engenharia de requisitos são realizadas. Quanto mais cedo forem geradas estimativas, menos detalhes serão obtidos e a margem de erro é maior. Quanto melhores habilidades e maior a experiência dos envolvidos no processo de elicitação de requisitos, maiores são as chances das estimativas serem confiáveis.

2.6 Gestão de requisitos

Os requisitos mudam e precisam ser monitorados, rastreados e gerenciados. A engenharia de requisitos é uma atividade executada durante todo o projeto. A princípio ela é mais intensamente executada para que todos os requisitos sejam elicitados, mas provavelmente diminuirá de intensidade no decorrer do tempo na medida em os requisitos forem mais conhecidos, desenvolvidos e atenderem às expectativas dos usuários.

Porém, cada mudança de requisito pode afetar vários elementos do projeto, tais como outros requisitos, funcionalidades e interfaces do sistema. A fim de gerenciar o encadeamento de mudanças utilizam-se técnicas de rastreabilidade, ou seja, identificação dos relacionamentos do requisito com os demais elementos do projeto.

Pressman (2006, p. 121) lista alguns tipos de tabelas de rastreabilidade que podem ser usadas quando há preocupação em identificar o impacto da mudança em um determinado requisito:

- **Tabela de rastreamento de características**: mostra relação do requisito com características importantes do sistema.
- **Tabela de rastreamento de fontes**: identifica a fonte de cada requisito.
- **Tabela de rastreamento de dependência**: relaciona os requisitos uns com os outros.
- **Tabela de rastreamento de interfaces**: mostra relação do requisito com as interfaces internas e externas do sistema.

Após a mudança em um requisito e a devida atualização dos elementos afetados, as estimativas de todos esses elementos também são afetadas e devem ser atualizadas. Isso permite analisar o impacto e a viabilidade da mudança no contexto do projeto.

2.7 Atividades da engenharia de requisitos

Durante o início do processo de engenharia de requisitos, na fase de concepção, Pressman (2006, p. 122) indica alguns passos necessários para colocar o projeto em andamento:

- **Identificação dos interessados**: um interessado é "quem quer que se beneficie de modo direto ou indireto do sistema que está sendo desenvolvido" (Pressman, 206, p. 122), tais como gerentes de negócio, gerentes de produto, engenheiros de software, engenheiros de suporte. Essas pessoas fornecerão entradas para os requisitos de usuário.
- **Reconhecimento de diversos pontos de vista**: os requisitos de usuário iniciais são explorados através de questões e comparações dos diversos pontos de vista dos interessados e precisam ser compostos para formar um conjunto consistente de requisitos.
- **Trabalho em busca de colaboração**: é necessário buscar a colaboração de todos os interessados no sistema, evitando conflitos entre as partes.
- **Formulação das questões iniciais**: definir questões livres de contexto para o cliente e os demais interessados no projeto.

Depois disso, na fase de levantamento de requisitos formais, Pressman (2006, p. 124) fornece as seguintes abordagens:

- **Coleta colaborativa de requisitos**: os interessados no projeto se reúnem e discutem a necessidade e a justificativa do novo produto, cada um expõe uma lista de objetivos que consideram importantes para o sistema, os quais são discutidos e combinados para formar uma lista geral, que então seria usada para derivar os requisitos do sistema.
- **Implantação da função de qualidade**: a *Quality Function Deployment* (QFD) é uma técnica que procura traduzir as necessidades dos clientes em requisitos que enfatizam o que possui mais valor para o cliente. Esta técnica cobre todo o

processo de Engenharia de Software, mas é muito bem aplicada na engenharia de requisitos. Um ponto importante é sua classificação de requisitos em:

  - **Requisitos formais**: refletem objetivos e metas para o produto que garantem a satisfação do cliente.
  - **Requisitos esperados**: requisitos implícitos no produto, que o cliente pode não fazer deixar óbvio, mas sua falta geraria insatisfação.
  - **Requisitos excitantes**: requisitos que vão além das expectativas do cliente.

- **Cenários de usuário**: consiste na criação de um conjunto de cenários de uso que descrevam como o software será usado.

A estimação do software depende do resultado das atividades de engenharia de requisitos. Alguns produtos de trabalho comumente gerados e que podem ser usados na estimação são:

- lista de clientes, usuários e outros interessados;
- descrição do ambiente técnico do sistema;
- lista dos requisitos e restrições;
- conjunto de cenários de uso;
- protótipo desenvolvido para definir melhor os requisitos.

2.8 Modelo de análise

Segundo Pressman (2006, p. 134), “o modelo de análise é um instantâneo dos requisitos em um determinado tempo”. Quando é necessário obter uma posição atualizada do projeto ele deve ser atualizado para refletir as mudanças dos requisitos. Um modelo de análise inicial correto fornece uma base sólida para o restante do projeto, embora mudanças sejam certas.

O modelo de análise pode se basear em diferentes elementos, tais como cenários, classes, comportamentos e fluxo. Linguagens como a UML (*Unified Modeling Language*), com seus diversos diagramas, são utilizadas para descrever os diferentes elemento e aspectos do sistema (Pressman, 2006, p. 153).

Em um processo de desenvolvimento tradicional, o modelo de análise é o ponto de partida para a tarefa de estimação do projeto de software. Os elementos da análise servem como entrada para os modelos e técnicas de estimação.

2.9 Histórias de usuário

Os modelos ágeis utilizam histórias de usuário para capturar as necessidades dos clientes em um projeto de software. Uma história de usuário é uma descrição em primeira pessoa e em alto nível de uma ação que o usuário efetua no sistema (Cohn 2004).

Histórias de usuário são escritas em primeira pessoa e em linguagem simples com a finalidade de melhorar o entendimento entre a equipe de desenvolvimento e o cliente, pois uma das maiores barreiras no entendimento do problema é na comunicação (Cohn, 2004). Portanto, ao invés de documentos técnicos e formais, as histórias de usuário enfatizam a forma verbal de comunicação.

Ao escrever uma história de usuário evitam-se detalhes considerados desnecessários e inconvenientes (Koskella, 2008, p. 326). Isso significa que ela deve ser escrita em poucas palavras e evitando abranger mais de uma funcionalidade. Em alguns casos, quando uma funcionalidade é complexa por natureza, várias histórias de usuário podem refletir cenários específicos dela. Além disso, uma história muito detalhada poderia prescrever tecnicamente uma solução, isto é, influenciando demasiadamente em como o desenvolvedor irá fazer sua implementação.

2.10 Backlog

O *backlog* é uma lista do que ainda é necessário fazer para concluir o produto ou uma iteração (Schwaber, 2004). Em geral, o *backlog* é composto pelo conjunto de histórias de usuário necessárias para concluir o produto ou pelo conjunto de atividades necessárias para concluir as histórias de usuário selecionadas e estimadas para a iteração. Schwaber (2004) define os dois tipos de *backlog*:

- ***Backlog* do produto**: a lista de histórias necessárias para concluir o produto, incluindo as estimativas para cada uma delas.
- ***Backlog* do *sprint***: a lista de tarefas necessárias para completar as histórias de usuário selecionadas para a entrega do próximo incremento funcional do produto.

O *backlog* pode ser mantido em ordem de importância ou prioridade para facilitar o planejamento das próximas iterações e o controle da iteração atual.

As estimativas de conclusão da iteração, conclusão do produto e velocidade da equipe são derivadas do gerenciamento do *backlog*. O gerente ajusta as estimativas na medida em que os dados reais de tempo são coletados durante a execução das atividades.

2.11 Conclusão

A engenharia de requisitos propõe abordagens para lidar com a inevitável dificuldade em definir os requisitos de um sistema de software e construir e manter o modelo de análise, ou seja, a definição do que será construído.

Entretanto, é necessário talento, experiência e esforço para obter requisitos adequados nesse processo devido a barreiras relacionadas à comunicação, escopo e mutabilidade.

Com lista inicial de requisitos, pode ser realizada a estimação inicial do software e então criado um cronograma baseado nas funcionalidades.

Porém, o processo de desenvolvimento de software executado no projeto afetará o cronograma. O projeto não é constituído apenas de tarefas de programação sequencialmente executadas. Cada processo é composto de atividades de desenvolvimento e gerenciamento que incluem aspectos além do software em si, demandando atividades extras, dividindo o desenvolvimento em fases e iterações, fornecendo atividades de ajuste e artefatos de saída que permitam o gerenciamento do projeto. Portanto, a atividade de estimação deve considerar o processo para ajustar as estimativas geradas.

No próximo capítulo são apresentados modelos de processos de desenvolvimento de software e gerenciamento de projetos que foram propostos para organizar o desenvolvimento de software.

CAPITULO III

NOÇÕES SOBRE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

3.1 Introdução

Um sistema de software é construído através de um processo de desenvolvimento, seja ele formal ou empírico. Diversos modelos de processos surgiram ao longo do tempo, dentro os quais os principais são apresentados neste capítulo.

Ao adotar processos formais e reconhecidos aproveita-se o conhecimento e a experiência dos especialistas que os criaram. Isso inclui boas práticas, listas de verificação, atividades pré-definidas, entre outros. Existe o risco de usar indevidamente o processo, executando todas as atividades propostas sem a devida reflexão, desperdiçando tempo com o que não é importante e gerando uma falsa segurança baseada apenas no fato de se usar um processo conhecido.

Por outro lado, um processo empírico depende da experiência, habilidade e conhecimento da equipe. Profissionais experientes e maduros obtêm vantagens com a liberdade desse tipo de processo porque eles já adotam boas práticas de desenvolvimento, sabem quais atividades são necessárias para completar suas tarefas e sabem como prevenir e resolver problemas.

Existem também processos de gerência utilizados durante o desenvolvimento de software. Alguns foram criados especificamente para projeto de software e outros adaptados de modelos de gerência de projetos. Alguns modelos de desenvolvimento incluem elementos de gerenciamento e vice-versa. Em decorrência disso, podem existir lacunas num projeto que executa um processo incompleto Algumas organizações adotam processos complementares, de forma a preencher essas lacunas.

3.2 Definição de modelo de processo e processo de software

Um modelo de processo é uma abstração de um processo (Sommerville, 2003, p. 36). Esses modelos representam as abordagens utilizadas no desenvolvimento de software dentro das organizações. Com base nesses modelos, diversos processos foram propostos para o desenvolvimento de software com a finalidade de se construir um produto melhor, de menor custo e mais rapidamente.

Um processo de desenvolvimento de software consiste num conjunto de atividades e resultados associados que geram um software (Sommerville, 2003, p. 7). Em geral, os processos de desenvolvimento têm como foco os aspectos técnicos, como especificação, desenvolvimento, validação e evolução do software e devem prover transparência e flexibilidade para facilitar o gerenciamento do projeto.

O processo adotado por uma organização também é uma abstração em relação ao processo executado num determinado projeto. Em geral, o processo da organização é adaptado de acordo com as necessidades específicas do projeto.

3.3 Importância do gerenciamento de projetos

Existe a necessidade de gerenciar o processo de desenvolvimento de software através de modelos, processos, atividades e ferramentas específicos. O desenvolvimento de um software ganha sentido no contexto de um negócio e de uma organização. É importante alinhar os requisitos de negócio com o produto de software e gerenciar as atividades de

desenvolvimento, verificando prazo, custo e qualidade para que o projeto não termine em fracasso do ponto de vista do negócio (Sommerville, 2003, p. 60).

Os processos de desenvolvimento podem incluir atividades de gerenciamento, tal como o Processo Unificado (Rational, 2001), mas existem modelos e processos específicos para gerenciamento de projetos. É possível adotar uma combinação de processos complementares de acordo com as necessidades do projeto e da organização.

A estimação de software é importante para o gerenciamento de um projeto. Decisões como o cancelamento de um projeto podem ser tomadas com base em estimativas de custo e prazo necessários para desenvolver um determinado sistema de software. Estimativas adequadas resultam em decisões gerenciais acertadas, enquanto estimativas irreais causam prejuízo.

Além disso, dados do esforço real comparado com o esforço estimado fornecem uma importante ferramenta para ajuste das estimativas do projeto corrente e nos projetos futuros. A experiência e os dados agregados através desta comparação permitem estimar as atividades futuras com maior acurácia e atuar mitigando riscos de descumprimento de prazos acordados. Para isso, o gerente deve possuir os registros organizados das atividades concluídas.

3.4 Classificação dos modelos em relação à burocracia e à iteratividade

A estimação deve considerar o quanto o processo onera as atividades de desenvolvimento. Quanto mais atividades extras, documentação e rigor forem exigidos, maior será o fator de ajuste necessário para as estimativas.

Este estudo considera modelos mais burocráticos e com maior rigor como modelos tradicionais. Modelos com pouca burocracia, também chamados de empíricos, incluem os modelos de processos ágeis.

Kroll (2006) classifica alguns modelos num gráficos cujo eixo horizontal é o grau de disciplina adotado e o eixo vertical o nível de iteratividade. Na Figura 2, o autor posiciona o modelo de maturidade de processo CMM com grau elevado de burocracia e baixa iteratividade, enquanto os modelos ágeis com pouca burocracia e alta iteratividade. O CMMI, uma evolução do CMM, é mais iterativo e menos burocrático. Já os processos ágeis são bastante iterativos e muito pouco burocráticos.

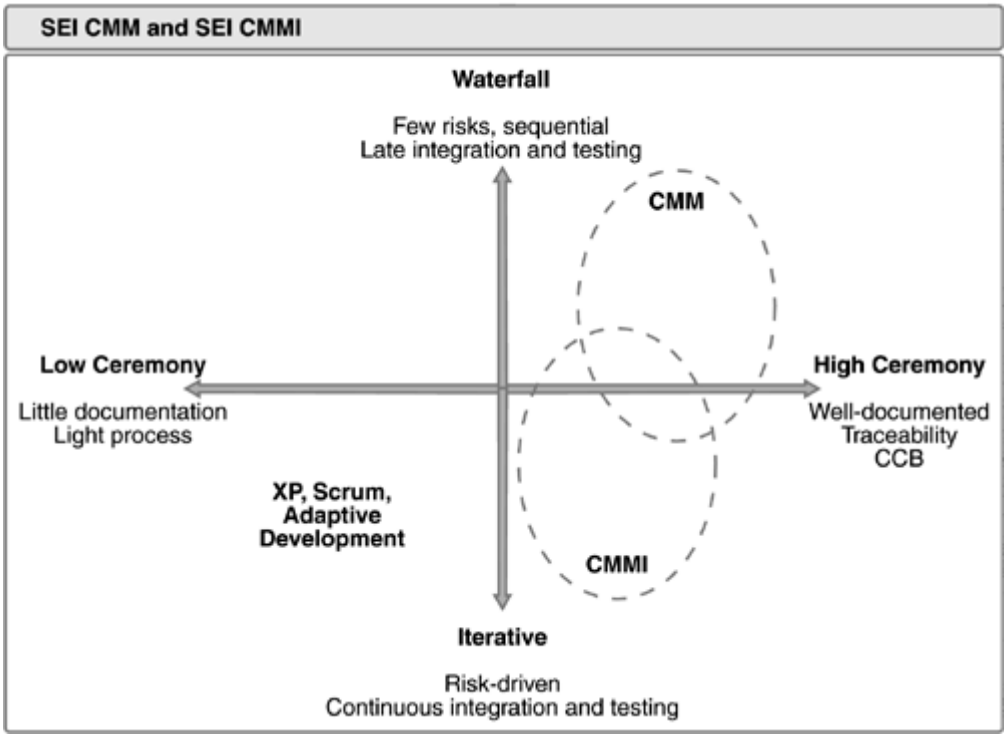

Figura 2 – Grau de burocracia de processos tradicionais (Kroll, 2006, p. 36).

