IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Alberto Ferreira Delgado

Um Método para Seleção de Processos de Desenvolvimento de Software com Ênfase em XP e RUP

São Paulo

2008

Alberto Ferreira Delgado

Um Método para Seleção de Processos de Desenvolvimento de Software com Ênfase em XP e RUP

Projeto de pesquisa apresentado ao Instituto de Pesquisas Tecnológicas – IPT, para obtenção do título de mestre em Engenharia de Computação.

Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Kechi Hirama

São Paulo

Junho de 2008

DEDICATÓRIA

A minha avó, Ditinha, que me educou e fez parte de todos os melhores momentos de minha infância.

Ao meu pai, Luiz, que me permitiu estudar nas melhores escolas; à minha mãe, Ruth, que sempre me incentivou a estudar.

A minha esposa, Evelise, com quem compartilho as alegrias e tristezas de minha vida.

AGRADECIMENTOS

A Deus, em primeiro lugar, que me possibilitou ter saúde para chegar até o final deste mestrado.

Ao Professor Doutor Kechi Hirama, meu orientador, que me ajudou desde o início a transformar um conjunto de boas idéias em um trabalho coeso e objetivo; Agradeço principalmente a paciência que ele teve em como me orientar na adequação de idéias que não se aplicavam bem no escopo do trabalho.

Aos membros da banca, Professor Doutor Reginaldo Arakaki e Professor Doutor Vilmar Pedro Votre, que com suas importantes e minuciosas observações e críticas, ajudaram a concluir esta dissertação;

A minha esposa, Evelise Delgado, pela paciência com que suportou os longos períodos de estudo e em alguns casos de ausência demandados pelo mestrado;

Aos professores do IPT, pelo conhecimento que pude adquirir ao longo do curso;

A todos os funcionários do mestrado profissional do IPT, que em algum momento me auxiliaram a chegar ao final deste trabalho.

A todos os meus mais sinceros agradecimentos.

Resumo

Os processos ágeis têm sido freqüentemente citados como uma alternativa viável para se produzir software mais rápido sem perda de qualidade. Porém, é necessário analisar o ambiente onde o software está sendo desenvolvido, antes de se optar pelo uso do processo mais adequado.

Neste trabalho é proposto um método para seleção do processo de desenvolvimento mais adequado, dado o ambiente organizacional onde o software está sendo desenvolvido. Assim, é possível identificar antes do inicio de um projeto se é melhor utilizar um processo ágil, como XP, ou um processo mais tradicional, como RUP. Apresenta-se também um exemplo de utilização do método de seleção proposto.

Palavras-chave: Método de Seleção; Desenvolvimento de Software; Processo de Desenvolvimento de Software; Processos orientados a planejamento; Processos Tradicionais; Processos Ágeis; Métodos Ágeis; XP; RUP; Extreme Programming; Rational Unified Process

Abstract

A Software Development Process Selection Method with XP and RUP emphasis

Agile methods have been usually referenced as a feasible choice to produce high quality software faster. Furthermore, it is necessary to analyze the work environment, where the software is going to be developed, before choosing the appropriate development process.

This paper describes a method to allow a proper selection of the development process, considering the work environment where the software will be developed. This method allows identifying when an agile process, like XP, could be considered more appropriate than a traditional one, like RUP. This paper also describes a utilization example of the proposed selection method.

Keywords: Selection Method; Software Development; Software Development Process; Plan-Driven Process; Traditional Process; Agile Process; Agile Methods; XP; RUP; Extreme Programming; Rational Unified Process

Lista de Ilustrações

Lista de Tabelas

Lista de Siglas e Abreviaturas

ASD	Agile Software Development
CIO	Chief Information Officer
DSDM	Dynamic Systems Development Model
FDD	Feature Driven Development
ISO	International Organization for Standardization
RUP	Rational Unified Process
RAD	Rapid Application Development
TI	Tecnologia da Informação
XP	Extreme Programming

Sumário

1 Introdução

1.1 Motivação

Nos últimos anos, ocorreu um rápido crescimento do interesse nos chamados processos ágeis (FOWLER, 2005). Esse crescimento foi ocasionado pela necessidade, percebida pelos desenvolvedores, de retirar passos desnecessários dos processos mais tradicionais e maximizar a velocidade de entregas que agreguem valor ao negócio (LITTLE, 2005) (BECK, 2001).

Entre os processos ágeis, XP (*Extreme Programming*) é reconhecidamente o mais popular e também o que possui mais estudos de caso a seu respeito. Ele é um processo que possui boa documentação de suas práticas, o que ajuda em sua implantação inicial (UDO; KOPPENSTEINER, 2003) (FOWLER, 2005).

Processos tradicionais, como o RUP, são descritos como um conjunto de atividades e artefatos para desenvolvimento de software e sistemas computacionais. Na implantação destes processos, em um determinado ambiente, devem-se definir quais atividades e artefatos são adequados. Esses processos tradicionais foram criados para atender a qualquer tamanho de equipe e projeto, mas nem sempre definem claramente quando escolher uma atividade ou um determinado artefato. Em um projeto que utiliza o RUP, encontrar as atividades e artefatos que são realmente necessários é uma tarefa difícil, que consome muito tempo até sua completa adaptação. Esta adaptação necessita da experiência dos desenvolvedores no processo e pode, em alguns casos, até inviabilizar sua implantação inicial (KROLL; KRUCHTEN, 2003) (COCKBURN, 2000). Por outro lado, os processos ágeis apresentam uma série de práticas que devem ser seguidas para se obter o sucesso, sem necessidade de grandes adaptações, o que facilita sua implantação inicial. (BECK, 2004)

No entanto, apenas adotar um processo ágil sem uma análise inicial pode se tornar mais um problema do que uma solução, pois mesmo os processos ágeis possuem situações de falhas que, quando não corretamente conhecidas e tratadas, podem levar o projeto ao fracasso. Por esta razão, os processos tradicionais também precisam continuar sendo considerados como uma alternativa ao desenvolvimento de sistemas computacionais. Isto porque, se por um lado, os processos ágeis, como XP, prometem aumento na satisfação do cliente, menor taxa de defeitos e menor tempo para o desenvolvimento do software, os processos tradicionais, como o RUP, procuram dar previsibilidade, estabilidade e alto grau de segurança. Ambas as abordagens têm seus pontos fortes e fracos e dependendo do ambiente de trabalho, um processo pode ser mais

adequado que o outro. Não considerar essas diferenças de ambiente, na escolha do processo a ser usado, pode levar qualquer projeto ao fracasso. (BOEHM; TURNER, 2003)

Definir um único processo para todos os projetos dentro de uma empresa não é uma boa prática no desenvolvimento de software. Encontrar o processo mais adequado, de acordo com o ambiente organizacional analisado, é a resposta a ser encontrada para cada novo projeto de software. (LINDVALL; RUS, 2000) (LITTLE, 2005) (BOEHM; TURNER, 2003) (COCKBURN, 2000).

1.2 Objetivo

Este trabalho tem como objetivo propor um método para a seleção do processo de desenvolvimento mais adequado para uma organização, logo no inicio da etapa de desenvolvimento de um projeto de software.

Não é escopo deste trabalho definir ou tratar a implantação do processo indicado, o método deve ser usado como uma ferramenta de apoio na seleção do processo mais adequado, seguindo fatores bem definidos.

1.3 Resultados Esperados e Contribuições

O impacto do uso de software nas organizações e na cultura da sociedade continua intenso. Sua importância aumenta continuamente, e a comunidade de engenharia de software procura constantemente processos para desenvolver melhor, mais rápido e com menos custo (PRESSMAN, 2001). Por essas razões, encontrar um método que permita uma seleção antecipada do processo de desenvolvimento mais adequado é uma contribuição essencial para a comunidade de engenharia de software.

Alguns autores sugerem que ter uma diversidade de processos de desenvolvimento é algo importante para a Engenharia de Software. Sendo assim, uma contribuição pretendida é colaborar nesta discussão, mostrando por meio de uma análise crítica que apenas mudar para um tipo de processo de desenvolvimento não é a única solução viável para as organizações. (LINDVALL; RUS, 2000) (BOEHM, 2002).

Este trabalho espera obter um método de seleção para permitir a escolha objetiva entre dois tipos de processos de desenvolvimento. Ambos devem se mostrar bastante adequados à realidade atual das organizações. É pretendido também que o uso conjunto dos dois processos

em uma mesma empresa, com a seleção do mais adequado no início de cada projeto, permita atender melhor a diversidade de características de projetos dentro da organização.

1.4 Metodologia

Para a elaboração deste trabalho, o primeiro passo foi definir seu objetivo. Após claramente definido este objetivo, foi iniciada uma pesquisa sobre os modelos e tipos de processos de desenvolvimento que podem ser considerados adequados às necessidades atuais das organizações. Chegou-se então a escolha de dois deles, apresentando o motivo da opção por cada um.

Uma vez escolhidos os dois tipos de processos de desenvolvimento, que em conjunto podem alcançar uma maior abrangência na solução dos problemas atuais das organizações, estes são descritos de maneira resumida, para se analisar se realmente são exemplos adequados como processos-alvo do método de seleção.

Após a escolha de XP e RUP como exemplos de processos de desenvolvimento, uma pesquisa foi realizada a fim de se obter os conceitos para a seleção entre processos ágeis e processos tradicionais. O uso destes conceitos deu o embasamento necessário na seleção entre estes dois tipos de processo, descritos no presente trabalho.

Uma vez finalizado o levantamento das referências bibliográficas, o método é descrito de forma a permitir a seleção entre processos de desenvolvimento com base em fatores que possam ser levantados no início de cada projeto.

Por fim, é apresentado um exemplo de utilização do método de seleção proposto, por meio de casos reais de XP e RUP. Permitindo então a conclusão final do trabalho e a análise dos resultados obtidos.

1.5 Organização do Trabalho

No capítulo 1, Introdução, são descritos: a motivação, o objetivo, os resultados esperados, a metodologia de pesquisa e a organização do trabalho.

No capítulo 2, Processos de Desenvolvimento de Software, é apresentado uma visão geral dos processos de desenvolvimento de software.

No capítulo 3, *Extreme Programming* (XP), são descritos os valores, princípios e práticas do XP, utilizado como exemplo de processo ágil a ser escolhido.

No capítulo 4, *Rational Unified Process* (RUP), é descrito o RUP, utilizado como exemplo de

processo tradicional a ser escolhido.

No capítulo 5, Fatores para Seleção de Processos, são descritos os principais conceitos utilizados pelo método de seleção de processo deste trabalho, que permitem justificar a seleção de um processo ágil ou tradicional.

No capítulo 6, Método para Seleção de Processo de Desenvolvimento de Software, são mostrados os passos que orientam a escolha do processo mais adequado no desenvolvimento de um projeto de software. É mostrada também uma análise dos fatores usados na seleção, em comparação com as práticas e atividades dos processos XP e RUP.

No capítulo 7, Aplicação do Método de Seleção, é mostrado um exemplo de utilização do método de seleção.

No capítulo 8, Conclusões, são apresentados um resumo do trabalho, uma análise geral dos resultados obtidos e sugestões para futuras pesquisas.

2 Processos de Desenvolvimento de Software

2.1 Introdução

Este capítulo inicia com a descrição dos modelos de processos usados para o desenvolvimento de software. Após a apresentação dos principais modelos de desenvolvimento, alguns processos são apresentados e discutidos. Com base nesta discussão inicial, eles são classificados quanto a seu grau de formalismo e ciclo de desenvolvimento. Por fim, são definidos os processos de desenvolvimento que fazem parte deste trabalho, com a justificativa da escolha de cada um deles.

2.2 Modelos de Desenvolvimento de Software

Um processo de desenvolvimento de software pode ser definido como um conjunto de tarefas necessárias para se obter um software de alta qualidade. Os processos de desenvolvimento compõem a base de sustentação para a engenharia de software (Figura 1). Na engenharia de software, são os processos que garantem que as tecnologias aplicadas durante o desenvolvimento funcionam bem em conjunto, permitindo desta forma um desenvolvimento controlado e dentro de um tempo aceitável. Os processos também permitem manter o foco em qualidade, fator essencial na engenharia de software (PRESSMAN, 2001).

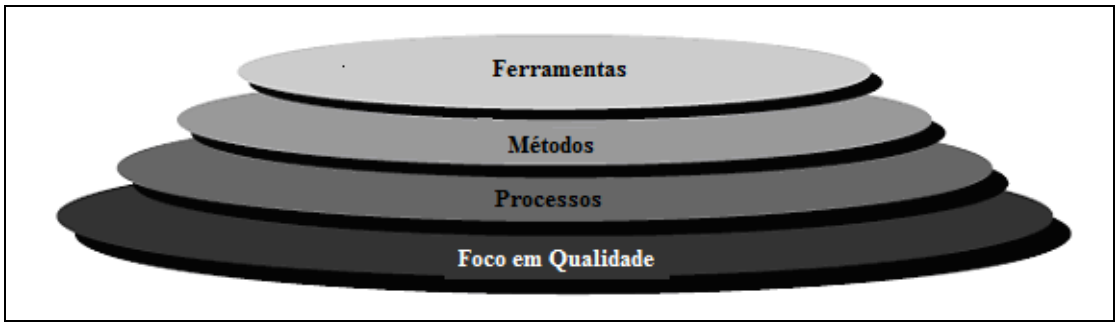

Figura 1 – As camadas da engenharia de software - Fonte: (PRESSMAN, 2001)

Os métodos na engenharia de software descrevem, de maneira mais técnica, como realizar cada uma das atividades do processo. Eles contêm o conjunto de tarefas a serem realizadas incluindo análise, projeto, construção, teste e manutenção. As ferramentas na engenharia de software

provêem suporte automatizado ou semi-automatizado para os métodos e processos (PRESSMAN, 2001).

Na engenharia de software descrevem-se também modelos de processos de desenvolvimento. Modelos estes que permitem um adequado desenvolvimento de um projeto de software; os mais comuns são descritos a seguir.

2.2.1 O modelo seqüencial ou linear

Este modelo, também conhecido como modelo cascata ou ciclo de vida clássico, sugere uma abordagem seqüencial e sistemática ao desenvolvimento de software. Devido ao fato de um software ser sempre parte de um sistema maior, torna-se essencial o entendimento de todos os elementos do sistema, para derivar então um subconjunto de requisitos de software. As atividades de levantamento dos requisitos de sistema são denominadas de engenharia de requisitos, que engloba as etapas de análise e projeto no modelo linear. Para os requisitos que demandam o desenvolvimento de software, o ciclo é então continuado por meio da codificação e testes. A figura 2 ilustra esse modelo linear ou seqüencial (PRESSMAN, 2001).

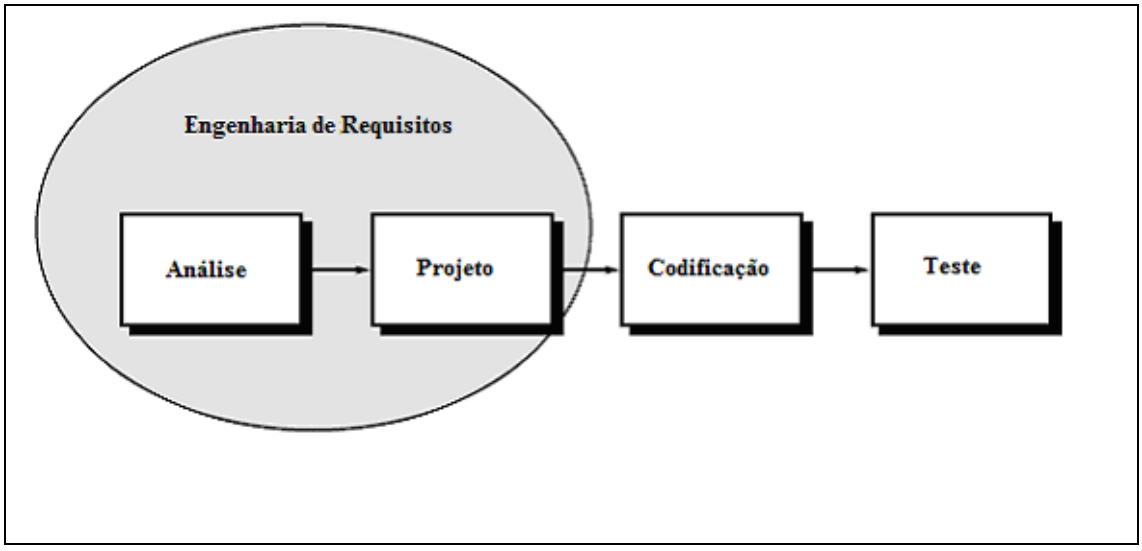

Figura 2 – O modelo seqüencial ou linear – Fonte: (PRESSMAN, 2001)

2.2.2 O modelo baseado em prototipação

Alguns problemas comuns na fase inicial de um projeto: o cliente define um conjunto de objetivos para o software, mas não é capaz de definir de maneira detalhada os requisitos; o desenvolvedor tem dúvidas sobre a eficácia de um algoritmo. Nestes casos e em vários outros uma abordagem baseada em prototipação pode ser a melhor escolha (PRESSMAN, 2001).

O paradigma da prototipação inicia-se com o levantamento dos requisitos com o cliente. O desenvolvedor e o cliente definem os objetivos gerais do software, identificam os requisitos já conhecidos e quais áreas ainda precisam de melhor definição. Um projeto de software rápido é realizado e um protótipo é então criado para validar ou entender melhor algum ponto do sistema. O protótipo é testado em conjunto com o cliente e é então usado para refinar os requisitos. As iterações ocorrem conforme o protótipo vai sendo ajustado, para se adequar às necessidades do cliente; ao mesmo tempo, permite ao desenvolvedor entender melhor o que ele precisa desenvolver (Figura 3) (PRESSMAN, 2001).

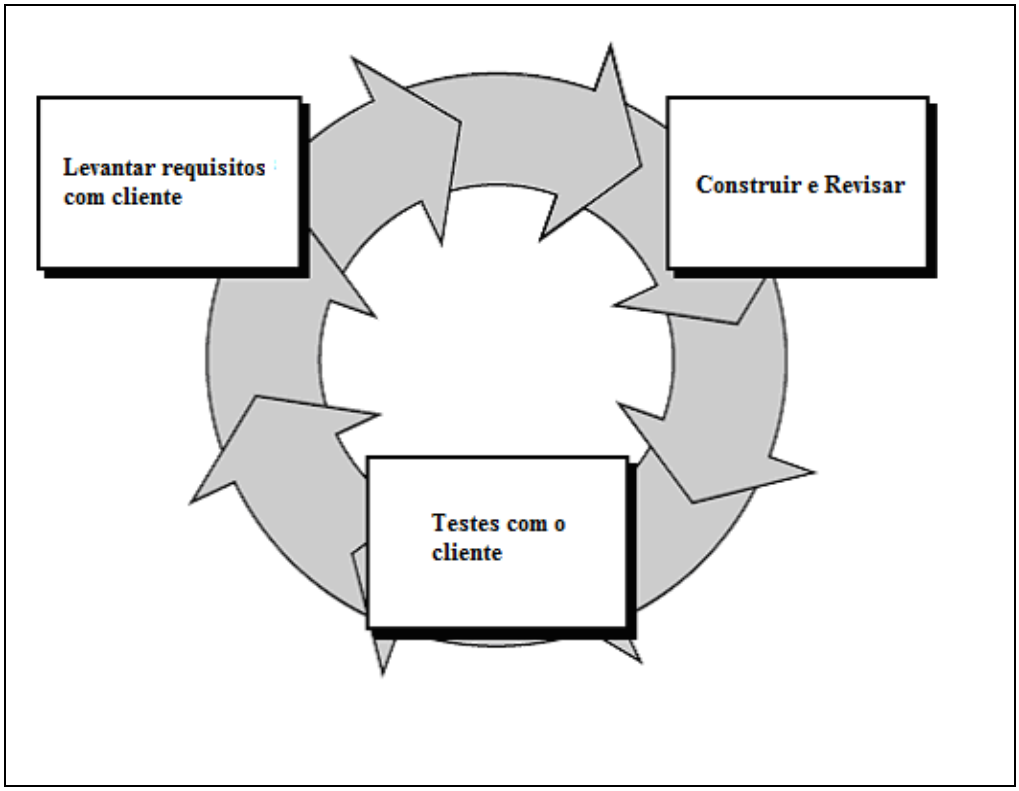

Figura 3 - O modelo baseado em prototipação – Fonte: (PRESSMAN, 2001)

O protótipo tem como objetivo principal identificar melhor os requisitos, mas caso o protótipo gere alguma versão funcional de software, pode ser usado como sistema inicial. Porém, o ideal é fazer o protótipo rapidamente não se preocupando se pode ser aproveitado depois, e sim, que ele logo esclareça os requisitos que precisam ser mais bem definidos (PRESSMAN, 2001).

2.2.3 O modelo RAD *(Rapid Application Development)*

RAD é um modelo incremental de processo de desenvolvimento que enfatiza um ciclo extremamente pequeno de desenvolvimento. O RAD é uma adaptação em alta velocidade do modelo seqüencial. A rapidez deve ser alcançada usando-se uma construção baseada em componentes. Se os requisitos estão bem entendidos e o escopo bem definido, o RAD permite a

criação de um sistema completamente funcional em 60 a 90 dias (PRESSMAN, 2001). O RAD possui as seguintes fases (Figura 4):

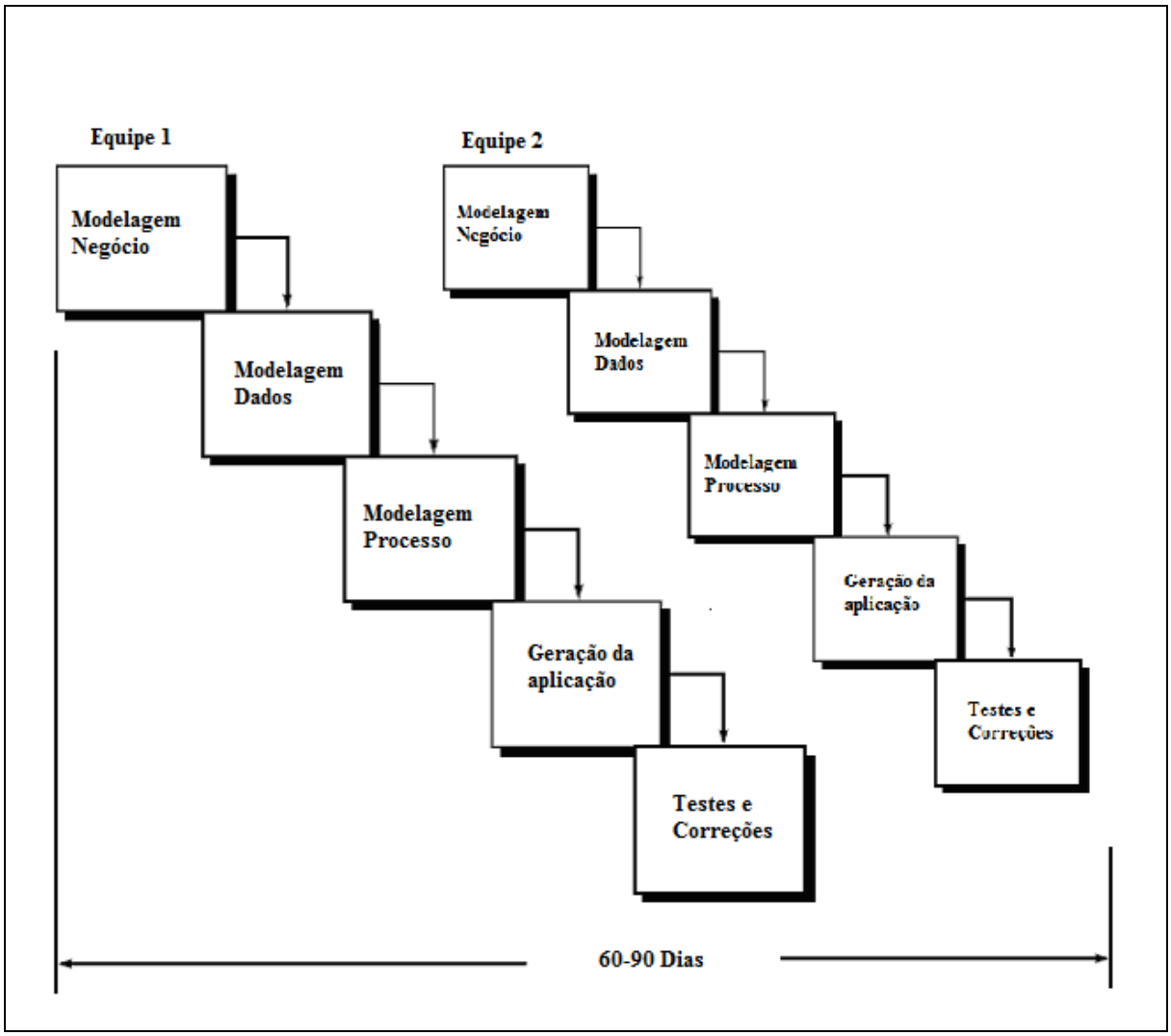

Figura 4 – O modelo RAD – Fonte: (PRESSMAN, 2001)

Modelagem de negócio: O fluxo de informação é modelado de forma a responder às questões: que informações direcionam o processo de negócio? Quem gera estas informações? Onde as informações devem ser levadas? Quem processa estas informações?

Modelagem de dados: Os fluxos de informação definidos pela modelagem de negócio são mapeados como objetos de dados, necessários para suportar o negócio.

Modelagem de processo: Os objetos de dados, definidos na modelagem de dados, são transformados para implantar o fluxo necessário para uma determinada função do negócio.

Geração da aplicação: O RAD é baseado em técnicas de geração automática de software com o uso de ferramentas que permitam a especificação em alto nível do sistema, sem a necessidade

de uso das linguagens de programação convencionais (Java, C++, etc.). RAD parte da premissa de alta reutilização de componentes e criação de componentes que possam ser reutilizados.

Testes e correções: Como o RAD é baseado em reuso, boa parte da etapa de testes já deve ter sido realizada anteriormente, permitindo reduzir o tempo de teste.

O RAD permite que um sistema seja dividido entre várias equipes de desenvolvimento em paralelo, cada uma criando um componente que deve ser integrado depois ao sistema (PRESSMAN, 2001).

2.2.4 O modelo de software evolutivo

Existe um crescente reconhecimento de que o software evolui ao longo de um período de tempo. Os requisitos de negócio e do produto mudam freqüentemente conforme o desenvolvimento vai sendo realizado; portanto, criar um planejamento linear considerando todo o ciclo de desenvolvimento do produto, não se mostra realista. Os prazos exíguos solicitados para atender necessidades dos clientes tornam necessário um desenvolvimento evolutivo, onde um conjunto principal de funções possa ser disponibilizado com rapidez. O modelo evolutivo é baseado em criar várias iterações de ciclo de desenvolvimento, permitindo que o produto seja completado de maneira incremental (PRESSMAN, 2001). Existem alguns modelos derivados deste modelo evolutivo descritos a seguir.

2.2.4.1 O modelo incremental ou iterativo

O modelo incremental, também denominado modelo iterativo, combina elementos do modelo linear ou seqüencial - aplicados repetidas vezes - com a característica iterativa da prototipação. Como ilustrado na figura 5, o modelo incremental aplica diversas iterações lineares gerando, no final de cada incremento, um software completamente funcional, mas que ainda não possui todos os requisitos do produto final. Esse software gerado pode ser entregue em ambiente de produção e ser gradualmente incrementado conforme as novas iterações vão sendo realizadas. Por exemplo, no desenvolvimento de um editor de textos as funções básicas de edição são entregues no primeiro incremento (iteração), incluindo o gerenciamento de arquivos e a edição de documentos. No segundo incremento, são incorporadas a verificação gramatical e algumas funções avançadas de edição. Este modelo é continuado por quantos incrementos forem necessários até ter o produto final (PRESSMAN, 2001).

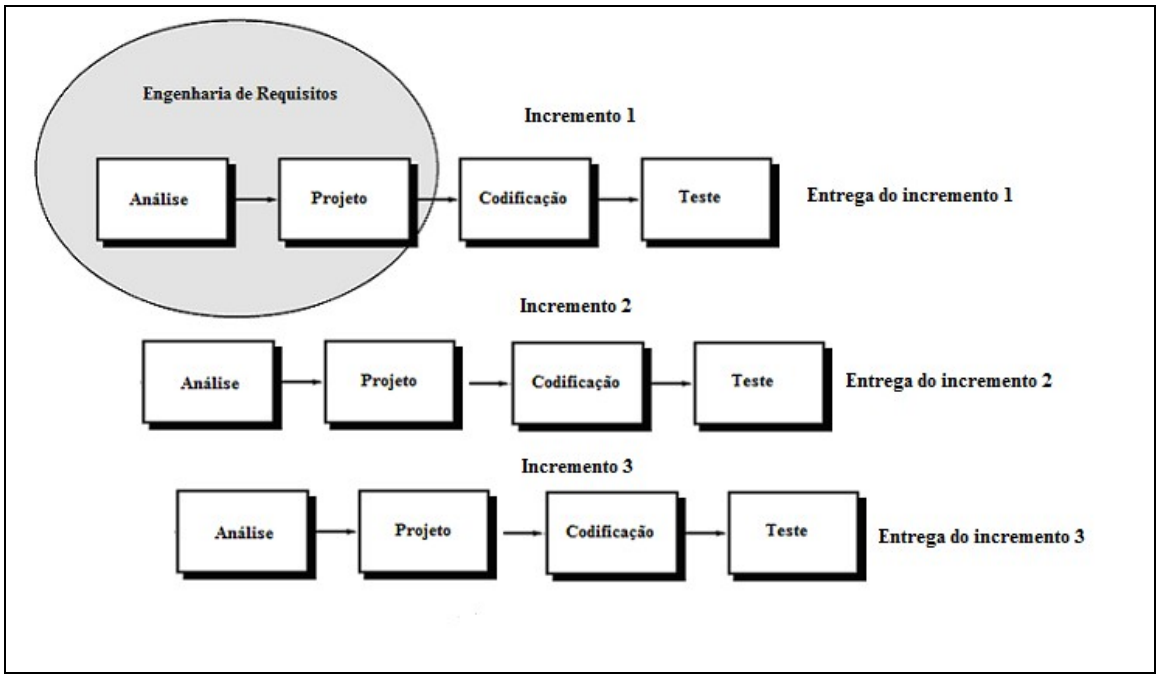

Figura 5 – O modelo incremental – Fonte: (PRESSMAN, 2001)

O modelo incremental, assim como o modelo baseado em prototipação, é iterativo por natureza. Esta iteratividade permite que os pontos ainda incertos do produto possam ser mais bem definidos conforme o sistema é desenvolvido. As funções básicas e já bem definidas fazem parte do primeiro incremento. As demais funções são distribuídas pelos outros incrementos de acordo com a prioridade definida. Porém, diferentemente do modelo baseado em prototipação, o modelo incremental tem seu foco na produção de versões funcionais do produto ao final de cada incremento e não apenas na validação dos requisitos funcionais (PRESSMAN, 2001).

