UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DOS EFEITOS DO *TURNOVER* NA PRODUTIVIDADE DE PROCESSOS DE *SOFTWARE* TRADICIONAIS E HÍBRIDOS

WILLIAM CHAVES DE SOUZA CARVALHO

Uberlândia - Minas Gerais
2012

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

WILLIAM CHAVES DE SOUZA CARVALHO

ANÁLISE DOS EFEITOS DO *TURNOVER* NA PRODUTIVIDADE DE PROCESSOS DE *SOFTWARE* TRADICIONAIS E HÍBRIDOS

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de *Software*.

Orientador:
Prof. Dr. Pedro Frosi Rosa

Co-orientador:
Prof. Dr. Michel dos Santos Soares

Uberlândia, Minas Gerais
2012

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Análise dos efeitos do *turnover* na produtividade de processos de *software* tradicionais e híbridos**" por **William Chaves de Souza Carvalho** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 07 de Maio de 2012

Orientador: _____
Prof. Dr. Pedro Frosi Rosa
Universidade Federal de Uberlândia

Co-orientador: _____
Prof. Dr. Michel dos Santos Soares
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Edgard Lamounier Júnior
Universidade Federal de Uberlândia

_____
Prof. Dr. Selma Shin Shimizu Melnikoff
Universidade de São Paulo

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Maio de 2012

Autor:	**William Chaves de Souza Carvalho**
Título:	**Análise dos efeitos do *turnover* na produtividade de processos de *software* tradicionais e híbridos**
Faculdade:	**Faculdade de Computação**
Grau:	**Mestrado**

_____

Autor

Dedicatória

Dedico este primeiramente aos meus pais, e sobretudo à minha mãe, que infelizmente nos deixou mais cedo do que seria razoável tolerar. E também é para o João Pedro, que um dia compreenderá o porquê de ter assistido, ao meu lado, a tantos desenhos enquanto o seu papai escrevia.

Agradecimentos

Agradeço especialmente ao professor e *yadid* Pedro Frosi pelo suporte às tantas idas e vindas. Não fosse pela sua obstinada paciência, generosidade e palavras certas nos momentos adequados, estou certo de que desfecho teria sido diferente deste. O professor Michel, por sua vez, foi um grande incentivador e também um amigo a quem agradeço pelo trabalho meticuloso de polimento das palavras deste texto para que ele se tornasse publicável. Também sou profundamente grato aos bons amigos que me acompanham e incentivam e às inúmeras pessoas que contribuíram, direta ou indiretamente, para a finalização deste trabalho. É provável que algumas delas não se dêem conta de como me ajudaram, seja com palavras, idéias ou apoio, mas de qualquer forma, cada qual à sua maneira, tem minha gratidão. No entanto, nada disso teria sido possível sem o amparo do amor incondicional da Ana Beatriz, que mais do que ninguém, sabe o quanto esta etapa é importante. Seria tão temerário quanto insuficiente tentar verbalizar como sou grato pelo seu apoio, carinho e resignação frente a tantas ausências e impaciências, no entanto, gostaria de registrar que você é a melhor companheira de viagem que eu poderia ter e que eu te amo muito.

It matters not how strait the gate,
How charged with punishments the scroll,
I am the master of my fate:
I am the captain of my soul.

William Ernest Henley (Invictus)

Resumo

Este trabalho apresenta um estudo de caso experimental conduzido para avaliar os efeitos do *turnover* na produtividade de projetos de desenvolvimento de *software*. Para isso foram coletados e analisados dados de 27 projetos reais, desenvolvidos de acordo com um processo baseado no método RUP ou com base em um modelo híbrido que combina práticas do RUP e Scrum. Foi utilizado o modelo de Stutzke para calcular a perda de produtividade dos projetos devido ao *turnover*, levando em consideração parâmetros de duração do tempo de contratação, assimilação e *mentoring*. A análise dos dados foi feita com base em um modelo de medição e em indicadores de desempenho de produto, pessoas, tecnologia, risco e qualidade. Os resultados obtidos indicam que os projetos RUP perdem, em média, 5,25% de produtividade devido ao *turnover*, enquanto que os projetos híbridos perdem 11,86%. Estas informações corroboram com a proposição da pesquisa de que os projeto RUP são menos sujeitos que os projetos híbridos aos efeitos do *turnover* na produtividade. Sugere-se, ao final, que trabalhos futuros criem extensões do modelo de Stutzke que permitam prever e contingenciar, ainda nas fases iniciais do projeto, as perdas de lucratividade dos projetos, mediante o uso de ferramentas e heurísticas apropriadas.