A Figura 3 apresenta a mesma ideia, incluindo agora o Processo Unificado. Devido a suas características de adaptação, o Processo Unificado abrange uma grande faixa do gráfico. Dependendo do projeto, ele pode ser adaptado para ser mais ou menos iterativo e disciplinado.

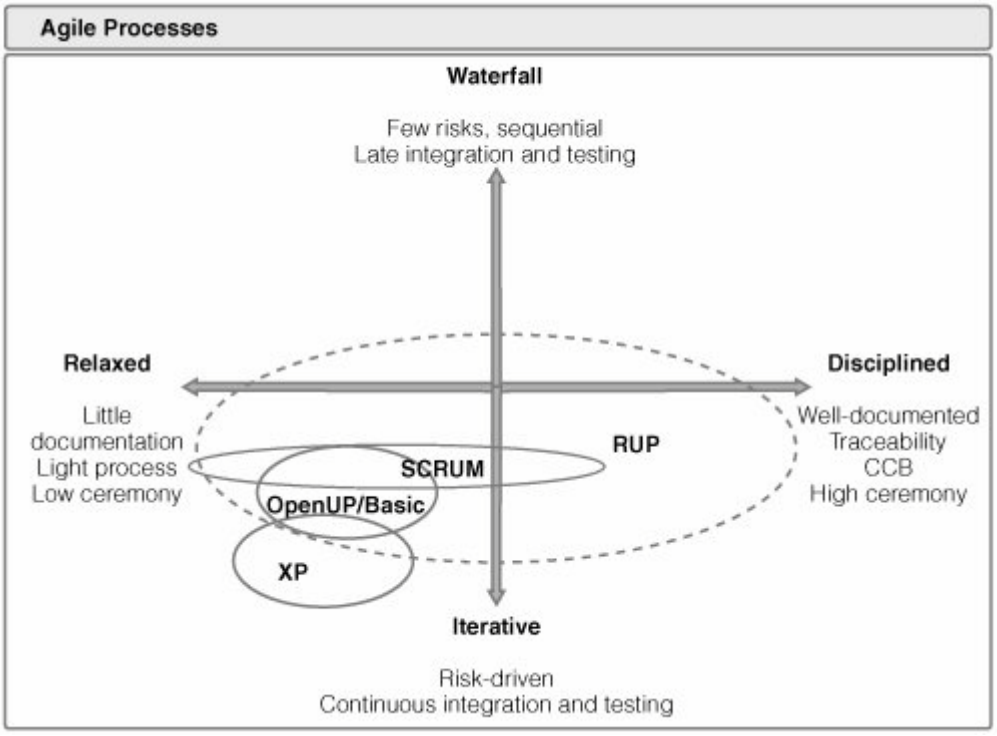

Figura 3 – Grau de burocracia dos processos ágeis (Kroll, 2006, p. 45).

3.5 Modelos de desenvolvimento e gerenciamento de software

Os modelos de desenvolvimento de software são abstrações das abordagens de desenvolvimento utilizadas nas organizações. Eles podem ser aplicados em diferentes processos individualmente, combinados e com variações.

Os modelos afetam diretamente a forma como o projeto é gerenciado. Os modelos iterativos, por exemplo, definem que haverá atividades de planejamento em cada iteração ao longo do desenvolvimento.

A compreensão dos princípios, benefícios e desvantagens desses modelos auxilia no desenvolvimento e na estimação de um software ao fornecer uma visão geral da abordagem utilizada no projeto.

3.5.1 Modelos incrementais

Os modelos incrementais englobam os modelos de desenvolvimento cuja constituição é de pequenos ciclos de desenvolvimento realizados de forma iterativa, ou seja, a cada ciclo novos incrementos são adicionados no software (Pressman, 2006, p. 40), que ganha funcionalidades no decorrer do projeto. Em cada iteração é produzida uma versão parcial funcional do software.

Este modelo possui vantagens em relação a um modelo sequencial de desenvolvimento. Os incrementos podem ser planejados para gerir os riscos técnicos, uma boa prática adotada por processos modernos (Campos, 2009). Ele também absorve melhor as mudanças nos requisitos, principalmente quando alguns deles ainda não foram claramente entendidos.

Dependendo das funcionalidades a serem desenvolvidas em cada incremento, atividades de gerenciamento devem realizar o respectivo planejamento, estimação e negociação dos requisitos para cada iteração.

Entretanto, o modelo incremental possui alguns problemas. Algumas funcionalidades dependem de outras, podendo ser interdependentes, por isso pode haver bloqueios no desenvolvimento. Além disso, alterações em requisitos já desenvolvidos invalidam o cronograma e as estimativas.

3.5.2 Modelos evolucionários

Nos modelos evolucionários o software é ajustado, melhorado e agrega novas funcionalidades, tornando-se mais completo, a cada ciclo de desenvolvimento (Pressman,

2006, p. 42). Sistemas de software precisam se adaptar com o passar do tempo. Não há como forçar um desenvolvimento linear até o produto final. A evolução gradual do produto é uma abordagem para solucionar esse problema.

Estes modelos também são iterativos, porém diferem dos modelos incrementais porque acomodam melhor situações onde apenas os requisitos básicos são entendidos, mas os detalhes somente serão conhecidos posteriormente.

Entretanto, nos modelos evolucionários é difícil estimar e planejar a quantidade de iterações necessárias para construir o produto completo, pois a maioria das técnicas de gestão e estimativa de projeto é baseada na disposição linear das atividades (Pressman, 2006, p.47).

A estimação e o planejamento em projetos que adotam o modelo evolucionário devem ser constantemente revisados na medida em que mudanças nos requisitos são detectadas. Na medida em que o projeto evolui, se houver gerenciamento adequado, os ajustes nas estimativas e o replanejamento provavelmente irão convergir para resultados mais próximos da realidade.

3.5.3 Modelos ágeis

O “Manifesto para o Desenvolvimento Ágil de Software” (Agile Manifesto, 2009), assinado no ano de 2001 por alguns proeminentes desenvolvedores de software, deu início um movimento emergente que busca formas de desenvolvimento de software mais ágeis. Este documento enfatiza alguns princípios já conhecidos com o objetivo de superar os desafios modernos do desenvolvimento de software (Pressman, 2006, p. 58).

Os princípios fundamentos dos modelos ágeis são:

- indivíduos e interações em vez de processos e ferramentas;
- softwares funcionando em vez de documentação abrangente;
- colaboração do cliente ao invés de negociação de contratos;
- resposta a modificações em vez de seguir um plano.

Os modelos ágeis procuram estabelecer apenas um conjunto mínimo de organização e disciplina, deixando as demais decisões a cargo da equipe. Há uma suposição de que uma equipe com experiência e diversidade de conhecimentos saberia como coordenar o seu trabalho e se auto-organizar, portanto qualquer tipo de burocracia inibiria a plena capacidade dos indivíduos. Os “agilistas” também defendem que não basta apenas incluir alguns ajustes e boas práticas nos modelos tradicionais, é preciso livrar-se da “roupagem velha”.

O objetivo dos modelos ágeis em geral não é solucionar definitivamente os desafios da Engenharia de Software, mas prover o ambiente mais adequado para o desenvolvimento de software.

Além disso, os defensores e praticantes dos modelos ágeis, chamados “agilistas”, defendem a ideia de que os processos ágeis são os mais adequados para responder às altas taxas de mudanças de requisitos decorrentes da dinâmica dos negócios da atualidade.

Em contrapartida a todos os benefícios dos modelos ágeis, a correta aplicação dos diversos processos ágeis necessita de uma equipe experiente e de indivíduos capacitados e motivados. Além disso, projetos de softwares com alto grau de complexidade exigirão documentação detalhada. Tanto o cliente como os engenheiros de software podem criar barreiras técnicas e pessoais para aceitar o modo de trabalho estabelecido por alguns processos ágeis. Problemas nestes e em outros aspectos podem levar o projeto ao fracasso.

Os modelos ágeis enfatizam que a estimação das histórias de usuário e respectivas tarefas deve ser feita pela equipe de desenvolvimento, pois quem efetivamente executa o trabalho seria mais capacitado a estimá-lo do que pessoas que podem nem estar envolvidas com o projeto. Porém, delegar essa função exigirá organização, experiência e habilidade por parte da equipe. Além disso, os próprios agilistas reconhecem o fato de que desenvolvedores tendem a gerar estimativas otimistas, sendo necessário ao gerente ajustá-las posteriormente (Astels, 2002, p.70).

Em geral, adotar um modelo ágil não implica em restrições quanto à técnica de estimação, desde que a mesma atenda os princípios ágeis. Contudo, técnicas que envolvam a equipe como um todo e enfatizem a comunicação, tal como o *Planning Poker* (ver capítulo IV), são mais recomendadas.

3.5.4 Modelagem ágil

A modelagem ágil consiste em um conjunto de princípios consistentes com a filosofia dos modelos ágeis de desenvolvimento. Ela não é uma técnica ou método em si, mas estabelece uma filosofia para nortear a modelagem do sistema de software.

Adotar um processo ágil não significa que não haverá documentação, mas que esta deve ser produzida de acordo com os princípios ágeis de desenvolvimento. Ambler (2002) apresenta alguns princípios da modelagem ágil na tabela abaixo.

Tabela 1 – Princípios da Modelagem Ágil (Ambler, 2002)

Modelar com uma finalidade	Somente criar diagramas, documentação e especificações se houver uma meta específica para isso.
Usar modelos múltiplos	Escolher algumas das múltiplas formas de modelagem existentes que sejam necessárias e representativas.
Conservar apenas o que for necessário	Atualizar os modelos no decorrer do projeto conforme as mudanças nos requisitos gera um trabalho considerável, então somente o que for realmente relevante deve ser conservado.
O conteúdo é mais importante que a representação	A preocupação deve ser em transmitir a ideia e não com os formalismos de um modelo, isto é, as informações contidas nos artefatos devem ser apenas as suficientes para a situação atual.
Conhecer os modelos e ferramentas	Saber como usar corretamente os modelos e ferramentas ajuda na decisão da forma como modelar um problema. Os diversos diagramas possuem características e limitações que são descartados conforme o uso.
Adaptar localmente	A modelagem deve ser adaptada às necessidades do projeto e da equipe.

É possível aplicar esses princípios em qualquer modelo de desenvolvimento, mas uma contribuição que os processos ágeis trouxeram foi a ênfase em deixar de usar o “processo pelo processo”, ou seja, usar o processo apenas como um meio para atingir os objetivos da organização e não como um fim em si.

3.5.5 Prototipagem

A prototipagem é uma técnica que pode ser aplicada em modelos iterativos. Ela consiste em produzir uma iteração inicial do software baseada em requisitos de alto nível a fim de testar a viabilidade do projeto e a satisfação do cliente (Pressman, 2006, p. 42). Ela também serve como um mecanismo para identificação dos requisitos, pois permite uma verificação antecipada do que está sendo produzido.

Tudo o que foi produzido deveria ser descartado, pois a construção inicial rápida é geralmente desorganizada e sem qualidade. O custo de corrigir as funcionalidades e a arquitetura de um protótipo pode superar um novo desenvolvimento. A linguagem de programação e as ferramentas de prototipagem podem não ser ideais para o ambiente de produção. Entretanto, a perda pode ser minimizada incluindo o descarte no planejamento.

Existem riscos nesta abordagem. O cliente pode não entender que o protótipo deve ser descartado. Além disso, partes do sistema construídas de forma ineficiente podem permanecer no produto final.

A prototipagem aumenta a qualidade das estimativas quando ela atinge o objetivo de esclarecer os requisitos. Esta técnica é importante quando o domínio é complexo ou há dificuldades de comunicação com os usuários.

3.6 Processos de desenvolvimento de software

3.6.1 Modelo Waterfall

O modelo Waterfall é um modelo sequencial (ver Figura 4) de desenvolvimento que, em tese, funcionaria bem quando os requisitos fossem bem conhecidos e poucas mudanças fossem esperadas.

Figura 4 - Adaptado de Pressman (2006, p. 39).

Esse modelo é criticado por apresentar alguns problemas (Pressman, 2006, p. 39). Primeiramente, projetos reais raramente são sequenciais. Ajustes realizados gerariam um tipo de iteração confusa em meio às fases do modelo. Além disso, os requisitos definidos com antecedência sofrerão mudanças difíceis de absorver. Por último, o cliente somente recebe algo executável ao fim do ciclo. Pressman cita o trabalho de Bradac, que verificou que a linearidade deste modelo leva a bloqueios frequentes, ou seja, todos os participantes do projeto precisam esperar todos os demais completarem a fase atual antes de iniciar a próxima fase.

O gerenciamento de um projeto com o modelo Waterfall é mais difícil que em outros modelos. Como o planejamento, que inclui a estimação, é realizada apenas no início, não há muitos pontos de verificação e ajuste. O ajuste das estimativas exigiria atividades de planejamento durante o projeto, o que não é previsto no modelo.

3.6.2 Modelo Spiral

O modelo Spiral é um modelo orientado a riscos cujas iterações iniciais consistem em modelos de papel ou protótipos do software (Pressman, 2006, p. 44). Nas iterações posteriores

são produzidas versões do sistema cada vez com mais funcionalidades. Um replanejamento é realizado a cada iteração levando em conta o retorno obtido do cliente.

Esse modelo é um dos mais adaptáveis e gerenciáveis, pois permite absorver com mais facilidade as mudanças nos requisitos de software ao longo do tempo devido à sua natureza iterativa e incremental. Além disso, as iterações podem ser estendidas além da entrega do produto final, incluindo novas versões do produto, finalizando apenas quando o produto for retirado de uso.