2.2.4.2 O modelo em espiral

O modelo em espiral combina a natureza iterativa da prototipação com aspectos de controle do modelo seqüencial. Nesse modelo, o software é desenvolvido de maneira incremental, sendo que nos primeiros incrementos a entrega pode ser um projeto em papel ou um protótipo do sistema. O modelo em espiral é dividido em diversas tarefas, cada uma pertencente a um determinado grupo de tarefas. Esses grupos são chamados regiões de tarefas; normalmente existem entre três e seis regiões de tarefas (PRESSMAN, 2001).

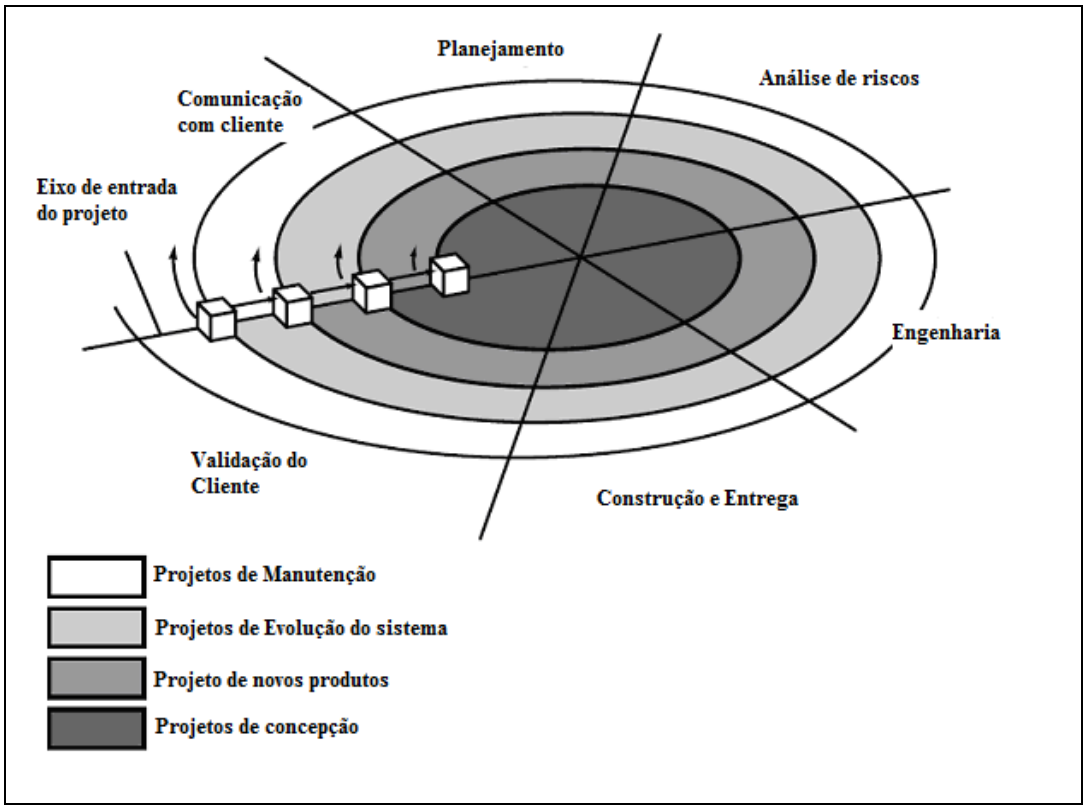

Figura 6 – O modelo em espiral – Fonte: (PRESSMAN, 2001)

A figura 6 é exemplo de um modelo em espiral com seis regiões, que são divididas na seguinte ordem de execução: Comunicação com o cliente, Planejamento, Análise de riscos, Engenharia ou Projeto, Construção e Entrega e a Validação do cliente. Cada uma das regiões possui um conjunto de tarefas que, dependendo do tamanho do projeto, podem ser mais ou menos formais e o processo de desenvolvimento é iniciado pela comunicação com o cliente. A equipe deve, então, conduzir o projeto seguindo a espiral no sentido horário, a partir do ponto mais próximo do centro da espiral. A primeira volta em torno da espiral pode resultar em uma especificação do produto. A segunda passagem pode produzir um protótipo do sistema. As passagens subseqüentes podem gerar versões mais completas do software final (PRESSMAN, 2001).

2.3 Processos de desenvolvimento de software

A indústria de software reconhece que realizar o desenvolvimento sem controle pode resultar em falhas, na entrega do produto final. A busca por uma forma estruturada de desenvolvimento de software levou à criação de processos, em princípio baseados no modelo em cascata (HARRIS; HEVNER; COLLINS, 2006).

Ao longo do tempo, muitos processos foram criados com o objetivo de estruturar o processo de desenvolvimento de software, alguns já nem são discutidos atualmente. Um estudo recente

mostrou que, 85% dos responsáveis pelas áreas de TI nas empresas (CIO), indicaram a agilidade como parte essencial de suas estratégias de negócio. Em contrapartida, outros estudos apontam que apenas ter agilidade, sem controle, pode representar o fracasso de qualquer projeto (HARRIS; HEVNER; COLLINS, 2006). A partir de pesquisas como estas, foi percebida a necessidade de ter-se um processo de desenvolvimento de software que responda rapidamente às mudanças, mas que permita o controle necessário sob sua execução.

Apesar da necessidade de agilidade nas organizações, por uma variedade de razões, nem todos já se encontram preparados para usar realmente os processos ágeis por completo. De fato, apenas 17% das empresas nos Estados Unidos e Europa usam processos ágeis no desenvolvimento de software, de acordo com um estudo feito pela empresa *Forrester Research's* em uma pesquisa chamada "*Enterprise Agile Adoption in 2006*" (WAILGUM, 2007). Porém, outra pesquisa, feita nos Estados Unidos, indica que pouco mais de 50% dos desenvolvedores já utilizam processos ágeis no seu dia-a-dia, e outra grande parcela ainda utiliza outros tipos de processos (SCHINDLER, 2008). Devido a essa divisão de abordagens, existe muita discussão para se indicar qual o processo ideal para cada organização (BOEHM; TURNER, 2003). Outro ponto amplamente discutido é como controlar essa agilidade de forma a não transformar o desenvolvimento em algo sem controle (*ad hoc*), como existia antes da criação do ciclo em cascata (HARRIS; HEVNER; COLLINS, 2006).

Levando-se em conta os motivos descritos acima, o modelo de desenvolvimento iterativo (incremental) é o que se enquadra melhor nas necessidades atuais das organizações (HARRIS; HEVNER; COLLINS, 2006). Dentro deste modelo existem dois grupos de processos que devem ser considerados: os tradicionais, com foco no planejamento, e os ágeis, com foco na execução. Desta forma, ambos podem ser considerados adequados ao cenário atual das organizações (BOEHM; TURNER, 2003). Considerando-se os principais processos em uso atualmente, todos permitem o uso de várias iterações até o término do projeto. No entanto, é importante que cada iteração realize todas as etapas do processo para ser considerada completa, podendo usá-las em maior ou menor grau, mas passando por todas até a entrega de algum software testado e instalado (FOWLER, 2005). O ponto que precisa ser avaliado, então, é quando utilizar um processo baseado em planejamento (processo tradicional), ou quando usar um processo com foco na execução, os chamados processos ágeis (BOEHM; TURNER, 2003).

2.3.1 Processos tradicionais (foco no planejamento)

Os processos tradicionais, baseados em planejamento, estão em uso há muitos anos nas organizações. Entre suas principais características estão: dar previsibilidade, manter a estabilidade e dar confiança em relação aos prazos de entrega acordados. (BOEHM; TURNER, 2003). Entres estes processos baseados em planejamento, alguns exemplos são descritos a seguir:

- **RUP –** O RUP (*Rational Unified Process*) foi criado no final dos anos 80 pela *Rational Software Corporation*, é um processo de engenharia de software que define claramente a responsabilidade de cada pessoa participante do processo. O RUP é centrado no planejamento de arquitetura e baseado em casos de uso. Neste processo é definido claramente o ciclo de vida de um projeto e todos os principais marcos de decisão a ser considerados (KROLL;KRUCHTEN, 2003).
- ***Cleanroom*** – É um processo baseado em planejamento com foco principal na equipe. Permite tornar o desenvolvimento de software mais gerenciável e previsível, porque é baseado em um controle estatístico de qualidade. A filosofia do *cleanroom* é evitar dependência com dispendiosos processos de remoção de defeitos, concentrando-se em escrever o código correto, já na primeira vez, e verificando se está correto antes mesmo dos testes (LINGER, 1994).

2.3.2 Processos ágeis (foco na execução)

Em contrapartida aos processos com foco no planejamento surgiram os chamados processos ágeis, com foco na execução e na entrega rápida de software funcional, procurando dar uma resposta mais ágil às mudanças. Entre estes processos ágeis podem-se destacar os seguintes:

- ***Extreme Programming (XP) –*** Criado por Kent Beck, é o processo ágil mais conhecido e o que gera maior interesse. XP tem como foco o desenvolvedor de software e iterações freqüentes com o cliente. XP também define regras para o ambiente ideal de desenvolvimento: Toda equipe em uma mesma sala, equipes de dez ou menos desenvolvedores e cliente dedicado em tempo integral ao projeto. Assim como os demais processos ágeis, o desenvolvimento ocorre em diversas pequenas iterações (três semanas ou menos) (UDO; KOPPENSTEINER, 2003) (FOWLER, 2005).
- ***Crystal Methods*** **–** Desenvolvido por Alistair Cockburn, possui um conjunto de atividades representado por uma matriz entre o tamanho da equipe, as prioridades do

projeto e a complexidade do sistema, que permite definir qual processo da família *Crystal* utilizar. Embora este processo sirva como um guia, as atividades que devem ser seguidas são selecionadas pela equipe do projeto. Neste processo, a comunicação e as pessoas devem estar em primeiro lugar e o processo em segundo plano. Apresenta um controle limitado sobre o andamento do projeto. Existem apenas duas regras absolutas neste processo: Use sempre um desenvolvimento iterativo e incremental; faça seminários ou reuniões para discutir cada prática adotada deste processo e como melhorar. Nesta família de processos o que pode ser considerado efetivamente como processo ágil é o *Crystal Clear* (UDO; KOPPENSTEINER, 2003) (FOWLER, 2005) (COCKBURN, 2004).

- ***Adaptive Software Development*** **(ASD) –** desenvolvido por Jim Highsmith, foi inspirado pelo chamado *Rapid Application Development* (RAD). ASD foi criado para funcionar em ambientes complexos e de muitas incertezas. ASD reconhece que as falhas, ao longo do projeto, são naturais e que as equipes devem continuamente se adaptar durante o projeto às situações específicas. O ciclo de vida no ASD envolve um processo iterativo de especulação, colaboração e aprendizado. O planejamento ocorre na fase inicial de especulação, o desenvolvimento do sistema ocorre na fase de colaboração. A fase de aprendizado ocorre durante todo o projeto, de forma que a equipe se adapte aos problemas que aparecem no decorrer do desenvolvimento, melhorando os processos adotados. O papel do gerente de projeto é o de facilitar a colaboração dentro da equipe (UDO; KOPPENSTEINER, 2003) (FOWLER, 2005).

- ***Scrum*** **–** desenvolvido por Ken Schwaber e Jeff Sutherland, este é o processo que possui mais ênfase no gerenciamento de projetos quando comparado aos outros processos ágeis existentes. Controlar projetos distintos requer diferente técnicas de controle, para os quais *Scrum* usa processos de monitoração bem definidos e rotinas de realimentação constante. O desenvolvimento ocorre em ciclos normalmente de 30 dias chamados *sprints.* Diariamente reuniões de acompanhamento do projeto são realizadas (duração de 30 minutos ou menos). Nessas reuniões são identificados os obstáculos que podem comprometer o andamento do projeto. Relatórios de andamento das atividades são gerados. Após a definição do que deve ser feito na próxima iteração (*sprint),* nenhum requisito mais pode ser alterado para o mesmo *sprint* (UDO; KOPPENSTEINER, 2003) (FOWLER, 2005).

- ***Feature Driven Development (FDD)*** **–** Diferentemente dos demais processos ágeis, FDD é baseado em processos repetitivos. FDD possui uma ênfase maior na modelagem de arquitetura que os outros processos ágeis. Os projetos se iniciam com um protótipo que é continuamente melhorado conforme o projeto amadurece, transformando-se no produto final. Cada função é definida como uma necessidade do negócio e é priorizada, de forma que possa ser entregue em iterações de duas semanas ou menos. FDD possui boa ênfase no gerenciamento do projeto e os requisitos são controlados durante todo o ciclo de vida do projeto. Variações de uma função que ultrapassem 10% devem ser reportadas ao gerente do projeto para que sejam tomadas decisões de escopo (UDO; KOPPENSTEINER, 2003) (FOWLER, 2005).
- ***Dynamic Systems Development Model (DSDM)*** **–** Criado na Inglaterra, na metade da década de 1990, também se baseou no RAD. DSDM é controlado por um consórcio denominado *"DSDM Consortium of vendors and experts in the field of Information System (IS) development"*, possui extensa documentação, serviços e suporte associado. Embora esteja disponível apenas para membros do consórcio, a entrada como membro é relativamente acessível para qualquer um. DSDM é o único processo ágil compatível com a norma ISO 9000. Isso faz com que esse processo se torne atrativo para empresas que precisam se adequar às regras da ISO. DSDM possui três áreas principais: Modelagem funcional, Projeto e a implementação. Existem pequenos processos dentro de cada uma destas áreas, mas todos são iterativos (UDO; KOPPENSTEINER, 2003) (FOWLER, 2005).

2.4 Definição dos processos a serem usados no método de seleção

Considerando as necessidades atuais nas organizações, os processos iterativos se mostram os mais adequados; dentre estes existem os processos tradicionais com foco no planejamento, e os processos ágeis. O método proposto por este trabalho tem como objetivo a escolha entre processos tradicionais em comparação com processos ágeis, para cada novo projeto a ser iniciado. Portanto, é necessária a escolha de pelo menos um processo de cada um destes tipos para servir de exemplo dentro do método de seleção.

Para a escolha dos processos a serem usados como exemplo dentro do método de seleção, deve-se antes classificar alguns dos processos descritos. Quando se comparam processos, devem-se levar em conta dois aspectos principais: nível de formalismo e ciclo de desenvolvimento. Classificar e comparar estes processos dentro destes dois eixos permite ver se estão dentro do

ciclo de desenvolvimento e nível de formalismo adequado ao estudo desse trabalho. Estes fatores podem ser descritos da seguinte forma:

Nível de formalismo: Exibido no eixo horizontal da figura 7, este fator determina o nível de documentação e rastreabilidade exigido pelo processo (KROLL;KRUCHTEN, 2003).

Ciclo de desenvolvimento: Exibido no eixo vertical da figura 7. No caso do ciclo em cascata determina uma abordagem linear, com integração e testes apenas no final do desenvolvimento. O modelo iterativo é uma abordagem direcionada pela análise dos riscos, com integração e testes sendo realizados durante todo o desenvolvimento (KROLL;KRUCHTEN, 2003).

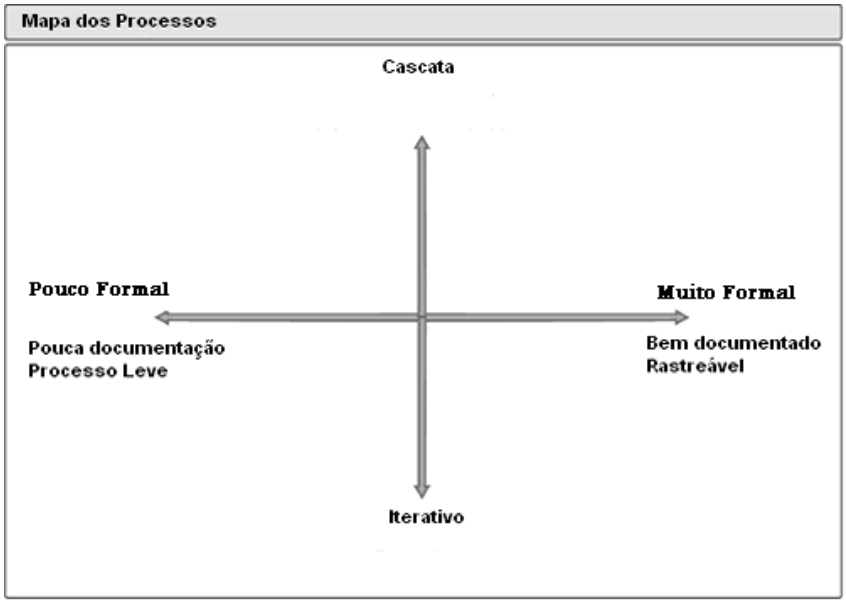

Figura 7 – Comparação entre processos – Fonte: (KROLL;KRUCHTEN, 2003)

Por ser um processo amplamente conhecido, o RUP foi a primeira opção para se analisar como exemplo de processo tradicional. Considerando-se apenas a configuração mais formal do RUP, ele pode ser efetivamente classificado com um alto grau de formalismo, mas que permite o uso de iterações (KROLL;KRUCHTEN, 2003) (COCKBURN, 2000). Porém, configurações mais leves do RUP oferecem um contraste interessante, pois apesar de ser um processo com foco no planejamento, permite o aprendizado durante a execução, típico dos processos ágeis. Devido a essa característica até certo ponto híbrida, o RUP foi escolhido para esse estudo, pois se aproxima muito bem das necessidades de flexibilidade demandadas pelas organizações, mas preserva o foco no planejamento. (HARRIS; HEVNER; COLLINS, 2006). A figura 8 ilustra as classificações do RUP em relação aos 2 aspectos de comparação mostrados.

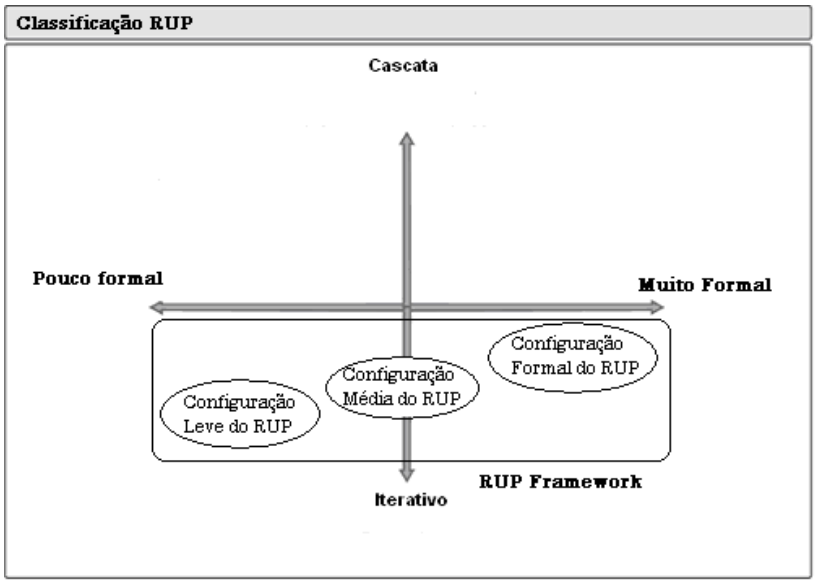

Figura 8 – Classificação do RUP - Fonte: (KROLL;KRUCHTEN, 2003)

Em oposição aos tradicionais, os processos ágeis procuram ser mais um método do que um processo, definindo um guia de práticas que devem ser seguidas. Estas, em conjunto, permitem obter a agilidade desejada sem perda de qualidade. A classificação dos processos ágeis em relação aos aspectos de comparação descritos anteriormente é mostrada na figura 9.

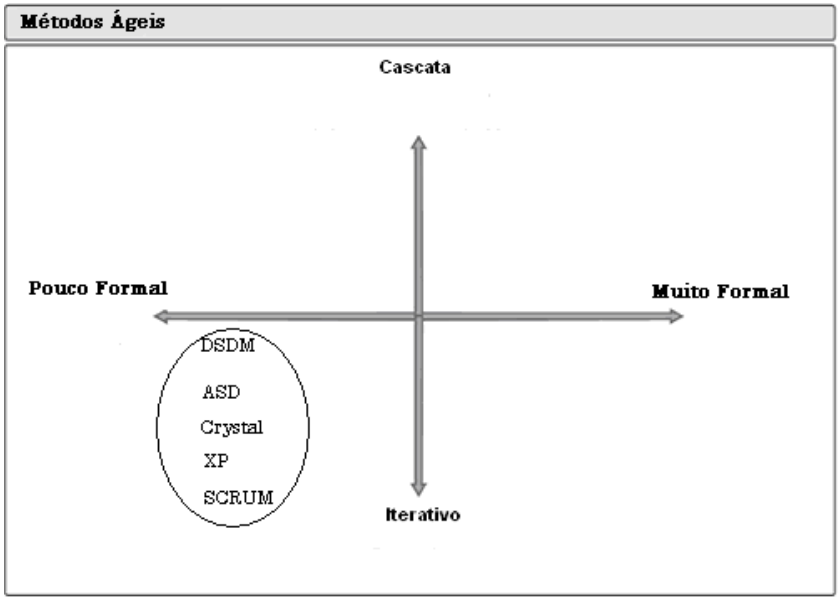

Figura 9 – Classificação dos processos ágeis – Fonte: (KROLL;KRUCHTEN, 2003)

Todos os processos ágeis são classificados, sob o aspecto de formalismo, como pouco formais e altamente iterativos e podem em princípio ser escolhidos. Porém, como XP é o mais conhecido e o que possui mais estudos de caso a seu respeito (SALO, 2004), foi o processo ágil escolhido para este trabalho.

2.5 Considerações finais do capítulo

Conforme visto neste capítulo, existem muitos processos de desenvolvimento de software. Cada um deles tem características únicas, mas de uma maneira geral são formas diferentes para tentar estruturar o desenvolvimento de software, com o objetivo de diminuir ao máximo os problemas deste tipo de indústria. Existem dois grandes grupos de processos de desenvolvimento: os processos tradicionais (foco no planejamento), que tem maior formalismo e menor iteratividade, e os processos ágeis (foco na execução) com menos formalismo e muita iteratividade.

Para atender às necessidades de flexibilidade e agilidade das organizações, sem perder o controle necessário no desenvolvimento, é necessário encontrar o tipo de processo mais adequado. Isso também ajuda a mitigar o risco da volta ao início do desenvolvimento de software, quando não existia o mínimo controle ou previsibilidade das entregas, o chamado *ad hoc*, que pode se tornar uma realidade usando como justificativa a agilidade demandada pelas empresas. A falta de controle é algo não aceitável nos dias atuais, considerando-se o valor que o software agrega nas organizações. Cada vez mais, as empresas dependem de software para conduzir seu negócio e ter retorno de seus investimentos. Isto se tornou ainda mais essencial com o aumento no uso de computadores e com o crescimento do acesso à internet. Esses fatos aumentaram consideravelmente a importância de produzir software de alta qualidade, com cada vez mais agilidade. Por isso, escolher o processo correto para permitir um desenvolvimento controlado e com a agilidade necessária é algo essencial para qualquer organização.

No método de seleção apresentado neste trabalho, é dado o direcionamento para se encontrar qual das abordagens de desenvolvimento de software pode ser considerada ideal para cada projeto. Os próximos capítulos detalham melhor os processos RUP e XP utilizados neste trabalho como sugestões de processo tradicional e ágil respectivamente.

3 *Extreme Programming* (XP)

3.1 Introdução

Neste capítulo são mostrados os valores, princípios e práticas que fazem parte do XP. Estes conceitos formam a base para que o desenvolvimento de um determinado software possa realmente ser considerado desenvolvido com o processo XP.

3.2 Valores do XP

Antes de se iniciar qualquer prática, é necessário entender os valores que estão por trás delas. Beck (2004) define cinco valores para guiar o desenvolvimento de software: comunicação, simplicidade, realimentação, coragem e respeito.

3.2.1 Comunicação

A boa comunicação dentro da equipe e entre as equipes pode facilitar muito o desenvolvimento do software. Esse é valor que faz mais diferença dentro de uma equipe XP. Quando um novo projeto é iniciado, possui suas particularidades, mas normalmente também dispõe de coisas em comum com outros projetos já realizados. Os pontos em comum podem estar tanto no nível técnico como gerencial, mas quase sempre alguém já passou por uma situação semelhante e conhece uma boa alternativa para solucionar o problema (BECK, 2004).

3.2.2 Simplicidade

Beck (2004) sugere que seja encontrada a solução mais simples para o novo sistema que está sendo desenvolvido. Fazer um sistema simples apenas para resolver o problema atual não é trivial, mas é o que se deve procurar fazer.

Esse valor não deve ser confundido com fazer um sistema simples demais. A idéia é realizar somente o necessário, nada mais, nada menos. Por exemplo, se o sistema tem requisitos críticos de segurança ou desempenho, o mais simples possível é diferente de outro caso, cuja segurança não é uma preocupação primária (BECK, 2004).

3.2.3 Realimentação *(Feedback)*

Nenhum processo deve durar para sempre. Deve-se refletir sobre como está o andamento do processo e o que pode ser melhorado. Essa realimentação pode aparecer de diversas maneiras (BECK, 2004):

- Coleta de opiniões de toda a equipe sobre uma idéia ou prática;
- Verificação da qualidade do código após a implementação de uma nova idéia;
- Análise de melhoria na elaboração de testes;
- Análise da execução dos testes;
- Verificação do resultado final após a entrega do sistema.

É importante sempre refletir sobre como o software está sendo desenvolvido e gerar realimentação a partir disso (BECK, 2004).

3.2.4 Coragem

Coragem é uma ação efetiva quando se está de frente a uma incerteza. Se o problema é conhecido, alguma ação em relação a ele deve ser tomada. Algumas vezes coragem pode se manifestar por meio de paciência. Se um problema existe, mas sua origem ainda não é conhecida, deve-se ter coragem para ser paciente e encontrar o real problema. (BECK, 2004)

Coragem por si só é algo perigoso, mas juntando-se com os outros valores isso se torna poderoso. Por exemplo, não se deve tomar uma ação considerada corajosa sem comunicar a todos que podem sofrer impacto por essa ação (BECK, 2004).

3.2.5 Respeito

Todos os outros quatro valores do XP dependem deste que está na base de tudo: respeito. Se os membros de uma equipe não se preocupam uns com os outros e com o que estão fazendo, XP não funciona. Se os membros da equipe não se preocupam com o resultado, nada pode salvar o projeto (BECK, 2004).

3.3 Princípios do XP

Valores são conceitos muito abstratos para guiar o comportamento humano. Diversos documentos existem para descrever apenas como deve ser a comunicação ideal, mas encontrar a forma mais efetiva de comunicação não é tarefa simples. Para se atingir uma boa comunicação dentro da equipe o contexto e os princípios intelectuais devem ser considerados. Os princípios existem para preencher a lacuna entre os valores e as práticas do XP. Beck (2004) cita os princípios descritos a seguir.

3.3.1 Humanidade *(Humanity)*

Pessoas desenvolvem software. Este simples fato pode invalidar boa parte das recomendações dadas por processos mais tradicionais existentes. Freqüentemente, processos de desenvolvimento de software não atendem às necessidades humanas. Pensar como se software não fosse escrito por humanos, acarreta um aumento significativo nos custos sobre os aspectos humanos que cada participante da equipe possui. Isso também pode ser prejudicial ao negócio, e sabe-se que empresas que não motivam seus funcionários, apresentam um alto índice de pedidos de demissão, pela falta de oportunidades e ações em prol do funcionário. Todo ser humano precisa de alguns aspectos básicos para poder ser um bom desenvolvedor de software:

- Segurança básica: não ter receio de perder o emprego, se machucar no trabalho, etc;
- Comprometimento: Oportunidade e habilidade de se sentir útil perante sua sociedade ou dentro de sua empresa;
- Sentir-se parte: Contribuir para metas que compartilha com os demais;
- Crescimento: Oportunidade de melhorar suas habilidades e perspectivas;
- Intimidade *(Intimacy)*: Habilidade de entender e ser entendido profundamente pelos outros. Não misturar isto com detalhes da vida privada, isto deve ser separado do trabalho.

As práticas do XP procuram preservar estes aspectos básicos que todo ser humano necessita para se sentir bem em seu trabalho, que é parte de sua vida.

3.3.2 Aspectos Econômicos *(Economics)*

O projeto é pago pelo cliente. É importante respeitar o custo do projeto e verificar se o que está sendo feito traz valor e se atende às metas e necessidades do cliente. Por exemplo, desenvolver primeiro e rapidamente as funções que trazem o maior retorno para o negócio, fazem com que o projeto tenha um maior valor para o cliente.

Dois aspectos econômicos afetam o desenvolvimento de software: valor gasto ao longo do tempo e valor futuro do sistema e da equipe. O valor gasto ao longo do tempo diz que o desenvolvimento de software tem maior benefício quando gasta menos no início, consegue receber retorno financeiro rapidamente, e gasta o restante do dinheiro depois de já ter um

retorno inicial. O valor futuro do sistema e da equipe define que se deve aproveitar o entrosamento da equipe em projetos futuros, ou aproveitar o sistema além do que se propôs inicialmente. Este princípio não define uma flexibilidade desnecessária em um sistema, mas procura a melhor relação de custo benefício para o que se está desenvolvendo. A preocupação com o quanto está sendo gasto deve ser responsabilidade de todos.

3.3.3 Benefício mútuo

Toda atividade deve beneficiar a todos diretamente envolvidos. Esse é o princípio mais importante do XP e o mais difícil de adequar a cada equipe. Soluções que demandam um grande trabalho de uma pessoa para beneficiar o trabalho de outra podem ser atrativas em determinadas situações, mas prejudicam as relações de trabalho e o resultado do projeto. Documentação excessiva é um exemplo da violação desse princípio; fazer uma determinada pessoa escrever em excesso tudo que fez, com o objetivo de facilitar a uma outra entender o que foi feito, num futuro possível, não deve ser uma prática adotada. Uma maneira de facilitar o entendimento do sistema que seja boa a todos, deve ser obtida. Este problema é resolvido no XP de maneira benéfica por meio de testes automatizados, simplicidade de código, padrões de código, etc.