Palavras chave: produtividade, processos híbridos, turnover.

Abstract

This dissertation presents an experimental case study conducted to aim to evaluate the effects of turnover on the software development projects productivity. To do so, they were collected and analyzed data from 27 software projects, developed in accordance with a process based on RUP or in a hybrid process that combines RUP and Scrum practices. The Stutzke's mathematical model was used to calculate the lost productivity due to the turnover of projects, taking into account duration of hirement, assimilation and mentoring. Data analysis was did based on a measurement model and performance indicators of product, people, technology, risk and quality. The results indicate that projects based on RUP lost an average of 5.25% of productivity due to the turnover while the hybrid process lose 11.86%. These results corroborates the research proposition that the project RUP are less likely than the hybrid projects to the effects of turnover on productivity. As suggestions to future researches the author states that the creation of Stutzke model extensions for predicting turnover in the early stages of the project, in order to help organizations to avoid projects loss of profitability.

Keywords: productivity, hibrid process, turnover.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BEP	Break-even Point
BFPUG	Brazilian Function Point Users Group
CFPS	Certified Function Point Specialist
CM	Configuration Management
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CMU	Carnegie Mellon University
COCOMO	COnstructive COst MOdel
COTS	Commercial Of The Shelf
DET	Data Element Type
DoD	Department of Defense
EI	External Input
EIA	Electronic Industries Alliance
ELF	External Logical File
EO	External Output
EPG	Engineering Process Group
EQ	External Query
FA	Fator de Ajuste
FFP	Full Function Point
FFP	Full Function Point
FLP	Fractional Loss of Productivity
FP	Feature Points
FTE	Full Time Equivalent
FTR	File Types Referenced
GG	Generic Goal
GP	Generic Practice
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFPUG	International Function Points Users Group
ILF	Internal Logical File
IPD-CMM	Integrated Product Development Capability Maturity Model
ISO	International Organization for Standardization
MA	Measurement and Analysis

PA	Process Area
P-CMM	People Capability Maturity Model
PMC	Project Monitoring and Control
PP	Project Planning
PPQA	Process and Product Quality Assurance
PSP	Personal Software Process
REQM	Requirements Management
RET	Record Element Types
RUP	Rational Unified Process
SECM	Systems Engineering Capability Model
SEI	Software Engineering Institute
SG	Specific Goal
SLOC	Source Lines of Code
SP	Specific Practice
SW-CMM	Capability Maturity Model for Software
TSP	Team Software Process
UCP	Use Case Points
USDP	Unified Software Development Process
WBS	Work breakdown structure
XP	eXtreme Programming

Capítulo 1

Introdução

Na história bíblica da Torre de Babel, Deus causou o fracasso do projeto de construção da torre interrompendo a comunicação entre os operários ao criar diversas linguagens. Na história moderna da Tecnologia da Informação, também foram criadas diversas linguagens cada qual pretendendo atender nichos específicos de necessidades organizacionais. Assim como na história bíblica, a diversidade de linguagens, modelos e *frameworks* tem causado confusão e desentendimento e cada qual, a seu modo, se propõe a reverter esta situação ao assegurar o advento de uma visão integradora das disciplinas do ciclo de vida do *software*.

Os primeiros modelos de processo de desenvolvimento propuseram que a construção de sistemas de *software* acontecesse de acordo com a aplicação sistemática de princípios de Engenharia de *Software*, incluindo as áreas de Qualidade e Engenharia de Sistemas [Becker e Whittaker 1997, Prowell et al. 1999]. Os métodos utilizados nestes processos davam ênfase às atividades das fases iniciais do projeto: entendimento de requisitos, planejamento e elaboração de arquiteturas bem estabilizadas, fartamente documentados em contratos e especificações [Boehm e Turner 2003a, Clements et al. 2003]. Alguns exemplos de processos de software baseados no paradigma tradicional são Cleanroom [Becker e Whittaker 1997, Prowell et al. 1999], PSP [Humphrey 1997], TSP [Humphrey 2000] e RUP [Jacobson et al. 1999, Kruchten 2003].