3.6.3 Processo Unificado

Segundo Kroll (2003, p. 32), o Processo Unificado é uma abordagem iterativa, centrada na arquitetura e dirigida por casos de uso. Este processo de gerenciamento e desenvolvimento de software foi uma tentativa de combinar as melhores características dos modelos de desenvolvimento (Pressman, 2006, p. 51).

O Processo Unificado enfatiza boas práticas de desenvolvimento (Kroll, 2003, p. 151), o que pode aumentar a qualidade das estimativas. Mitigar os riscos arquiteturais considerados mais impactantes o mais cedo possível através de provas de conceito, por exemplo, ajuda na compreensão do tamanho real do problema. Dessa forma, evita-se a falsa impressão do bom andamento do projeto enquanto os maiores desafios são deixados por último, os quais provavelmente irão invalidar qualquer estimativa. Além disso, se os riscos forem difíceis de superar, o custo de cancelamento do projeto não é tão grande como em fases mais avançadas.

Na verdade, esse processo funciona como um *framework* que permite a composição de processos específicos. Ele define um conjunto de atividades e produtos de trabalho que podem ser incluídos no processo de uma organização, de acordo com suas necessidades e de seus projetos. As atividades e os produtos de trabalho podem variar em quantidade, tamanho e detalhamento conforme o nível de gerenciamento desejado, além de outros fatores. O grau de burocracia pode ser ajustado de acordo com as necessidades do projeto.

O Processo Unificado pode ser adotado em pequenos e grandes projetos. Kroll (2003, p. 88) descreve aplicações do processo em projeto de apenas uma pessoa até projetos distribuídos com centenas de desenvolvedores, sendo o nível de burocracia em cada tipo de projeto compatível com a necessidade.

3.6.4 Extreme Programming

Beck (1999) publicou um trabalho que deu origem ao *Extreme Programming* (XP), um processo que utiliza uma abordagem de desenvolvimento orientada a objetos, iterativa e

incremental (Novak, 2002). Ele é composto algumas fases que ocorrem a cada iteração (Pressman, 2006, p. 63).

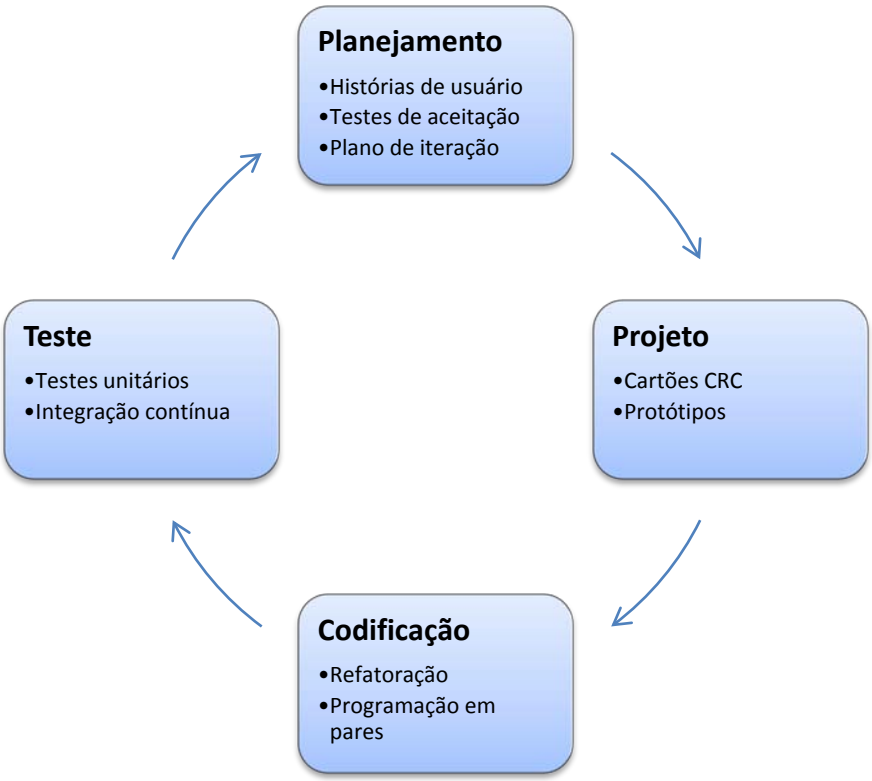

Figura 5 - Adaptado de Presman (2006, p. 64)

Na fase de planejamento cria-se um conjunto de histórias escritas pelo cliente em cartões de indexação, incluindo um valor associado ao negócio. O custo de cada história é definido com base nas avaliações dos desenvolvedores, conforme os princípios do modelo ágil. Algumas histórias são selecionadas para serem desenvolvidas na iteração atual.

A fase de *design* tem por objetivo criar uma representação simples das histórias que serão desenvolvidas na iteração. Utilizam-se preferencialmente cartões CRC (Classe-Responsabilidade-Colaborador), o único produto de trabalho que faz parte do processo. Cada história de usuário deve ter pelo menos um teste de aceitação para cada cenário identificado. O teste de aceitação de uma história de usuário tem por objetivo demonstrar que o comportamento do sistema corresponde ao esperado pelo usuário.

Durante a fase de codificação, a recomendação é não começar desenvolvendo as funcionalidades do sistema e sim codificando testes unitários para o que será desenvolvido. A codificação é realizada com a recomendação da programação em pares, onde duas pessoas trabalham juntas numa estação de trabalho. O ideal seria que, ao final de cada dia, o código

produzido por cada par fosse integrado aos demais na estratégia conhecida como integração contínua.

O *Extreme Programming* incentiva a refatoração, ou seja, o processo de modificar um sistema de tal modo que ele não altere o comportamento externo do código, mas melhore a qualidade interna do software através de revisões arquiteturais, melhoria de desempenho, alterações no código para aumentar a clareza, entre outros (Astels, 2002, p. 149). É um modo disciplinado de modificar e simplificar o projeto que minimiza as chances de introdução de defeitos. Isso pode ser feito porque os testes automatizados garantem que as funcionalidades do sistema continuem consistentes.

O acompanhamento do progresso de desenvolvimento é realizado através dos testes de aceitação automatizados, embora não se descarte os testes de aceitação do usuário na entrega de uma versão. Dessa forma, o progresso é medido de acordo com uma visão aproximada à que o usuário possui sobre as funcionalidades que ele espera. Isso facilita o alinhamento dos objetivos do usuário com aqueles da equipe de desenvolvimento.

3.6.5 Scrum

O *Scrum* é um processo ágil de gerenciamento de projetos criado por Jeff Sutherland e Ken Schwaber na década de 90 (Schwaber, 2008). Ele é um processo empírico para desenvolvimento de produtos, isto é, não é restrito ao desenvolvimento de software, podendo ser aplicado em qualquer tipo de projeto cujas características do produto façam com que seu desenvolvimento não seja completamente previsível. Para que seja possível utilizar o *Scrum*, a equipe deve possuir todas as habilidades necessárias para executar as tarefas do projeto.

O *Scrum* foi desenvolvido sobre os pilares da transparência, onde os aspectos que afetam os resultados são visíveis para aqueles que gerenciam os resultados, da inspeção, onde os diversos aspectos do processo devem ser inspecionados com uma frequência suficiente para que variações inaceitáveis no processo possam ser detectadas, e da adaptação, pois se um ou mais aspectos do processo estão fora dos limites e o produto resultante for inaceitável, o gerente do projeto deverá ajustar o processo ou o material sendo processado o mais rápido possível para minimizar desvios posteriores.

Esses pilares procuram facilitar o gerenciamento do projeto. Os riscos podem ser mitigados assim que surgem e adaptações realizadas sempre que necessário. A flexibilidade evita que o processo torne-se uma obstrução para que a equipe atenda as necessidades imediatas do negócio.

Existem apenas três papéis que os membros de uma equipe Scrum podem assumir. O *Scrum Master* é responsável por garantir que o processo seja compreendido e seguido, como um gerente de projeto, o *Product Owner* por maximizar o valor do trabalho que a equipe desenvolve, representando o cliente, priorizando e explicando as funcionalidades, e o Time é todo o pessoal que executa efetivamente o trabalho.

O *Scrum Master* é responsável por manter estimativas atualizadas do projeto. As estimativas do projeto como um todo e da iteração atual são parte importante dos pilares acima descritos, pois elas são geradas com base na inspeção, possibilitam a transparência e são base para a adaptação necessária.

O *Scrum* define apenas quatro produtos de trabalho: o *Product Backlog*, que consiste em uma lista dos requisitos do produto (histórias de usuário, por exemplo), podendo ser alterado a qualquer momento do projeto; o *Burndown da Release*, um gráfico que mostra a soma das estimativas de trabalho restantes do *Product Backlog* ao longo do tempo; o *Sprint Backlog*, uma lista de tarefas que a equipe deve cumprir para gerar o próximo incremento do produto; e o *Burndown da Sprint*, um gráfico da quantidade de trabalho restante da iteração atual (*sprint*).

Os gráficos de *burndown* fornecem as estimativas atualizadas do projeto. Eles são gerados a partir do *backlog* e da composição de dados estimados e coletados da velocidade da equipe em cada atividade ou história de usuário. Através desses gráficos o gerente pode responder a qualquer momento sobre o estado da iteração (*sprint*) e do projeto (*release*). Porém o gráfico geral do release nem sempre é produzido, pois as histórias de usuário das próximas iterações podem sofrer mudanças drásticas.

As iterações do *Scrum* são dividias em seis fases, descritas na Tabela 2.

Tabela 2 – Princípios da Modelagem Ágil (Schwaber, 2008)

Reunião de planejamento da *release*	Tem o propósito de estabelecer um plano e metas para a iteração que está começando. Nessa reunião o *backlog* inicial do *release* é criado.
Sprint	É a iteração em si, que gera um novo incremento do produto contendo a implementação dos requisitos elicitados através das histórias de usuário selecionadas. A indicação da duração é de uma até quatro semanas, dependendo do nível de experiência da equipe.
Reunião de planejamento do *sprint*	Realização de planejamento da iteração.
Revisão da *sprint*	O incremento é apresentado aos clientes para obter o *feedback* e

	outras informações necessárias ao planejamento do próximo *sprint*.
Retrospectiva do *sprint*	A equipe se reúne para analisar as decisões e ações que cada um considera terem sido acertadas ou erradas, por exemplo, sobre ferramentas ou técnicas adotadas, a fim de melhorar o processo de desenvolvimento da equipe.
Daily Scrum	Reunião rápida, com duração média de 15 minutos, onde cada membro da equipe expõe resumidamente o que está fazendo e se há alguma barreira a ser superada.

A estimação inicial geral e de cada iteração ocorre nas reuniões de planejamento, enquanto o controle e as atualizações das estimativas são feitos através dos gráficos de *burndown*.

3.6.6 Combinação de *Extreme Programming* e *Scrum*

Por serem processos ágeis, *Scrum* e a *Extreme Programming* compartilham de princípios comuns e podem ser usados em conjunto. *Scrum* é um processo de gerenciamento de projetos, enquanto a *Extreme Programming* é um processo de desenvolvimento de software. Esses dois tipos de processos geralmente não são excludentes. Kniberg (2006) descreve suas experiências com os processos combinados e afirma que não há conflito entre eles.

Essa combinação é um exemplo de um processo que engloba duas esferas necessárias para o desenvolvimento de um software. Processos de gerenciamento como o *Scrum* não definem aspectos específicos do desenvolvimento de software, enquanto processos de desenvolvimento como o XP o fazem.

Quando o processo de uma organização não aborda devidamente as áreas necessárias e restam lacunas a serem preenchidas, atividades necessárias podem deixar de ser executadas ou serem feitas empiricamente, de forma marginal ao processo e sem gerenciamento.

3.6.7 Test Driven Development

O *Test Driven Development* (TDD) é um processo de desenvolvimento de software orientado por casos de testes. A partir dos cenários de uso identificados para um sistema, codificam-se testes para cada um deles. Durante o desenvolvimento, escreve-se o código necessário para obter sucesso em cada teste. Por fim, melhora-se a arquitetura e o código em

geral usando os testes para garantir que o funcionamento esperado não foi afetado (Koskela, 2008, p. 33).

Este processo procura alinhar o desenvolvimento com os requisitos de negócio através dos testes de aceitação, garantindo que o software atenderá as necessidades de negócio. Se os testes de aceitação refletem o comportamento esperado pelos usuários, então o critério de avaliação do software é o resultado da execução dos testes. Isso permite avaliar e acompanhar a qualidade do sistema de um ponto de vista semelhante ao do usuário, evitando que sejam empreendidos esforços em não conformidade com as necessidades de negócio.

O TDD pode melhorar a acurácia da estimação. Ela procura garantir que a equipe tenha o entendimento das funcionalidades através da criação precoce dos testes. Assim, como a equipe precisa que conhecer os requisitos para a criação dos testes, ela terá uma noção mais apurada do problema, o que pode levar a estimativas melhores.

3.6.8 Feature Driven Development

O *Feature Driven Development* (FDD) é um processo prático, iterativo e incremental de desenvolvimento de software orientado a objetos com foco nas características do sistema. Uma característica “é uma função valorizada pelo cliente que pode ser implementada em duas semanas ou menos” (Pressman, 2009, p. 71). A cada iteração de incremento do software algumas características que agregam valor ao produto são estimadas, selecionadas e então desenvolvidas de forma que o resultado do respectivo teste passe a ser de sucesso.

O objetivo deste processo é valorizar o produto focando as características que o cliente necessita. O foco em características facilita a compreensão do produto pelo cliente e no entendimento deste com a equipe de desenvolvimento.

O FDD pode dificultar a atividade de estimação quando as características são muito abstratas em relação ao software. A visão do produto através de características pode ser mais compreensível do ponto de vista do usuário, mas para a equipe de desenvolvimento é um desafio tentar prever o esforço de desenvolvimento necessário para uma característica descrita em alto nível.

3.7 Conclusão

Conhecer e entender os diferentes modelos e processos de desenvolvimento de software e gerenciamento de projetos é importante para uma estimação adequada. Os processos e

modelos fornecem abordagens práticas para o desafio de desenvolver software de acordo com as reais necessidades do cliente.

Há uma tendência dos processos modernos em buscar flexibilidade, menor burocracia e manter o foco no produto de negócio. Isso significa diminuir o tempo que a equipe gasta com atividades desnecessárias ou sem sentido de modo a não encarecer o produto e tornar possível que a organização coloque seu foco no negócio e não em processos, técnicas ou tecnologias.

O próximo capítulo apresenta noções de estimativas, além de técnicas e modelos de estimação. As práticas e abordagens dos processos de software apresentadas neste capítulo e as noções de engenharia de requisitos apresentadas no capítulo anterior fornecerão uma base para a correta compreensão da atividade de estimação e de sua execução dentro dos processos de desenvolvimento.