3.3.4 Similaridade

Este princípio diz que se deve procurar usar uma solução adotada no passado ou existente na literatura para a resolução de novos problemas. Deve-se tentar reutilizar a estrutura de uma solução já criada e adaptar a este novo contexto. Este princípio não trata o reuso de software, mas a reutilização de idéias que deram certo no passado.

3.3.5 Melhoria contínua

Não existe processo perfeito, sempre há algum ponto que pode ser melhorado. Todos os membros da equipe devem, por princípio, procurar problemas que estejam prejudicando o bom andamento do processo e sugerir melhorias. Deve-se analisar sempre os resultados obtidos por um projeto e quais pontos podem ser otimizados para gerar um melhor resultado, em uma próxima oportunidade.

3.3.6 Diversidade

Equipes em que todos pensam e agem de forma similar não são efetivas. Deve-se procurar uma diversidade de habilidades, comportamentos e perspectivas dentro da equipe para se atingir um melhor resultado.

3.3.7 Reflexão

Boas equipes não fazem somente seu trabalho, elas sempre refletem e questionam como o estão fazendo e por que estão realizando cada atividade.

3.3.8 Fluxo contínuo

Deve-se criar um fluxo contínuo de entrega de software com valor para o negócio. Não pensar no desenvolvimento de software como uma série de fases separadas, mas sim, em um fluxo onde todas as fases ocorram simultaneamente e gerem sempre como saída um software que traga valor ao negócio.

Não se deve tentar corrigir um problema de fluxo, aumentando cada fase, ao invés disso deve-se entender porque o fluxo contínuo não está dando certo, corrigindo o problema em sua origem.

3.3.9 Oportunidade

Os obstáculos que são encontrados ao longo de um projeto devem ser encarados como oportunidades para futura melhoria e não como problemas.

3.3.10 Redundância

As práticas de um processo devem apresentar redundância, para prevenir a ocorrência de uma falha grave, que comprometa a confiança entre os membros da equipe. Por exemplo, utilizar uma tarefa de revisão de código, feita por uma pessoa diferente de quem criou o código, pode evitar que erros graves sejam levados até os testes finais do sistema. Esse é um exemplo de prática que previne a ocorrência de defeitos, aumentando a confiança da equipe em seu próprio trabalho de desenvolvimento. Isso é essencial no XP, pois a confiança é um grande eliminador de etapas de controle excessivas de um processo. Portanto, devem-se executar atividades redundantes, uma validando a outra, para garantir que a confiança não seja perdida entre os membros da equipe.

3.3.11 Falha

Aceitar que falhas possam ocorrer é uma grande fonte de aprendizado. Quando não se sabe qual opção pode ser mais bem sucedida, para implementação de uma estória, deve-se tentar de todas as formas possíveis até encontrar a solução adequada. Mesmo se todas falharem, tem-se um valoroso aprendizado destas falhas, evitando que sejam repetidas no futuro. É preciso evitar discussões longas em busca de idéias perfeitas, e procurar por idéias que podem funcionar e colocá-las à prova.

Deve-se evitar que a mesma falha ocorra novamente, mas aceitar que possam ocorrer falhas é importante para o processo. A preocupação com falhas deve existir, mas quando não há certeza, arriscar pode ser o caminho mais curto para o sucesso.

3.3.12 Qualidade

Um projeto não é entregue mais rapidamente, aceitando-se níveis menores de qualidade do que o necessário. Sacrificar a qualidade não é a maneira mais eficiente de se conseguir controle. Qualidade não é uma variável de controle e não faz com que o desenvolvimento fique mais lento.

Qualidade não é somente um fator econômico, as pessoas devem ter orgulho do trabalho que realizaram. Mesmo se um desenvolvedor não tem conhecimento para fazer seu trabalho com toda a qualidade necessária, deve se esforçar e fazer o melhor que conseguir, mas sempre deve tentar deixar seu trabalho bem feito.

3.3.13 Passos pequenos

É sempre tentador fazer grandes mudanças em grandes passos, mas o ideal é sempre dar pequenos passos, um de cada vez. Isso permite perceber com mais rapidez se o caminho que está sendo seguido é o correto.

3.3.14 Responsabilidade deve ser aceita

Responsabilidade não pode ser simplesmente atribuída, ela deve também ser aceita por quem a recebe. Se alguém decide atribuir à outra pessoa uma responsabilidade, somente esta pode dizer se é ou não capaz de recebê-la. É responsabilidade de quem aceitou a tarefa estimar o tempo e complexidade da mesma. Junto com a responsabilidade deve ser verificada a autoridade. Ambas devem estar alinhadas para que o desenvolvimento ocorra normalmente.

Os princípios podem ser utilizados para facilitar o entendimento das práticas do XP. É importante conhecê-los e entender os motivos da criação de cada prática, para não se tentar improvisar novas práticas quando não se encontra uma que pareça adequada à necessidade. É com o entendimento desses princípios que as práticas do XP são mais facilmente entendidas e conseqüentemente aplicadas. (BECK, 2004)

3.4 Práticas do XP

Na edição mais recente das práticas do XP (BECK, 2004), algumas nomenclaturas e uma nova divisão sobre cada prática do XP foram propostas. Abaixo segue um resumo de como Beck (2004) descreve as práticas primárias e secundárias do XP.

3.4.1 Práticas primárias do XP

3.4.1.1 Estórias

O planejamento deve conter unidades de novas funções que sejam visíveis para o cliente. Cada estória descreve uma nova função, que deve ser incorporada ao sistema. Ex: “Incluir tecla de atalho para colocar em negrito um texto”; “Permitir que usuário cancele a compra caso não confirme o cartão de crédito”, etc. Uma estória não deve ter detalhes sobre a função, deve conter apenas um breve resumo e quem demandou a função. Quando a estória entrar no desenvolvimento, os detalhes são obtidos com quem fez a solicitação da mesma.

3.4.1.2 Equipe no mesmo local físico

O desenvolvimento do software deve ocorrer em um ambiente onde toda a equipe esteja no mesmo local físico. Esse ambiente deve respeitar o direito de privacidade de cada um para assuntos particulares. Por isso, devem existir pequenos espaços privativos nas proximidades. Outra forma de garantir a privacidade é permitir intervalos durante o horário de trabalho, para que os interesses particulares de cada membro da equipe possam ser resolvidos em outro local.

Essa prática não afirma que XP não possa ser aplicado sem esse ambiente comum. Porém, quando isso não for viável, a agilidade pode ser prejudicada se a comunicação não fluir com facilidade.

3.4.1.3 Equipe completa

A equipe deve contar com pessoas de habilidades e perspectivas distintas para o projeto ter sucesso. A constituição de uma equipe completa pode variar muito, dependendo do contexto,

mas um conjunto de todas as habilidades necessárias deve existir dentro da equipe. Se um grupo de habilidades ou conhecimento é necessário, deve-se trazer uma pessoa com tais requisitos para a equipe. Se outra pessoa não é mais útil em determinada equipe, deve-se colocá-la em outra equipe, onde suas qualidades possam ser mais bem aproveitadas. Se o projeto tem uma necessidade grande por alterações em banco de dados, deve-se trazer alguém com esse conhecimento. Quando as alterações em banco de dados não forem mais necessárias, esta pessoa pode ir para outra equipe.

3.4.1.4 Espaço informativo de trabalho

É importante que o ambiente informe algo sobre o andamento do trabalho em si. Desta forma, um observador que esteja interessado no andamento do trabalho pode ter uma idéia geral do andamento apenas caminhando dentro deste ambiente. Um exemplo de como fazer isto é colocar cartões descrevendo cada função (estória) do sistema a ser desenvolvido em um quadro, dividindo as estórias em grupos: prontas; para esta semana; para próxima versão a ser lançada; para serem estimadas; para o futuro, etc.

3.4.1.5 Trabalho produtivo

A equipe deve trabalhar apenas quantas horas puder para continuar sendo produtiva. Esse tempo não pode atrapalhar as demais atividades de sua vida. Em resumo, a prática diz que o horário de trabalho deve ser respeitado e que as horas extras e excesso de trabalho devem ser evitados.

3.4.1.6 Programação em pares

Devem-se escrever todos os produtos de software sempre com duas pessoas sentadas lado a lado em uma mesma máquina. A estação de trabalho e os móveis devem ser configurados de forma que as duas pessoas fiquem sentadas confortavelmente uma ao lado da outra. A programação em pares é um diálogo entre duas pessoas, programando simultaneamente. Enquanto uma está digitando, a outra pode estar pensando na arquitetura de uma nova classe ou pensando no teste. Para uma programação eficiente:

- Ambos devem se preocupar que o outro tenha tarefas a fazer;
- Realizar sessões de discussão para refinar o sistema;
- Expor e clarificar idéias;

- Ter iniciativa e tentar ajudar seu parceiro a achar uma solução, quando perceber que ele está parado com algum problema do sistema;
- Garantir que ambos estão seguindo as práticas do XP.

Não é necessário que todas as horas do dia de um programador sejam utilizadas com a programação em pares, mas todos os dias isto deve ser feito durante o desenvolvimento.

3.4.1.7 Ciclos semanais de planejamento

Deve-se planejar o trabalho uma semana de cada vez. Uma reunião no início de cada semana deve ser feita para:

- Revisar o progresso desde a última reunião, incluindo a análise se o progresso atual está de acordo com o que era esperado;
- Fazer com que o cliente do projeto escolha as estórias que devem ser implementadas durante esta semana;
- Dividir as estórias em tarefas. Cada membro da equipe escolhe as tarefas que ficam com ele. Cada um faz as estimativas de suas tarefas.

No início da semana, devem ser escritos os testes automatizados que são executados quando as estórias escolhidas forem finalizadas. O resto da semana é utilizado para construir as estórias, de forma que elas passem nos testes definidos. A meta é ter as novas estórias prontas para integração com o sistema no final de cada semana (Esse incremento das funções pode estar em ambiente de desenvolvimento ou em produção).

3.4.1.8 Ciclos trimestrais de planejamento

A separação entre temas e estórias deve existir para retirar a necessidade de detalhamento sobre funções mais específicas nesse planejamento. Os temas são ainda menos específicos que as estórias, abrangendo um escopo pouco definido. É o que mais se assemelha aos requisitos da engenharia de software tradicional.
Deve-se fazer um planejamento em mais alto nível com o que se espera para o próximo trimestre. Durante este planejamento deve-se:

- Identificar gargalos, especialmente os que estão fora da equipe;
- Planejar os consertos no sistema;

- Planejar os temas para o próximo trimestre;
- Criar um grupo de estórias que fazem parte dos projetos a serem realizados no próximo trimestre;
- Definir onde o projeto se enquadra dentro da estratégia da organização.

3.4.1.9 Tarefas sobressalentes

Em qualquer planejamento devem-se incluir tarefas menos prioritárias que possam ser canceladas, caso o cronograma esteja em atraso. Sempre é possível incluir estórias após o planejamento inicial se estiver adiantado, mas também deve haver margem de manobra clara, quando ficar atrasado.

3.4.1.10 Construção do sistema em até 10 minutos

A construção do sistema deve ser automatizada e não pode levar mais que 10 minutos. Uma construção de sistema que demora mais tempo que isso, acaba não sendo realizada com a freqüência recomendada pelo XP. Com isso, a realimentação sobre o andamento do sistema fica mais lenta. Em suma, deve ser possível executar todos os testes automatizados do sistema e sua construção, em até 10 minutos. Este tempo foi considerado ideal após observações feitas por Beck (2004) em vários projetos de software usando XP.

3.4.1.11 Integração contínua

É importante procurar fazer a integração de todos os componentes do sistema de forma contínua. O recomendado é integrar o sistema diariamente, ou até mesmo várias vezes durante um dia. A tarefa de integração de um sistema é algo que, quanto mais se esperar para realizar, se torna uma tarefa cada vez mais complexa. Em alguns casos esse tempo pode até mesmo superar o de construção do sistema. A recomendação de Beck (2004) é fazer a integração diariamente após cada sessão de programação em pares.

3.4.1.12 Programação orientada a testes

Deve-se definir um teste automatizado, antes de iniciar qualquer nova codificação. Em princípio, esse teste não é bem sucedido, já que não existe o código implementado. Depois, é necessário implementar o código, de forma que o teste seja bem sucedido. Colocar claramente em um teste o que o código deve fazer, facilita o entendimento e comunicação com os demais membros da equipe.

Esses testes têm uma limitação de, em princípio, verificar apenas pequenos trechos do sistema. Ou seja, verifica apenas a classe ou componente que está sendo desenvolvido e não o sistema inteiro. Porém, conforme a experiência da equipe aumentar, mais itens podem ser incluídos como parte do teste. Devido à limitação de escopo desses testes, eles costumam executar muito rapidamente. Desta forma, milhares de testes podem fazer parte da construção diária do sistema. Isto dá mais segurança na hora de fazer alterações.

3.4.1.13 Arquitetura incremental

Convém investir na arquitetura do sistema diariamente. É preciso que essa arquitetura se enquadre exatamente nas necessidades do sistema vigente. Quando se perceber que a mesma está defasada, deve-se planejar o trabalho, de forma que isso seja gradualmente alinhado com as novas necessidades do sistema. É neste ponto que a reengenharia de código é aplicada na prática.

A idéia dessa prática é evitar que toda a análise e arquitetura do sistema tenham que ser feitas no início, com longas sessões. A partir de um estudo de Barry Boehm (1960 apud Beck, 2004), que indicava que o custo de mudança no final é muito maior que no início, permaneceu por décadas a seguinte filosofia: "Faça toda a arquitetura do sistema no início, pois não existirá outra chance". Esta afirmação não é válida no XP. Nos processos ágeis, a idéia principal é diminuir o custo de mudança com testes automatizados, habilidade em fazer reengenharia do sistema, etc. Desta forma, no XP a idéia é fazer a arquitetura evoluir aos poucos conforme a necessidade.

3.4.2 Práticas secundárias do XP

Segundo Beck (2004) as práticas secundárias são difíceis de ser implementadas, antes que todas as práticas primárias já estejam em uso com sucesso. Um resumo delas é apresentado a seguir, apenas para conhecimento:

Envolvimento real do cliente: Deve-se ter o cliente como parte do projeto.

Entregas incrementais: Na substituição de sistemas legados, deve-se fazer gradualmente a substituição deste sistema.

Manutenção de equipes: Se uma equipe é efetiva, deve-se procurar mantê-la junta por um bom tempo.

Diminuição gradual da equipe: Se uma equipe está com sua capacidade de desenvolvimento em crescimento, deve-se procurar manter ao longo dos projetos a mesma quantidade de trabalho, mas diminuindo o tamanho da equipe.

Análise de causa raiz: Toda vez que um problema for encontrado depois de uma entrega, deve-se corrigir não somente o problema, mas também sua causa.

Código comum: Um membro da equipe pode codificar qualquer parte do sistema. Não existe código de um determinado membro da equipe, todos são proprietários de todo o código do sistema.

Código e testes: Devem-se manter apenas o código dos testes e o do sistema, como artefatos permanentes. Todo o restante é temporário e não precisa ser mantido por todo o ciclo de vida do sistema.

Código único: Deve existir apenas um código do sistema. Pode-se abrir um desenvolvimento paralelo por um tempo, mas é necesssário voltar a apenas um código do sistema o mais rápido possível.

Entregas diárias: É importante se colocar algum software novo em ambiente final para o cliente, todos os dias.

Contrato com escopo negociável: Contratos de software devem ser escritos onde os requisitos de tempo, custo e qualidade sejam fixos. Porém, deve-se ter escopo negociável para o software, mesmo após o fechamento do contrato.

Pagamento por uso: O sistema pode ser pago não pelo seu desenvolvimento, mas pelo seu uso efetivo após estar em produção.

3.5 A equipe completa do XP

Para um desenvolvimento de software com sucesso uma diversidade de perfis de pessoas deve fazer parte da equipe. Os papéis sugeridos por Beck (2004) são descritos a seguir.

3.5.1 Testadores

Os testadores auxiliam os clientes na elaboração de testes automatizados do sistema. Eles também devem ensinar as técnicas de testes para os programadores. Uma vez que os testes para um dado ciclo semanal tenham sido definidos, escritos e não estejam funcionando (já que o código ainda não foi implementado), os testadores continuam escrevendo novos testes para

definir novas funções, que vão sendo especificadas durante o desenvolvimento. Eles podem também trabalhar na elaboração de testes de desempenho do sistema.

3.5.2 Projetistas de interação

Os projetistas auxiliam o cliente na elaboração das estórias, fazem a avaliação do sistema em busca de novas oportunidades para estórias, criam metas e personagens simulando os possíveis usos do sistema. Com isso, eles procuram definir os comportamentos necessários do sistema, e auxiliam na definição da interface com o usuário.

3.5.3 Arquitetos

Os arquitetos procuram pela necessidade em larga escala de reengenharia do sistema. Quando encontram, eles mesmos planejam e coordenam esse processo com a equipe, definindo qual a nova arquitetura desejada. Eles garantem que nenhum excesso ou problema de arquitetura exista no sistema. Os arquitetos também podem elaborar testes para direcionar como uma mudança de arquitetura deve ser realizada.

3.5.4 Gerente de projeto

Os gerentes de projeto têm como papel principal facilitar a comunicação dentro da equipe. Eles coordenam a comunicação com os clientes, com os fornecedores e o restante da organização, e atuam criando a documentação das atividades da equipe, para comunicar com as equipes externas. Eles devem lembrar e acompanhar a equipe com relação ao progresso das atividades e têm o dever de manter as informações atualizadas, de forma que sempre seja possível fazer uma apresentação do progresso para os executivos, ou os demais envolvidos no projeto.

O planejamento no XP é uma atividade e não uma fase. O papel do gerente de projeto é fazer o planejamento inicial do ciclo semanal e manter tudo atualizado com a realidade. A atividade pode ser iniciada com um dia por semana, mas o tempo ideal para planejar as tarefas da semana é uma hora. Isso somente pode ser alcançado com a prática no processo, mas é algo possível segundo Beck (2004).

3.5.5 Gerente do produto

Os gerentes do produto escrevem as estórias dos ciclos semanais e definem os temas e estórias para o ciclo trimestral. Eles são responsáveis pela definição de prioridade das mesmas e por esclarecer à equipe pontos que não ficaram claros sobre uma determinada estória. Eles devem sempre ter em mente o interesse do produto e não os interesses técnicos.

3.5.6 Executivos

Os executivos devem prover para a equipe coragem, confiança e realimentação sobre seu trabalho. Precisam mostrar como o trabalho da equipe está alinhado com os objetivos da corporação. Em resumo, o executivo é a pessoa que deve informar claramente o que a empresa pretende com o produto, ele define o escopo em alto nível do produto e faz a comunicação com os altos níveis da empresa.

3.5.7 Escritores técnicos

Os escritores técnicos devem usar linguagem escrita ou figuras para descrever as funções de forma clara para o público alvo. Eles são responsáveis pela explicação do sistema ao cliente. Devem ser um dos primeiros a ver as novas funções do sistema, mesmo quando ainda são apenas protótipos. Precisam pensar em como cada função pode ser explicada para o cliente e usuários finais. Eles são responsáveis pela documentação técnica do sistema, que pode se apresentar de diversas formas: tutoriais, vídeos, áudio, etc.

3.5.8 Usuários

Os usuários representam o usuário final do sistema, auxiliam na escrita e escolha das estórias. Eles auxiliam também na hora de tomar decisões de domínio durante o desenvolvimento. Os usuários devem ter grande experiência e conhecimento sobre sistemas similares, para realmente ter valor no projeto.

3.5.9 Programadores

Os programadores são responsáveis pelo desenvolvimento do sistema e pela estimativa de tempo na programação de cada estória. Eles dividem cada estória em tarefas e também estimam o tempo de cada uma, refinando a estimativa de tempo da estória. Eles escrevem testes unitários e o código para implementar as estórias.

3.6 Considerações finais do capítulo

Como foi descrito no capítulo anterior, as empresas atualmente estão em busca de processos de desenvolvimento de software cada vez mais flexíveis, visando atender à demanda crescente por software dentro das organizações. O processo ágil XP foi criado exatamente para atender a essa

necessidade, em detrimento de processos mais formais com pouca flexibilidade ao longo do desenvolvimento de software.

Algumas práticas descritas neste capítulo como: ciclos semanais de planejamento, estórias, temas, integração contínua e arquitetura incremental visam dar a flexibilidade demandada pelas organizações, permitindo que seja natural, durante o desenvolvimento, existir alterações de escopo de uma função ou mesmo a definição de novas funções.

Existe a definição de poucos papéis dentro do processo XP, para torná-lo algo fácil de ser compreendido. Para se obter a flexibilidade necessária, XP tem como ponto forte a manutenção de todos os membros da equipe juntos, visando facilitar a comunicação e o conhecimento tácito, em detrimento do conhecimento escrito e formal, ganhando maior agilidade no desenvolvimento.

Pelas razões descritas neste capítulo, o XP é efetivamente um processo ágil, sendo utilizado como exemplo deste tipo de processo, neste trabalho. No próximo capítulo é descrito o RUP, com foco no planejamento, para melhor conhecimento de sua estrutura, permitindo então verificar se ele é realmente adequado, como exemplo de processo tradicional, para o método de seleção proposto neste trabalho.

4 *Rational Unified Process* (RUP)

4.1 Introdução

Neste capítulo são descritos os princípios, atividades, papéis, fases e alguns outros elementos que compõem o processo RUP. As informações deste capítulo servem como base para a análise do RUP, frente aos fatores do método de seleção proposto por este trabalho.

4.2 Princípios

Segundo Kroll e Kruchten (2003), os princípios fundamentais que devem ser considerados para se obter sucesso na utilização do RUP são:

- Tratar os maiores riscos de forma rápida e contínua: Ao invés de deixar os riscos de negócio, técnicos e demais para serem analisados no final do projeto, deve-se levantar logo os maiores riscos e criar planos de ação o mais cedo possível.
- Garantir que está sendo entregue algo de valor ao cliente: Devem-se documentar os requisitos de forma que sejam facilmente compreendidos. Porém, estes requisitos devem ser desenvolvidos e monitorados durante as fases de elaboração, construção e testes, como forma de garantir que o produto final esteja em conformidade com os requisitos solicitados.
- Manter o foco em versões executáveis de software: Documentos, diagramas de arquitetura, planos de teste são bons artefatos, mas são indicadores muito fracos e subjetivos sobre o progresso de um projeto. A avaliação e importância desses artefatos são secundárias quando comparadas com o código propriamente dito. O produto final sempre deve ser um software executável; portanto, quanto mais código existir para ser entregue, melhor é o indicador real do progresso.
- Acomodar as mudanças durante o projeto: As aplicações atuais são muito complexas para permitir que todos os requisitos e arquitetura sejam coletados logo no início. Isso significa que se deve desenvolver um sistema bom o suficiente na primeira iteração e adaptá-lo ao longo do tempo às mudanças necessárias.
- Criar uma arquitetura base logo no início: Muitos riscos podem ser mitigados por meio de uma boa arquitetura, implementação e testes. Por isso, deve-se criar logo no início do projeto uma arquitetura base que permita avaliar o impacto das mudanças.

- Construir o sistema com componentes: Aplicações construídas com foco em componentes são mais adaptáveis às mudanças, o que pode reduzir muito o custo das manutenções. Componentes facilitam o reuso, possibilitando a construção de aplicações com alta qualidade de maneira mais rápida.
- Trabalhar todos em conjunto, como uma só equipe: O desenvolvimento de software tornou-se um trabalho de equipe, como nos esportes coletivos. O desenvolvimento iterativo enfatiza a necessidade de boa comunicação e espírito de equipe entre todos os participantes, para se atingir um bom resultado final.

4.3 A estrutura do RUP

O RUP pode ser caracterizado por meio de duas estruturas ou dimensões. A estrutura dinâmica mostra como os processos dentro do RUP – mostrados por meio de ciclos, fases, iterações e marcos – podem ser definidos ao longo do ciclo de vida do projeto. A estrutura estática descreve como os elementos do processo – atividades, disciplinas, artefatos e papéis – estão logicamente agrupados (KROLL;KRUCHTEN, 2003). A figura 10 mostra como estas estruturas podem ser visualizadas.

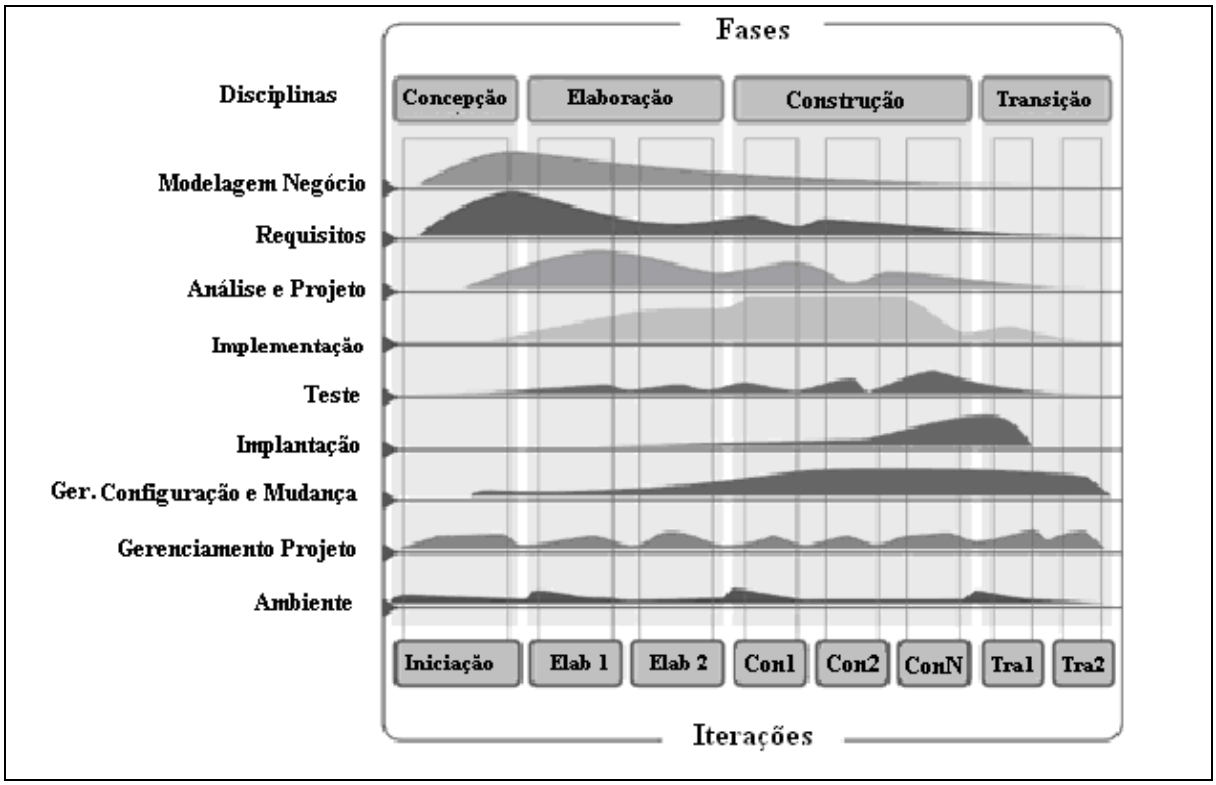

Figura 10 – A estrutura do RUP – Fonte: (KROLL;KRUCHTEN, 2003)

4.4 As fases do RUP

O ciclo de desenvolvimento do RUP é dividido em quatro fases. Cada ciclo encerra-se com a entrega de um produto completo de software. No entanto, apesar destas quatro fases serem realizadas em seqüência, é importante ter em mente que o RUP é um processo iterativo, com foco no planejamento e orientado a riscos. O RUP propõe o uso de várias iterações, sendo cada uma delas compostas pelas quatro fases existentes no processo (Concepção, Elaboração, Construção e Transição). (KROLL;KRUCHTEN, 2003)

Em todas as fases do RUP é importante que se tenha clareza sobre os objetivos que devem ser atingidos, no final da fase. Isso permite uma escolha mais correta de quais atividades devem ser executadas e quais artefatos devem ser gerados. (KROLL;KRUCHTEN, 2003)

4.4.1 Fase de Concepção

Na fase de concepção deve-se entender o escopo e os objetivos do projeto (Figura 11). Também é nessa fase que se deve verificar a viabilidade de prosseguir com o projeto.

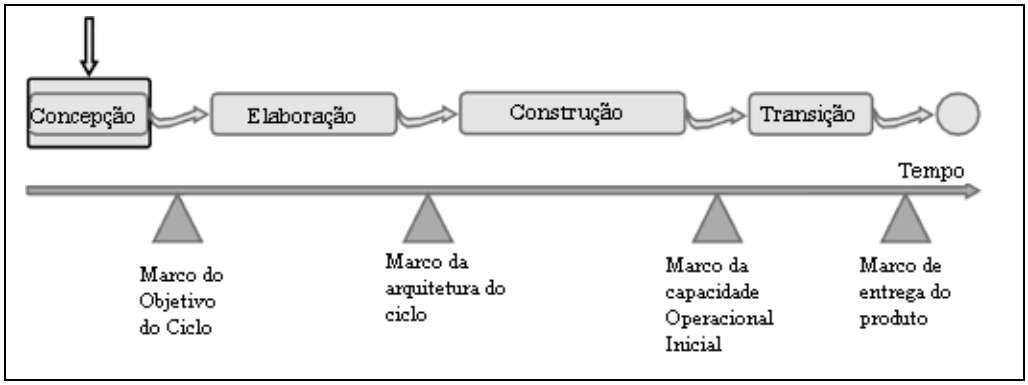

Figura 11 – Fase de Concepção no RUP – Fonte: (KROLL;KRUCHTEN, 2003)

Os cincos principais objetivos da fase de concepção são:

4.4.1.1 Entender o que deve ser construído

Para garantir o entendimento comum do que deve ser construído, faz-se necessária a elaboração de um documento com a visão do sistema em alto nível. É necessário prover uma breve descrição sobre o que o sistema deve fazer, os benefícios que traz e qual problema ele pretende resolver. Não se deve exagerar em detalhes, para evitar que alguns envolvidos percam o foco do problema entre tantos detalhes que não agregam valor ao entendimento do problema. Devem-se levantar quais são os usuários do sistema e que funções eles demandam.