O ambiente de negócios também evoluiu e se diversificou, exigindo maiores esforços de desenvolvimento de *software* para atender as necessidades de mercado. Além de exigir que fronteiras organizacionais e geográficas sejam ultrapassadas, os gestores são compelidos a orquestrar competências dos mais variados perfis, adequando-se a cronogramas restritos para atender expectativas agressivas dos clientes [Humphrey 1989]. As inovações envolvem desde sistemas distribuídos e equipes transdisciplinares [Humphrey 2010], até desenvolvimento concorrente, integração com COTS, aplicações *open source*, utilização de processos ágeis, aderência a padrões e modelos internacionais de qualidade.

Processo de *software* é o conjunto de atividades inter-relacionadas que permite identificar necessidades, modelar soluções técnicas, codificar e implantar sistemas automatizados que atendam as expectativas dos usuários [Nord e Tomayko 2006, Leffingwell 2011]. Os modelos de processo desempenham papel primordial na materilização das expectativas porque são eles que dispõem as ferramentas essenciais para aumentar a taxa de produtividade, diminuir o tempo de desenvolvimento e reduzir o custo dos serviços prestados [Nord e Tomayko 2006].

O aumento da complexidade – técnica e organizacional –, estabelecem terrenos fecundos para

a proliferação de modelos e *frameworks* de processos burocráticos e, por vezes, incompatíveis com o *time to market* exigido das unidades organizacionais. A cultura desenvolvida neste ambiente fértil em novidades, modas e tendências contribuiu, em parte, para a indústria de *software* se deparar com a diminuição da capacidade de entregar rapidamente produtos de alto valor agregado de acordo com rigorosos padrões de qualidade [Leffingwell 2011].

Ao longo das últimas décadas o desenvolvimento de *software* se tornou uma das atividades humanas mais importantes e desafiadoras. O código-fonte escrito atualmente corresponde a parcelas significativas da propriedade intelectual registrada no mundo [Humphrey 2010]. Em outras palavras, o mundo moderno depende de *software* [Leffingwell 2011]. Desde o início da década de 2000, os processos de desenvolvimento de *software* podem ser categorizados de acordo a aplicação, filosofia e princípios que os fundamenta [Boehm e Turner 2003a]:

- Processos de *software* baseados no paradigma tradicional são mais recomendados para direcionar o desenvolvimento de projetos grandes e complexos, no entanto, processos ágeis são mais adequados para projetos pequenos e de baixo risco [Boehm e Turner 2003a, Boehm e Turner 2003b, Cohen et al. 2004, Nord e Tomayko 2006, Ramsin e Paige 2008].
- Em contrapartida, processos ágeis tem se mostrado adequados quando o objetivo é aumentar a sinergia da equipe [Cockburn 2000, Layman et al. 2004, Williams e Cockburn 2003, Ilieva et al. 2004] ou melhorar a capacidade de resposta às solicitações de mudança feitas ao longo do desenvolvimento da aplicação.

Como os processos do paradigma tradicional não atendiam satisfatoriamente às necessidades dos diversos atores atuantes no mercado tecnologia da informação – em franca expansão –, ainda na década de 1990 pesquisadores, órgãos governamentais e representantes da indústria de *software* se dedicaram à busca de novas abordagens [Jones 2007].

Em 2001, com a publicação do Manifesto Ágil, atinge-se o clímax das atividades de um movimento que buscava alternativas ao paradigma tradicional. Nele, seus signatários recomendavam desenvolvimento colaborativo, reuniões de evolução de requisitos baseadas em versões funcionais do *software*, resposta rápida a requisições de mudança e crescimento orgânico da aplicação a partir de iterações de prazo determinado [Beck et al. 2001]. Entretanto, seja qual for o processo adotado para desenvolver um *software*, a equipe alocada para fazê-lo pode ter sua composição alterada ao longo do ciclo de vida do projeto.