CAPITULO IV

NOÇÕES E TÉCNICAS DE ESTIMAÇÃO DE SOFTWARE

4.1 Introdução

A estimação de software é fundamental para qualquer projeto. Estimativas de custo, esforço e prazo são geralmente demandadas por clientes e o gerente do projeto precisa ter uma base para o planejamento e para tomar decisões no decorrer do projeto. A estimação também contribui para um maior entendimento do problema e provê um horizonte para a conclusão do projeto ou da iteração.

Existem diversas técnicas, abordagens e modelos de estimação. Este capítulo apresenta alguns deles com a finalidade de prover ao leitor uma visão geral sobre o tema.

4.2 Definições relacionadas à estimação

Estimar é prover uma visão do projeto clara o suficiente para que a gerência possa tomar boas decisões de como gerenciar o projeto para que o mesmo atinja seus objetivos (McConnell, 2006). Um dos propósitos da estimação é determinar se esses objetivos são tangíveis o suficiente para que o mesmo possa ser gerenciado de forma a atingir esses objetivos (Pressman, 2009, p. 519).

Boas estimativas são as que levam a decisões de negócio corretas (McConnell, 2006). A organização sofre prejuízo ao executar um projeto onde o orçamento e os esforços reais ultrapassam os valores estimados em uma determinada escala. Portanto, boas estimativas seriam aqueles que fornecessem a aproximação necessária para que o orçamento e o esforço fossem estabelecidos com uma margem adequada.

McConnell (2006) define boas estimativas do ponto de vista estatístico como aquelas que variam em 25% ou menos, durante pelo menos 75% das vezes. O critério arbitrário do autor reforça o conceito de que boas estimativas são as que proporcionam boas aproximações suficientes para que a maioria das decisões sobre projetos estejam corretas.

Uma estimativa não é uma medida, mas uma suposição sobre algo que se espera ser verdadeiro. Técnicas e modelos matemáticos podem dar a falsa impressão de acurácia, mas previsões sobre o software são sempre baseadas em julgamentos subjetivos com base em uma definição abstrata do sistema.

Estimativas não devem ser ajustadas para se adequar ao objetivo de negócio da empresa ou do cliente, tal como para atender um prazo arbitrário. O esforço de desenvolvimento e a complexidade do software não mudam devido a esses aspectos. Quando é necessário entregar um software antecipadamente, devem-se ajustar outras variáveis para que o esforço necessário seja atingido em um tempo menor.

Uma estimativa não é um compromisso (McConnell, 2006). Os executivos de uma organização geralmente querem um compromisso e um plano para alcançar um objetivo de negócio. Embora o compromisso e o plano possam ser baseados em estimativas, é importante diferenciar os conceitos.

Além disso, a estimação deve considerar diversas influências sobre o desenvolvimento de software. Ao se estimar devem ser considerados fatores relacionados ao problema, ao cliente, ao ambiente de desenvolvimento e às pessoas envolvidas, os quais influenciam diretamente no projeto de software.

Segundo McConnel, a atividade de estimação seria análoga a uma arte ao invés de um processo científico (McConnell, 2006). Embora técnicas e modelos forneçam uma abordagem consistente para a estimação, o conhecimento, a experiência e as habilidades do estimador são determinantes para a qualidade das estimativas.

4.3 Grau de incerteza das estimativas

As estimativas possuem grau de incerteza variável no decorrer do projeto (McConnell, 2006). Elas são incertas por natureza (Pressman, 2009, p. 520), mas a incerteza pode diminuir ao passo em que medidas do projeto são coletadas e o problema conhecido mais detalhadamente. Este pensamento é exemplificado na Figura 6, onde Cohn (2006, p. 4) apresenta o Cone da Incerteza.

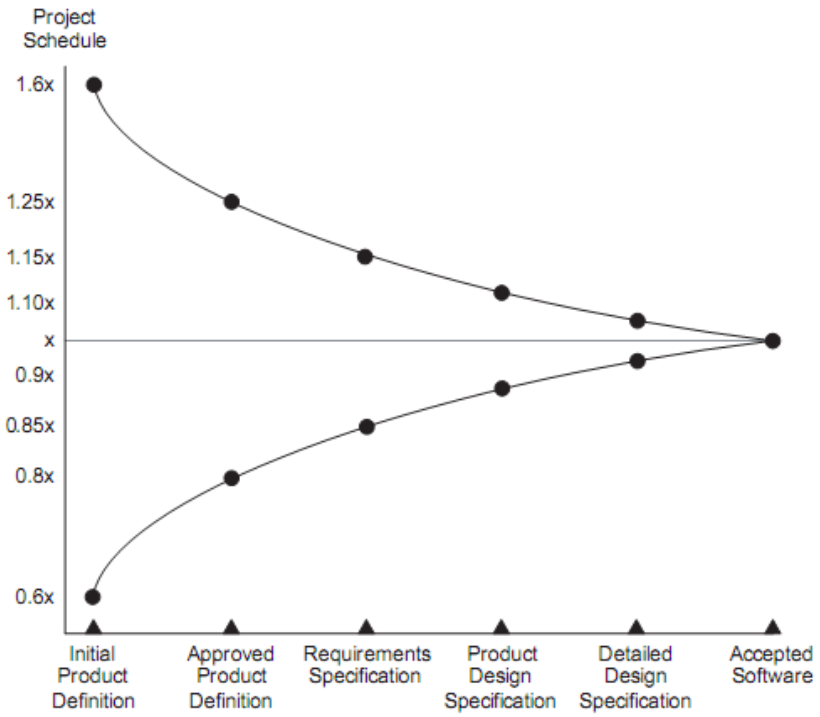

Figura 6 - Cone da Incerteza (Cohn, 2006, p. 4)

As estimativas podem ser analisadas do pondo de vista probabilístico. Ao invés de um valor absoluto, considera-se uma faixa de valores com maior probabilidade de conter o valor real. No início do projeto a incerteza atinge seu ápice, mas diminui na medida em que as tarefas são concluídas, os riscos mitigados e o gerente pode avaliar o que realmente foi realizado em contraste com o que foi previamente planejado. A incerteza é mínima ao final do projeto, quando todas as atividades estão concluídas e o esforço necessário para completá-las contém o valor que reflete a realidade.

Entretanto, o grau de incerteza representado pelo Cone da Incerteza (Figura 6) reflete a menor incerteza possível num determinado ponto do projeto. McConnell (2006) afirma que o comum é a incerteza ser maior do que a apresentada no gráfico, principalmente em projetos onde não se tomam as precauções necessárias em cada fase através de um gerenciamento adequado.

4.4 Barreiras e influências na estimação de software

Não existe uma forma de estimação determinística. Esforços e recursos foram despendidos para criar e aperfeiçoar técnicas e modelos de estimação, mas características como mutabilidade dos requisitos e a intangibilidade do software os tornam inerentemente difícil de estimar.

Barreiras na elicitação de requisitos fazem parte dos principais fatores que tornam a estimação difícil. Como apresentado no capítulo II, os requisitos mudam ao longo do tempo. Muitos tipos de domínios sofrem mudanças muito rapidamente, outros são muito complexos, há casos onde os usuários não conseguem definir suas necessidades ou o conhecimento está fragmentado de tal forma que não é possível identificar com precisão o que deve ser desenvolvido.

As características do software também dificultam a tarefa de estimação. Identificar a solução completa para um problema é um desafio. O software é um produto abstrato, desenvolvido em um processo diferente da engenharia convencional, pois cada software é único. Tudo isso torna desafiadora a tarefa de prever o esforço de desenvolvimento.

Além disso, os projetos de software são imprevisíveis e ocorrências podem invalidar a estimação realizada. Brooks (1975) afirma que “nossas técnicas de estimativa são mal desenvolvidas [...] elas não refletem um pressuposto não mencionado, que não é muito verdadeiro, isto é, de que tudo irá correr bem.” Brooks acrescenta que a pressão externa pode resultar em estimativas tendenciosas e ruins, pois “frequentemente falta aos gerentes de software firmeza de fazer o pessoal esperar por um bom produto”.

Imprevistos podem ocorrer de diversas formas no decorrer do projeto. Alguns tipos de imprevistos incluem aspectos já discutidos neste estudo sobre o software e os requisitos. Pressman (2009, p. 520) identifica e classifica uma série de fatores que impactam no custo e no esforço de desenvolvimento, conforme a tabela a seguir.

Tabela 3 – Fatores que impactam no desenvolvimento de software (Pressman, 2009, p. 520)

Fatores	Exemplos
Humanos	Pedidos de demissão, problemas pessoais, habilidades dos indivíduos.
Ambientais	Dificuldade de comunicação com cliente ou entre a equipe de desenvolvimento, mudanças no negócio, no escopo ou no domínio.
Políticos	Mudanças em leis, nas políticas da empresa.
Técnicos	Ferramentas de desenvolvimento apresentam problemas ou possuem limitações difíceis de contornar, mudanças e restrições tecnológicas.

Existem problemas que, quando presentes em um projeto, influenciam negativamente na estimação, a qual dificilmente refletirá a realidade. McConnell (2006) identifica alguns desses problemas, os quais estão descritos na tabela abaixo.

Tabela 4 – Influências negativas sobre a estimação (McConnell, 2006)

Processo de desenvolvimento caótico	Ocorre quando há falha em uma ou mais partes do projeto. Por exemplo: requisitos não elicitados adequadamente desde o início, falta de envolvimento dos usuários na validação dos requisitos, práticas de codificação ruins, inexperiência da equipe, plano de projeto incompleto, abandono do plano nos momentos de pressão, falta de gerenciamento automatizado do código.
Requisitos instáveis	Mudanças são esperadas, mas os requisitos precisam estar bem definidos em algum momento, caso contrário a estimação não será adequada nem mesmo em fases avançadas do projeto.
Atividades omitidas	Umas das fontes mais comuns de problemas é o esquecimento de incluir atividades essenciais. Alguns exemplos comuns são: instalação do sistema, conversão de dados, adequação e uso de bibliotecas de terceiros, ajuda ao usuário, interfaces com sistemas externos. Além disso, alguns requisitos não funcionais também são esquecidos, tais como desempenho, confiabilidade, escalabilidade, segurança, entre outros.
Otimismo sem fundamento	Desenvolvedores tendem a gerar estimativas otimistas com fator de vinte a trinta por cento, em média.
Subjetividade e	Tendências pessoais conscientes ou inconscientes do estimador

tendência pessoal	geram erros de estimação para mais ou para menos.
Estimativas informais	Estimativas repassadas de desenvolvedores para seus superiores, sem que haja tempo hábil para avaliar criteriosamente a situação, provavelmente serão bem diferentes da realidade.
Precisão não necessária	O termo *precisão* não deve ser confundido com *acurácia*. Acurácia significa estar próximo da realidade, enquanto precisão está mais relacionada ao nível de detalhamento, por exemplo, o número de casas decimais de um resultado. Estimativas demasiadamente precisas e detalhadas não necessariamente são verdadeiras. Uma estimativa com precisão de horas não tem utilidade. Uma boa prática seria estimar com base em semanas, meses ou o que for adequado para o tamanho do projeto, resultando numa maior acurácia.

Além disso, as estimativas também são influenciadas por características do projeto e da equipe. A influência pode ser positiva ou negativa, dependendo de cada característica. McConnell (2006) cita quatro características do projeto e da equipe descritas na tabela a seguir.

Tabela 5 – Características do projeto que influenciam estimativas (McConnell, 2006)

Tamanho do projeto	O esforço por linha de código aumenta proporcionalmente ao tamanho do projeto. É importante descobrir o porte do projeto para depois ajustar as estimativas individuais.
Tipo de software	O rendimento da equipe varia dependendo da área para a qual o software está sendo desenvolvido. Por exemplo, softwares desenvolvidos para uma intranet chegam a ter um rendimento vinte vezes maior do que softwares construídos para a aeronáutica.
Fatores pessoais	Estudos mostram que o desempenho pode variar numa escala de dez vezes entre indivíduos. As estimativas devem ser ajustadas conforme o desempenho dos desenvolvedores envolvidos.
Linguagem de programação ou tecnologias	Ao usar o número de linhas de código como base para as estimativas, deve-se usar um fator de ajuste dependendo da linguagem utilizada. O mesmo pode ser estendido para outras

	tecnologias envolvidas.

Pressman (2009, p. 525) também destaca algumas influências que devem ser consideradas na estimação. Segundo o autor, mesmo que o software a ser estimado fosse relativamente estático em relação ao ambiente e aos requisitos, existem fatores externos que influenciam diretamente na qualidade das estimativas, tais como:

- o grau de precisão com que o planejador estimou o tamanho do produto a ser construído;
- a precisão para transformar a estimativa de tamanho em esforço humano, tempo e custo;
- experiência do gerente;
- acesso a boas informações históricas.

Enfim, a estimação é uma atividade desafiadora que possui diversas influências e dificuldades intrínsecas oriundas das barreiras do desenvolvimento de software e da natureza dos requisitos, das várias influências e das características do projeto, além do risco de problemas que podem ocorrer durante o projeto.

4.5 Princípios gerais de estimação

Existem princípios que podem ser aplicados a qualquer forma de estimação. Embora as diferentes técnicas e modelos de estimação sejam diferentes em diversos aspectos e aplicáveis em diferentes contextos, a natureza das estimativas permanece a mesma.

Quanto mais cedo ocorrer a estimação, piores serão as estimativas (Pressman, 2009, p. 524). A estimação antecipada de todo o software tende a gerar estimativas piores do que em casos onde elas são feitas em fases posteriores do projeto (Pressman, 2009, p. 520). Embora muitas vezes seja necessário estimar todo o software, é importante refazer a estimação quando se deseja obter uma posição atualizada sobre o estágio de desenvolvimento.

Além disso, informações de projetos semelhantes já completados tendem a deixar as estimativas melhores. Dessa forma, é possível prever com mais exatidão o esforço necessário ao se basear em experiências anteriores.

A decomposição do projeto e do software em unidades menores também ajuda na estimação. Isso facilita ao estimador manter o foco e ter uma visão mais clara do que ele está

estimando. Além disso, é mais fácil encontrar informações históricas de componentes de software semelhantes já desenvolvidos.

Estimativas de componentes do software podem ser feitas utilizando medidas relativas (Cohn, 2006, p. 34). Em técnicas que utilizam este recurso, atribui-se um valor relativo a cada componente ou funcionalidade do sistema. Então a estimativa final é calculada baseando-se em dados de projetos semelhantes ou com base no que já foi desenvolvido do projeto corrente. Dessa forma, não é necessário refazer as estimativas individuais dos componentes do sistema no decorrer do projeto, sendo necessário apenas ajustar o fator usado para calcular a estimativa final de acordo com as avaliações do projeto.