4.4.1.2 Identificar as funções principais do sistema

Nesta etapa é importante decidir quais casos de uso são mais importantes ou mais relevantes em termos de arquitetura para o sistema. É preciso saber quais os casos de usos são mais críticos, que devem ser trabalhados o mais cedo possível. O gerente do projeto e o arquiteto interagem em conjunto com os principais envolvidos no projeto, de forma a determinar quais são os casos de uso mais relevantes.

4.4.1.3 Definir ao menos uma possível solução

Dado que um dos objetivos da fase de concepção é determinar a viabilidade do projeto, uma saída necessária desta fase é ter ao menos uma solução viável de arquitetura para o problema, caso contrário o projeto deve ser descontinuado. Essa solução de arquitetura deve levar em conta fatores que sejam determinantes ao projeto como: custo, tempo e risco.

4.4.1.4 Entender as dimensões de custo, prazo e risco

Entender o que construir é importante, mas é crucial entender como deve ser construído e quanto deve custar. Nessa etapa pode ser necessário criar documentos de casos de uso de negócio. Deve-se estimar se é possível entregar no prazo e com o custo aprovado. Os riscos devem ser mapeados e estar dentro de níveis aceitáveis.

4.4.1.5 Definir qual processo seguir e quais ferramentas utilizar

É importante que as equipes do projeto compartilhem uma visão comum de qual processo seguir e que ferramentas devem ser utilizadas. Nesse ponto deve-se criar um documento que descreva quais partes do RUP devem fazer parte do desenvolvimento do software.

No fim da fase de concepção, o primeiro marco do processo deve ter sido atingido (Figura 11). Este marco define o objetivo da iteração atual do ciclo de desenvolvimento, composto de:

- Definição do escopo e estimativa inicial de tempo e custo (a ser refinada depois);
- Concordância entre os envolvidos que o conjunto correto de requisitos foi capturado e que existe um entendimento comum destes requisitos;
- Concordância na estimativa de tempo, estimativa de custo, priorizações, riscos e se o processo de desenvolvimento escolhido é adequado;

- Concordância de que os riscos iniciais foram identificados e que existe uma estratégia para mitigar esses riscos, caso ocorram.

4.4.2 Fase de Elaboração

A meta primária da fase de elaboração é definir uma arquitetura base definitiva, com o objetivo de prover uma estrutura estável para a fase de construção (Figura 12). A arquitetura base nasce de uma análise dos requisitos mais significativos em termos de arquitetura.

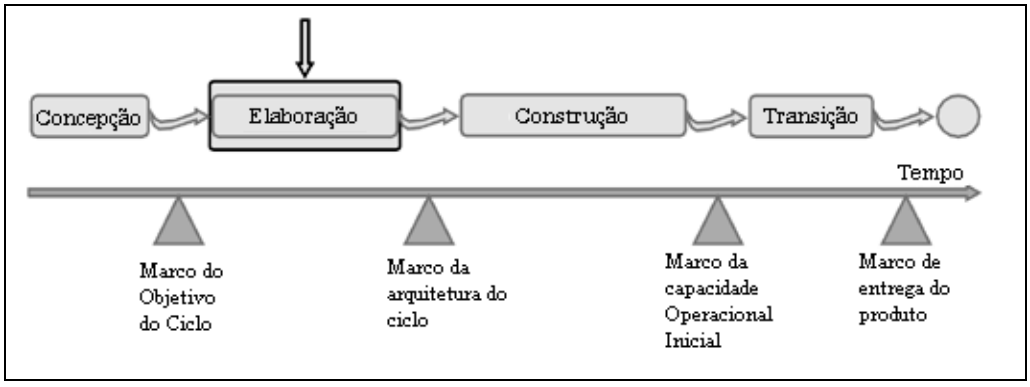

Figura 12 – Fase de Elaboração no RUP – Fonte: (KROLL;KRUCHTEN, 2003)

Essa meta primária pode ser subdividida em quatro objetivos a serem atingidos. Cada um deles visa tratar uma grande área de risco:

4.4.2.1 Obter um entendimento mais detalhado dos requisitos

Durante a elaboração pretende-se obter um detalhamento mais profundo dos requisitos, já que na fase de concepção existia apenas uma breve descrição dos mesmos. Nesse momento, os detalhes técnicos devem ser analisados com mais detalhes. É importante verificar se a arquitetura base realmente suporta os requisitos do sistema. A maioria dos casos de uso deve ser detalhada na elaboração.

4.4.2.2 Projetar, implementar, validar e obter uma arquitetura de referência.

Na elaboração é quando se cria a estrutura base do sistema. Pode-se fazer necessário um teste com um protótipo base desta estrutura, como forma de confirmar que esteja adequado. Nesse protótipo pode existir apenas uma simulação da função real para validar a arquitetura, mas pelo menos as interfaces e componentes do sistema devem ser definidos e validados.

4.4.2.3 Mitigar os riscos essenciais e produzir estimativas mais precisas

Deve-se definir como tratar os riscos. Normalmente isto é obtido como resultado do detalhamento dos requisitos e testes da arquitetura.

4.4.2.4 Refinar o processo de desenvolvimento

O processo definido na concepção pode ser refinado com um melhor conhecimento dos requisitos. Deve-se refletir sobre as lições aprendidas e aplicá-las em projetos futuros. O ambiente de desenvolvimento deve estar funcional, e sem barreiras que impeçam o início do desenvolvimento.

No fim da fase de elaboração, existe o marco de definição da arquitetura do ciclo de desenvolvimento (Figura 12). Neste marco, os seguintes objetivos devem ter sido atingidos:

- A visão e requisitos devem estar estáveis;
- A arquitetura base deve estar estável;
- As abordagens a serem usadas nos testes devem ter sido comprovadas;
- Os planos de execução para cada iteração devem possuir um nível suficiente de detalhes para seguir com o desenvolvimento;
- As estimativas para cada iteração da fase de construção devem ser aceitas;
- Todos os envolvidos devem concordar que o documento de visão está de acordo com o desejado;
- As estimativas atuais de tempo e custo foram aceitas pelos envolvidos.

4.4.3 Fase de Construção

Na fase de construção é quando a maior parte do trabalho é realizada (Figura 13). Nessa fase os casos de uso e a arquitetura são mais detalhados; o sistema deve ser implementado e os testes realizados para se obter, ao final da fase, um sistema executável a ser entregue.

Mesmo que os principais riscos tenham sido levantados e os planos para eles tenham sido criados na elaboração, é na fase de construção que muitos novos riscos e problemas surgem e têm que ser trabalhados. A arquitetura base criada na elaboração serve para permitir o crescimento do sistema durante a fase de construção, mas não garante que representa a arquitetura final do sistema quando entregue.

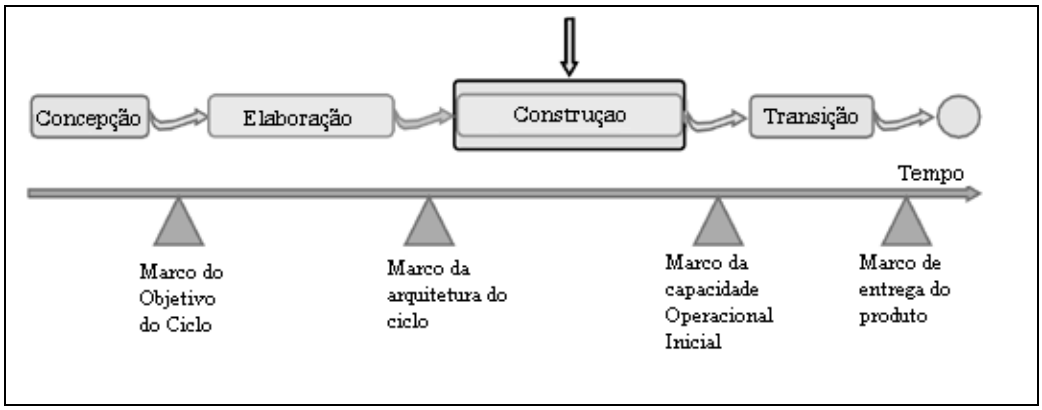

Figura 13 – Fase de Construção no RUP – Fonte: (KROLL;KRUCHTEN, 2003)

Os objetivos da fase de construção são:

4.4.3.1 Reduzir o custo de desenvolvimento e conseguir algum paralelismo

Deve-se procurar uma otimização dos recursos alocados ao projeto, assim como evitar refazer trabalhos ou realizar tarefas desnecessárias. Mesmo em sistemas pequenos geralmente é possível construir em paralelo componentes do sistema, permitindo, assim, uma melhor distribuição de trabalho entre as pessoas e equipes.

4.4.3.2 Desenvolver iterativamente um produto completo

Ao fim da fase de construção, deve existir uma versão funcional completa do sistema, a qual deve conter os requisitos definidos pelos objetivos da iteração atual.

A fase de construção normalmente tem mais de uma iteração. Usualmente a construção possui mais iterações que as demais fases do RUP, tipicamente têm entre duas a quatro iterações. A tabela 1 mostra um exemplo de um plano de iterações da fase de construção.

Tabela 1 – Plano de iterações do RUP – Adaptado de: (KROLL;KRUCHTEN, 2003)

Requisitos	**Componentes**	**Testes**
Fim da fase Elaboração		
15 Casos de uso identificados 8 casos de uso descritos em detalhes	18 componentes identificados 4 tiveram 50% do código implementado, incluindo todas as interfaces. 10 possuem as *interfaces* e parte da lógica (10 a 20% do código final). Camadas mais baixas da arquitetura foram parcialmente implementadas. Testes unitários realizados.	Testes inicial de stress e carga foram realizados, baseados nos principais casos de uso. Função dos 4 principais casos de uso foram testados completamente.
Fim da 1a iteração da fase de Construção		
12 casos de uso descritos em detalhes 3 com algum detalhe	18 componentes identificados (1 não é mais necessário devido à eliminação de um caso de uso) 10 foram parcialmente implementados. 8 possuem 50% do código implementado, incluindo todas as interfaces. Camadas mais baixas da arquitetura foram completamente implementadas. Testes unitários realizados.	Testes de stress e carga continuaram sendo realizados. Todos os testes funcionais foram realizados.
Fim da 2a iteração da fase de Construção		
1 dos casos de uso ainda não detalhado foi retirado do escopo devido a limitações de tempo Mais 14 casos de uso foram encontrados e detalhados	18 componentes identificados (1 não é mais necessário devido à eliminação de um caso de uso). 10 foram quase completamente implementados. 8 possuem 50% do código implementado, incluindo todas as interfaces. Testes unitários realizados.	Testes de stress e carga continuaram sendo realizados. Todos os testes funcionais foram realizados.
Fim da 3a iteração da fase de Construção		

O planejamento de cada iteração é guiado pelas funções a serem implementadas, descritas como casos de uso do sistema. Também devem ser analisados os principais riscos técnicos que podem

ocorrer na próxima iteração. Devem-se selecionar os casos de uso ainda não implementados, que são mais prioritários para o cliente.

Na primeira iteração, e em alguns casos também na segunda, inicia-se a fase de construção com a implementação apenas parcial dos casos de uso. Dessa forma, pretendem-se identificar possíveis novos riscos ainda não mapeados. Uma vez definidos os casos de uso, ou partes de casos de uso que devem ser implementados, devem-se identificar quais componentes precisam ser criados para permitir a função descrita pelo caso de uso. Esta identificação dos componentes envolvidos permite refinar melhor as estimativas de tempo e custo. Nessa fase, é possível obter então uma estimativa mais real do tempo e custo do desenvolvimento.

No fim da fase de construção existe o marco de definição da capacidade operacional (Figura 13). Neste ponto devem-se atingir os seguintes objetivos:

- O produto deve estar estável e maduro para ser entregue ao ambiente de produção;
- Todos os envolvidos devem estar preparados para a fase de transição;
- Os custos atuais em comparação com o planejado devem continuar aceitáveis.

4.4.4 Fase de Transição

A fase de construção termina com a entrega de um sistema completamente funcional, mas ainda em versão preliminar (deve passar por testes com alguns usuários finais). É na fase de transição que os ajustes finais e correções de defeitos são realizados. O foco da fase de transição é validar que o software, que está sendo entregue, esteja em concordância com os requisitos solicitados. A fase de transição é normalmente composta de uma ou duas iterações que incluem testes do produto, preparação para entrega final e pequenos ajustes de problemas encontrados nos testes. Sistemas mais complexos podem necessitar de muitas iterações na fase de transição. No final de cada iteração uma nova versão mais refinada do sistema é gerada (Figura 14).

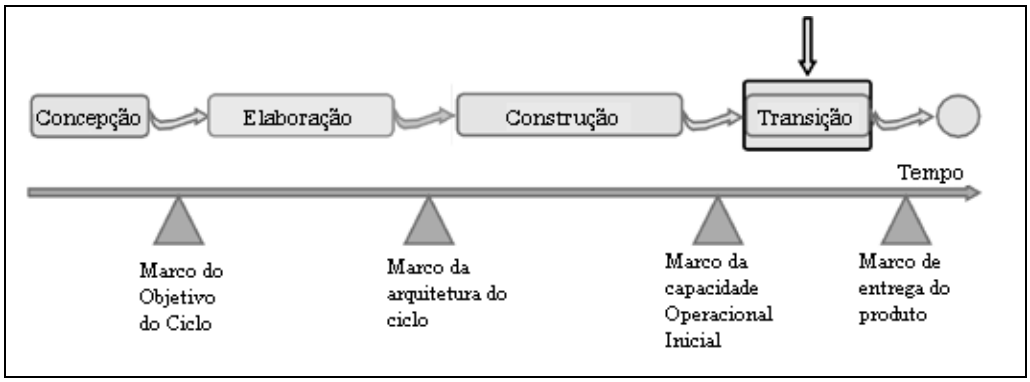

Figura 14 – Fase de Transição no RUP – Fonte: (KROLL;KRUCHTEN, 2003)

Os principais objetivos da fase de transição são descritos a seguir.

4.4.4.1 Testes finais com usuários reais

Na fase de transição normalmente existe a necessidade de ajustes finos e correções de problemas. Melhorias de desempenho e usabilidade também são definidas nesse ponto.

4.4.4.2 Treinamento dos usuários e operadores do sistema

O treinamento visa garantir que a empresa a qual está adotando o sistema é capaz de usá-lo e mantê-lo em funcionamento.

4.4.4.3 Preparar ambiente de produção

Para ter o sistema em completo funcionamento pode ser necessário adquirir novas máquinas, mais espaço físico ou mesmo recursos humanos. Qualquer migração de dados, necessária para adoção do novo sistema, também deve ser realizada nessa fase.

4.4.4.4 Preparação de produto comercializável

Especialmente no caso de produtos de mercado, que ainda precisam ser vendidos a clientes externos, é importante ter uma versão fácil de instalar do sistema. Deve-se treinar o pessoal de vendas e de suporte para garantir o sucesso na comercialização do produto.

4.4.4.5 Aceitação final do cliente do projeto

O cliente do projeto deve validar que o sistema entregue está de acordo com a visão aprovada para o mesmo.

4.4.4.6 Registrar lições aprendidas

Todos os erros e acertos do projeto devem ser documentados. Este documento de lições aprendidas deve ser considerado em novos projetos, assim como nas reflexões com o intuito de melhorar o processo de desenvolvimento.

No fim da fase de transição, o marco de entrega do produto deve ser atingido (Figura 14). Este deve conter os seguintes itens:

- O cliente final deve estar satisfeito com o produto;
- O tempo e custo do projeto devem ser analisados. Deve-se avaliar se foi aceitável em comparação com o inicialmente previsto;
- Caso o tempo ou custo tenha sido muito maior do que o aprovado, ações corretivas para projetos futuros devem ser definidas.

4.5 As disciplinas do RUP

A seguir são descritas as disciplinas do RUP (KRUCHTEN, 2003):

4.5.1 Gerenciamento do projeto

O propósito do gerenciamento em um projeto de software é balancear os objetivos de negócio, riscos e superar os obstáculos, para entregar um produto de qualidade. Este produto deve atender às necessidades do cliente do projeto e dos usuários finais. O objetivo da disciplina de gerenciamento de projetos, no RUP, é tornar a tarefa de gerenciamento o mais simples possível, provendo um direcionamento de como atuar nesta área.

A disciplina possui três itens:

- Prover um processo para o gerenciamento de projetos de software;
- Descrever práticas para auxiliar no planejamento, execução e controle de projetos;
- Prover um processo para gerenciamento de riscos.

A disciplina de gerenciamento de projeto do RUP não cobre todos os aspectos de gerenciamento de projeto tradicional como:

- Gerenciamento de pessoas: Contratação, treinamento, etc;
- Gerenciamento de custos;

- Gerenciamento de contratos com fornecedores e clientes.

Esta disciplina foca nos aspectos mais específicos de um desenvolvimento iterativo de software:

- Planejar um projeto iterativo por meio das fases do ciclo de vida do RUP;
- Planejar uma iteração específica;
- Gerenciamento de riscos;
- Monitorar o progresso de um desenvolvimento iterativo.

4.5.2 Modelagem do negócio

Na disciplina de modelagem de negócio, os objetivos são: entender a estrutura e dinâmica de trabalho da empresa onde o sistema deve ser instalado; entender os problemas atuais da empresa onde o sistema deve ser instalado e identificar possíveis melhorias; assegurar que os clientes, os usuários finais e desenvolvedores tenham um entendimento comum sobre a empresa alvo; derivar os requisitos de sistema para suportar a empresa alvo. Para atingir estes objetivos a disciplina de modelagem de negócios descreve como desenvolver uma visão da empresa alvo.

A modelagem de negócio deve seguir padrões similares aos usados na engenharia de software, para facilitar a leitura dos modelos.

4.5.3 Engenharia de requisitos

Os objetivos da engenharia de requisitos são: estabelecer e manter a concordância entre o cliente e os demais envolvidos com o projeto, sobre o que o sistema deve fazer; prover aos desenvolvedores do sistema um melhor entendimento sobre o que deve ser construído; definir as fronteiras ou escopo do sistema; prover a base para o planejamento do conteúdo técnico de cada iteração; prover a base para a estimativa de tempo e custo do sistema; definir a interface com o usuário, com foco nas necessidades e objetivos do usuário final.

O RUP define um requisito como uma condição ou capacidade que o sistema deve possuir. Os requisitos estão divididos em dois tipos:

- Requisitos funcionais: Definem o comportamento do sistema frente ao usuário final. Determinam os usos possíveis que o sistema pode ter.

- Requisitos não funcionais: Este é o conjunto de requisitos necessários para garantir a qualidade do sistema, dando suporte aos requisitos funcionais. Os requisitos não funcionais são criados visando garantir requisitos de usabilidade, confiabilidade, desempenho ou para facilitar a manutenção e o uso do sistema.

4.5.4 Análise e Projeto

O objetivo da disciplina de análise e projeto é traduzir os requisitos numa especificação que descreva como implementar o sistema. Para realizar essa tradução é necessário entender bem os requisitos e transformá-los em uma arquitetura de sistema, por meio da seleção da melhor estratégia de construção.

A análise tem como objetivo transformar os requisitos do sistema, com o intuito de mapear a área em que o sistema deve atuar. Essa transformação é guiada pelos casos de uso e depois é complementada com os requisitos não funcionais. A análise cobre apenas os aspectos funcionais do sistema e visa garantir sua correta implementação, por meio de especificações de classes e demais artefatos de análise. Por outro lado, o objetivo da etapa de projeto é adaptar o resultado da análise às restrições impostas pelos requisitos não funcionais. Por exemplo, requisitos de desempenho, ambiente de desenvolvimento, etc. A arquitetura é um refinamento da análise, seu foco é otimizar a arquitetura do sistema, enquanto garante a cobertura completa dos requisitos.

4.5.5 Implementação

São propósitos da disciplina de implementação: definir a organização do código por meio da implementação de subsistemas organizados em camadas; implementar classes e objetos por meio de componentes; testar os componentes desenvolvidos como unidades funcionais; integrar os componentes para obter uma versão executável completa do sistema.

O escopo dos testes, dentro da disciplina de implementação, resume-se aos testes unitários. Testes de sistema ou testes de integração pertencem ao escopo da disciplina de testes. Três conceitos precisam ser conhecidos na disciplina de implementação:

- Construção do sistema *(Builds)*: uma versão operacional do sistema ou de parte de um sistema, para demonstrar um conjunto de capacidades que o produto final pode realizar.

- Integração: diferentes componentes de software são integrados como um sistema. A integração é realizada ao longo de toda fase de construção, não existe uma grande integração apenas no final.
- Protótipos: São criados para reduzir riscos e avaliar a viabilidade de implementação do sistema. Protótipos podem ser criados com o objetivo de explorar um determinado comportamento do sistema, verificar aspectos arquiteturais, trazer apenas conhecimento para ser usado no futuro, ou mesmo como base para fazer um desenvolvimento incremental, onde o protótipo acaba se tornando o sistema final.

4.5.6 Testes

Em vários aspectos a disciplina de teste atua como um prestador de serviço para as demais disciplinas. Os testes têm como ênfase, primariamente, avaliar ou atingir a qualidade desejada do produto. Para atingir essa qualidade, algumas práticas são realizadas: encontrar e documentar defeitos e problemas no software gerado; aconselhar o gerenciamento sobre a qualidade observada do software; avaliar se as premissas definidas, durante o levantamento de requisitos e na definição de escopo, se mostraram verdadeiras na prática; validar que o produto funciona conforme solicitado e que os requisitos foram construídos corretamente.

A principal diferença da disciplina de teste para as demais disciplinas está na forma que se deve pensar. Enquanto as demais disciplinas focam em fazer a coisa certa, garantir a consistência e a melhor arquitetura, a disciplina de teste deve procurar pensar no que pode dar errado, qual teste pode causar erros no software e expor as falhas, a serem corrigidas antes de ir para produção.

A disciplina de teste avalia as premissas, confere se os riscos ocorreram, analisa as incertezas que foram geradas pelas demais disciplinas. Teste não deve ser uma atividade isolada ou uma fase de um projeto para se atingir a qualidade desejada. Para que o desenvolvimento ocorra realmente com qualidade, os testes devem ser realizados ao longo de todo o ciclo de desenvolvimento.

Deve-se testar as funções, arquitetura, desempenho enquanto ainda é possível realizar as correções e antes de se tornarem problemas reais no software final. Os testes devem garantir a qualidade do sistema. Isso não significa apenas verificar se existem defeitos no sistema, mas também se ele faz o que deveria fazer. A qualidade deve garantir que as funções, o desempenho, a usabilidade e a confiabilidade do produto final estejam em concordância com as necessidades do cliente e usuário final.

Os testes podem ser classificados em quatro níveis:

- Testes unitários: A menor porção testável de um componente (uma unidade) é testada, normalmente pelo desenvolvedor, durante a implementação.
- Testes de integração: cada unidade testável do sistema é avaliada junto com outras unidades do sistema, visando observar a correta integração das mesmas.
- Teste de sistema: Um sistema é testado, que pode ser um subsistema de outro.
- Teste de aceitação: O sistema completo é testado com todos seus componentes e subsistemas.

4.5.7 Gerenciamento de configuração e mudança

O propósito do gerenciamento de configuração e mudança é manter a integridade dos artefatos criados durante o projeto. Durante o desenvolvimento do projeto muitos artefatos de valor são gerados. Garantir que esses artefatos estejam com suas versões corretas e facilitar a localização e reuso é tarefa dessa disciplina. Ao mesmo tempo a equipe do projeto precisa manter o controle da evolução e das mudanças no produto. Capturar e gerenciar estas requisições de mudanças é papel do gerenciamento de mudanças. Finalmente deve trabalhar em conjunto com a gerência do projeto, reportando as versões dos artefatos, quais artefatos existem, as mudanças solicitadas e seus impactos.

4.5.8 Ambiente de desenvolvimento

O propósito da disciplina de ambiente de desenvolvimento é prover suporte ao desenvolvimento de software por meio de ferramentas e padrões a serem utilizados na construção do sistema. É responsabilidade dessa disciplina definir a configuração do RUP que deve ser utilizada, que artefatos precisam ser gerados, refletir e sugerir melhorias no processo, ferramentas e linguagens a serem utilizadas no desenvolvimento.

4.5.9 Implantação

A disciplina de implantação tem como propósito implantar o produto final desenvolvido no ambiente a ser utilizado pelo usuário final. As seguintes atividades fazem parte dessa disciplina:

- Testar o software no ambiente operacional final (beta teste);
- Empacotar o software para entrega final;

- Distribuir o software;
- Instalar o software no ambiente de produção;
- Treinar o usuário final e equipe de vendas;
- Migrar os sistemas atuais e converter banco de dados.

4.6 Considerações finais do capítulo

O RUP descreve um processo complexo, que pode ser utilizado com o nível de formalismo alto, quando necessário, mas que permite a adaptação desta formalidade para cada caso. Porém, essa adaptação no formalismo não é claramente definida pelo processo.

Diferentemente do XP, o RUP possui fases bem definidas de concepção e elaboração, dando o foco em planejamento característico desse tipo de processo, mesmo quando considerado o uso de várias iterações dentro do RUP. A fase de elaboração de cada iteração sugere a necessidade de requisitos mais estáveis, para o processo caminhar pelas suas fases posteriores de construção e implantação, o que não ocorre no XP.

Pelas razões descritas neste capítulo, o RUP é efetivamente um processo tradicional com foco no planejamento, sendo adequado como exemplo deste tipo de processo. No próximo capítulo são descritas as pesquisas que serviram de base para os fatores utilizados pelo método de seleção deste trabalho.

5 Fatores para Seleção de Processos

5.1 Introdução

O principal objetivo deste trabalho é obter um método que permita uma adequada seleção entre processos tradicionais e ágeis a cada novo projeto. Por essa razão, foi realizada uma pesquisa por fatores que permitam a criação desse método, analisando trabalhos de autores reconhecidos e bem referenciados por suas contribuições acadêmicas já realizadas nessa área. Entre os autores foram encontrados três trabalhos muito adequados aos objetivos desse trabalho e que mostram conceitos relevantes para a criação do método de seleção proposto. Neste capítulo são descritos três trabalhos, que servem como base do método para seleção de processos de desenvolvimento.

5.2 Os princípios e conceitos de Cockburn

Cockburn (2000) sugeriu um método para seleção de processo baseado em número de pessoas e criticidade. O princípio base do método de Cockburn (2000) sugere considerar o uso de múltiplos processos de desenvolvimento a serem utilizados, de acordo com o projeto e o contexto de desenvolvimento (COCKBURN, 2000) (LITTLE, 2005). Três conceitos foram estabelecidos neste estudo de Cockburn (2000):

- **Tamanho do processo:** é o número de elementos de controle incluindo produtos gerados (Ex: documentação), padrões usados, número de atividades, marcos do projeto, quantidade de medições de qualidade, entre outros.
- **Densidade do processo:** é o grau de detalhe que cada um dos itens definidos pelo processo necessita. Por exemplo, define o nível de detalhe exigido na documentação dos requisitos do projeto. Quanto maior a densidade, mais rígidos são os controles sobre cada item do processo.
- **Peso do processo:** é o tamanho do processo multiplicado por sua densidade.

5.2.1 Princípios para seleção de processos

Segundo Cockburn (2000) quatro princípios devem ser considerados na seleção do tipo de processo (tradicional ou ágil) a ser utilizado:

a) Peso do processo

“Um grupo grande precisa de um processo mais pesado” (Cockburn, 2000).

Um processo é mais pesado quando contém maior número de elementos (papéis, produtos, padrões, etc.). Devido ao fato de que um processo existe primordialmente para coordenar pessoas, quanto maior o número de pessoas, maior deve ser o processo, necessitando um melhor detalhamento de cada atividade. Portanto, este princípio estabelece que se um processo foi criado para coordenar o trabalho em equipes grandes, não funciona bem para uma equipe pequena, e vice-versa.

b) Criticidade do sistema

“Um sistema mais crítico - aquele cujos defeitos não detectados produzirão mais danos - precisa ter um maior controle sobre a quantidade de defeitos em sua construção, exigindo um maior formalismo” (Cockburn, 2000).

A criticidade de um sistema é definida em 4 níveis de perda:

- **Perda de conforto:** no caso de falha do sistema, as pessoas apenas ficam menos confortáveis com o resultado obtido.
- **Perda discreta de dinheiro:** a falha do sistema produz a perda de valores relativamente pequenos de dinheiro.
- **Perda irrecuperável de dinheiro:** a falha do sistema pode levar até mesmo à falência da empresa.
- **Perda de vidas:** pessoas podem morrer se o sistema falhar.

Esse princípio estabelece que, dependendo da criticidade do sistema sendo desenvolvido, um maior gasto com a prevenção contra falhas é justificável. Por outro lado, quando o sistema não é suficientemente crítico para justificar um processo mais pesado, deve-se buscar outro, mais leve. Ou seja, o processo deve estar de acordo com o nível de perda que a falha nele pode causar.

c) Custo e Benefício do processo

“Um aumento relativamente pequeno no tamanho ou densidade de um processo adiciona uma quantia relativamente grande aos custos do projeto” (Cockburn, 2000).

Interromper o desenvolvimento para coordenar pessoas e atualizar documentos de requisitos são atividades que consomem tempo. Esse princípio não determina quando estas atividades são

necessárias ou não, mas pede para ser levado em conta que, adicionar elementos e controles a um processo custa tempo e dinheiro.

Existe uma correlação entre o tamanho do processo, tamanho do projeto e tamanho do problema. Para relativamente poucas pessoas, um processo pequeno é necessário. Com menos peso, o trabalho se torna mais produtivo. Com maior produtividade, esses indivíduos podem resolver problemas relativamente maiores. Evidentemente, existe um limite para o tamanho do problema com que uma equipe pequena pode lidar, mas a eficiência da equipe é melhor relativamente.

Por outro lado, um projeto com mais pessoas requer maior trabalho de coordenação, o que indica a necessidade de um processo mais pesado. A produtividade individual da equipe diminui em processos mais pesados. Dessa forma, para um mesmo tamanho do problema, mais pessoas são necessárias. No entanto, o peso do processo cresce de maneira mais lenta que o número de pessoas.