A rotatividade de profissionais, ou *turnover*, é um elemento fundamental a ser considerado ao planejar e controlar o projeto, pois a substituição de membros da equipe gera efeitos colaterais na produtividade porque a senioridade dos profissionais não é homogênea [Stutzke 1995]. Estas variações interferem nos objetivos quantitativos estabelecidos para qualidade e para desempenho do projetos, uma vez que eles se baseiam nas necessidades dos clientes, dos usuários finais, da organização e dos responsáveis pela implementação do projeto. A variação da produtividade, por exemplo, pode ser determinante para a violação de acordos de nível de serviço ou desempenho de processo [Paulk e Chrissis 2002].

1.1 Contextualização

Ao longo do tempo, os sistemas tornaram-se maiores e com a missão de integrar e conciliar os objetivos estratégicos de diversos segmentos de mercado, o quê, consequentemente, contribuiu para o aumento de sua complexidade [Humphrey 2010]. Isso mudou a engenharia de *software* em alguns aspectos significativos desde que os primeiros modelos de melhoria de processo surgiram. Seja qual for sua finalidade ou complexidade, produtos de *software* não são palpáveis, e sim, a materialização de pensamentos e abstrações intangíveis em forma de código binário, desenvolvido de acordo com um processo. O reconhecimento da relevância do papel dos modelos de processo aliado ao aumento exponencial da demanda por sistemas baseados em *software* nas últimas cinco décadas impeliu pesquisadores e profissionais da indústria a criar vários *frameworks* de processos para estruturar, gerenciar, padronizar e controlar as atividades do ciclo de vida de desenvolvimento de *software* [Humphrey 1989, Humphrey 2010].

1.2 Problematização

Projetos grandes, complexos ou críticos são mais facilmente prejudicados pela falta de rigor e previsibilidade dos métodos ágeis, ao passo que projetos pequenos e de baixo risco podem ser prejudicados pela falta de simplicidade e flexibilidade do paradigma tradicional [Boehm e Turner 2003a, Boehm e Turner 2003b, Boehm e Turner 2004, Alves 2011]. Todavia, independentemente do paradigma adotado, a maioria dos projetos que dura mais do que poucas semanas enfrenta *turnover*: a perda ou realocação de profissionais capacitados antes da finalização do trabalho. Apesar de este fenômeno ter potencial para afetar todos os tipos projetos, ele afeta, sobretudo, aqueles de grande volume de esforço e longa duração.

Projetos de desenvolvimento de *software* são especialmente sensíveis aos efeitos do *turnover* porque construção de *software* é um trabalho intensivamente baseado em conhecimento [Stutzke 1995]. Programadores, analistas e engenheiros de *software* precisam desenvolver modelos mentais complexos do sistema, entender o ambiente e os processos do projeto. Mesmo os profissionais mais capacitados e experientes precisam de tempo para assimilar a arquitetura e o *design* do produto, entender o funcionamento da infraestrutura de *hardware*, configurar os *frameworks* e preparar os ambientes de desenvolvimento [Stutzke 1995].

Apesar da relevância e recorrência do tema, pouco se sabe sobre os benefícios reais da utilização de processos baseados num ou noutro paradigma. Os estudos são preliminares e as evidências esparsas, sobretudo no que se refere a pesquisas experimentais sobre a variação da taxa de produtividade. Poucos investigadores se debruçaram sobre os impactos do *turnover* na produtividade, apesar deles serem potencialmente expressivos [Boehm e Turner 2003a, Stutzke 1995, Abdel-Hamid e Madnick 1991].

A pesquisa de Alves [2011] investigou diversas opções de integração ágil e tradicional e definiu uma abordagem híbrida em que o rigor tradicional foi mantido em subprocessos críticos e permitiu obter ganhos reais de produtividade graças a adoção de práticas ágeis. O *framework* resultante da integração de práticas do Scrum com processos do RUP resultou no modelo híbrido utilizado pela empresa cujos projetos foram analisados neste trabalho. A abordagem metodológica utilizada por Alves [2011] no seu estudo comparativo multi-projeto forneceu subsídios para comparar os

efeitos do *turnover* na produtividade de projetos tradicionais e projetos híbridos.