A qualidade de uma estimativa pode ser verificada utilizando um segundo método de estimação. Quando o estimador está incerto sobre uma estimativa e deseja uma certeza maior sobre a mesma, pode-se realizar uma nova estimação através de outra técnica ou modelo a fim de averiguar se a primeira estimativa era confiável.

Entretanto, não é recomendado investir esforço demasiado na estimação. Esforços exagerados em melhorar as estimativas podem resultar num efeito contrário ao esperado, gerando distorção nos valores obtidos. Cohn (2006, p. 54) fez um levantamento sobre os esforços para obter melhores estimativas e percebeu que, a partir de um determinado momento, quanto mais tempo é investido nessa tarefa, mais o resultado estará longe da realidade (Figura 7).

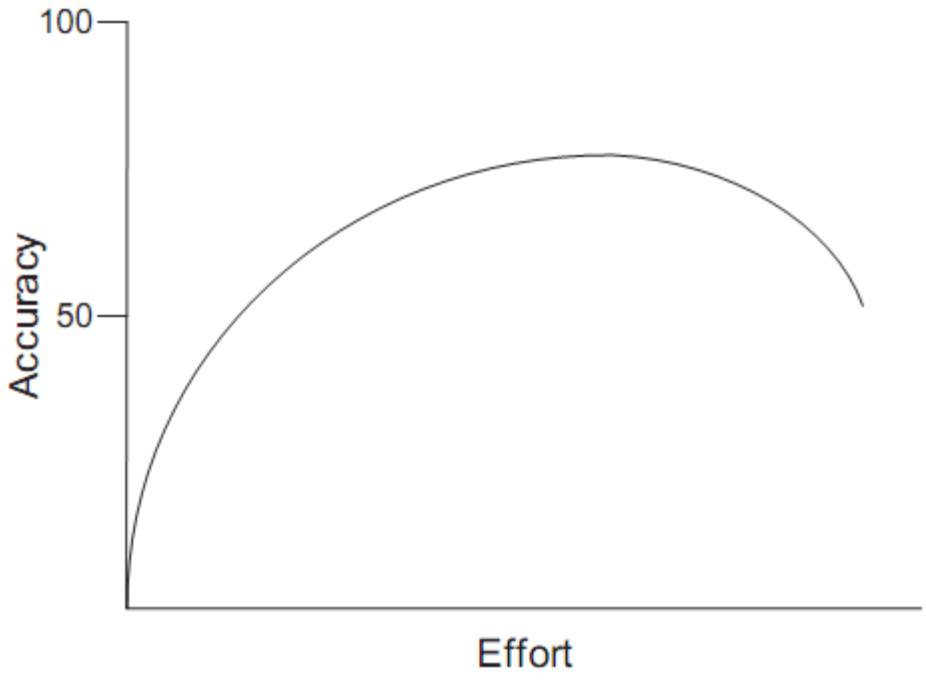

Figura 7 - Gráfico de Escorço x Acurácia das Estimativas (Cohn, 2006, p. 54)

4.6 Estimação indireta

Algumas das técnicas de estimação são indiretas (McConnell, 2006). Por exemplo, dividir o sistema em componentes e estimar baseando-se nas características dos componentes ou estimar através de uma classificação das funcionalidades utilizando lógica *fuzzy* (categorizando cada funcionalidade como muito pequena, pequena, média, grande ou muito grande) são formas indiretas de estimação. Após a estimação dos elementos do software calcula-se a estimativa final de acordo com a técnica utilizada.

4.7 Aplicabilidade das técnicas e modelos de estimação em processos

A escolha de uma técnica ou modelo de estimação em um projeto pode ser definida com base no processo de desenvolvimento adotado. Características do processo podem inviabilizar ou recomendar o uso de determinada técnica ou modelo de estimação. Processos ágeis que utilizam histórias de usuário para ilustrar as funcionalidades do sistema dificultam a aplicação de técnicas de estimação que dependem de um modelo de análise e design.

Técnicas de estimação que exigem detalhamento do sistema são mais recomendados para processos disciplinados ou que exigem maior burocracia. Em geral, essas técnicas de estimação partem do modelo de análise procurando antecipar o tamanho e a complexidade do software (Pressman, 2009, p. 357). O modelo de análise deve estar detalhado o suficiente para permitir que seus elementos sejam estimados. Como os processos burocráticos dão grande importância ao planejamento antecipado como um dos fatores mais críticos para o sucesso ou a falha de um projeto (Pressman, 2009, P. 519) então uma técnica de estimação mais detalhada torna-se adequada.

Por outro lado, técnicas empíricas baseadas em julgamento humano de funcionalidades ou características de alto nível de um software são recomendadas quando a organização adota processos com pouca burocracia, tal como processos ágeis.

4.8 Estimação em processos ágeis

A estimação nos processos ágeis é, em geral, baseada em histórias de usuário (Cohn, 2006, p. 34). A unidade de medida comumente utilizada é a *story point* (ponto de história), que indica o tamanho de uma história de usuário que está no backlog do produto, numa escala definida para o projeto em questão. Os *story points* atribuídos às histórias de usuários possuem valor relativo entre as várias histórias, com o objetivo de permitir a comparação.

Uma história de usuário é detalhada em tarefas individuais para a equipe de desenvolvimento no início da iteração em que foi selecionada para ser implementada. O esforço das tarefas é então estimado em horas. É recomendado que uma tarefa não tenha duração de mais de um dia de trabalho.

A estimação em processos ágeis é feita pela equipe. Os agilistas enfatizam que a estimação é mais bem feita pelos indivíduos que efetivamente criam o software, ao invés de um analista ou gerente.

Além disso, a estimação deve ser em equipe, de modo que todos ou a maioria concorde com cada estimação. Deve-se evitar que um membro influencie o outro na estimação inicial, pois um membro da equipe poderia ter receio de ter uma opinião muito diferente sobre determinadas estimativas. Cada membro da equipe teria o potencial de apresentar uma perspectiva sobre um determinado requisito que os demais não conseguiram visualizar. Dessa forma, haveria uma verificação coletiva de cada estimativa.

4.9 Modelo básico da atividade de estimação

A atividade estimação pode ser dividida em três tarefas básicas, independente do modelo ou técnica utilizada (McConnell, 2006).

Primeiramente, deve-se realizar a escolha da técnica de estimativa considerando o tamanho e o estágio atual do projeto, o processo de desenvolvimento (iterativo, sequencial, evolutivo) e a acurácia possível.

Em seguida, o estimador deve contar, computar e julgar de forma adequada os elementos que servirão como base das estimativas, tais como linhas de código, horas de trabalho ou pontos de história;

Finalmente, deve-se calibrar a contagem realizada. Isso é feito utilizando dados históricos de projetos passados ou de iterações anteriores do mesmo projeto e avaliando características internas e externas que influenciam o projeto. Desse modo, a contagem pode ser realmente considerada uma estimativa.

4.10 Técnicas e modelos de estimação

4.10.1 Estimação orientada a objetos

Sistemas orientados a objetos podem ser estimados através de modelagem orientada a objetos, como a UML (*Unified Modeling Language*). Atribui-se valores a cada objeto do sistema e, assim, pode-se chegar a uma estimativa geral.

Algumas medidas sugeridas por Pressman (2009, p. 506) são:

- número de scripts de cenários de interação entre usuário e o sistema;
- número de classes importantes e independentes do sistema;
- número de classes de apoio não relacionadas ao domínio (banco de dados, interface com usuário);
- número de classes de apoio relacionadas ao domínio;
- número de subsistemas.

4.10.2 Estimação orientada a casos de uso

É possível utilizar casos de uso para estimar um software. As estimativas de cada caso de uso possibilitariam planejar o projeto com um todo.

Entretanto, como casos de uso são muito abstratos e pessoas diferentes trabalham em diversos níveis de abstração. Não há parâmetros para definir uma medida padrão para o esforço necessário para implementar um dado caso de uso (Pressman, 2009, p. 507). Em decorrência disso, a estimação com casos de uso não é recomendada, sendo pouco utilizada.

4.10.3 Estimação orientada a componentes

Um sistema pode ser dividido em componentes de acordo com as características de seus componentes. Por exemplo, um sistema com arquitetura para a web poderia ter seus elementos classificados em páginas dinâmicas, páginas estáticas, tabelas de banco de dados, relatórios e regras de negócio (McConnell, 2006). Cada conjunto representa um componente, cujos elementos são semelhantes.

Dados históricos classificados de acordo com os tipos de componentes ou o julgamento de um especialista podem ser usados para estimar o número de linhas de código ou esforço necessário para desenvolver cada componente.

4.10.4 Estimação baseada no julgamento de um especialista

Esta técnica consiste em dividir o desenvolvimento do aplicativo de software em tarefas utilizando uma granularidade adequada, isto é, dividindo tarefas grandes, de longa duração ou complexas, em menores e utilizando faixas de estimativas com melhor e pior caso. Esses valores são usados para derivar uma estimativa final que possuiria grande probabilidade de sucesso.

Na medida em que o projeto evolui, o estimador compara os dados obtidos das tarefas já concluídas com as estimativas originais e calcula o erro relativo, ajustando os valores estimados para as tarefas futuras.

Através dessa técnica, um estimador que realiza seu trabalho de forma estruturada e não apenas intuitiva pode conseguir bons resultados (McConnell, 2006). Apesar de ser totalmente empírico, este é o método mais utilizado nas organizações.

4.10.5 Estimação por analogia

Consiste em estimar um projeto baseando-se em outro com arquitetura semelhante já concluído, ou seja, que possui dados reais sobre as tarefas realizadas.

Esta técnica pode gerar grandes erros se usado de forma indevida (McConnell, 2006). Primeiramente é necessário possuir dados precisos e detalhados sobre o projeto antecessor. Uma boa prática seria construir as novas estimativas como uma porcentagem em relação ao projeto anterior, comparando o tamanho de ambos. Ainda assim é necessário considerar diferenças importantes, tais como tecnologias diferentes, tamanhos muito distintos, equipe diferente, etc.

4.10.6 Estimação por Ponto de Função (*Function Point*)

A técnica de estimativa por ponto de função se propõe a medir o tamanho das funcionalidades de um sistema (Pressman, 2009, p. 357). Através de uma lista dos elementos do software a serem construídos, avaliações da complexidade desses elementos e dados históricos, seria possível estimar:

- custo e esforço do desenvolvimento;
- quantidade de erros encontrados nos testes;
- número de componentes e linhas de código projetadas no sistema.

As entradas necessárias para realizar a estimação são:

- número de entradas externas: entrada de dados pelo usuário ou por outra aplicação;
- número de saídas externas: saída que fornece informação ao usuário (relatórios, telas, etc.);
- número de consultas externas: chamada on-line com resposta imediata, como uma chamada de serviço por outra aplicação;
- número de arquivos lógicos internos: quantidade de agrupamentos de dados, tal como tabelas, arquivos de dados, entre outros;
- número de arquivos de interface externa: dados externos acessados pela aplicação.

Cada entrada é extraída de modelo da aplicação. A partir dos dados obtidos, cada elemento é contado, avaliado e categorizado em níveis de complexidade, geralmente de forma subjetiva, como simples, médio ou complexo. O resultado é utilizado para calcular uma contagem total dos pontos, conforme a figura abaixo:

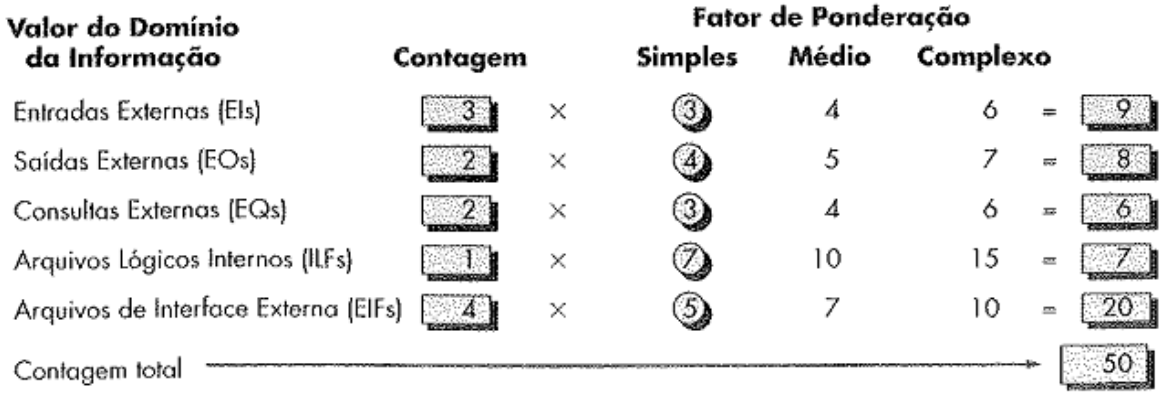

Figura 8 - Exemplo de Estimativa por Ponto de Função (Pressman, 2009, p. 359)

Alguns valores de ajuste (F_i) são definidos baseados em respostas com valores de “0” (não importante) até “5” (absolutamente essencial) às catorze descritas na tabela a seguir:

Tabela 6 – Fatores de ajuste da técnica de estimação por pontos de função (Pressman, 2009, p. 357)

1	O sistema precisa de backup?
2	Modos de transferência de dados específicos são usados para importar ou exportar

	dados?
3	Existe processamento distribuído?
4	O desempenho é crítico?
5	O sistema funcionará em um ambiente operacional existente, intensamente utilizado?
6	O sistema requer entrada de dados online?
7	A entrada online exige que a transação seja construída por várias telas ou operações?
8	Os arquivos lógicos internos são atualizados online?
9	As entradas, saídas, arquivos ou consultas são complexos?
10	O processamento interno é complexo?
11	O código foi projetado para ser reusado?
12	A conversão e instalação estão incluídas no projeto?
13	O sistema está projetado para instalações múltiplas em diferentes organizações?
14	A aplicação está projetada para facilitar modificações no uso do usuário?

Ao final, aplica-se a seguinte equação para obter o valor de pontos de função:

$$FP = \text{total de contagem} \times [0{,}65 + 0{,}01 \times \Sigma (F_i)]$$

Figura 9 - Equação para obter o resultado da métrica por pontos de função

Os níveis de complexidade e os fatores de ajuste são valores subjetivos, pois dependem completamente da experiência e intuição do estimador. Entretanto, eles mesmos podem ser ajustados através de um histórico de dados colhidos em projetos anteriores (Pressman, 2009, p. 505).

A quantidade de linhas de código de um sistema (LOC - *Line of Code*) pode ser derivada a partir dos pontos de função. Através de estimativas grosseiras da quantidade de linhas de código da linguagem de programação escolhida por ponto de função, a quantidade total de linhas de código é calculada, através de simples multiplicação, e utilizada para prever o tamanho do sistema.