Existem meios para se determinar o tamanho do projeto como pontos de função, pontos de caso de uso, mas determinar o real tamanho do problema, no início do projeto, é algo bem difícil. Muitas vezes o projeto pode ter seu escopo aumentado durante o desenvolvimento, isto pode ocorrer caso seja percebido que o problema é maior do que o inicialmente previsto. Além disso, a dificuldade pode variar, dependendo das pessoas envolvidas no projeto.

A figura 15 ilustra que, quanto maior o problema, mais pessoas são exigidas para resolvê-lo. Se o número de pessoas aumenta muito, os limites dos processos mais leves são atingidos, sendo necessário o uso de processos mais adequados para se coordenar equipes maiores.

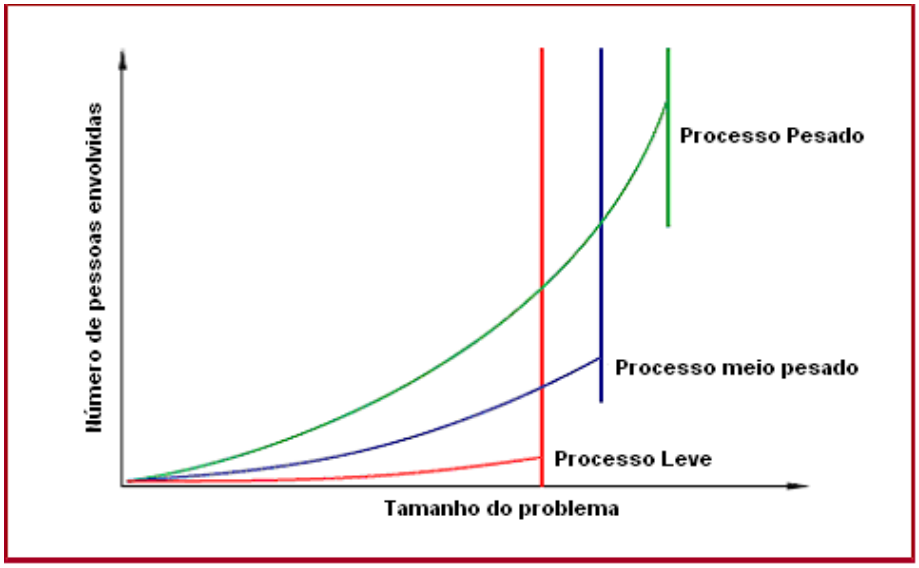

Figura 15 – O tamanho do problema, número de pessoas e peso do processo – Fonte: (COCKBURN, 2000)

d) Eficiência na Comunicação

"A forma mais efetiva de comunicação é face a face" (PLOWMAN, 1995 apud COCKBURN, 2000).

Esse princípio estabelece que pessoas próximas, com fácil contato entre si, desenvolvem software com maior facilidade. A figura 16 mostra que a eficiência na comunicação diminui quando, por exemplo, as pessoas passam da comunicação face a face para conversa por telefone, até tornar-se ainda menos eficiente por meio de documentos escritos.

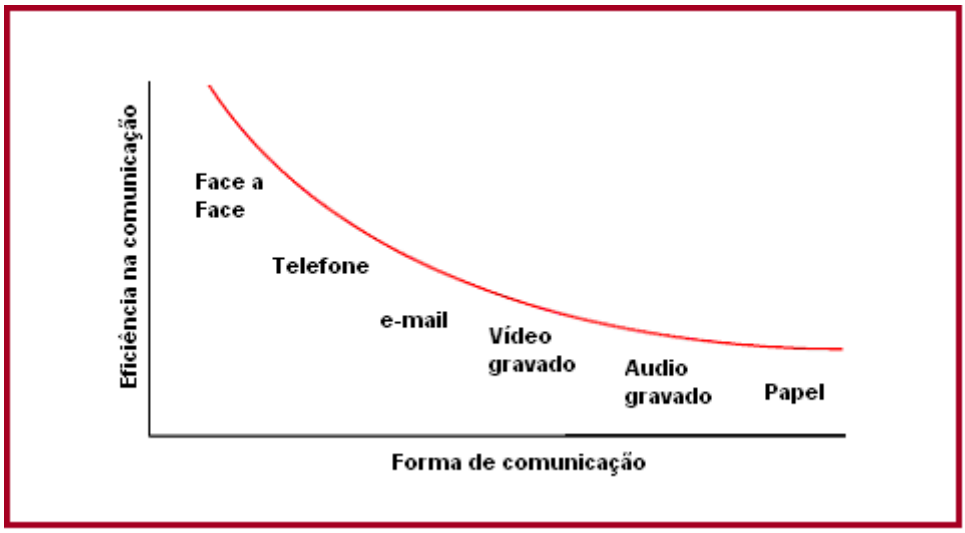

Figura 16 – Eficiência na comunicação – Fonte: (PLOWMAN, 1995 apud COCKBURN, 2000)

5.2.2 Fatores de projeto

Além dos princípios descritos, dois fatores devem ser considerados:

a) Conhecimento das prioridades do projeto

É muito importante que o patrocinador indique quais fatores são mais importantes para o projeto: tempo, qualidade ou custo. (COCKBURN, 2000). Além disso, ele pode informar outras prioridades do projeto a serem obrigatoriamente cumpridas.

b) Familiaridade da equipe com o processo

Esse fator deve ser considerado também na hora da escolha do processo a ser utilizado. Se um projeto tem data limite de entrega e a equipe não estiver familiarizada com o processo escolhido, mesmo que mais adequado em médio prazo, deve ser preterido em prol de outro

processo a que a equipe esteja acostumada, garantindo a data de entrega. O ideal é contar com pessoas que tenham conhecimento em cada um dos processos sob seleção (COCKBURN, 2000).

5.2.3 Matriz de seleção de processos

Com base nos fatores e princípios descritos anteriormente, Cockburn (2000) elaborou uma matriz que sugere a classificação do projeto de acordo com a sua criticidade e número de pessoas. Esta matriz é mostrada na figura 17.

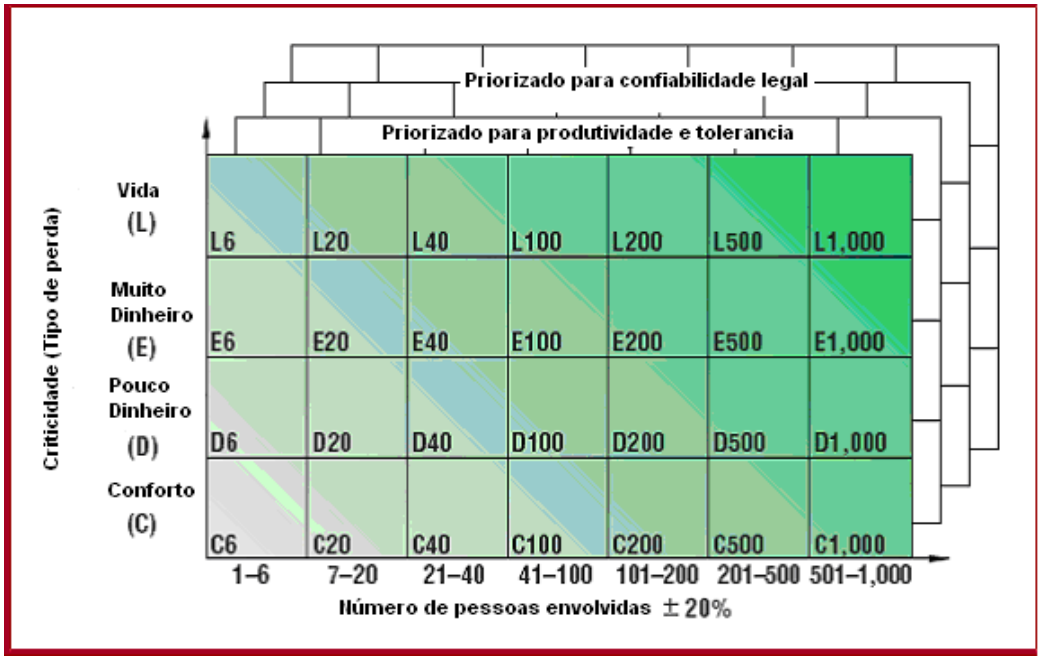

Figura 17 – Matriz de classificação de projetos – Fonte: (COCKBURN, 2000)

Cada célula indica o máximo que ela atende em termos de criticidade e número de pessoas. A prioridade define se a matriz deve ou não ser deslocada para a direita. Por exemplo, se um projeto tem seis pessoas e foi classificado com criticidade de perda de conforto, ficaria na célula C6, mas dependendo de como foi priorizado pode ser deslocado para outra célula. As prioridades citadas na figura 17 são apenas exemplos, outras podem ser estabelecidas e os deslocamentos de células na Matriz devem ser calibrados para a realidade de quem a utiliza.

5.3 Os cinco fatores de Boehm e Turner

Boehm e Turner (2003) propõem um modelo para seleção das atividades necessárias de um processo, baseado em uma análise de riscos. Nesse modelo, eles definem alguns valores e os principais fatores considerados na seleção das atividades a serem usadas, dentro de um processo. Nele é sugerido o levantamento dos riscos do projeto, centrado nos riscos de

ambiente, risco de uso de processos ágeis e de uso dos processos tradicionais. Cada risco é então classificado em cinco níveis: risco mínimo; risco moderado; risco grave controlável; risco muito grave controlável; risco não aceitável (Tabela 2).

Tabela 2 – Classificação do grau de risco – Fonte: (BOEHM; TURNER, 2003)

Grau Risco	Descrição
1	Mínimo
2	Moderado
3	Grave controlável
4	Muito grave controlável
5	Risco não aceitável

Boehm e Turner (2003) mostram um exemplo da análise de riscos. No exemplo foram comparados três projetos denominados Gerenciador de Eventos, SupplyChain.com e NISCM:

- **Pequeno com baixa criticidade:** O Gerenciador de Eventos é um sistema para controlar eventos como conferências e convenções. Este tipo de sistema se enquadra em padrões de riscos, normalmente observados em aplicações disponibilizadas por meio de *web services* (BOEHM; TURNER, 2003).
- **Criticidade Média:** O SupplyChain.com é um sistema para controle de toda a cadeia de fornecedores e clientes de uma empresa. Os riscos deste tipo de sistema foram derivados da experiência da *ThoughtWorks* em tentar usar o XP em projetos com mais de 50 pessoas (BOEHM; TURNER, 2003).
- **Muito grande e altamente crítico:** O NISCM é um sistema para o controle de crises de um país. Os riscos foram derivados de padrões de riscos encontrados pela agência de pesquisas avançadas norte-americana. Trata-se de um sistema desenvolvido por uma equipe com mais de 2000 pessoas (BOEHM; TURNER, 2003).

A tabela 3 mostra como é a análise de riscos dos três projetos citados, utilizando a pontuação conforme seu grau de risco (Tabela 2).

Tabela 3 – Mapeamento dos riscos – Fonte: (BOEHM; TURNER, 2003)

Riscos		Gerenciador de Eventos	Supplychain.com	NISCM
Riscos do ambiente	Incertezas tecnológicas	2	3	4
	Muitos envolvidos	1	2	4
	Complexidade do sistema	1	2	4
Riscos para uso de processos ágeis	Escalabilidade	1	3	5
	Uso de arquitetura simples	1	2	5
	Saída de pessoas da equipe	3	2	5
	Falta de pessoas habilitadas no processo	1	2	5
Riscos para uso de processos tradicionais	Mudança rápida de requisitos	5	3	3
	Necessidade de resultados rápidos	5	3	3
	Requisitos emergentes	5	3	3
	Falta de pessoas habilitadas no processo	2	2	3

Se algum dos riscos levantados for considerado como não aceitável, uma decisão já pode ser obtida em prol do processo que não apresente o mesmo grau de risco. No exemplo da tabela 3, riscos não aceitáveis foram localizados no projeto NISCM e Gerenciador de Eventos, permitindo a escolha de um processo tradicional e ágil respectivamente nesses casos.

Outro conceito importante, presente no trabalho de Boehm e Turner (2003), descreve o habitat dos processos ágeis e tradicionais. O habitat de um processo é definido como o conjunto de características que compõe o ambiente de desenvolvimento do projeto. Após o mapeamento dos riscos é importante verificar em qual habitat o projeto se enquadra, conforme descrito na tabela 4.

Tabela 4 - Habitat dos processos ágeis e tradicionais – Fonte: (BOEHM; TURNER, 2003)

Habitat dos processos ágeis e tradicionais			
Grupo da característica	**Características do projeto**	**Processos ágeis**	**Processos tradicionais**
Aplicação	Metas primárias	Agregar valor rápido. Resposta rápida a mudanças.	Previsibilidade, estabilidade e alto grau de confiabilidade.
	Tamanho	Equipe e projetos pequenos.	Equipe e projetos grandes.
	Ambiente	Turbulento, muitas mudanças, foco no projeto.	Estável. Poucas mudanças. Foco na organização.
Gerenciamento	Relação com o cliente	Clientes dedicados. Foco no desenvolvimento incremental de funções priorizadas.	Poucas Interações com o cliente. Foco no contrato.
	Planejamento e controle	Planos internalizados. Controle qualitativo.	Planos documentados. Controle quantitativo.
	Comunicação	Conhecimento tácito entre as pessoas.	Conhecimento documentado explícito.
Técnico	Requisitos	Estórias e casos de testes informais. Sujeito a mudanças não previstas	Projeto, capacidade, interface e qualidade formalizados. Evolução dos requisitos previstos, sem novos requisitos.
	Desenvolvimento	Arquitetura simples. Pequenos incrementos. Assume que a reengenharia de código tem baixo custo.	Planejamento extensivo da arquitetura. Grandes incrementos. Assume que a reengenharia de software tem custo alto.
	Testes	Casos de testes automatizados garantem os requisitos. Testes unitários automatizados.	Planos e procedimentos de teste documentados.
Pessoal	Clientes	Dedicados, próximos fisicamente, colaborativos, representativos, munidos de autoridade, comprometidos e dotados de conhecimento.	Idem, mas não necessariamente próximos fisicamente, podem não estar facilmente disponíveis.
	Nível dos Desenvolvedores (*)	Pelo menos 30% de especialistas nível 2 e 3 em tempo integral. Nenhum nível 1B ou nível -1.	50% do nível 3 no início. 10% durante o projeto. 30% de nível 1B aceitável; nenhum nível -1.
	Cultura	Conforto e poder de decisão por meio de muitos graus de liberdade (baseado em confiança).	Conforto e poder de decisão através de políticas e procedimentos (baseado em ordem).
(*) Consultar tabela 5			

Após a análise de riscos e da comparação do ambiente do projeto a ser desenvolvido, com o habitat ideal de cada tipo de processo, deve-se definir os cinco fatores de Boehm (2003). São sugeridos cinco fatores que devem ser analisados:

1. Nível de conhecimento: Indica a capacidade de cada membro da equipe em se adaptar às mudanças durante o desenvolvimento do projeto. O nível de conhecimento é

graduado conforme tabela 5 definida por Boehm (2003), que estende uma definição anterior feita por Cockburn (2001).

Tabela 5 – Nível de conhecimento da equipe – Fonte: (BOEHM; TURNER, 2003)

Nível	Características
3	Habilidade para revisar um processo e mudá-lo mesmo em situações completamente novas e imprevistas.
2	Habilidade para adaptar um processo a uma nova situação prevista.
1A	Com treinamento se torna hábil para executar passos discretos de um processo, pode dimensionar estórias para que caibam em um incremento; com experiência, pode se tornar nível 2.
1B	Com treinamento se torna hábil para executar passos de um procedimento, como reengenharias simples de código, codificar um programa simples; com experiência pode dominar algumas habilidades do nível 1A.
-1	Pode ter habilidades técnicas, mas é incapaz de colaborar ou de seguir um processo.

2. Criticidade: Indica o nível de perda que falhas podem causar à organização. São classificadas em 5 níveis: perda de conforto; perda discreta de dinheiro; perda irrecuperável de dinheiro; perda de uma vida ou perda de muitas vidas.
3. Número de pessoas envolvidas: Indica o número de pessoas envolvidas com o desenvolvimento do projeto.
4. Cultura: Indica a forma como a organização prefere lidar com a gestão de conhecimento dentro da empresa. Conhecimento tácito ou explícito (documentado).
5. Dinamismo: Indica o percentual de mudanças por mês previsto para o projeto.

A análise dos fatores propostos por Boehm e Turner (2003) deve ser feita colocando-se o projeto em um gráfico polar. Para exemplificar, analisando-se o projeto Supplychain.com tem-se um gráfico polar como mostrado na figura 18.

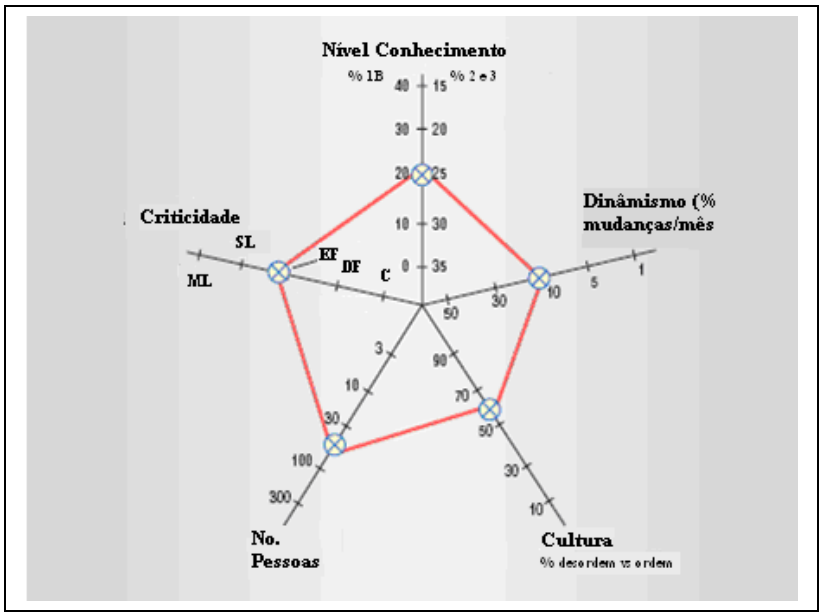

Figura 18 – Habitat do sistema Supplychain.com – Fonte: (BOEHM; TURNER, 2003)

Quanto menor a área do pentágono resultante do gráfico polar (Figura 18), maior a tendência aos processos ágeis. Se a área for grande, com os pontos tendendo às extremidades, o uso de processos tradicionais é sugerido. Observando-se o gráfico polar deste exemplo, comparando os riscos levantados e analisando os habitats de cada tipo de processo, pode-se concluir que no caso do projeto SupplyChain.com um processo ágil é mais adequado. A análise desse mesmo sistema, quando considerado em outra empresa, pode gerar um gráfico muito diferente. Fatores como a cultura da empresa, que pode ser baseada em excessiva documentação pela falta de confiança, ou mesmo empresas com equipes menos experientes, podem gerar um resultado distinto do gráfico.

5.4 Análise de Complexidade e Incerteza

Little (2005) criou um cartão de pontuação (*Scorecard*) baseado na complexidade e no grau de incerteza de cada projeto. Usando estes dois fatores e aplicando a fórmula definida por ele, pode-se graduar o projeto e encontrar o processo mais adequado.

5.4.1 Complexidade

O primeiro fator que deve ser analisado é a complexidade do projeto, cuja estrutura determina sua complexidade.

A tabela 6 resume como fazer a pontuação da complexidade. Os fatores que definem a complexidade são:

- Tamanho da equipe
- Criticidade
- Localização da equipe
- Capacidade da equipe
- Conhecimento do domínio
- Dependências externas

Tabela 6 – Atributos de complexidade e seus valores – Fonte: (LITTLE, 2005)

Pontuação de complexidade					
Pontos:	**1**	**3**	**5**	**7**	**10**
Tamanho da equipe	1	5	15	40	100 >
Criticidade	Especulativa.	Pequena base de usuários.	Mercado estabelecido.	Crítico com grande base de usuários.	Segurança crítica com grande exposição.
Localização da equipe	Mesma sala.	Mesmo prédio.	Distância viável por carro.	Mesmo fuso horário +/- 2h.	Vários locais do mundo.
Capacidade da equipe	Equipe de especialistas já formada.	Nova equipe de especialistas.	Equipe mista de especialistas e novatos.	Equipe com experiência limitada e poucos especialistas.	Equipe nova com maioria de novatos.
Conhecimento do domínio	Desenvolvedores conhecem domínio tanto quando os especialistas.	Desenvolvedores conhecem bem o domínio.	Desenvolvedores precisam de ajuda sobre o domínio.	Desenvolvedores tem pouco conhecimento do domínio.	Desenvolvedores não tem alguma idéia sobre o domínio.
Dependências externas	Nenhuma.	Limitada, pequena.	Moderada.	Significativa.	Integração com vários projetos.

Tamanho da equipe

O número de pessoas envolvidas no projeto contribui para o aumento da complexidade do mesmo.

Criticidade

Um projeto que envolve risco de vida, ou que seja vital ao negócio, deve ser tratado de maneira diferente dos casos em que a falha no projeto representa, no máximo, a perda do investimento realizado.

Localização da equipe

Ter todos os membros da equipe na mesma sala facilita muito a comunicação dentro do projeto. Em contrapartida, quanto mais separada estiver a equipe, mais difícil é a comunicação, criando obstáculos durante seu desenvolvimento.

Capacidade da equipe

Uma equipe formada de especialistas, que já vem trabalhando junto há algum tempo, pode até mesmo antecipar o que os demais membros irão precisar e como devem fazer suas atividades. Por outro lado, uma equipe somente com novatos reage de maneira menos previsível aos problemas. Em muitos casos, pode até não saber como resolver um problema sem a ajuda de especialistas.

Conhecimento do domínio

Se a equipe que vai desenvolver o sistema possui conhecimento total sobre o domínio da aplicação, o projeto se torna mais simples. Mas se esse conhecimento está apenas com os especialistas e o acesso a eles não for simples, a complexidade para o desenvolvimento aumenta.

Dependências externas

Esse atributo mede o grau de dependência da equipe com outras, fora do projeto ou mesmo com outros projetos, dentro da mesma empresa. Em geral, quanto mais dependências externas um projeto tem, maior é sua complexidade.

5.4.2 Incerteza

Após pontuar a complexidade do projeto, deve-se verificar qual o grau de incerteza que o mesmo apresenta. A tabela 7 resume como fazer a pontuação do fator de incerteza, que depende das condições do mercado e restrições do projeto (LITTLE, 2005).

Tabela 7 – Atributos de incerteza e seus valores – Fonte: (LITTLE, 2005)

Pontuação de Incerteza					
Pontos:	**1**	**3**	**5**	**7**	**10**
Incertezas do mercado	Entregas definidas e conhecidas. Obrigação contratual.	Pequenas mudanças nas expectativas e metas do mercado.	Necessidade de prospecção do mercado.	Incertezas significativas sobre o mercado.	Mercado novo, não testado e desconhecido.
Incertezas técnicas	Melhorias em arquitetura existente.	Arquitetura já conhecida.	Não existe certeza de qual arquitetura deve ser usada.	Não existe nada sobre a arquitetura do sistema. É necessário pesquisa.	Tecnologia completamente nova. Nova arquitetura.
Duração	1-4 semanas.	6 meses.	12 meses.	18 meses.	24 meses.
Dependências, flexibilidade do escopo	Escopo bem definido. Obrigações contratuais.	Muitas interfaces já existentes. Escopo não muito flexível.	Escopo possui alguma flexibilidade.	Escopo altamente flexível.	Nenhuma interface publicada. Escopo desconhecido.

Os indicadores primários que determinam o fator de incerteza são:

- Incertezas do mercado
- Incertezas técnicas
- Duração do projeto
- Dependências e flexibilidade de escopo

Incertezas do mercado

Se as necessidades do mercado são bem conhecidas, o projeto, provavelmente, pode ter seu controle reduzido. Em contrapartida, se as necessidades não são bem conhecidas, o controle do projeto se torna mais complexo e demanda maior habilidade de seu gerente, para atingir as metas. Essas incertezas do mercado podem levar a um maior número de mudanças nos requisitos, demandando maior agilidade no tratamento das mudanças.

Incertezas técnicas

Produtos maduros que usam tecnologias já comprovadas não apresentam muitas incertezas durante seu desenvolvimento. Equipes desenvolvendo produtos novos freqüentemente encontram muitas incertezas ao longo do projeto.

Duração do projeto

Quanto mais tempo o projeto durar, maior é a probabilidade de incertezas técnicas e de mercado afetarem as entregas.

Dependências e flexibilidade de escopo

É o grau de dependência que os outros projetos têm em relação ao que está sendo analisado. Isso pode limitar a capacidade de controle dentro do projeto. Mudanças freqüentes de interface ou escopo não são toleradas, quando muitos outros projetos dependem deste.

Tabela 8 – Fórmulas de Little – Fonte: (LITTLE, 2005)

Complexidade = $2^{\Sigma \log_{10} x_i}$	Onde: Xi é o valor encontrado para cada um dos atributos de complexidade.
Incerteza = $2^{\Sigma \log_{10} y_j}$	Onde: Yi é o valor encontrado para cada um dos atributos de incerteza.

Uma vez realizada a pontuação da incerteza e complexidade do projeto, deve-se usar a fórmula criada por Little (2005), mostrada na tabela 8, a fim de se obter os valores para complexidade e incerteza do projeto.

Little (2005) sugere quatro metáforas ao se classificar o projeto sob análise. Para se encontrar em qual delas o projeto se enquadra, deve-se localizar no gráfico da figura 19 o ponto exato para os valores de complexidade e incerteza encontrados.

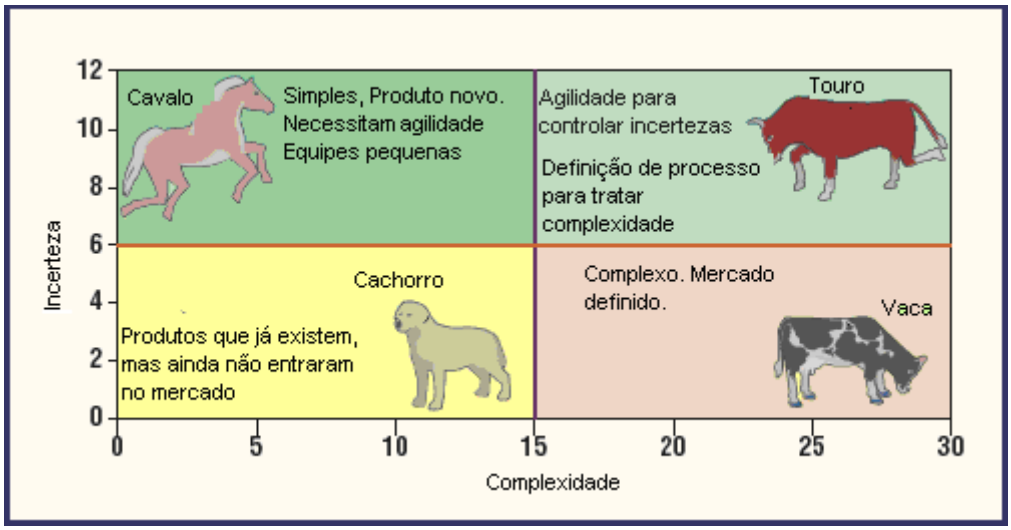

Figura 19- Os quadrantes de complexidade e incerteza – Fonte: (LITTLE, 2005)

Dependendo do quadrante em que o projeto for classificado, um processo mais ágil ou tradicional pode ser considerado mais adequado. A descrição de cada uma das metáforas, e que tipo de processo é mais adequado em cada quadrante, são descritos a seguir:

Cachorro: Projetos simples com baixo grau de incerteza. Em geral são produtos desenvolvidos por equipes pequenas. Nesse tipo de projeto, com baixa complexidade e incerteza, o melhor é deixar a equipe de desenvolvimento fazer seu trabalho. A documentação excessiva é ineficiente e inadequada.

Cavalo: Projetos simples com alto grau de incerteza. Novos produtos têm em geral tanto incertezas técnicas como de mercado.

Vaca: Projetos complexos com pouca incerteza. Produtos maduros que ainda permanecem com grandes equipes de projeto podem ser encaixados nessa metáfora. As vacas são normalmente grandes, mas não se movem com rapidez. Nesse tipo de projeto a agilidade é algo não tão necessário. Na realidade, o que realmente se faz necessário é um controle disciplinado, evitando ou reduzindo assim o impacto de mudanças, para os clientes ou projetos que dependem deste produto.

Touro: Projetos complexos com muitas incertezas. Este tipo de projeto cria problemas em todos os aspectos, sendo necessário, ao mesmo tempo, agilidade para lidar com as incertezas e certo formalismo para lidar com a complexidade. A metáfora do touro se aplica nesse caso, pois se trata de grandes projetos que podem facilmente fugir ao controle. Em geral possuem grande visibilidade dentro da empresa.

5.5 Considerações finais do capítulo

Conforme mostrado neste capítulo é de grande importância realizar uma seleção consciente do processo de desenvolvimento a ser utilizado. O método de seleção de Cockburn (2000) determina que a seleção adequada do processo de desenvolvimento seja feita levando-se em conta o número de pessoas e a criticidade do projeto. A partir desses fatores nasceu uma família de processos de desenvolvimento denominada *Crystal* (Cockburn, 2001). Embora o método de seleção usado na família *Crystal* seja interessante, outros autores como Boehm e Turner (2003) e Little (2005) mostram que apenas o número de pessoas e criticidade não são suficientes em todos os casos. Os fatores número de pessoas, criticidade e a prioridade, descritos no trabalho de Cockburn (2000), são utilizados no método de seleção desse trabalho.

O segundo método analisado proposto no trabalho de Boehm e Turner (2003) categoriza os projetos de acordo com cinco fatores: número de pessoas, criticidade, dinamismo, nível de conhecimento e cultura. Os cinco fatores são usados no método de seleção desse trabalho. Outro ponto utilizado foi o conceito de habitat de cada tipo de processo. O método de Boehm, Turner (2003) também sugere a realização de uma análise de riscos para determinar as características de cada projeto. Porém, a análise de risco não foi utilizada, pois fazer este processo poderia tornar o método de seleção subjetivo e com variações muito grandes de resultado. Pretende-se definir um método objetivo, com fatores quantitativos para a seleção de processos de desenvolvimento. O motivo para se considerar uma análise de risco como algo subjetivo é que, para um mesmo risco pode-se ter uma classificação bem diferente, dependendo de quem faz a análise.

No terceiro método pesquisado, Little (2005) propõe um método bem objetivo baseado na análise de complexidade e incerteza. Pôde-se perceber que seu método tem objetivo bem similar ao desse trabalho, porém não direciona a seleção para nenhum processo específico de desenvolvimento mais conhecido, apenas ao acréscimo ou retirada de atividades usadas dentro do contexto da empresa onde ele atua. Por essa razão, seu trabalho sozinho não é suficiente para a escolha pretendida, que tem como objetivo a seleção entre processos amplamente conhecidos como RUP e XP. São utilizados então os fatores de complexidade e incerteza de Little e suas tabelas de pontuação, mas para a definição do processo adequado entre XP e RUP outros fatores precisam ser considerados.

O método de seleção do processo de desenvolvimento de software é baseado nos fatores e conceitos descritos nos trabalhos de Cockburn (2000), Boehm (2003) e Little (2005). Foram

usados XP e RUP como exemplos de processo ágil e tradicional respectivamente, que podem ser considerados mais adequados às necessidades atuais das organizações. O próximo capítulo descreve em detalhes o método de seleção proposto.

6 Método para Seleção de Processo de Desenvolvimento de Software

6.1 Introdução

Adotar um processo de desenvolvimento, sem fatores claros para sua escolha, pode se tornar mais um problema do que uma solução. Se por um lado, os processos ágeis como *Extreme Programming* (XP), prometem aumento na satisfação do cliente, menor taxa de defeitos e menor tempo para o desenvolvimento, por outro lado, os processos tradicionais como o RUP pretendem dar previsibilidade, estabilidade e alto grau de segurança. Ambas as abordagens têm seus pontos fortes e fracos e, dependendo do contexto, uma pode ser mais adequada que a outra. Não considerar essas diferenças na escolha do processo a ser usado, pode levar qualquer projeto ao fracasso. (BOEHM; TURNER, 2003)

Com base nas pesquisas realizadas e descritas no capítulo anterior, este capítulo descreve em detalhes o método de seleção de processos proposto e as suas atividades. São descritos cada um dos fatores que devem ser considerados, na hora de se optar pelo uso de um determinado processo para desenvolvimento de software. Com esses fatores pode-se definir qual o processo, ágil ou tradicional, é mais adequado para um determinado projeto de software. Após a descrição do método, uma análise é realizada, comparando os fatores do método de seleção com práticas do XP e RUP, ambos descritos anteriormente, para analisar se estas estão realmente em concordância com os conceitos usados pelo método de seleção.

6.2 Visão Geral

O método de seleção deste capítulo define as atividades necessárias para a escolha adequada do processo de desenvolvimento. A seqüência de atividades deste método foi ordenada respeitando-se a complexidade de cada uma, colocando sempre antes atividades que requerem informações que possam ser mais facilmente obtidas. Para definir as atividades por complexidade foi verificado o que existia em comum entre todos os trabalhos mencionados anteriormente e que servem de base deste método. Os fatores comuns são colocados logo no início. Quando somente o levantamento destes fatores mais básicos não permite a conclusão, outros, mais específicos, são necessários.

A seleção é dividida em três passos discretos, onde ao final de cada passo existe a possibilidade da decisão pelo processo de desenvolvimento mais adequado, conforme ilustrado na figura 20.

No **Passo 1** é onde os fatores citados como necessários em todos os trabalhos pesquisados são levantados. Porém, estes fatores isoladamente não permitem a definição do processo mais adequado se o habitat onde o projeto está inserido ainda não é conhecido. No **Passo 2** é onde o habitat é classificado, conhecendo este habitat deve-se verificar se ele apresenta condições mais favoráveis a um processo tradicional ou a um processo ágil. Após a aplicação do método por algumas vezes, se for verificado que o habitat não está se modificando entre os projetos, o passo 1 pode levar a decisão final apenas com os três fatores básicos de número de pessoas, criticidade e prioridade. Mas apenas no caso do habitat ser estável e não interferir no uso de nenhum dos tipos de processo o passo 1 é decisivo. Este cenário pode ocorrer, por exemplo, se a equipe de projeto se repetir e estiver dentro de uma mesma empresa. Quando o habitat não é constante, que permita a definição no passo 1, e ao mesmo tempo não se mostre como fator decisivo permitindo a decisão no passo 2, o passo 3 deve ser realizado. No **Passo 3** é onde os fatores técnicos de complexidade e incerteza são definidos e levam a uma conclusão final do processo de desenvolvimento mais adequado.

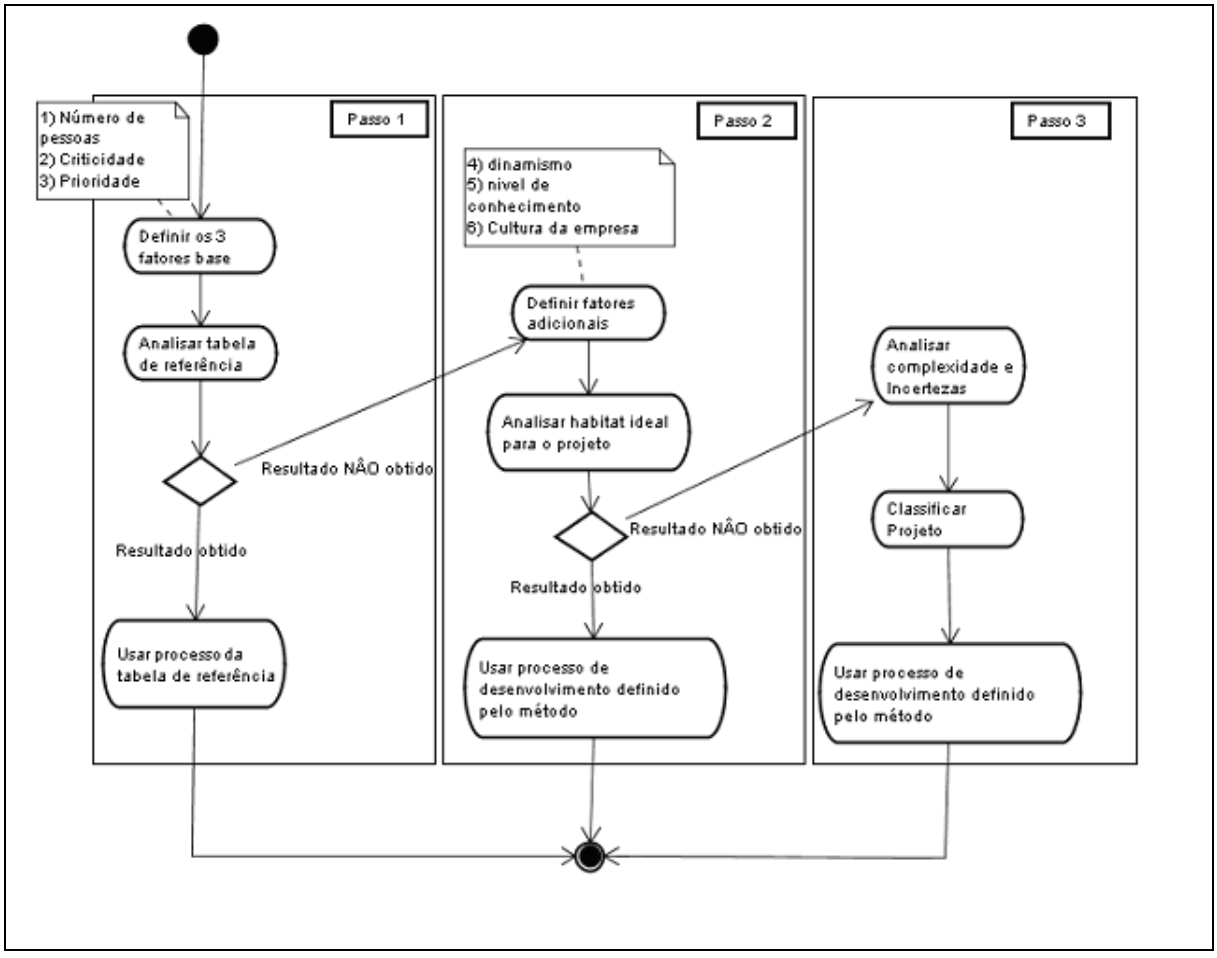

Figura 20 – Diagrama de atividades do método de seleção

6.3 Detalhamento dos Passos do Método

A seguir são descritos em detalhes os passos do método de seleção e as atividades que devem ser realizadas em cada passo.

6.3.1 Passo 1

O primeiro passo do método define e quantifica os fatores básicos. Este passo é importante para o correto andamento do processo de escolha, apesar de muitas vezes ainda não ser suficiente para a escolha definitiva. Visando permitir mais frequentemente uma decisão nesse passo, uma tabela de referência é descrita, e pode ser evoluída, conforme a pessoa que aplica o método perceber que os fatores do passo 2 estão iguais entre os diversos projetos. As atividades desse passo são descritas a seguir.

6.3.1.1 Definir os três fatores base

Três fatores extraídos do estudo de Cockburn (2000) e Boehm e Turner (2003) fazem parte desta atividade neste passo do método de seleção:

Fator 1)Número de pessoas: Para a realização do passo 1 o gerente do projeto precisa levantar quantas pessoas estão envolvidas no desenvolvimento do projeto. Apenas as pessoas diretamente envolvidas durante o desenvolvimento precisam ser contadas. Não devem ser considerados futuros envolvidos como, por exemplo, a área de marketing ou central de atendimento, a menos que estejam diretamente envolvidos com o desenvolvimento do projeto. O gerente do projeto deve conhecer esse número no início, até mesmo porque deve realizar as alocações necessárias de recursos.

Fator 2)Criticidade: Após conhecer o número total de envolvidos, o gerente do projeto deve definir a criticidade do projeto, que deve ser classificada em um dos cinco níveis possíveis, que foram levantados a partir dos níveis de criticidade definidos nos três trabalhos que embasaram este método (COCKBURN, 2000)(BOEHM; TURNER, 2003)(LITTLE, 2005). Os cinco possíveis níveis de criticidade são:

a) Perda de conforto (Nível 1): baixa criticidade, em caso de falha não prejudica nem a imagem nem as finanças da empresa. Por exemplo, um sistema interno para avaliação de funcionários.

b) Perda de dinheiro em pequena base de usuários (Nível 2): Representa uma perda relativamente pequena de dinheiro, mas que deve ser considerada. A exposição é pouca na imagem da empresa, já que a base de usuários não é significativa.

c) Perda de dinheiro em mercado estabelecido (Nível 3): Representa uma perda considerável de dinheiro em um mercado estabelecido que não tolere uma falha grave. Com essa criticidade, uma falha grave pode representar um fracasso definitivo da empresa no mercado onde está entrando, mas não representa risco para a empresa como um todo.

d) Perda de dinheiro em ambiente crítico e irrecuperável (Nível 4): Representa uma perda irrecuperável de dinheiro. Esta criticidade pode até mesmo representar a falência da empresa.

e) Perda de vidas (Nível 5): Projetos que podem levar a perda de vidas sempre representam grandes investimentos em dinheiro e não podem ter falhas de nenhuma natureza.

O gerente de projeto deve definir a criticidade formalmente com o patrocinador e com o cliente do projeto. Existindo o acordo entre os envolvidos sobre a criticidade do projeto, ele pode usar o valor no método de seleção de processo de desenvolvimento.

Fator 3)Prioridade: A prioridade do projeto é o valor mais simples de ser percebido. Esta prioridade é definida de acordo com as necessidades do projeto. Deve ser determinada com base em: tempo, custo ou escopo/qualidade. A prioridade tempo ocorre quando existe claramente uma data de entrega limite, como o caso de um sistema para as eleições ou copa do mundo. A prioridade custo ocorre quando existe um investimento máximo pré-definido em moeda corrente. Por exemplo, o custo de desenvolvimento do sistema não pode ultrapassar a 10.000 reais. A prioridade denominada escopo ou qualidade ocorre, quando o mais importante é garantir que todos os requisitos sejam entregues na íntegra, sem faltar nenhum detalhe e sem falhas. Isso é mais comum em sistemas altamente críticos. Não é possível combinar as prioridades, sempre se deve ter claramente definida qual delas deve guiar o projeto entre: tempo, custo ou escopo/qualidade.

Depois de realizado o levantamento dos três fatores base, essa primeira atividade do passo 1 está finalizada. O método então leva a segunda atividade deste passo.

6.3.1.2 Analisar tabela de referência

É mostrada aqui a tabela de referência que pode ser evoluída ou modificada, de acordo com a necessidade de quem a utiliza. Porém, é importante ter conhecimento que essa tabela foi

extraída dos princípios mostrados por Cockburn (2000). Ou seja, para modificá-la estes princípios listados a seguir precisam ser considerados:

- Um grupo grande precisa de um processo mais pesado.
- Um sistema mais crítico exige um maior formalismo.
- Um aumento relativamente pequeno no tamanho ou densidade de um processo adiciona uma quantia relativamente grande aos custos do projeto.
- A forma mais efetiva de comunicação é face a face.

Eles são essenciais para uma análise e evolução da tabela de referência desse passo. Caso a análise ainda não seja possível, deve-se usar a tabela de referência padrão do método de seleção.

6.3.1.3 Usar processo da tabela de referência

Uma vez que o método esteja sendo aplicado recentemente, dentro de uma empresa onde ainda não exista histórico para modificar a tabela, deve-se utilizar a tabela de referência mostrada nessa atividade (Tabela 9).

Com os valores levantados para os três fatores base, deve-se procurar uma linha na tabela de referência com esses valores. Na linha encontrada deve-se verificar qual o direcionamento que o método de seleção sugere.

Tabela 9 – Valores de referência para o passo 1

Linha	Número de pessoas	Criticidade	Prioridade	Conclusão
1	Entre 1 e 20	1	Tempo	Ágil se fatores do passo 2 já se mostrarem estáveis. Senão passo 2
2	21 ou mais	1	Tempo	Seguir passo 2
3	Entre 1 e 20	2 ou 3	Tempo	Ágil se fatores do passo 2 já se mostrarem estáveis. Senão passo 2
4	21 ou mais	2 ou 3	Tempo	Seguir passo 2
5	Entre 1 e 20	1	Escopo/ Qualidade	Ágil se fatores do passo 2 já se mostrarem estáveis. Senão passo 2
6	21 ou mais	1	Escopo/ Qualidade	Tradicional se fatores do passo 2 já se mostrarem estáveis. Senão passo 2
7	Entre 1 e 6	2	Escopo/ Qualidade	Ágil se fatores do passo 2 já se mostrarem estáveis. Senão passo 2
8	Entre 7 e 20	2 ou 3	Escopo/ Qualidade	Seguir passo 2
9	21 ou mais	3	Escopo/ Qualidade	Tradicional se fatores do passo 2 já se mostrarem estáveis. Senão passo 2
10	Entre 1 e 20	1	Custo	Ágil se fatores do passo 2 já se mostrarem estáveis. Senão passo 2. Senão passo 2
11	21 ou mais	1	Custo	Seguir passo 2
12	Entre 1 e 6	1	Custo	Ágil se fatores do passo 2 já se mostrarem estáveis. Senão passo 2. Senão passo 2
13	7 ou mais	2 ou 3	Custo	Seguir passo 2
14	Qualquer número de pessoas	4	Qualquer prioridade	Seguir passo 2
15	Qualquer número de pessoas	5	Qualquer prioridade	Tradicional se fatores do passo 2 já se mostrarem estáveis. Senão passo 2

6.3.2 Passo 2

No segundo passo do método, fatores adicionais, ainda não levantados no passo 1, devem ser definidos e quantificados. Este passo 2 permite em muitos casos a decisão do melhor processo de desenvolvimento. As atividades deste passo são descritas a seguir.

6.3.2.1 Definir fatores adicionais

Nesta atividade são definidos os valores para os três fatores adicionais:

Fator 4)Nível de conhecimento: O nível de conhecimento é um fator que deve ser calculado considerando-se a equipe de desenvolvimento que está no projeto. A classificação deve ser feita de acordo com a tabela 5, já mostrada anteriormente, mas repetida aqui na tabela 10 para facilitar o entendimento deste método de seleção. Fazendo a classificação de cada pessoa que compõe a equipe de desenvolvimento nesses níveis de conhecimento, tem-se a porcentagem de pessoas em cada nível. Essa porcentagem permite a análise posterior do habitat do projeto. O nível de conhecimento da equipe de desenvolvimento é bastante relevante na decisão pelo processo de desenvolvimento de software. Processos baseados em agilidade e confiança, como os processos ágeis, necessitam de mais pessoas nos níveis 2 e 3. Processos mais formais e com regras claras das atividades dependem menos de pessoas qualificadas no nível 2 ou 3.

Fator 5)Dinamismo: O quinto fator a ser quantificado é o dinamismo da empresa, indicado pela porcentagem de mudanças em requisitos por mês. É um número que deve ser obtido pelo levantamento histórico de projetos anteriores que possuam características similares. Caso não exista outro projeto similar ou não exista base histórica na empresa para consulta, deve-se acordar com o cliente uma estimativa para esse valor. Por exemplo, se tivermos um projeto com 100 requisitos onde mais do que 10% deles sofrem mudanças consideráveis no seu escopo, até mesmo surgindo novos requisitos, tem-se a preferência por processos com maior agilidade para comportar melhor essas mudanças.

Fator 6)Cultura da empresa: O sexto fator a ser analisado é a cultura da empresa. Esse fator também é definido com base no histórico. Assim como no caso do dinamismo, deve-se fazer uma análise do comportamento das pessoas envolvidas no projeto atual em situações anteriores. Com isso, pode-se encontrar a porcentagem de pessoas que trabalham bem em ambiente informal e as que só trabalham bem em ambiente mais formal. Se o número de pessoas que preferem formalismo (documentação escrita) for maior que o número de pessoas que preferem certa informalidade (predomínio de comunicação verbal), a cultura é considerada

como baseada na ordem. Quando a cultura permite o predomínio do conhecimento tácito, ela é definida como baseada em conhecimento tácito, ou seja, permite que o conhecimento esteja com as pessoas e nem sempre obrigando um alto grau de documentação escrita.

Tabela 10 – Nível de conhecimento da equipe – Fonte: (BOEHM; TURNER, 2003)

Nível	Características
3	Habilidade para revisar um processo e mudá-lo mesmo em situações completamente novas e imprevistas.
2	Habilidade para adaptar um processo a uma nova situação prevista.
1A	Com treinamento torna-se hábil para executar passos discretos de um processo, pode dimensionar estórias para que caibam em um incremento; com experiência, pode se tornar nível 2.
1B	Com treinamento torna-se hábil para executar passos de um procedimento, como reengenharias simples de código, codificar um programa simples; com experiência pode dominar algumas habilidades do nível 1A.
-1	Pode ter habilidades técnicas, mas é incapaz de colaborar ou de seguir um processo.

6.3.2.2 Analisar o habitat ideal para o projeto

Após o levantamento dos valores para cada um dos fatores deste passo 2 e utilizando-se ainda de três fatores do passo 1, tem-se como resultado um gráfico polar conforme figura 21.

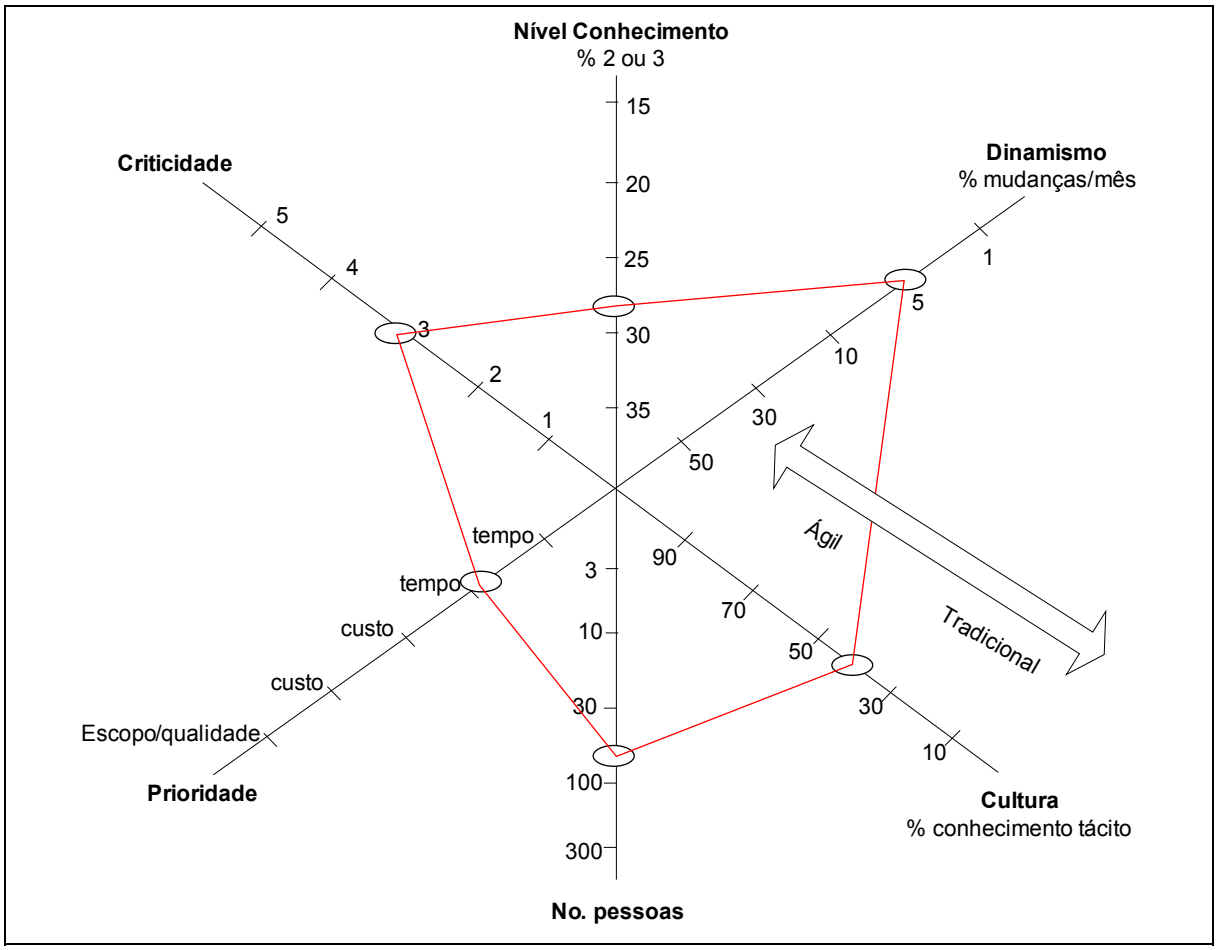

Figura 21 – Exemplo de gráfico do passo 2.

Quanto mais pontos houver, próximos ao centro no gráfico resultante, maior é a tendência ao uso de processos ágeis. Da mesma forma, se a maioria dos pontos tenderem às extremidades indica a tendência ao uso de processos tradicionais. Neste exemplo, pode-se observar três pontos mais próximos da extremidade, dois pontos mais próximos do centro e um ponto em posição intermediária.

A tabela de habitats de cada processo de desenvolvimento criada por Boehm e Turner (2003) foi analisada e modificada para possibilitar uma conclusão objetiva nesse passo. A tabela 11 descreve como analisar cada um dos seis fatores em comparação com os habitats de cada processo.

Tabela 11 – Fatores e habitats dos processos – Adaptado de: (BOEHM; TURNER, 2003)

Habitat dos processos ágeis e tradicionais			
Grupo	**Fator**	**Processos ágeis**	**Processos tradicionais**
Projeto	Prioridade	Agregar valor rápido. Resposta rápida a mudanças. Ideal para prioridade tempo e custo.	Previsibilidade, estabilidade e alto grau de confiabilidade. Ideal para prioridade Escopo/Qualidade.
	Número de pessoas	Equipe e projetos pequenos. Entre 1 e 50 pessoas	Equipe e projetos grandes. 30 ou mais pessoas.
	Dinamismo (% de mudanças/mês)	Turbulento, muitas mudanças, foco no projeto. Mais de 10% de mudanças por mês.	Estável. Poucas mudanças. Foco na organização. Menos de 10% de mudanças por mês.
	Criticidade	Arquitetura simples. Pequenos incrementos. Assume que a reengenharia de código tem baixo custo. Ideal para projetos com criticidade entre 1 a 3.	Planejamento extensivo da arquitetura. Grandes incrementos. Assume que a reengenharia de software tem custo alto. Ideal para projetos com criticidade entre 3 a 5.
Pessoal	Nível dos Desenvolvedores	Pelo menos 25 % no nível 2 ou 3 em tempo integral. Nenhum nível 1B ou -1	10% no nível 2 ou 3. 30% de nível 1B aceitável. Nenhum nível -1
	Cultura	Conforto e poder de decisão através de muitos graus de liberdade (conhecimento tácito)	Conforto e poder de decisão através de políticas e procedimentos (conhecimento escrito)

Obtendo os valores de cada fator deve-se realizar uma análise comparativa com os habitats de cada processo de desenvolvimento (ágil e tradicional). Neste exemplo (Figura 21) a seguinte análise é realizada:

1. Número de pessoas: 100 pessoas.

   Análise: Considerando um projeto com este número de pessoas, o habitat de um processo tradicional é mais indicado.

2. Criticidade: Nível 3.

   Análise: Esse fator de criticidade está no limite entre o uso de um processo ágil ou tradicional, por este motivo não é conclusivo neste caso qual o habitat mais adequado.

3. Prioridade: Tempo.

   Análise: Esse tipo de prioridade indica o uso preferencial de processos ágeis.

4. Nível de conhecimento: 28% no nível 2 ou 3.

   Análise: Com essa taxa de pessoas nos níveis 2 ou 3 o uso de processos ágeis é mais adequado.

5. Dinamismo: 5 % de mudanças por mês.

   Análise: Indica que o uso de processos tradicionais é mais adequado, dado a pequena taxa de mudanças.

6. Cultura da empresa: 40% preferem conhecimento tácito. 60% o conhecimento escrito.

   Análise: Indica o uso preferencial de processo tradicional.

No exemplo citado, o número de pessoas, o dinamismo e a cultura da empresa estão mais adaptados ao habitat dos processos tradicionais. Porém, o nível de conhecimento e a prioridade estão mais adaptados ao habitat dos processos ágeis. A criticidade pode ser classificada no limite entre os dois habitats.

6.3.2.3 Usar o processo de desenvolvimento definido pelo método

Com a análise de habitat de cada processo, em alguns casos, a tendência para um dos tipos de processos de desenvolvimento (ágil ou tradicional) já pode ser percebida. Mas, nesse exemplo, não é possível chegar a uma conclusão precisa, pois temos três pontos indicando processos tradicionais e dois indicando processos ágeis. No mínimo, quatro dos seis pontos devem apontar os processos tradicionais, para uma conclusão já nesse passo; portanto o passo 3 deve ser realizado nesse exemplo.

6.3.3 Passo 3

Os dois passos iniciais do método de seleção de processos permitem, em alguns casos, a decisão por um processo ágil ou tradicional, mas outras vezes apenas no terceiro passo pode-se obter matematicamente uma conclusão. Todos os valores e informações obtidos nos dois passos iniciais são muito importantes para este. As atividades deste passo são descritas a seguir.

6.3.3.1 Analisar complexidade e Incerteza

São definidos aqui os valores para os dois fatores adicionais deste passo:

Fator 7)Complexidade do projeto: A complexidade do projeto é definida pela análise do número de pessoas, criticidade, localização, nível de conhecimento, conhecimento do domínio e as dependências externas, explicados anteriormente no trabalho de Little (2005). Essa análise e levantamento permitem realizar a pontuação de complexidade conforme tabela 12.

Tabela 12 – Encontrar a complexidade do projeto – adaptado de: (LITTLE, 2005)

Pontuação de complexidade					
Pontos:	**1**	**3**	**5**	**7**	**10**
Qtde. Pessoas	Até 3	Até 10	Até 30	Até 100	Mais que 100
Criticidade	1	2	3	4	5
Localização	Mesma sala	Mesmo prédio	Distância viável por carro	Mesmo fuso horário +/- 2h	Vários locais do mundo
Nível conhecimento	>25% 2 ou 3 0% 1B ou -1	>25% 2 ou 3 30% 1B ou -1	15 a 25% 2 ou 3 >30% 1B ou -1	<15% 2 ou 3 >30% 1B ou -1	0% 2 ou 3
Conhecimento do domínio	Desenvolvedores conhecem todo o domínio	Desenvolvedores conhecem bem o domínio	Desenvolvedores precisam de ajuda sobre o domínio	Desenvolvedores com pouco conhecimento do domínio	Desenvolvedores não têm nenhuma idéia sobre o domínio
Dependências externas	Nenhuma	Limitada, pequena	Moderada	Significativa	Integração com diversos projetos

Fator 8)Grau de incerteza: Após ter pontuado a complexidade, deve-se fazer a pontuação para os atributos que definem o grau de incerteza do projeto. Esta é definida pela análise das incertezas de mercado, técnicas, duração do projeto e flexibilidade de escopo. A pontuação de incerteza deve ser feita com base na tabela 13.

Tabela 13 – Encontrar a incerteza do projeto – adaptado de: (LITTLE, 2005)

Pontuação de Incerteza					
Pontos:	**1**	**3**	**5**	**7**	**10**
Incertezas do mercado	As entregas são definidas e conhecidas.	Pequenas mudanças são possíveis	Necessidade de prospecção do mercado.	Incertezas significativas do mercado.	Mercado novo, não testado e completamente desconhecido.
Incertezas técnicas	Melhorias em arquitetura existente.	Arquitetura já conhecida.	Não existe certeza de qual a arquitetura deve ser usada.	Não existe nenhuma documentação sobre a arquitetura do sistema. É necessário pesquisa.	Tecnologia completamente nova. Nova arquitetura.
Duração	1-4 semanas	6 meses	12 meses	18 meses	24 meses
Flexibilidade do escopo (Dinamismo)	1% de mudanças por mês.	5% de mudanças por mês.	10% de mudanças por mês.	30% de mudanças por mês.	50% de mudanças por mês.

6.3.3.2 Classificar Projeto

Uma vez realizada a pontuação de cada um dos atributos de complexidade e incerteza, deve-se encontrar os valores finais para os dois fatores, nesse terceiro passo do método de seleção. Para se obter o valor da complexidade e incerteza do projeto, devem-se utilizar as fórmulas mostradas na tabela 14.