4.10.7 Modelo COCOMO II

COCOMO (*Constructive Cost Model*) é um modelo para estimar custo, esforço e cronograma durante o planejamento de projetos de software. Ele foi inicialmente publicado

por Barry Boehm em 1981 e posteriormente foi lançada uma versão atualizada denominada COCOMO II, em 1995, refletindo a dramática evolução das práticas de desenvolvimento de software (CSSE).

O modelo COCOMO II possui um programa de afiliados que investem técnica e financeiramente no seu desenvolvimento, incluindo grandes empresas e laboratórios de pesquisa. Ele é um modelo aberto, isto é, pode ser usado por qualquer organização, e faz uso intensivo de dados históricos.

O modelo utiliza a seguinte fórmula para calcular o esforço:

$$Esforço\ Nominal = A \times Tamanho^{B}$$

Figura 10 - Fórmula de cálculo do esforço do projeto (Boehm, 2005)

Segundo Boehm (2005), a constante A captura o efeito linear do tamanho do projeto no esforço necessário. B é um fator exponencial de ajuste de acordo com o tamanho do projeto. O Tamanho se refere à dimensão do projeto e seu valor pode ser medido em milhares de linhas de código, pontos de objeto ou pontos de função, geralmente derivado de outra técnica de estimativa.

O fator exponencial B é ajustado para cada projeto a partir de um conjunto de fatores definidos no modelo COCOMO II, os quais levam em consideração, por exemplo:

- se o projeto foi precedido por outro semelhante;
- se os riscos e a arquitetura são claramente definidos;
- o entrosamento da equipe;
- a maturidade do processo.

Além disso, o Esforço Nominal calculado pode ser ajustado por fatores relacionados ao produto, à plataforma, ao projeto e mesmo também pessoais. Por exemplo, o fator DATA permite o ajuste pelo tamanho do banco de dados, RELY considera a confiabilidade exigida do software, CPXL a complexidade do produto, ACAP a capacidade dos analistas, TOOL o uso de ferramentas de desenvolvimento e assim por diante.

Cada fator recebe uma classificação numa escala de cinco níveis (muito baixo, baixo, nominal, alto, muito alto). O ajuste é realizado multiplicando-se os fatores individuais pelo Esforço Nominal.

Clark (1998) define o COCOMO como um modelo paramétrico, ou seja, que utilizada uma representação matemática idealizada do mundo real baseada em parâmetros quantitativos ou qualitativos. Embora os parâmetros sejam derivados, em sua maioria, de julgamentos subjetivos, o modelo propõe que, através dos ajustes adequados, é possível obter uma estimativa com 70% de confiabilidade (Boehm, 2005).

4.10.8 PROBE

O PROBE (*Proxy Based Estimating* – Estimação Baseada em Proxy) é uma técnica de estimação indireta baseada em objetos e dados históricos (Humphrey, 2000, p.12).

Primeiramente, os engenheiros identificam os objetos necessários para a construção do produto de software e então determinam aproximadamente o tipo e o número de funções de cada objeto. Com base em dados históricos, compara-se cada objeto com objetos similares e estima-se o tamanho do mesmo em linhas de código usando regressão linear.

A estimação do esforço é realizada de forma similar. Com base nos dados históricos, aplica-se a regressão linear para estimar o esforço necessário para o desenvolvimento. Os dados devem mostrar uma relação direta entre o tamanho do programa e o tempo de desenvolvimento.

Para gerar o cronograma, cada engenheiro deve estabelecer em que ele irá trabalhar no projeto semanalmente ou diariamente, calculando-se assim o tempo necessário para conclusão de suas tarefas.

Durante o desenvolvimento, os engenheiros devem armazenar os dados de tempo e tamanho do projeto atual para planejamentos futuros.

4.10.9 Planning Poker

Alguns processos ágeis, como o *Scrum*, encaram o projeto de software como um jogo. Nessa linha, alguns autores consideram a melhor forma de estimar uma espécie de *poker* com as histórias de usuário do *backlog* (Cohn, 2006, p. 55).

No *Planning Poker*, todos os membros da equipe de desenvolvimento se reúnem. Primeiramente, cada membro recebe cartas. Cada carta contém um valor da escala *de story points* definida para o projeto. Para cada história de usuário, os membros escolhem secretamente a carta com o valor que consideram ser o tamanho daquela história. Então, todos mostram suas cartas ao mesmo tempo. Quando há consenso, a estimativa seria confiável. Se um ou mais valores forem discrepantes, os membros discutem entre si o motivo porque a

história de usuário seria mais ou menos complexa. Após entrarem num consenso, uma nova rodada de cartas é realizada. Isso é feito até que todos estejam de acordo com a estimativa.

Os “agilistas” afirmam que esta técnica tem funcionado bem pelos seguintes motivos (Cohn, 2006, p. 59):

- aqueles que sabem como fazer o trabalho são os que fazem as estimativas;
- o diálogo faz com que os estimadores tenham que justificar suas estimativas, portanto eles pensam bem no que estão fazendo;
- a discussão em grupo, segundo alguns estudos, levam a melhores estimativas.

O *Planning Poker* geralmente é feito de forma mais intensa, porém menos detalhada, no início do projeto, de modo a gerar as estimativas iniciais necessárias para o planejamento geral e o número de iterações.

No início de cada iteração, sugere-se levar algo em torno de uma hora para detalhar as estimativas das histórias de usuário que serão implementadas, as quais, que nesse ponto, estão dividias em tarefas. O tempo de estimar varia de acordo com o tamanho da iteração.

4.10.10 Estimação por velocidade e aceleração

A velocidade de desenvolvimento é a quantidade de *story points* que uma equipe consegue implementar em um determinado intervalo de tempo (Cohn, 2006, p. 39). A velocidade pode ser calculada medindo-se o tempo que a equipe levou para concluir cada ponto de história.

A estimação das próximas histórias pode ser feita ou ajustada de acordo com a velocidade da equipe. Além disso, basta aplicar a velocidade às histórias de usuário restantes para obter-se a estimativa geral de conclusão do projeto.

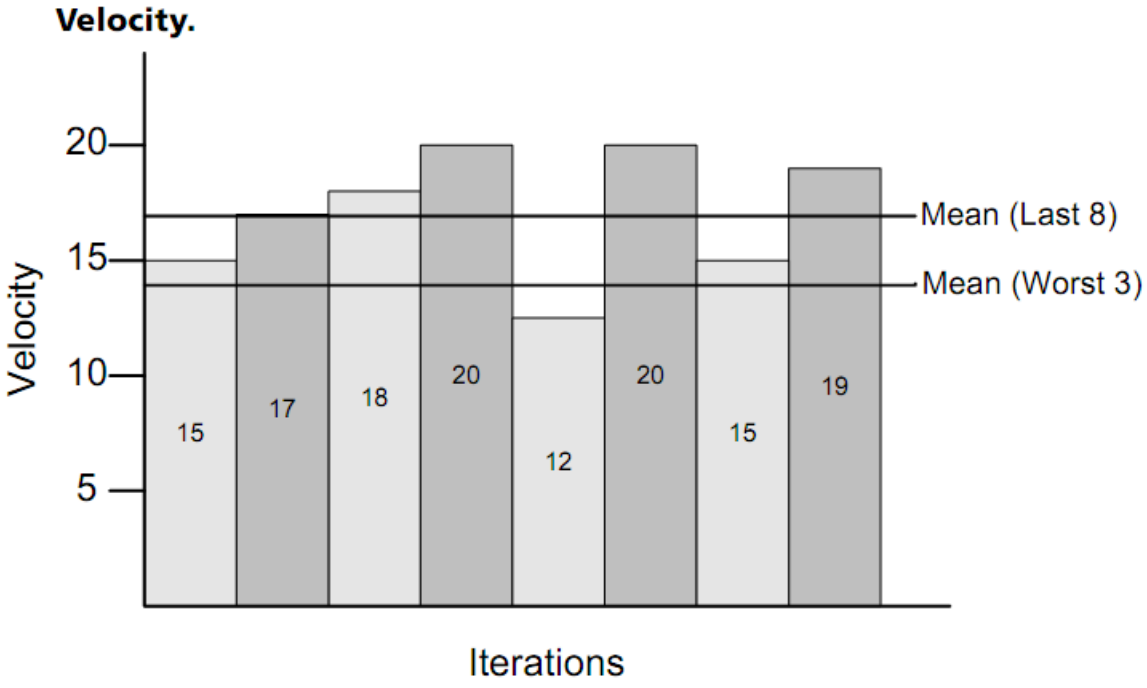

Figura 11 - Gráfico de Velocidade (Cohn, 2006, p. 237)

A Figura 11 contém um gráfico de velocidade por iteração. A velocidade média permite saber se o projeto está avançando de acordo com o esperado.

A aceleração do desenvolvimento pode ser calculada quando há uma tendência de aumento ou declínio da velocidade de desenvolvimento. Decisões gerenciais podem ser tomadas antecipadamente se os desenvolvedores estão terminando as histórias de usuário mais rapidamente ou mais lentamente a cada iteração.

4.11 Conclusão

O uso das técnicas e modelos de estimação depende do contexto. Alguns elementos são complementares, como a técnica de Pontos de Função e o modelo COCOMO, enquanto outros são incompatíveis, tal como o *Planning Poker* e técnicas semelhantes.

Embora não se possa afirmar definitivamente a superioridade absoluta de um modelo ou técnica, a utilização de um modelo ou técnica adequada pode guiar o estimador a melhores resultados, alavancando suas habilidades através de uma metodologia bem definida.

Por outro lado, qualquer estimador que se apoia em determinado modelo ou técnica como uma “bala de prata” sem compreender a natureza das estimativas, inevitavelmente irá se deparar com uma realidade muito diferente da esperada.

Entender o que são estimativas e no que consiste a tarefa de estimação é fundamental e mais importante do que adotar uma técnica ou modelo em específico.

CAPITULO V

ESTUDO DE CASO

5.1 Introdução

Este capítulo apresenta um estudo de caso prático de estimação num contexto comum de um projeto de desenvolvimento de software em uma empresa a fim de ilustrar a aplicação dos conceitos apresentados neste trabalho.

5.2 Contexto

Uma empresa de desenvolvimento de software trabalha primariamente com softwares "de prateleira" (*Commercial Off-The-Shelf* - COTS) e faz adaptações sob demanda para seus diversos clientes no segmento financeiro. Um projeto de desenvolvimento pode ser criado quando se identifica a necessidade de um novo sistema ou a partir da solicitação de clientes para implementação de novas funcionalidades ou modificações nas já existentes.

Os sistemas são desenvolvimentos na plataforma Java com arquitetura web, onde o usuário acessa através de um navegador web. A arquitetura dos sistemas adota o padrão *Model-View-Controller* (MVC). O *model* (modelo) consiste em um conjunto de componentes que disponibiliza serviços que realizam cálculos, atualizam as entidades do sistema e fazem integrações com outros sistemas. A *view* (visão) consiste num conjunto de tecnologias para geração de interfaces web exibidas no navegador do usuário a partir dos dados do *model*. O *controller* (controlador) consiste em um componente que recebe ações e dados do usuário, atualiza o *model* e redireciona o usuário para a *view* adequada.

No início de um projeto, a elicitação dos requisitos é realizada por um analista de negócios, que cria o artefato chamado de Especificação de Requisitos de Software (ERS). A ERS pode conter, além dos requisitos, o *design* da solução. O *design* pode ser feito pelo analista de negócios quando este entende da solução ou por um Analista de Sistemas da equipe. Então o gerente do projeto, cujo papel é geralmente desempenhado pelo coordenador da equipe que irá desenvolver o software, faz o planejamento e a estimação inicial.

No planejamento, o gerente cria um cronograma, que pode um gráfico de Gantt contendo as atividades de desenvolvimento e testes para cada funcionalidade da ERS. A partir dessas informações é possível negociar o prazo com o cliente e alocar a equipe de acordo com o esforço necessário.

O processo de desenvolvimento da empresa pode ser no modelo *Waterfall* ou iterativo, dependendo do tamanho do projeto e da necessidade do cliente. Porém, o desenvolvimento não é incremental, sendo a implementação das funcionalidades simplesmente distribuída sequencialmente pelo projeto.

5.3 Detalhes do projeto

Uma nova funcionalidade de “estorno de aditamento” de contrato financeiro foi solicitada em um dos sistemas da empresa. Um novo projeto foi iniciado para tratar esta demanda.

Um aditamento consiste em uma alteração contratual realizada a partir do segundo mês de vigência do contrato, pois alterações dentro do mês de entrada não caracteriza um aditamento.

Um estorno de aditamento consiste em desfazer as alterações realizadas no processo de aditamento considerando todos os impactos nas integrações com outros sistemas e restaurando todos os aspectos originais do contrato antes do aditamento.

O diagrama a seguir ilustra os casos de uso relacionados a aditamento:

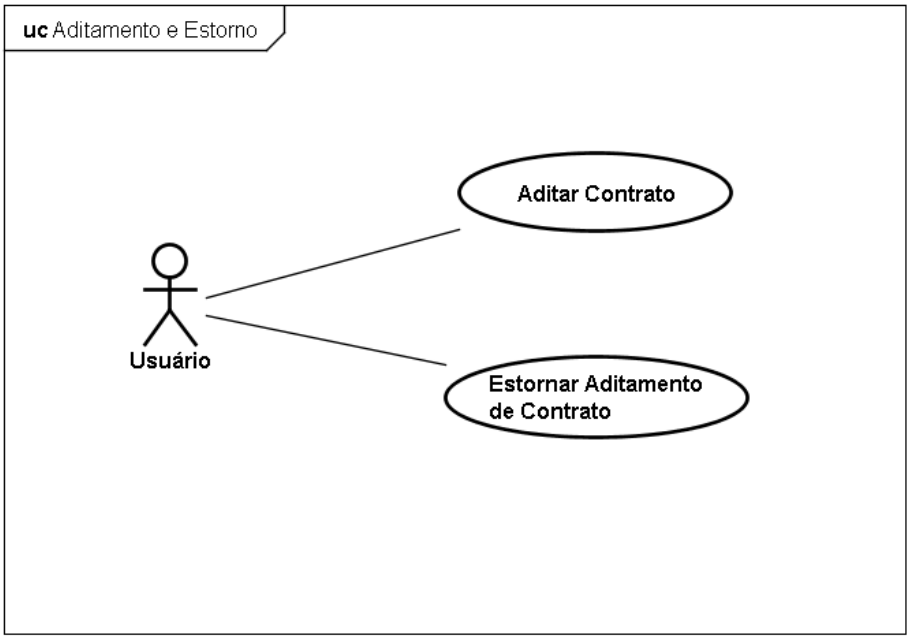

Figura 12 – Casos de Uso do projeto do Estudo de Caso

Ambos os conceitos são de domínio da equipe, pois funcionalidades semelhantes já foram implementadas diversas vezes, porém sem registros de tempo das atividades realizadas, de modo que não há dados históricos para a estimação no projeto atual.