Tabela 14 – Fórmulas para definir complexidade e incerteza. Adaptado de: (LITTLE, 2005)

Complexidade = $2^{\Sigma \log_{10} x_i}$	Ex: Se for definida a pontuação 3 para todos os atributos de complexidade, deve-se calcular o log de 3 na base 10 para cada atributo, que resulta 0,477. Fazendo-se a somatória tem-se o número 2,86 como potência na fórmula. Por fim, elevando-se 2 a 2,86 tem-se a complexidade definida em 7,27.
Incerteza = $2^{\Sigma \log_{10} y_j}$	Ex: Se for definida a pontuação 5 para todos os atributos de incerteza, deve-se calcular o log de 5 na base 10 para cada atributo, que resulta 0,699. Fazendo-se a somatória tem-se o número 2,79 como potência na fórmula. Por fim, elevando-se 2 a 2,79 tem-se a incerteza definida em 6,94.

Após definidos os valores de complexidade e incerteza, deve-se então encontrar em qual das metáforas de Little (2005) se encontra o projeto. Considerando-se os valores colocados no exemplo da tabela 14, o projeto é classificado conforme figura 22.

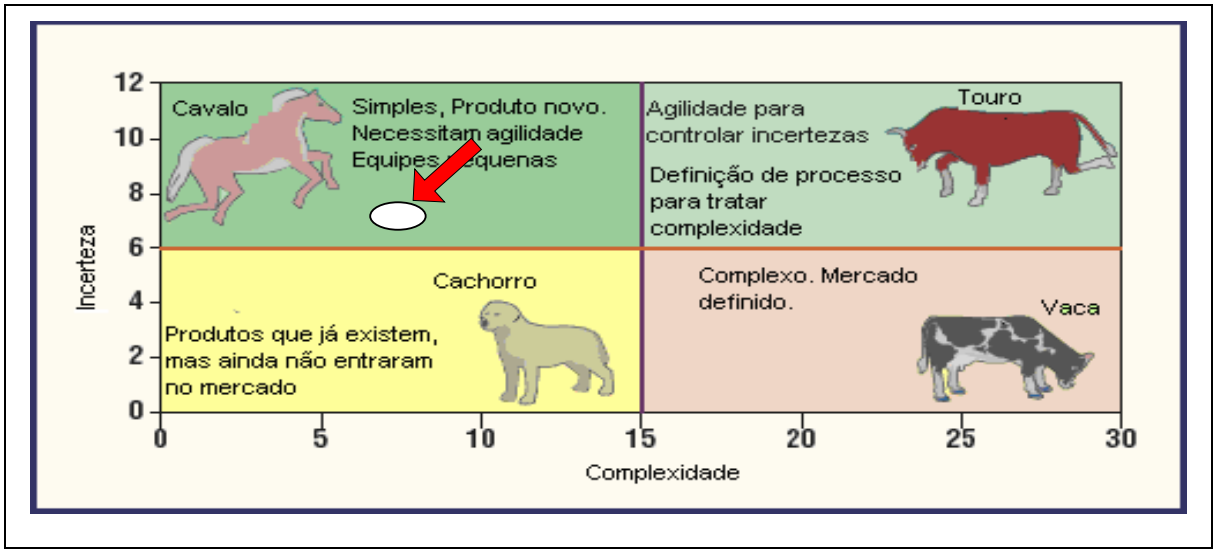

Figura 22 – Classificação do projeto – Adaptado de: (LITTLE, 2005)

6.3.3.3 Usar o processo de desenvolvimento definido pelo método

Após a localização de qual quadrante se encontra o projeto sob análise, a conclusão do melhor processo de desenvolvimento pode ser obtida. A seguir são descritos os processos mais adequados a cada metáfora (Little, 2005) utilizada por este método de seleção:

Cachorro: O uso de processos ágeis é mais indicado neste caso.

Cavalo: O uso de processos ágeis é mais indicado neste caso.

Vaca: O uso de processos tradicionais é mais indicado neste caso.

Touro: O uso de processos tradicionais é mais indicado neste caso, mas é recomendável o uso de algumas práticas de processos ágeis, para lidar com as incertezas.

Os processos ágeis devem ser utilizados se a complexidade do projeto tiver como resultado um valor entre 0 a 15, independentemente do grau de incerteza. Mas são ainda mais indicados se a incerteza for superior a 6. Os projetos com a complexidade acima de 15 pedem o uso de processos tradicionais. Porém, caso a incerteza seja superior a 6 e a complexidade for superior a 15, o ideal é balancear o processo tradicional com alguns conceitos de agilidade, para lidar com as incertezas, ou ampliar bastante a fase de concepção e elaboração como forma de dirimir as incertezas.

6.4 Análise do Método de Seleção

O método de seleção proposto neste trabalho possui três passos a serem seguidos. Por meio deles, pode-se decidir pelo uso de um processo ágil ou tradicional.

Faz-se, nessa parte do trabalho, uma análise crítica de cada um dos passos adotados pelo método de seleção, em comparação com as características dos processos de desenvolvimento RUP e XP. Isto é necessário para garantir que os processos selecionados como exemplos de processo tradicional e ágil respectivamente, sejam realmente compatíveis com os conceitos utilizados pelo método.

6.4.1 Os fatores de seleção usados pelo passo 1

- Número de pessoas.
- Criticidade.
- Prioridade (tempo, custo ou qualidade).

A seguir é feita a análise dos fatores do passo 1 de maneira individual, para se permitir uma melhor comparação com os processos XP e RUP.

6.4.1.1 Número de pessoas

O método sugere que apenas em equipes pequenas ou médias, de no máximo 50 pessoas, o uso de processos ágeis pode ser considerado mais adequado, sendo ainda mais indicado para equipes pequenas de até dez pessoas. Isto não impossibilita o uso em um projeto com mais envolvidos, mas uma divisão em projetos menores é necessária, para que a equipe de cada projeto dividido fique com o tamanho ideal.

Analisando as práticas primárias do XP pode-se perceber claramente essa tendência em manter equipes pequenas. Por exemplo, a prática de sentar todos próximos em um mesmo ambiente é fortemente recomendada pelo XP. Com um menor número de pessoas, a prática é mais facilmente utilizada.

Apesar do RUP não apresentar uma citação clara do número mínimo de envolvidos, considerando a quantidade de papéis e artefatos necessários para atender o processo, pode-se perceber que o RUP não foi pensado para equipes pequenas. O RUP sugere uma adequação do processo para equipes menores, porém não indica exatamente como pode ser simplificado. Sendo assim, pode-se concluir que o RUP mais formal não é adequado para equipes pequenas. No caso de equipes médias, entre 10 e 20 pessoas, o RUP já pode ser considerado tão adequado quanto o XP. Porém, pelo seu formalismo, tende a entregar com menos agilidade, mas com maior rastreabilidade.

6.4.1.2 Criticidade

Os processos ágeis devem ser usados, preferencialmente, em sistemas cuja criticidade esteja entre os níveis 1 a 3, definido neste método de seleção. Os processos tradicionais são mais recomendados nos níveis de criticidade entre 3 a 5.

Pelo próprio fato do XP não exigir muita formalidade, pode-se facilmente entender porque não é adequado no caso de risco de perda de vidas. Além disso, práticas como arquitetura incremental e estórias servem para ilustrar como é a dinâmica de um projeto XP. No XP, o mais importante é entregar unidades funcionais muito rapidamente e não se preocupar tanto com a arquitetura final do sistema, que deve ser ajustada ao longo do projeto e não planejada totalmente no início. Sistemas mais críticos necessitam de um maior formalismo já que podem representar inclusive perda de vidas humanas.

No caso do RUP, a rastreabilidade de todo o processo de desenvolvimento é possível, pelos artefatos exigidos em cada fase. A arquitetura base já é definida logo no início e todos os riscos

são mapeados. Isso permite inclusive que em sistemas críticos, um nível de teste mais rigoroso seja obrigatório, para mitigar os maiores riscos do projeto.

6.4.1.3 Prioridade (tempo, custo ou qualidade)

No passo 1 da seleção do processo ideal é mencionada a importância de se definir qual a prioridade do projeto, logo no início. Os projetos com tempo ou custo mais rigorosos pedem um processo mais ágil.

Nesse critério não existe apenas uma ou outra prática do XP que seja aderente, mas todas elas são pensadas para trazer um retorno mais rápido ao cliente, permitindo criar um sistema com custo mais baixo e em menos tempo. Por exemplo, práticas como espaço informativo de trabalho, estórias e ciclos semanais de planejamento, são sempre executadas de forma a gerar rapidamente um software funcional que gradualmente recebe novas funções, de acordo com o tempo e orçamento previstos.

O formalismo do RUP é algo que pode ser considerado excessivo, quando se trata de um sistema onde tempo ou orçamento são restritos. Projetos, onde a qualidade e formalismo de contratos sejam explícitos, são mais adequados para o uso de processos tradicionais como RUP.

6.4.2 Os fatores de seleção usados pelo passo 2

- Nível de conhecimento da equipe de desenvolvimento.
- Dinamismo (Quantidade de mudanças nos requisitos).
- Cultura da empresa.

A seguir é feita a análise dos fatores do passo 2, de maneira individual, para se permitir uma melhor comparação com os processos XP e RUP:

6.4.2.1 Nível de conhecimento da equipe de desenvolvimento

No passo 2 da seleção do método é mostrado que o ideal, para se usar os processos ágeis, é quando a equipe de projeto possui um percentual mínimo de 25% de pessoas nos níveis 2 e 3. No caso dos processos tradicionais o percentual mínimo é de 15%.

Nenhuma prática da XP aponta diretamente à necessidade de uma quantidade maior de pessoas nos níveis 2 ou 3. Porém, considerando práticas como arquitetura incremental, integração contínua e geração do sistema em até 10 minutos, é possível entender que realmente em projetos XP isto é mais necessário. Outro ponto é que, devido à ausência de controles mais

apurados, por meio de documentação ou atividades que facilitem a distribuição do conhecimento, um maior nível de conhecimento é requerido para o sucesso do projeto.

Processos mais formais como RUP têm menor necessidade de grande número de pessoas experientes, já que o próprio processo define artefatos para repassar o conhecimento. Alguns exemplos: casos de uso, diagramas de seqüência, entre outros.

Desta forma, no caso do RUP apenas 15% das pessoas precisam ser do nível 2 e 3. No XP este número deve ser de pelo menos 25%. Em ambos os casos, com menos de 15% de pessoas no nível 2 e 3 a qualidade necessária do produto, correria o risco de não ser alcançada e nenhum dos processos pode auxiliar.

6.4.2.2 Dinamismo (Mudanças nos requisitos)

No passo 2, quando as mudanças de requisitos no mês superam 10%, existe uma preferência pelo uso de processos mais ágeis. Quando o número é inferior a 10%, os processos tradicionais podem ser utilizados.

Práticas como: ciclos semanais de planejamento, envolvimento real do cliente e estórias foram criadas para esses cenários dinâmicos, com um grande nível de mudanças. A natureza adaptativa da XP foi idealizada exatamente para lidar com esse tipo de ambiente, sendo mais adequado em ambientes com muitas mudanças.

No caso do RUP, apesar dele lidar com as mudanças, o formalismo sugerido na gestão de mudanças e as análises de impactos tendem a ser fatores que limitam o excessivo número delas. Em ambientes turbulentos pode ocorrer uma paralisação no desenvolvimento em projetos usando RUP, até que as mudanças sejam todas avaliadas.

6.4.2.3 Cultura da empresa

Empresas, cujas culturas incentivam a criatividade, onde os funcionários são motivados a contribuir com o processo e não somente a seguir regras e procedimentos, são as ideais para uso de processos ágeis. Empresas que exigem controle maior sobre as atividades são mais adequadas aos processos tradicionais. Este fator sozinho não deve limitar a escolha do processo, pois mesmo em empresas onde a cultura direciona para um determinado processo, uma iniciativa de mudança no processo de desenvolvimento pode ser aceita, quando bem justificada. Esse método de escolha pode contribuir muito com a justificativa.

Os próprios valores da XP, tais como coragem e comunicação, práticas como programação em pares, equipe completa e espaço informativo de trabalho mostram-se mais adequados em empresas onde a confiança está mais nas pessoas e no trabalho em equipe do que nos processos. Empresas em que os funcionários não têm tanta liberdade de criar, ou o formalismo é mais requerido, tendem a optar por processos que garantam a rastreabilidade do erro; o RUP permite esta rastreabilidade.

6.4.3 Os fatores de seleção usados pelo passo 3

- Complexidade.
- Grau de Incerteza.

A seguir é feita a análise dos fatores do passo 3 de maneira individual, para se permitir uma melhor comparação com os processos XP e RUP:

6.4.3.1 Complexidade e grau de incerteza do projeto

No passo 3 têm-se os dois últimos fatores. Neste passo, nota-se que projetos de pequena até média complexidade são os ideais para o uso de processos ágeis. Quando possuem um alto grau de incerteza essa recomendação torna-se ainda mais forte.

Práticas para eliminar tarefas de documentação como a programação orientada a testes, integração contínua, entre outras, foram criadas no XP para lidar com muitas incertezas no início do projeto. Como as equipes do XP são preferencialmente pequenas, sistemas com menor complexidade enquadram-se melhor em suas práticas.

No RUP, por princípio, já se pede para analisar os riscos e mitigar aqueles que oferecem maior perigo ao projeto. Isso é muito diferente do XP, que foca em novas funções prontas o quanto antes. Porém, no caso de sistemas complexos, não julgar os riscos e apenas pensar em novas funções, não parece o mais adequado. Os processos tradicionais mostram-se mais aderentes nos sistemas mais complexos. Em projetos com muitas incertezas e pouco complexos, essa fase de análise do RUP tende a se prolongar, mesmo quando a complexidade é baixa. Nesse caso um processo mais ágil é mais adequado.

6.5 Considerações finais do capítulo

As atividades e passos descritos neste capítulo permitem a escolha do processo de desenvolvimento mais adequado, de maneira quantitativa e objetiva. Considerando a

flexibilidade requerida pelas empresas e o tempo escasso para este tipo de seleção, ter um método que possa ser rapidamente aplicado é algo muito importante. Utilizando os fatores mostrados é possível encontrar o ambiente ideal para a utilização de processos ágeis ou tradicionais, em pouco tempo.

No entanto, o método de seleção proposto pode não ser aplicável em empresas que não possuam base histórica para a definição do habitat do projeto no passo 2. Os fatores de nível de conhecimento, dinamismo e cultura dependem de análise histórica para gerarem resultados confiáveis. Empresas sem este histórico podem usar o método, mas a eficiência dele pode ser comprometida se os valores levantados com o cliente não se mostrarem reais ao longo do projeto. Por exemplo, quando o dinamismo não é um fator que possa ser obtido pelo histórico de projetos similares, um acordo a respeito do máximo dinamismo aceito pode ser definido. Porém, se a necessidade no projeto se mostrar diferente do combinado, isto pode gerar um problema para o projeto, levando inclusive o método a possivelmente indicar um processo inadequado.

A tabela 15 mostra um resumo de todas as atividades e passos do método de seleção deste trabalho. Para complementar o entendimento sobre o método de seleção, o próximo capítulo mostra um exemplo de utilização dele, permitindo avaliar, com base em casos reais, qual o processo de desenvolvimento de software é mais adequado em cada ambiente.

Tabela 15 – Resumo das atividades do método de seleção de processos

Resumo	Atividade	Localização
Levantar número de pessoas.	Definir os 3 fatores base	Passo 1
Definir criticidade do projeto.		
Definir prioridade do projeto (Escopo/Qualidade, Tempo ou Custo).		
Analisar se a tabela de referência pode ser adaptada.	Analisar tabela de referência	
Escolher método de desenvolvimento mais adequado ou tomar decisão de necessidade do passo 2.	Usar processo da tabela de referência	
Verificar Nível de conhecimento.	Definir fatores Adicionais	Passo 2
Definir dinamismo.		
Definir cultura da empresa.		
Encontrar habitat ideal do projeto.	Analisar Habitat do projeto	
Analisar fatores em comparação com cada habitat.		
Escolher método de desenvolvimento mais adequado ou tomar decisão de necessidade do passo 3.	Usar o processo de desenvolvimento definido pelo método	
Calcular complexidade do projeto.	Analisar complexidade e Incertezas	Passo 3
Calcular incerteza do projeto.		
Aplicar fórmula de Little.		
Encontrar valores de complexidade e incerteza.	Classificar Projeto	
Escolher processo de desenvolvimento mais adequado.	Usar o processo de desenvolvimento definido pelo método	

7 Aplicação do Método de Seleção

7.1 Introdução

Neste capítulo são descritos casos reais de uso de XP e RUP obtidos de artigos da literatura, usados para contextualizar a aplicação do método de seleção, proposto por este trabalho. Os dados obtidos destes casos reais são usados como entrada para cada atividade do método de seleção.

Após a aplicação do método de seleção, é mostrado qual o processo de desenvolvimento mais adequado em cada um dos cenários descritos. Por fim, é discutido se o método de seleção direcionou para o caminho adequado, quando analisado em comparação com os casos reais.

7.1.1 Um caso real de uso do XP

7.1.1.1 Descrição do cenário de aplicação do método

O caso real de uso de XP, usado como exemplo de utilização do método, foi obtido de uma empresa chamada FST, na Itália. Esta desenvolve software para Internet (MURRU, 2003).

O principal motivo da escolha deste exemplo foi em virtude da característica do ambiente de desenvolvimento na FST. Eles utilizavam o processo RUP e a linguagem de programação Java, na maioria de seus projetos, um ambiente encontrado com facilidade em empresas de desenvolvimento. Dessa forma, o caso é uma boa base de informação, na comparação entre os processos enfatizados nesse trabalho. O relato da FST mostra como a implantação do XP foi benéfica nesse ambiente (MURRU, 2003).

Foram realizados dois projetos-piloto entre 2001 e 2003, onde as práticas de XP foram testadas. O projeto 1 tinha como objetivo desenvolver alguns componentes de criptografia, enquanto o projeto 2 foi criado para desenvolver um portal de aquisições via Internet. Cada um deles durou três meses e foi realizado por três pessoas. Além das três pessoas do desenvolvimento, mais uma fez o papel de cliente do projeto. Cinco dos programadores tinham pouca experiência, no máximo dois anos, em programação. Eles não estavam familiarizados com o uso do XP, mas foram treinados para usar esse processo. A tabela 16 mostra como foram adotadas as práticas do XP para os dois projetos.

Tabela 16 – Estudo de caso FST – adaptado de: (MURRU, 2003)

Prática	**Grau de adoção**		**Comentários**
	Projeto 1	**Projeto 2**	
Ciclos semanais de planejamento	nada	completo	Melhorou muito o controle do projeto. Possibilitou avaliar freqüentemente a prioridade de cada função.
Pequenas iterações	completo	completo	Iterações de duas semanas foram utilizadas.
Metáforas (*Metaphors*)	nada	nada	Não conseguiram aplicar. Mas a prática não é mais considerada essencial no XP.
Arquitetura simples	parcial	completo	Somente incluía as funções solicitadas, sem usar muito tempo em definições de arquitetura.
Programação orientada a testes	completo	completo	Problemas para automatizar testes de interface com o usuário. Programação em pares facilita muito no sucesso dessa prática.
Arquitetura incremental	completo	completo	Consome até 20% do tempo de codificação. Necessita de programadores experientes.
Programação em Pares	completo	completo	Prática aceita com facilidade e com excelente resultado. Facilita inclusive a adoção de outras práticas, já que, quando trabalhando individualmente, muitos resistiam a algumas práticas.
Código comum	completo	completo	Usado controle de versão.
Integração contínua	parcial	parcial	Uma vez por semana.
Não fazer hora extra (*Sustainable Pace*)	completo	completo	9 horas por dia no máximo.
Representante do cliente na equipe de projeto	completo	completo	Gerente do projeto pode fazer papel do cliente.
Padrões de código	parcial	parcial	Definido padrões de código para Java e C++.

A figura 23 mostra, em cada semana do projeto, o número total de funções previstas, ilustrando quantas funções estão finalizadas, foram descartadas ou estão incompletas.

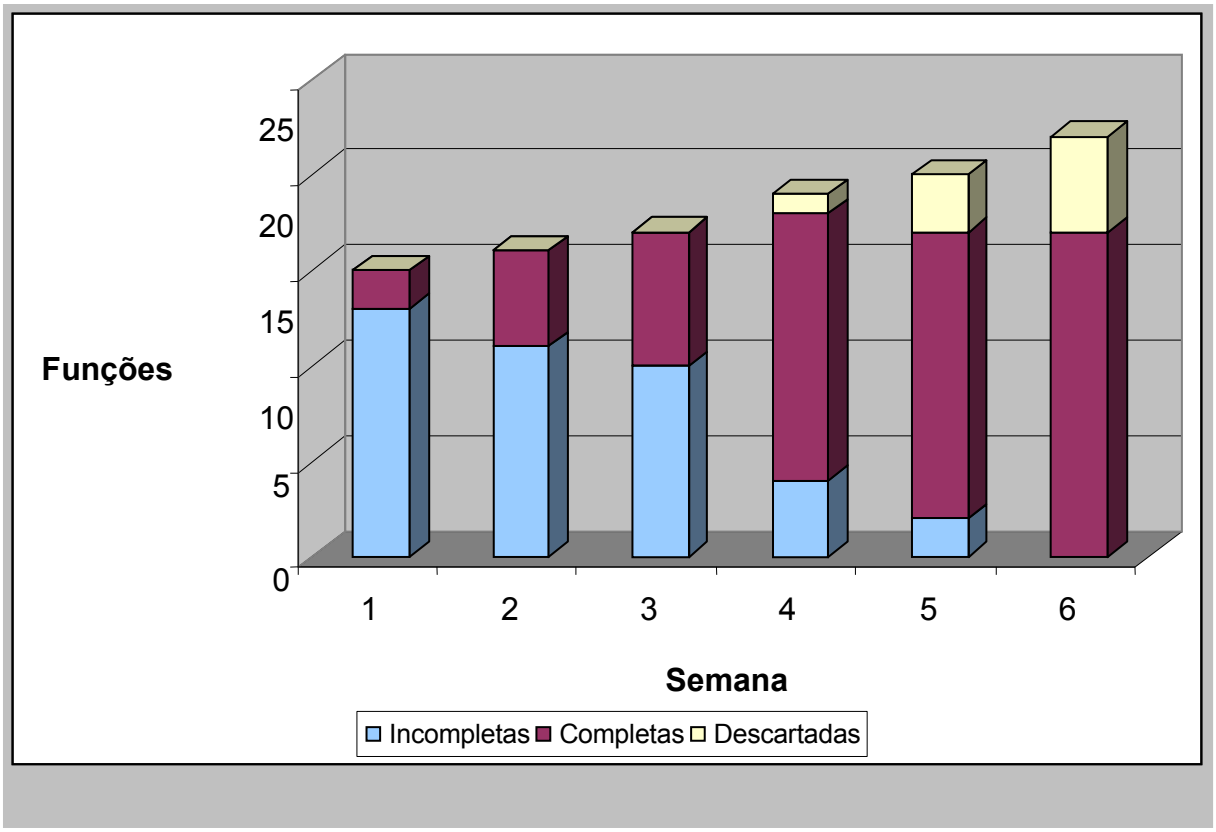

Figura 23 – Funções em cada iteração – Fonte: (MURRU, 2003)

Após a aplicação do XP na FST, foram realizadas nove entrevistas com as equipes de marketing, gerenciamento de projeto e qualidade, para se avaliar como as demais áreas da FST viram o XP. Nelas, foram levantados quais os maiores problemas que cada uma destas áreas identificou na adoção do XP. Os principais problemas levantados estão descritos na tabela 17.

Tabela 17 – Problemas para outras áreas – Fonte: (MURRU, 2003)

Área	Problema	Descrição
Marketing	Difícil de comunicar a nova metodologia de desenvolvimento	A idéia de que a natureza de desenvolvimento de software é adaptativa e que as mudanças de requisitos são normais, são difíceis de vender para quem tem pouca ou nenhuma experiência com software. Isso normalmente é entendido como uma desculpa pela falta de visão ou para evitar comprometer-se com datas de entrega.
	Clientes que não tem poder para priorizar requisitos	Alguns dos clientes da FST são grandes corporações, onde os responsáveis pela contratação somente fazem um papel político, coletando requisitos com seus superiores e repassando-os. Dessa forma, a preocupação deles é muito mais em não serem culpados pelo fracasso, do que em atingir um grande sucesso em conjunto. Preferem uma atitude conservadora com assinatura de documentos de requisitos e cronogramas detalhados a parcerias por um bom resultado.
Gerencia de projeto	Superexposição da equipe de desenvolvimento	XP expõe muito o processo interno de desenvolvimento para os clientes, fazendo com que estes possam decidir contratar programadores do projeto e montar suas próprias equipes. Com o cliente participando do projeto, as manobras para atender os requisitos ficam mais difíceis de serem feitas.
	Analistas líderes podem ficar tecnicamente desatualizados	Existe uma clara distinção entre programador e analista. Os analistas acabam se tornando líderes de equipe e raramente irão programar, isso faz com que eles fiquem tecnicamente obsoletos. Fica complicado uma mudança técnica ser adotada com eficiência por esses analistas.
Qualidade	Nenhum problema grave levantado	Não apontaram grandes problemas com adoção do XP. Inclusive a resposta foi positiva sobre a melhoria na qualidade do software com o uso do XP.

Os membros da equipe de desenvolvimento na FST ficaram entusiasmados com a adoção do XP. Uma das maiores lições aprendidas foi que não se deve deixar de lado nenhuma prática do XP, pois é o uso delas em conjunto que garante que o projeto não perca o controle, dentro da agilidade proposta pelo processo (MURRU, 2003).

Foi percebido, no estudo da FST, que com uma equipe experiente de programadores o ganho obtido com o XP é mais facilmente perceptível. Porém, mesmo em projetos com equipes inexperientes, o XP tem sido usado sem trazer muitos problemas, desde que a equipe esteja bem treinada em todas as práticas (MURRU, 2003).

A prática mais difícil de ser obtida foi conseguir o envolvimento do cliente durante todo o projeto, mas esta é uma prática saudável se pudesse ser realizada, já que ter o cliente sempre exercendo seu papel é algo que traz ganhos e contribui muito para o sucesso de um projeto. Para minimizar o problema foi colocado o líder da equipe de desenvolvimento fazendo o papel de cliente do projeto. As conversas com o cliente real foram reduzidas, elas somente se realizaram no final de cada ciclo de desenvolvimento (iteração). Isso possibilitou usar os documentos que os clientes exigem sem perder a agilidade que o XP necessita em seu dia a dia (MURRU, 2003).

7.1.1.2 A aplicação do método de seleção com uso do XP

Pode-se perceber pelo estudo da FST que a empresa em questão considerou o uso do XP muito positivo nos dois projetos descritos. Caso o método de seleção desse trabalho fosse aplicado no início do projeto, o resultado é conforme descrito a seguir.

7.1.1.2.1 Realização do Passo 1

O primeiro passo do método de seleção é onde os fatores básicos são encontrados:

Fator 1) Número de pessoas: Considerando os três programadores e o cliente do projeto, existiam quatro pessoas diretamente envolvidas. Além disso, houve interação com as demais áreas da empresa. Ou seja, mesmo com os envolvidos indiretos, o projeto tinha menos de sete pessoas, podendo ser enquadrado com número de pessoas em sete.

Fator 2) Criticidade: Nos dois projetos pode-se identificar que uma falha causaria apenas perda de conforto, pois não eram sistemas previstos para gerar receita à organização.

Fator 3) Prioridade: O principal objetivo nestes projetos era atender o escopo solicitado; não existiam definições claras com relação ao tempo máximo de cada. Mesmo considerando que um resultado rápido seria importante para mostrar na empresa a eficiência do XP, o projeto é classificado com prioridade no escopo, que é sua prioridade mais importante.

Após levantar os três fatores básicos descritos acima, deve-se procurar na tabela de referência do método de seleção, como este projeto pode ser classificado. Nesse caso, está na linha que indica entre 1 a 20 pessoas, com criticidade 1 e com prioridade no escopo/qualidade (Linha 5 da tabela 9). Esta linha da tabela indica "Ágil se fatores do passo 2 já se mostrarem estáveis. Senão passo 2". Nesse exemplo o método está sendo aplicado pela primeira vez, portanto a indicação é pela realização do passo 2.

7.1.1.2.2 Realização do Passo 2

No segundo passo, os fatores adicionais devem ser definidos e quantificados:

Fator 4) Nível de conhecimento: Das 7 pessoas do projeto apenas 1 pode ser enquadrada nos níveis 2 ou 3, de acordo com o relato. Ou seja, 15% das pessoas estão no nível 2 ou 3.

Fator 5) Dinamismo: Considerando que a empresa procurava por processos ágeis, indica que mudanças deveriam ser situações comuns na empresa. Considerando também a natureza da empresa (Internet), em geral, apresenta alto grau de dinamismo devido às características desse tipo de organização (BASKERVILLE, 2003). Para o exemplo, vamos considerar que a taxa é superior a 30% por mês.

Fator 6) Cultura da empresa: Com base nas entrevistas realizadas, pôde-se perceber que a empresa não tinha muitas restrições ao uso do novo método. Apenas algumas considerações das áreas de marketing e gerenciamento, mas a resistência era baixa. Porém, como a diretoria aprovava o novo método, pode-se considerar que a maioria estava disposta ao uso de conhecimento tácito, para usar neste exemplo foi considerado que 75% aceitavam bem o conhecimento tácito.

A próxima etapa é criar o gráfico polar com os seis fatores levantados neste exemplo (Figura 24), isso facilita a visualização do processo de desenvolvimento mais adequado e permite comparar com cada habitat de processo.

Fazendo a análise dos valores de cada fator e comparando com os habitats de cada processo de desenvolvimento, tem-se o seguinte:

1. Número de pessoas: 7 pessoas. Esse número de pessoas indica que o habitat dos processos ágeis é mais indicado.
2. Criticidade: Nível 1. Indica o uso dos processos ágeis como o mais adequado.
3. Prioridade: Escopo/Qualidade. Indica o uso dos processos tradicionais como o mais adequado.
4. Nível de conhecimento: 15% (nível 2 e 3). Com esta taxa de pessoas nos níveis 2 ou 3 o uso de processos tradicionais é mais adequado.
5. Dinamismo: 30% de mudanças por mês. Indica que o uso de processos ágeis é mais adequado com essa alta taxa de mudanças.
6. Cultura da empresa: 75% preferem conhecimento tácito. Indica o uso de processos ágeis.

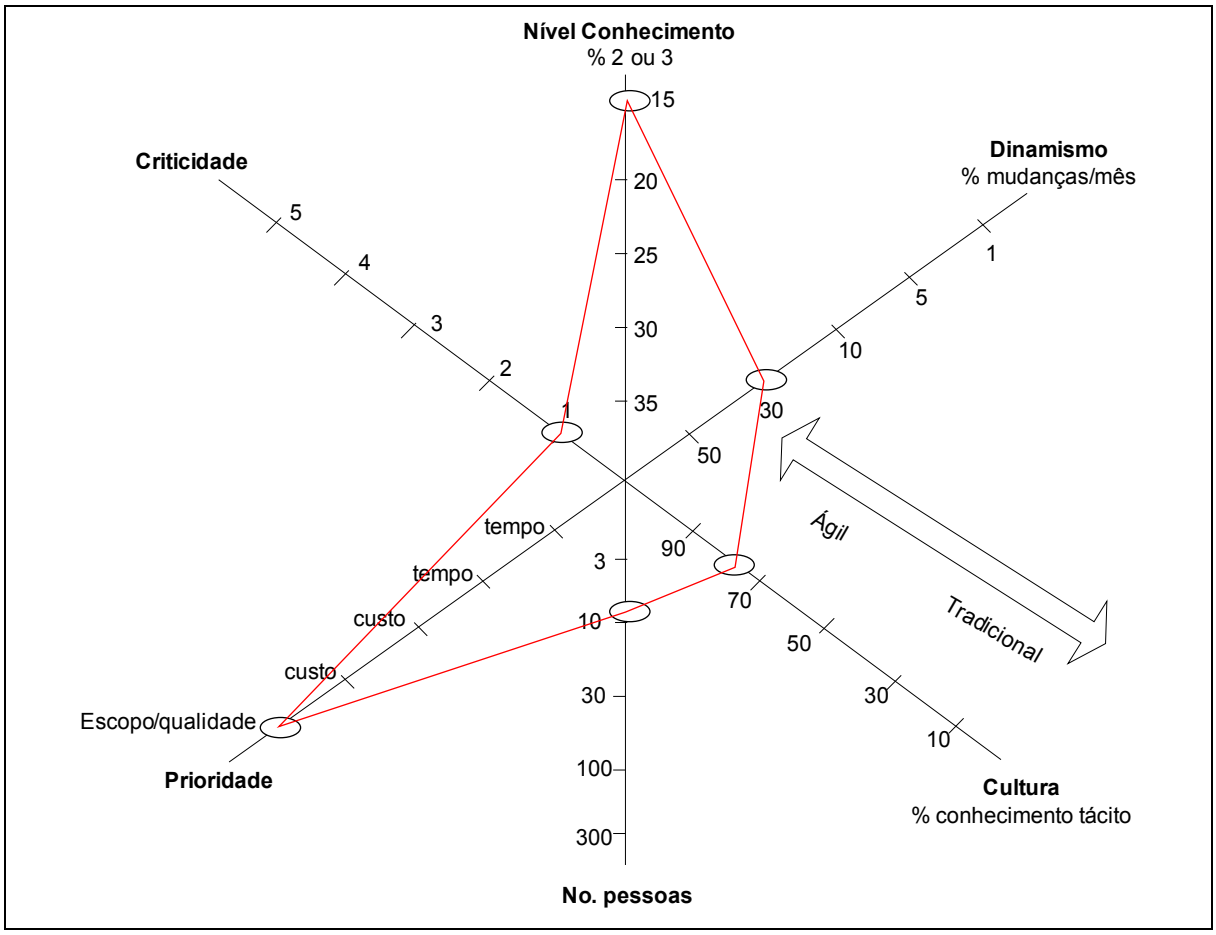

Figura 24 – Gráfico polar no estudo com XP

A conclusão que se pode obter nesse caso é que quatro pontos estavam adequados ao uso de processos ágeis; A figura está com uma área pequena, pois a maior parte dos pontos está próxima ao centro. No exemplo da FST, já é possível nesse passo a conclusão pelo uso de processos ágeis. Porém, para mostrar todos os passos foi realizado o terceiro passo, apenas para mostrar como funcionaria neste exemplo.

7.1.1.2.3 Realização do Passo 3

No terceiro passo uma conclusão definitiva deve ser obtida. Nos dois primeiros, se obtêm informações necessárias para pontuar corretamente este terceiro. No passo 3, a complexidade e incerteza do projeto são medidas. A tabela 18 mostra como é a pontuação de complexidade para este caso descrito. A tabela 19 mostra como é a pontuação de incerteza para o caso descrito.

Tabela 18 – Pontuação complexidade FST (XP)

Critério	Projeto 1	Projeto 2
Número de pessoas	3	3
Criticidade	1	1
Localização	1	1
Nível de conhecimento	5	5
Conhecimento do domínio	1	7
Dependências externas	1	5
Complexidade	2,26	6,58

Tabela 19 – Pontuação incerteza FST (XP)

Critério	Projeto 1	Projeto 2
Incertezas do mercado	1	7
Incertezas técnicas	5	5
Duração do projeto	3	3
Dinamismo	5	7
Incerteza	3,66	7,2

Aplicando-se a fórmula de Little (2005), já descrita anteriormente, ao projeto 1 tem-se um grau de complexidade em 2,26 e a incerteza em 3,66. Isso coloca o projeto 1 no quadrante denominado cachorro. No caso do projeto 2, a complexidade é de 6,58 e a incerteza ficaria em 7,2, e coloca o segundo estudo no quadrante denominado cavalo.

Em ambos os projetos, o método de seleção indica que o uso de processos ágeis realmente é o mais adequado, dado os quadrantes em que os projetos foram classificados. O relato da FST descreve um grande sucesso no uso do XP. Fazendo-se uma comparação do resultado do método com os resultados obtidos no caso real da FST, pode-se concluir que o XP mostrou-se realmente adequado ao contexto e o método de seleção, apontou para a direção adequada já no passo 2.

7.1.2 Um caso real de uso do RUP

7.1.2.1 Descrição do cenário de aplicação do método

O caso real de utilização do RUP analisado neste trabalho foi realizado na Coréia. Trata-se de um projeto feito dentro de uma universidade, onde o cliente era uma empresa de telecomunicações. O projeto foi criado visando o desenvolvimento de um componente *web* para a administração de currículos. Teve duração de um ano e foi divido em três semestres, possuía cinco pessoas na equipe de desenvolvimento. O cliente definiu os principais artefatos que precisavam ser entregues no final de cada semestre (KIM, 2005). A tabela 20 mostra quais produtos eram esperados no final de cada semestre.

Tabela 20 – Atividades em cada semestre caso real RUP – Fonte: (KIM, 2005)

Período	Atividade	Produto final
1o Semestre	Planejamento	Plano de projeto
	Engenharia de requisitos (Levantamento e especificação)	SOW
		Rascunho da especificação de requisitos (SRS)
		SRS final
2o semestre	Análise de requisitos	Documentos de análise, planos de teste.
	Arquitetura	Arquitetura detalhada do sistema
3o semestre	Construção	Executáveis, código fonte, manual do usuário

O projeto deveria seguir as boas práticas de gerenciamento de projeto. Apesar de não detalhar quais são estas boas práticas, pelos artefatos gerados conclui-se que o PMBOK foi utilizado (PMI, 2004). Dessa forma, um maior nível de formalismo já era exigido por contrato.

Alguns envolvidos já tinham experiência com o uso do RUP, mas nada foi mencionado sobre metodologias ágeis. Inicialmente a principal prioridade era atender aos requisitos e qualidade desejada para o produto final, mas ao longo do projeto ficou claro que foi necessário um ajuste de escopo, para conseguir manter a data e custo previsto (KIM, 2005).

Existia apenas uma fase para a definição de todos os requisitos, o que mostra que o nível de mudança nos requisitos é muito pequeno. Durante o projeto isso foi o que realmente ocorreu, somente foi necessário retirar funções para atender ao limite de tempo, mas não é relatada

nenhuma mudança nos requisitos iniciais. Todos os envolvidos eram alunos de pós-graduação com boa experiência no mercado de trabalho (KIM, 2005).

O projeto foi finalizado dentro do tempo e custo previsto, mas não atendeu a todos os requisitos inicialmente acordados. Porém, isso foi renegociado durante o projeto e não comprometeu a satisfação final do cliente. O projeto foi considerado um sucesso e o uso do RUP foi satisfatório. Não foi utilizado o RUP completo, e sim um RUP mais leve, usando-se apenas o necessário para atender aos artefatos solicitados pelo cliente. Nenhum teste de desempenho foi realizado, bem como a rastreabilidade dos requisitos não funcionais também deixou de ser realizada. A tabela 21 mostra as atividades de cada fase do projeto (KIM, 2005). Nas fases de concepção, elaboração e transição, apenas uma iteração foi utilizada. Na fase de construção três iterações foram utilizadas.

Tabela 21 – Objetivos de cada iteração no caso real RUP – adaptado de: (KIM, 2005)

Fase	Iteração	Objetivos
Concepção	Iteração 1	Levantamento de requisitos. Foi desenvolvida a visão do projeto.
Elaboração	Iteração 1	Modelos de arquitetura (UML).
Construção	Iteração 1	Protótipo do sistema
	Iteração 2	Implementação das funções principais.
	Iteração 3	Implementação de todas as funções.
Transição	Iteração 1	Entrega do produto.

7.1.2.2 A aplicação do método de seleção com uso do RUP

7.1.2.2.1 Realização do Passo 1

Para se iniciar as atividades do passo 1, três fatores base são levantados:

Fator 1) Número de pessoas: Conforme descrito por Kim (2005) o projeto tinha cinco pessoas envolvidas no desenvolvimento. Considerando o cliente, ao todo existem 6 pessoas envolvidas.

Fator 2) Criticidade: Trata-se de um sistema de administração de currículo para uma empresa de telecomunicações, algo com criticidade baixa nesse tipo de empresa. Portanto, pode-se classificar a criticidade do projeto como perda de conforto, nível 1.

Fator 3) Prioridade: A principal prioridade era atender o escopo solicitado com a qualidade necessária, porém o projeto precisava ser encerrado em um ano, que é uma limitação de tempo. Sendo assim, apesar de começar com prioridade em qualidade e escopo, o projeto poderia ter a prioridade alterada ao longo do desenvolvimento. Mas como a análise é realizada no início do projeto, deve-se considerar a prioridade em escopo/qualidade.

Com as informações desse exemplo e usando a tabela de referência do passo 1 tem-se como resultado a linha 5 da tabela 9, que indica: "Ágil, se fatores do passo 2 já se mostrarem estáveis. Senão, passo 2". Isso mostra uma tendência ao uso do XP. Porém, o passo 2 é necessário, se consideramos que o método está sendo aplicado pela primeira vez na empresa, não sendo possível considerar os fatores do passo 2 como estáveis.

7.1.2.2.2 Realização do Passo 2

No segundo passo os fatores adicionais devem ser definidos e quantificados:

Fator 4) Nível de conhecimento: Quase todos eram alunos de mestrado com boa experiência de trabalho; de acordo com a descrição, poderíamos classificar que mais de 35% estavam nos níveis 2 ou 3.

Fator 5) Dinamismo: Não havia expectativa de mudanças em requisitos. Por essa razão, o projeto teria seu dinamismo classificado em menos de 1% de mudanças/mês.

Fator 6) Cultura da empresa: Considerando as solicitações do cliente por artefatos específicos, pode-se perceber que existia uma preferência pelo conhecimento escrito e formal. Não é possível identificar se algum dos desenvolvedores preferiria o uso de processos ágeis,

mas não existia muita liberdade para tal, devido à cultura já estabelecida. Dessa forma, vamos classificar que todos eram favoráveis ao conhecimento mais formal e escrito.

A próxima etapa é criar o gráfico polar com os seis fatores, levantados neste exemplo (Figura 25), o que facilita a visualização do processo de desenvolvimento mais adequado e permite comparar com cada habitat de processo.

Fazendo a análise dos valores de cada fator e comparando com os habitats de cada processo de desenvolvimento, tem-se o seguinte:

1. Número de pessoas: 6 pessoas. Esse número de pessoas indica que o habitat dos processos ágeis é mais indicado.
2. Criticidade: Nível 1. Indica o uso dos processos ágeis como o mais adequado.
3. Prioridade: Escopo/Qualidade. Indica o uso dos processos tradicionais como o mais adequado.
4. Nível de conhecimento: mais de 35 % no nível 2 ou 3. Com essa taxa de pessoas nos níveis 2 ou 3 o uso de processos ágeis é mais adequado.
5. Dinamismo: inferior a 1% de mudanças por mês. Indica que o uso de processos tradicionais é mais adequado com essa taxa de mudanças.
6. Cultura da empresa: menos de 10% prefere conhecimento tácito. Indica o uso de processos tradicionais.

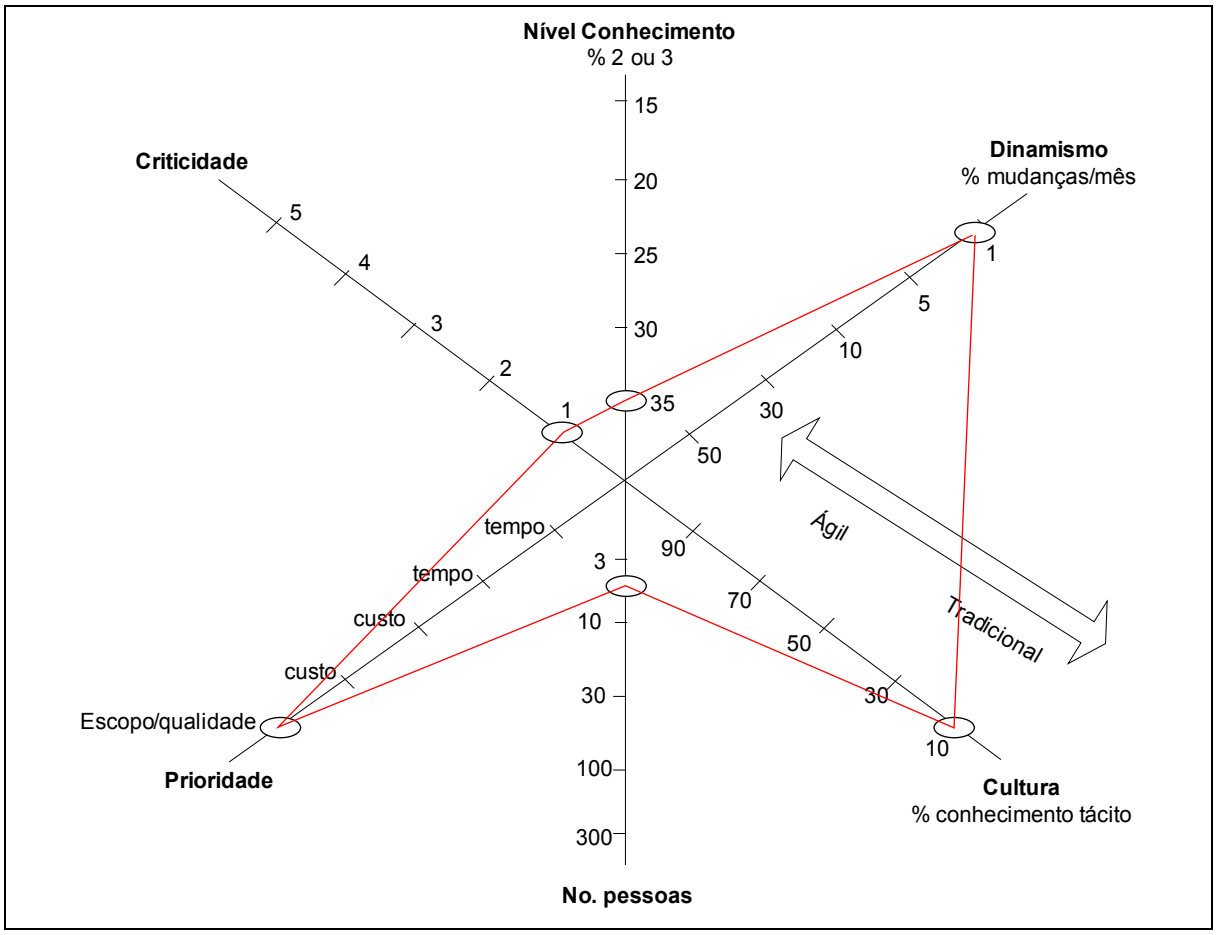

Figura 25 – Gráfico polar no estudo com RUP

A conclusão a que se pode chegar com os valores descritos neste exemplo, é que existiam fatores importantes que direcionariam para o uso de processos ágeis, incluindo o nível de conhecimento, a criticidade do projeto e o número de pessoas. Porém a cultura, o dinamismo e a prioridade apontavam fortemente para o uso de processos tradicionais. Nesse caso, três fatores apontam para o uso de processos ágeis e três apontam para o uso de processos tradicionais. Assim, não é possível uma conclusão ainda neste passo, o passo 3 precisa ser realizado para a conclusão final.

7.1.2.2.3 Realização do Passo 3

No terceiro passo devem-se usar as informações levantadas pelos passos anteriores, para pontuar cada um dos itens de complexidade e incerteza. A tabela 22 mostra como é a pontuação de complexidade, para este caso descrito. A tabela 23 mostra como é a pontuação de incerteza, para este caso descrito.

Tabela 22 – Pontuação de complexidade caso real RUP

Critério	Sistema de Currículo
Número de pessoas	3
Criticidade	1
Localização da equipe	5
Nível de conhecimento	1
Conhecimento do domínio	5
Dependências externas	3
Complexidade	5,10

Tabela 23 – Pontuação de incerteza caso real RUP

Critério	Sistema de Currículo
Incertezas do mercado	1
Incertezas técnicas	1
Duração do projeto	5
Dinamismo	1
Incerteza	1,62

Aplicando-se a fórmula do passo 3, para encontrar a complexidade e incerteza, tem-se uma complexidade de 5,10 e a incerteza de 1,62. Isso coloca o projeto no quadrante denominado cachorro, que indica o uso preferencial de processos ágeis. Observando os resultados do caso real, pode-se perceber que, para usar o RUP, muitas adaptações foram necessárias para tornar o processo mais leve, e um processo naturalmente mais leve poderia ser mais adequado. Ainda analisando os resultados, pode-se concluir que a direção apontada pelo método de seleção está correta, pois o projeto teve problemas com prazo e alguns requisitos não foram atendidos, principalmente os não funcionais. Dessa forma, apesar do sucesso apontado pelo relato de Kim (2005) no uso de RUP, o uso de um processo ágil é mais adequado, considerando todo o contexto do projeto.

7.2 Considerações finais do capítulo

Neste capítulo foi aplicado o método de seleção em situações reais de uso do XP e RUP. O exemplo de utilização do método de seleção, com o uso de dados reais, permite a análise objetiva dos resultados apontados pelo método, quando comparados com os resultados obtidos por cada um dos estudos analisados. O ponto principal foi avaliar que, nos dois exemplos, verificou-se que o método indica satisfatoriamente o processo de desenvolvimento mais adequado. Essa aplicação simulada do método facilita também o entendimento de como ele deve ser utilizado de maneira adequada.

No próximo capítulo os resultados obtidos, contribuições e trabalhos futuros são descritos, para conclusão final do trabalho.

8 Conclusões

8.1 Introdução

Este capítulo apresenta as principais conclusões e contribuições obtidas por este trabalho. Além disso, são sugeridas linhas de pesquisas adicionais, com o objetivo de aumentar a contribuição acadêmica.

8.2 Resultados obtidos

Este trabalho foi desenvolvido com o objetivo principal de definir um método objetivo, para a escolha do tipo de processo de desenvolvimento mais adequado a um determinado projeto. Ao longo do trabalho descreve-se como o método foi criado. No final, do trabalho a aplicação prática deste método foi demonstrada.

O uso do processo de software adequado a cada contexto de projeto é fundamental para se atingir sucesso. Por essa razão, o método obtido pode ser considerado uma ferramenta muito importante para as organizações atuais que procuram flexibilidade, mas sem perder o controle no desenvolvimento de um projeto de software.

Este trabalho também analisou a aderência de dois processos amplamente conhecidos de desenvolvimento de software, XP e RUP, aos fatores utilizados, permitindo a aplicação do método de seleção, sem adaptações, na escolha entre estes dois processos de desenvolvimento.

Outros processos de desenvolvimento que tenham a mesma aderência aos fatores descritos neste trabalho, também podem ser considerados como resultado final do método de seleção. Por exemplo, outros processos ágeis, ao invés de XP, podem ser considerados quando o método aponta para o uso preferencial desse tipo de processo, desde que a análise de sua aderência aos fatores seja realizada. O mesmo vale para a utilização de outros processos tradicionais, ao invés do RUP, quando o método aponta para o uso desse tipo de processo.

8.3 Contribuições deste trabalho

Ter um processo de desenvolvimento é essencial para o bom andamento e resultado em um projeto de software. Existem muitos disponíveis e uma contribuição importante deste trabalho foi apresentar, com ênfase em XP e RUP, dois tipos de processos de desenvolvimento muito adequados à realidade atual na engenharia de software.

Encontrar o tipo de processo mais adequado nem sempre é uma tarefa simples, portanto ter uma ferramenta de seleção ajuda na escolha, contribuindo com o sucesso do desenvolvimento do software. A principal contribuição deste trabalho foi ter criado um método objetivo de seleção, entre processos de desenvolvimento amplamente conhecidos e utilizados.

Também contribui com a discussão a respeito da necessidade de existir diversos processos de desenvolvimento de software, sendo cada um deles mais adequado a cada contexto específico. Foram discutidos aqui os processos com foco em planejamento e os processos ágeis, que possuem conceitos distintos para tratar o desenvolvimento de software, sendo cada um deles mais adequado em cada contexto distinto.

8.4 Sugestões de Trabalhos Futuros

A partir deste trabalho existem algumas linhas de pesquisa que podem dar continuidade a ele. Entre as possíveis extensões merecem ser destacadas as seguintes possibilidades:

- Analisar e determinar quais processos podem ser considerados resultados válidos da aplicação do método de seleção, além do XP e RUP. Analisar se processos tais como *SCRUM*, *Crystal* ou processos como *Cleanroom* são também aderentes aos fatores deste método de seleção deste trabalho.
- Realizar um estudo de caso real do desenvolvimento de um projeto após a aplicação do método. Ou seja, após a definição do processo mais adequado, o estudo de caso deveria, em paralelo, realizar um projeto de desenvolvimento usando duas equipes distintas, uma com RUP e outra com XP. No fim, analisar o resultado de cada projeto e validar se, realmente o processo inicialmente apontado pelo método de seleção, foi considerado como o mais adequado na prática.
- Chegar a uma fórmula no passo 2 de qual a área mínima adequada aos processos tradicionais e qual a área máxima adequada aos processos ágeis.

Referências bibliográficas

BASKERVILLE, R. et al. Is Internet-Speed Software Development Different?. IEEE Software. IEEE Computer Society Press, v.20, n.6, p 70-77, nov/dez 2003.

BECK, K. et al. Manifesto for Agile Software Development. 2001. Disponível em: <http://agilemanifesto.org/>. Acesso em 29 de Abril de 2007.

BECK, K. Extreme Programming Explained: Embrace Change - 2nd Edition. Massachusetts: Addison-Wesley, 2004. ISBN 0-321-27865-8

BOEHM, B. Get Ready for Agile Methods, with Care. IEEE Computer. IEEE Computer Society Press, v.35, n.1, p.64-69, jan 2002. ISSN: 0018-9162.

BOEHM, B; TURNER, R. Using risk to balance agile and plan-driven methods. IEEE Computer. IEEE Computer Society Press, v. 36, n. 6, p. 57-66, jun 2003. ISSN: 0018-9162

COCKBURN, A. Selecting a Project's Methodology. IEEE Software. IEEE Computer Society Press,v.17,n.4,p.64-71, jul/ago 2000. ISSN: 0740-7459.

COCKBURN, A. Agile Software Development. Boston: Addison-Wesley, 2001. ISBN: 0-201-69969-9

COCKBURN, A. Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley, 2004. ISBN: 0-201-69947-8.

FOWLER, M. The new methodology. 2005. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em 29 de Abril de 2007.

HARRIS, M; HEVNER, A.R; COLLINS, R.W. Controls in Flexible Software Development. Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06) Track 9, pp 216a, janeiro 2006.

HOUAISS, A. Dicionário Houaiss da Língua Portuguesa. 2007. Disponível em: <http://houaiss.uol.com.br/busca.jhtm?verbete=habitat&stype=k>. Acesso em 06 de junho de 2008.

KIM, S; CHOI, H. An evaluation of process performance for a small-team project – A case study. ICIS, p.308-313, Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05), 2005.

KROLL, P. KRUCHTEN, P. The Rational Unified Process Made Easy: A Practitioner's Guide to Rational Unified Process. Addison-Wesley Professional -1st edition, 2003. ISBN: 0-321-16609-4.

KRUCHTEN, P. The Rational Unified Process: An Introduction. Addison-Wesley Professional – 3rd Edition, 2003. ISBN: 0-321-19770-4.

LINDVALL, M; RUS, I. Process Diversity in Software Development. IEEE Software. IEEE Computer Society Press, v17, n.4, p.14-18, jul/ago 2000. ISSN: 0740-7459

LINGER, R.C. Cleanroom Process Model. IEEE Software. IEEE Computer Society Press, v.11, n.2, p.50-58, mar/abr 1994. ISSN: 0740-7459.

LITTLE, T. Context-adaptive agility: managing complexity and uncertainty. IEEE Software. IEEE Computer Society Press v.22, n.3, p.28-35, mai/jun 2005. ISSN: 0740-7459.

MURRU,O. et al. Assessing XP at a European Internet Company. IEEE Software. IEEE Computer Society Press v.20, n.6, p.37-43, mai/jun 2003. ISSN: 0740-7459.

PLOWMAN, L. The Interfunctionality of Talk and Text. Computer Supported Cooperative Work v.3, n.4, p.229-246, 1995.

PMI. A Guide To The Project Management Body Of Knowledge (PMBOK Guides). Project Management Institute 3rd edition, 2004.

PRESSMAN, R. Software Engineering – A practitioner's approach – 5th Ed. New York: McGraw Hill, 2001.

SALO, O. Improving Software Process in Agile Software Development Projects: Results from Two XP Case Studies. 30th EUROMICRO Conference (EUROMICRO'04), p. 310-317, 2004.

SCHINDLER, E. Getting Clueful: 7 Things CIOs Should Know About Agile Development CIO Magazine. Fev, 2008. disponível em:

http://www.cio.com/article/180402/Getting_Clueful_Things_CIOs_Should_Know_About_Agile_Development. Acesso em 16 de março de 2008.

UDO, N. KOPPENSTEINER, S. WILL AGILE DEVELOPMENT CHANGE THE WAY WE MANAGE SOFTWARE PROJECTS? AGILE FROM A PMBOK® GUIDE PERSPECTIVE. Disponível em: http://www.projectway.com/paper_agilevspmbok.pdf, 2003. Acesso em 29 de abril de 2007.

WAILGUM, T. How Agile Development Can Lead to Better Results and Technology-Business Alignment. disponível em: <http://www.cio.com/article/109751>, mai, 2007. acesso em 17 de março de 2008.

Referências Complementares

ABRAHAMSSON, P et all. New Directions on agile methods: a comparative analysis. Proceedings of the 25th International Conference on Software Engineering (ICSE'03), pp 244-254, Portland, Oregon, 2003.

AMBLER, S. Lessons in Agility From Internet-Based Development. IEEE Software. IEEE Computer Society Press v.19, n.2, p. 66-73, mar/abr 2002a. ISSN: 0740-7459.

AMBLER, S. Agile Modeling. John Wiley & Sons, New York, 2002b. ISBN: 0-471-20282-7.

CUSUMANO, M.A. YOFFIE, D.B. What Netscape Learned from Cross-Platform Software Development. Communications of the ACM, ACM Press, v. 42, n.10, p.72-78, out. 1999.

CUSUMANO, M.A. YOFFIE, D.B. Software Development on Internet Time. IEEE Computer, IEEE Computer Society Press, v.32, n.10, p.60-69, out 1999.

DINGSOYR, T. HANSSEN, G. K. Extending Agile Methods: Postmortem Reviews as Extended Feedback. 4th International Workshop on Learning Software Organizations (LSO'02), Chicago, Illinois, USA, 2002.

ELSSAMADISY, A. SCHALLIOL, G. Recognizing and responding to "bad smells" in extreme programming. ACM Press, p.617-622, Orlando, Florida, 2002. ISBN:1-58113-472-X.

HUMPHREY, W.S. TSP: Leading a Development Team. Addison-Wesley, September 2005. ISBN: 0-321-34962-8.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 12207: Software Life Cycle Process. Geneva, Switzerland, 2000.

LARMAN, C. Agile & Iterative Development : A manager´s guide. USA:Addison-Wesley, 2003. ISBN: 0-131-11155-8.

IANSITI, M. MACCORMACK, A. Developing Products on Internet Time. Harvard Business Review, v.75, n. 5, p.108-117, set/out 1997.

NERUR, S. MAHAPATRA, R. MANGALARAJ, G. Challenges of migrating to agile methodologies. Communications of the ACM, ACM Press, v.48, n.5, p.72-78, mai 2005.

SOMMERVILLE, I. Software engineering, 7th ed. New York: Addison-Wesley, 2004. ISBN: 0-321-21026-3.

JOHNSON, Jim et al. ChAOS: A recipe for success. 1998. Published Report. The Standish Group.

JOHNSON, Jim et al. ChAOS in the new Millenium. Published Report. The Standish Group.

Glossário

Habitat	Conjunto de circunstâncias físicas e geográficas que oferece condições favoráveis à vida e ao desenvolvimento de determinada espécie. No âmbito deste trabalho esta definição é aplicada para determinar um ambiente que permite condições favoráveis a cada tipo de processo de desenvolvimento (HOUAISS, 2007).
Método	Define os roteiros técnicos de como executar cada atividade para a construção de software (PRESSMAN, 2001).
Processo	É uma estrutura composta dos grupos das tarefas necessárias para a produção de software de alta qualidade. Esta estrutura define as recomendações que devem ser seguidas na produção do software (PRESSMAN, 2001).