Como há somente uma funcionalidade no projeto, o desenvolvimento seguirá o modelo *Waterfall*. Os requisitos serão elicitados, depois serão feitos o design e o planejamento de todo o projeto, seguidos pela estimação do esforço. Após o desenvolvimento da nova funcionalidade, uma versão executável do sistema será enviada para testes e então disponibilizada para o usuário final.

Correções serão realizadas durante a fase de testes internos da empresa e também, quando problemas ocultos forem encontrados durante os testes de aceitação dos usuários finais do sistema.

5.4 Análise e *Design* da Solução

O estorno de aditamento consiste em desfazer o aditamento de um contrato, portanto foi realizado um levantamento da funcionalidade de aditamento já existente e verificou-se que ela afeta vinte e duas entidades do sistema cujos dados são armazenados em tabelas no banco de

dados. Logo, o estorno deverá restaurar o estado de todas essas tabelas. O diagrama de sequência abaixo ilustra, em alto nível, o processo de estorno de aditamento:

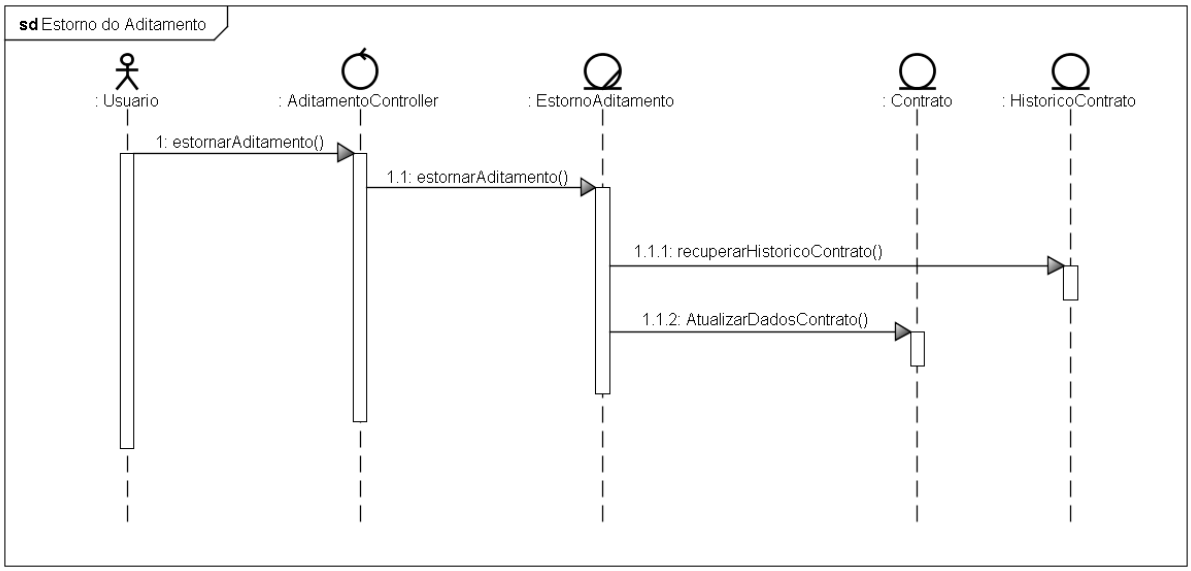

Figura 13 – Diagrama de Sequência da funcionalidade de Estorno de Aditamento

A fim de restaurar o estado anterior de uma entidade é necessário que exista um histórico de alterações. Algumas entidades já possuem tal histórico, enquanto doze delas não. Entretanto, as entidades que não possuem informações históricas não serão modificadas para minimizar o impacto nas demais funcionalidades do sistema. Tabelas de dados históricos auxiliares serão criadas, contendo os mesmos atributos de entidade e a data em que ocorreu a alteração. A funcionalidade de aditamento será alterada de modo que os dados anteriores de todas as entidades modificados sejam armazenados nessas tabelas de dados históricos.

No aditamento, uma interface de integração com um sistema de contabilidade é acionada através de serviços de componentes de software. No estorno do aditamento, deve ser acionado o serviço correspondente para gerar o efeito contrário do aditamento anterior.

Além disso, conforme o padrão dos sistemas da empresa, esse tipo de funcionalidade contará com duas interfaces com o usuário. A interface de **pesquisa de contratos** listará os contratos que foram aditados, contendo alguns filtros e uma tabela com o resultado, a qual permitirá selecionar um dos contratos para a realização do estorno. A interface de **confirmação do estorno** exibirá os dados do contrato após a seleção de um dos contratos na pesquisa e permitirá a confirmação do estorno do aditamento.

Os componentes do *model* do sistema deverão disponibilizar o serviço de pesquisa dos contratos aditados com os devidos filtros e o serviço principal que efetuará o estorno do

aditamento do contrato selecionado. Os serviços serão acionados pelos *controllers*, que, por sua vez, acionarão as *views* para exibir as interfaces do usuário.

O cenário de uso básico do sistema está representado no Diagrama de Atividades abaixo:

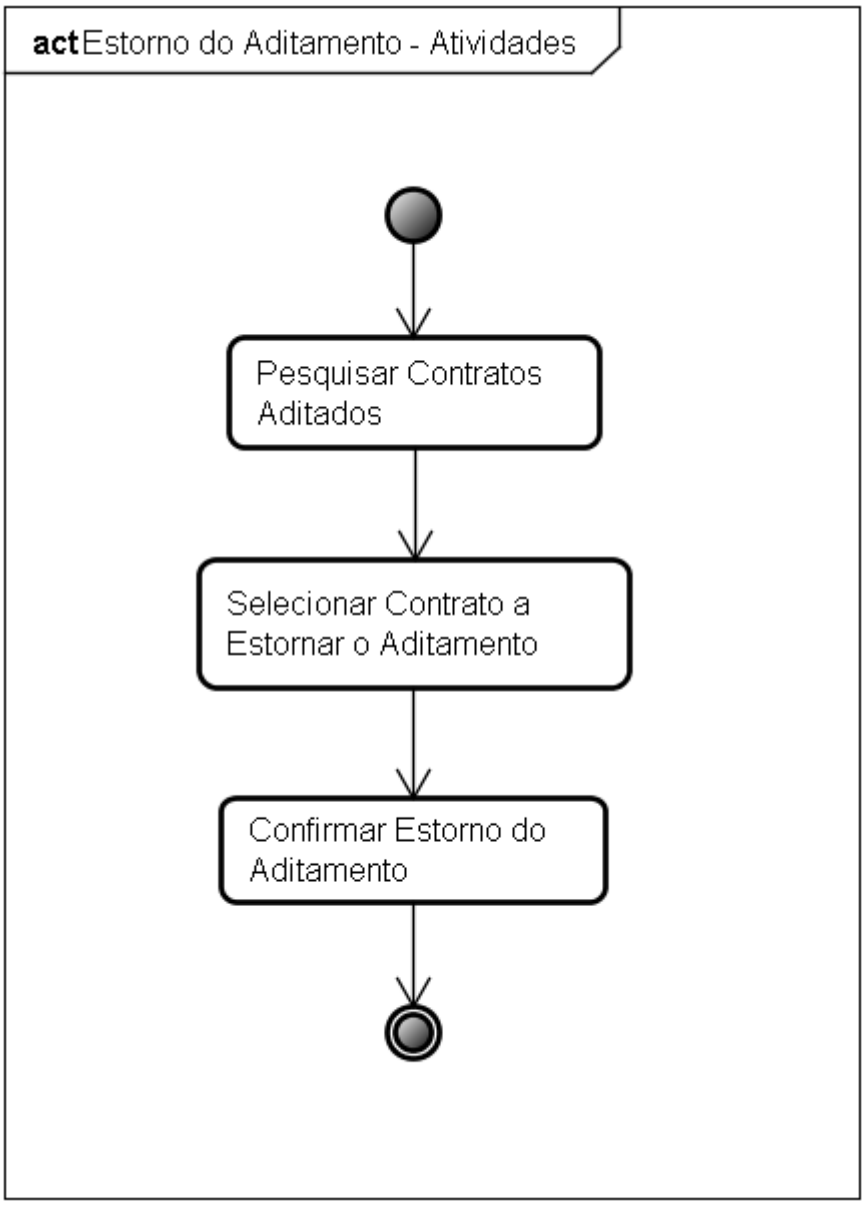

Figura 14 – Diagrama de Atividades da funcionalidade de Estorno de Aditamento

5.5 Estimação através da técnica de Pontos de Função

Através dos dados da análise e do design da funcionalidade de estorno de aditamento, obteve-se a tabela abaixo, a qual contém as entradas necessárias para a técnica de estimação por pontos de função:

Tabela 7 – Tabela com contagem de Pontos de Função do Estudo de Caso

Valor do Domínio	Contagem		Simples	Médio	Complexo		
Entradas Internas	2	x	3	4	6	=	6
Saídas Externas	2	x	4	5	7	=	8
Consultas Externas	1	x	3	4	6	=	6
Arquivos Lógicos Internos	1	x	7	10	15	=	15
Arquivos de Interface Externa	0	x	5	7	10	=	0
Contagem Total							35

Existem duas Entradas Internas, que são as duas interfaces com o usuário, pois tanto na pesquisa quanto na tela de confirmação o usuário deve interagir com a tela. As duas interfaces também são Saídas Externas, pois exibem dados dos contratos aditados. Elas são consideradas simples por possuírem poucos dados de entrada. Além disso, existe uma Consulta Externa, que consiste na integração com o sistema de contabilidade através do acionamento de um serviço de retorno imediato, a qual é considerada complexa por possuir mais um conjunto de dados com mais de quinze elementos. Além disso, existe um Arquivo Lógico Interno, isto é, apenas um conjunto de dados identificável pelo usuário. Apesar de vinte e duas tabelas ordinárias serem afetadas, mais as doze tabelas de dados históricos que deverão ser criadas, as

diversas rotinas CRUD reutilizadas e criadas não acrescentam complexidade. Por último, não existem Arquivos de Interface Externa.

Os seguintes fatores de ajuste (F_i) foram identificados:

- item 4, desempenho crítico, com valor 3;
- item 5, ambiente intensamente utilizado, com valor 2;
- item 10, processamento complexo, com valor 5.

Aplicando os valores obtidos na equação da técnica de estimação utilizada, o valor total de pontos de função resultante foi de 26,25.

É necessário multiplicar os pontos de função por um fator de horas por ponto de função de modo a estimar o esforço. Embora não haja dados históricos para apoiar a estimação no projeto atual, é possível estimar com base em dados médios registrados por organizações reconhecidas.

O *Brazilian Function Point Users Group* (BFPUG, 2013) apresenta diversas medidas de horas por ponto de função. Para a plataforma Java, as médias variam de 10 a 60 horas por pontos de função. Considerando a estimativa mais otimista, a funcionalidade de estorno de aditamento demandaria cerca de 260 horas de trabalho ou, utilizando como base o número de 160 horas de trabalho em um mês, o desenvolvimento levaria pouco mais de um mês e meio, contando com um desenvolvedor.

Com base no total de pontos de função estimados, é possível ainda derivar a estimação do tamanho do software em linhas de código (LOC). Pressman (2006, p. 505) apresenta uma tabela onde a linguagem Java possui uma média de 63 linhas por pontos de função. Assim, pode-se estimar o desenvolvimento de 1.653,75 linhas de código para a funcionalidade de estorno de aditamento.

5.6 Análise dos resultados

A estimativa de esforço obtida através da técnica de estimação por pontos de função é razoável quando comparada com o julgamento de um especialista, isto é, alguém com conhecimento sobre o software e o requisito em questão.

Entretanto, caso o desenvolvimento da funcionalidade do estudo de caso fosse realizado por um desenvolvedor com pouca experiência ou pouco conhecimento do software, a estimação deveria ser feita com base num número de horas por ponto de função mais pessimista.

5.7 Conclusão

O estudo de caso ilustra uma situação comum das empresas de desenvolvimento de software e exemplifica a aplicação de uma técnica de estimação de software.

A análise crítica do resultado de uma estimação é importante para evitar que equívocos no detalhamento das funcionalidades gerem grandes distorções. Caso o resultado gere desconfiança no estimador, deve-se revisar cada passo da estimação.

Além disso, embora coeficientes genéricos como a média de horas por pontos de função possam ser utilizados inicialmente, é essencial registrar os dados do desenvolvimento para uso em estimações futuras a fim de se obter um coeficiente mais adequado para o cálculo de esforço, que resulte em estimativas mais próximas à realidade no contexto do projeto e da equipe.

CONCLUSÕES E TRABALHOS FUTUROS

6.1 Introdução

Este capítulo apresenta as conclusões com base nos tópicos discutidos neste trabalho, relaciona possíveis trabalhos futuros que poderiam estender esta pesquisa e avalia a principal contribuição da mesma.

6.2 Conclusões

O primeiro capítulo estabeleceu algumas bases teóricas sobre o desenvolvimento de software com base na Engenharia de Software, uma disciplina que trata dos aspectos do desenvolvimento de software e provê abordagens para que o desenvolvimento atinja os objetivos do negócio.

O software não é fabricado em série como um produto “tradicional”, ele é desenvolvido. Tomando como exemplo um modelo de automóvel, o processo de fabricação do mesmo é definido linearmente do início ao fim. Ao invés disso, o desenvolvimento de software pode ser comparado ao projeto de criação do modelo do automóvel, o que ocorre antes da fabricação em série.

A arquitetura de um software é importante para a estimação, pois ela determina os tipos de componentes que irão compor o software e os relacionamentos entre eles, afetando diretamente a complexidade e as regras de desenvolvimento.

Existem barreiras no desenvolvimento e dificuldades intrínsecas na estimação de software. Algumas delas tem sua origem em características do software como intangibilidade,

complexidade e mutabilidade. Outras são oriundas de fatores humanos, como a dualidade de visão entre desenvolvedores, que pensam em nível funcional, e gerentes, que muitas vezes são orientados a cronograma e custos.

O segundo capítulo apresentou noções sobre Engenharia de Requisitos, uma atividade que busca o entendimento do software a ser construído e o gerenciamento das mudanças no decorrer do desenvolvimento.

Requisitos podem ser representados em vários níveis de detalhamento, mas para o desenvolvimento de software é necessário que eles sejam definições matematicamente formais das funcionalidades do software. Requisitos não existem por si só, isto é, eles não são levantados, encontrados ou identificados, mas são elicitados e definidos por um analista.

Requisitos são fundamentais para a estimação, pois fornecem a abstração necessária do problema para que os engenheiros de software cheguem ao modelo da solução que represente o instantâneo dos requisitos num determinado momento.

Estimativas são derivadas do modelo de análise e design. Estes modelos representam uma abstração da solução a ser desenvolvida em forma de software.

Ainda que algumas abordagens de estimação não incluam um modelo formal, os estimadores acabam usando um modelo de análise implícito em suas mentes, pois para chegar a uma estimativa de desenvolvimento eles precisam mentalizar a arquitetura e os componentes a serem desenvolvidos.

Os requisitos mudam ao longo do tempo por definição. O software é construído com base num modelo que representa os requisitos elicitados num determinado momento.

As mudanças nos requisitos devem ser gerenciadas para que seja possível identificar o impacto no software. A análise de impacto pode se beneficiar da rastreabilidade, isto é, uma relação entre os componentes do software que permita identificar quais deles devem ser modificados para atender uma mudança de requisito. Neste ponto, a arquitetura exerce um papel fundamental ao estabelecer a forma de organização e comunicação entre os componentes.

A qualidade da estimação no decorrer do desenvolvimento depende do gerenciamento de mudança dos requisitos, da qualidade da arquitetura e da rastreabilidade para que o estimador tenha consciência das alterações que são necessárias no software. Sem isso, é provável que o desenvolvimento siga sem um horizonte visível.

O terceiro capítulo discorreu sobre processos de desenvolvimento e gerenciamento de software. Processos são fundamentais para organizar o desenvolvimento de software e evitar o caos. Todo desenvolvimento segue um processo, seja ele bem definido, informal ou empírico.

Os modelos de processo propostos pela Engenharia de Software são abstrações de processos que as organizações podem implementar em seus projetos.

Alguns processos focam mais o gerenciamento do projeto, enquanto outros o desenvolvimento do software. Por isso, processos podem ser usados de forma complementar em um projeto de desenvolvimento de software.

Os processos podem ser classificados de acordo com a burocracia ou disciplina exigida. Processos burocráticos facilitam o gerenciamento, mas acrescentam uma carga adicional aos envolvidos no projeto. Processos empíricos, como os processos ágeis, procuram eliminar essa carga, baseando-se na confiança e interação entre a equipe ao invés da formalização.

A estimação de software deve considerar o processo adotado. Estimativas devem ser ajustadas de acordo com as atividades definidas e os produtos de trabalho exigidos pelo processo vigente, pois estes elementos impactam na produtividade da equipe. Além disso, estimativas de prazo e cronograma podem ser ajustadas e atualizadas de acordo com a iteratividade do processo e do modo como as funcionalidades são implementadas, isto é, num modelo evolucionário ou incremental.

Os processos ágeis introduziram uma séria de conceitos ao desenvolvimento de software procurando ser bastante adaptáveis às mudanças de requisitos, minimizar o tempo de resposta a estas mudanças e, dessa forma, atender as necessidades mais imediatas dos clientes. Para atingir esses objetivos, o processo é reduzido a um mínimo de disciplina e organização e espera-se que a equipe de desenvolvimento esteja motivada e se auto-organize para atender as necessidades imediatas do projeto.

Na prática, várias atribuições do gerente do projeto recaem sobre a equipe de desenvolvimento. Quando esta equipe possui maturidade e habilidade suficiente para arcar com essas responsabilidades, o que inclui planejar e estimar suas próprias tarefas, o processo ágil realmente pode otimizar o desenvolvimento software. Em contrapartida, o desenvolvimento pode chegar ao caos se os indivíduos envolvidos não corresponderem a estas expectativas.

Em decorrência disso, mesmo os “agilistas” às vezes utilizam certas práticas contrárias ao modelo ágil, como em casos onde o responsável pelo projeto ajusta as estimativas da equipe.

O quarto capítulo apresentou noções de estimação de software e algumas das técnicas comumente utilizadas em processos tradicionais e ágeis.

Estimativas são importantes do ponto de vista gerencial, pois clientes e gerentes precisam ter um horizonte sobre o desenvolvimento de um software.

O objetivo da estimação não é simplesmente fornecer uma data de entrega, mas possibilitar decisões corretas de negócio para que os objetivos da organização sejam atingidos ou ainda a correta avaliação desses objetivos para averiguar se eles são realmente tangíveis.

Estimativas não devem ser precisas, ou seja, utilizar uma unidade de medida pequena ou casas decimais, elas precisam ter acurácia, que significa estar próximo à realidade.

Embora estimativas geralmente sejam representadas numericamente, elas representam uma ordem de grandeza e não valores reais e absolutos. Manipular matematicamente os números das estimativas é um erro e leva à falsa sensação acurácia. Por exemplo, dobrar a quantidade de programadores não diminui pela metade o tempo de codificação. Esta e outras armadilhas são evitadas quando se entende a natureza real da estimação, que não consiste em medir algo, mas tentar prever sua ordem de grandeza.

Além disso, uma estimativa não deve ser confundida com o compromisso da equipe de desenvolvimento em concluir o trabalho num determinado prazo, tampouco com o plano da organização para atingir uma meta. É um erro grave manipular estimativas para adequá-las a uma determinada agenda. Cabe ao gerente do projeto usar as estimativas para calcular os recursos necessários para cumprir um plano ou então negociar os prazos do projeto.

Estimativas possuem um grau de incerteza. No de correr do projeto, quanto mais cedo a estimação for realizada, maior a incerteza. Em teoria, à medida que o desenvolvimento evolui, a incerteza diminui proporcionalmente ao conhecimento mais detalhado sobre o problema e a solução. Entretanto, isto somente será a realidade se houver o gerenciamento adequado, caso contrário o desenvolvimento pode acabar num ciclo interminável de correções e mudanças.

Bons estimadores devem considerar influências externas sobre o projeto e o software. Isto inclui fatores humanos, do ambiente do projeto, políticos e técnicos. Certas características do projeto também devem ser levadas em conta, como o tamanho do projeto, tipo de software, experiência da equipe e restrições tecnológicas.

Além disso, a qualidade da estimação é afetada por diversos fatores, tais como falha na elicitação de requisitos, falta de experiência da equipe, atividades omitidas, otimismo sem fundamento dos desenvolvedores e tendências pessoais do estimador.

Embora muitas variáveis influenciem na estimação, não é recomendável investir esforço demasiado nesta atividade, assim como se deve evitar exagero de detalhes. Diversos autores verificaram que, a partir de um determinado ponto, esforços e detalhes adicionais desviam as estimativas da realidade.

O mesmo princípio é aplicado à precisão desnecessária. Estimar um projeto de vários meses em horas de trabalho, embora seja uma unidade precisa, passa uma falsa sensação de acurácia. Neste caso, seria melhor adotar uma unidade de dias ou semanas de trabalho.

Diferentes técnicas de estimação foram apresentadas. Em geral, elas consistem em contar e julgar os elementos da solução proposta, em seguida aplicar uma fórmula prescrita e, finalmente, calibrar os resultados obtidos através de dados históricos e da avaliação das influências externas e características do projeto. A arquitetura do software é importante para a identificação e classificação dos elementos do sistema.

Algumas técnicas de estimação utilizam elementos do problema ao invés da solução, tal como o *Planning Poker*. Neste caso, o estimador acaba por construir mentalmente a ponte de ligação entre as necessidades dos clientes e da solução proposta, isto é, o modelo de análise e design.

Outras técnicas, como o *Function Point*, utilizam elementos pré-definidos que não levam em consideração a arquitetura do software. Técnicas deste tipo podem levar a desvios na estimação, pois nem sempre um software pode ser representado adequadamente no modelo exigido pela técnica.

A qualidade da estimação depende diretamente da habilidade, do conhecimento e da experiência do estimador. Em geral, o gerente do projeto é responsável pela estimação. Ele pode delegar esta atividade como um todo ou em parte para um ou mais membros da equipe, como nos processos ágeis. Porém, é necessário avaliar se as pessoas envolvidas estão capacitadas, afinal, é um consenso que desenvolvedores em geral são muito otimistas quando fornecem estimativas.

Alguns autores consideram a estimação como uma arte, outros como um processo de engenharia. Ambos contam com certa razão, pois tanto um bom artista como um bom engenheiro precisam de treino, técnica e habilidade para exercer seus talentos com perfeição. Além disso, o próprio desenvolvimento do software é considerado por alguns como um processo artesanal. No entanto, embora o talento dos indivíduos seja um elemento importante no desenvolvimento e na estimação, o conhecimento e o uso de técnicas adequadas irá potencializar esse talento.

Este estudo pretendia incluir uma comparação entre as técnicas de estimação com a finalidade de identificar as melhores técnicas disponíveis. Mas estimativas são previsões. Não há como comparar objetivamente técnicas que procuram antecipar o futuro.

A escolha da técnica de estimação em um projeto depende do que o estimador considerar conveniente. Assim, este estudo buscou trazer conhecimentos essenciais sobre estimação e os elementos que mais o influenciam para que o estimador tenha uma visão mais ampla sobre esta atividade e tome decisões conscientes.

O estimador deve conhecer o processo de desenvolvimento o mais detalhadamente e abrangentemente possível, ser realista e sincero quanto ao tempo de cada atividade e saber negociar os prazos com a área de negócios, que sempre exigirá entregas antecipadas.

6.3 Trabalhos Futuros

Como base teórica, o presente estudo limitou-se a apresentar conceitos sobre aplicativos de software, engenharia de requisitos, processos de desenvolvimento e gerenciamento de software que influenciam na estimação de software. Além disso, conceitos de estimação e diversas técnicas foram apresentados, concluindo com um estudo de caso com a aplicação de uma delas.

Outros trabalhos podem ser realizados comparando como as técnicas de estimação se encaixam nos diversos processos existentes. Algumas técnicas necessitam de determinados modelos que não são recomendados em processos ágeis, por exemplo. Por outro lado, o *Planning Poker* pode ser de difícil de execução quando requisitos formais estão especificados e não há estórias de usuário.

Outra possibilidade de estudo complementar seria analisar a usabilidade das técnicas de estimação, ou seja, verificar na prática se as técnicas são usadas corretamente ou se elas podem levar a armadilhas. Uma técnica pode se apresentar boa em teoria, mas a falha em entender a técnica pode levar a estimativas equivocadas.

Além disso, uma análise mais detalhada das técnicas usadas em processos ágeis poderia ser útil para verificar se essas técnicas trazem algum benefício em relação às técnicas mais tradicionais. Poderia-se analisar como esses métodos percorrem a distância entre o problema a solução para fazer a estimação ou ainda analisar as contradições e os pontos em que os “agilistas” discordam entre si.

REFERÊNCIAS BIBLIOGRÁFICAS

Agile Manifesto website, Disponível em http://agilemanifesto.org/. Acesso em: 18/12/2009.

AMBLER, S. **What Is Agile Modeling (AM)?**, Disponível em www.agilemodeling.com, 2002.

ASTELS, D.; MILLER, G.; NOVAK, M. **Extreme programming: guia prático**, Campus, 2002.

BECK, K. **Extreme Programming Explained: Embrace Change**, Addison-Wesley, 1999.

BFPUG. **Qual a produtividade do JAVA?**, disponível em http://www.bfpug.com.br/Produtividade_Java.htm, 18/01/2013.

BOEHM, B. **Anchoring the Software Process**, IEEE Software, v. 13, n. 4, jul. 1996.

BOEHM, B.; CLARK, B., HOROWITZ, E. et al. **An Overview of the COCOMO 2.0 Software Cost Model**, Software Technology Conference, 1995.

BROOKS, F., **No Silver Bullet: Essence and Accidents of Software Engineering**, Computer, Vol. 20, Nº 4 (Abril de 1987) pp. 10-19.

BROOKS, F., **The mythical Man-Month**, Addison-Wesley, 1975.

CAMPOS L.M.L., LIMA A.S. **Gerenciamento de Projetos de Desenvolvimento de Software com o RUP e o PMBOK**, 2009.

CHIN, G. **Agile Project Management: how to succeed in the face of changing project requirements**. NY: Amacon, 2004.

CLARK, B. **COCOMO II**, PSM Users' Group Conference, disponível em http://www.psmsc.com/UG1998/Presentations/cocomo2%201998.pdf, 1998, Acessado em 26/09/2012.

COHN, M., **Agile Estimating and Planning**, Robert C. Martin Series, Prentice Hall, 2006.

COHN, M. **User Stories Applied: For Agile Software Development**, Addison-Wesley Professional, 2004.

CSSE. **COCOMO II**, Center for Systems and Software Engineering, disponível em http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html, Acessado em 26/09/2012.

HUMPHREY, W. **The Personal Softwares Process (PSP)**, Pittsburg: Software Engineering Institute, 2000.

KNIBERG, H. **Scrum e XP direto das Trincheiras**, InfoQ, 1.ed. 2006

KOSKELA, L. **Test Driven: Pratictal TDD and Acceptance TDD for Java Developers**, Manning: Greenwich, 2008.

KROLL, P., KRUCHTEN, P. **Rational Unified Process Made Easy**, Boston: Addison Wesley, 2003.

KROLL P., MacIsaac B. **Agility and Discipline Made Easy: Practices from OpenUP and RUP**, 2006.

LEITE, J.C. **Requisitos de Software**, disponível em http://engenhariadesoftware.blogspot.com/2007/05/requisitos-de-software.html, 2007, Acessado em 14/12/2011.

MCCONNELL, Steve. **Classic Mistakes**, disponível em http://www.stevemcconnell.com/ieeesoftware/bp05.htm, IEEE Software, Vol. 13, No. 5, September 1996, Acessado em 03/06/2012

MCCONNELL, Steve. **Professional Software Development**, Addison Wesley, 2003.

MCCONNELL, Steve. **Software Estimation: Demystifying the Black Art**, Washington: Microsoft Press, 2006.

PRESSMAN, Roger S. **Engenharia de Software**, 6. ed., São Paulo: Makron Books, 2006.

RATIONAL. Rational Unified Process: Best Practices for Software Development Teams, 2001, disponível em www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf**,** acessado em 03/10/2012.

SCHWABER, K. **Scrum Development Process**, OOPSLA'95: Workshop on Business Object Design and Implementation, Stringer-Verlag, 1995.

SCHWABER, K. **Agile Process Management with Scrum**, Microsoft Press, 2004.

SCHWABER, K. SUTHERLAND, J. **Scrum Guide**, Scrum.Org, 2008.

SOMMERVILLE, I. **Engenharia de Software**, 6ª ed., São Paulo: Addison Wesley, 2003.