As perdas e atrasos decorrentes do *turnover* podem consumir uma fração significativa do lucro esperado de um projeto [Stutzke 1995] e, apesar dos esforços realizados para mapear as implicações do uso de métodos tradicionais, ágeis e híbridos, os estudos são preliminares e o campo de investigação ainda é pouco explorado [Ilieva et al. 2004, Dybå e Dingsøyr 2008, Dybå e Dingsoyr 2009].

Por este motivo, existe especial interesse em entender e medir objetivamente os efeitos da rotatitividade de profissionais ao longo do projeto. Em outras palavras, o que se pretende medir e comparar é a fração perdida de trabalho – decorrente de *turnover* – por equipes de desenvolvimento que trabalham guiadas por processos tradicionais e processos híbridos dos membros da equipe. Intuitivamente, parece que a dinâmica reativa dos métodos ágeis contribua com a erosão da taxa de produtividade e, consequentemente, com as margens de lucro dos projetos. Assim, esta pesquisa objetiva responder a seguinte questão:

> Quais são oa efeitos do *turnover* na produtividade de projetos desenvolvidos com processos híbridos, utilizando como parâmetro de comparação um processo baseado no RUP?

Naturalmente, seria desejável responder a questão sob dois pontos de vista: quantitativo e causal. No entanto, a pesquisa se restringe a mensurar os efeitos do *turnover* na produtividade utilizando o modelo de Stutzke. A proposição[1] deste estudo é que a perda de produtividade de projetos que adotaram o processo híbrido seja maior do que as encontradas em projetos RUP.

Segundo Stutzke e Abdel-Hamid, uma taxa de *turnover* de 10% diminuiu o lucro e a produtividade média de um projeto em cerca de 3% [Abdel-Hamid e Madnick 1991, Stutzke 1995]. Considera-se que estes valores se aproximem aos dos efeitos do *turnover* em processos RUP, e dessa forma espera-se encontrar perdas maiores de produtividade em projetos baseados em métodos híbridos.

1.3 Metodologia

A metodologia foi estabelecida em função das limitações impostas pela natureza do contexto, como falta de controle sobre variáveis, aspectos éticos, custo, tempo e acesso a informações.

O primeiro passo do estabelecimento da metodologia e diretrizes da pesquisa foi a definição do contexto de sua realização. Foram utilizados dados de 27 projetos reais desenvolvidos por uma fábrica de *software* ao longo de três anos. É importante descrever o contexto de realização da pesquisa para permitir que diferentes estudos realizados em condições similares possam ser comparados [Petersen e Wohlin 2009b].

O objetivo da pesquisa é comparar os efeitos do *turnover* no processo RUP com aqueles verificados no processo híbrido. A concecução do objetivo da pesquisa será feita mediante a realização de um estudo de comparativo, em que as unidades de análise serão agrupadas em dois

[1] Uma proposição é similar a uma hipótese e especifica o que será esperado como resultado do estudo. Ela guia o pesquisador para a direção na qual procurar por evidências de modo a responder a questão do um estudo de caso [Yin 2010].

grupos: projetos RUP, identificados pelos acrônimos R_1 a R_n e projetos híbridos, identificados por H_1 a H_m.

A abordagem para a pesquisa foi a do estudo de caso, porque é experimental e visa investigar um fenômeno contemporâneo em seu contexto [Yin 2010]. Estudos de caso são particularmente importantes para avaliação industrial de métodos e ferramentas de Engenharia de *Software* [Kitchenham et al. 1995a]. De modo geral, a abordagem de estudo de caso é bem apropriada para muitos tipos de pesquisa em Engenharia de *Software*, pois os objetos de estudo são fenômenos contemporâneos, que são difíceis de controlar experimentalmente.

Não existem *frameworks* padrão para descrever o contexto de uma pesquisa que envolva processos híbridos – embora sua necessidade já tenha sido reconhecida [Petersen e Wohlin 2009a]. Para suprir esta falta são utilizados os critérios propostos por [Petersen e Wohlin 2009b] e descritos a seguir.

1.3.1 Empresa

O estudo foi realizado em uma empresa brasileira de Tecnologia da Informação (TI) cujas atividades são focadas em desenvolvimento de projetos de *software* customizados, consultoria e *outsourcing* de serviços de TI. As práticas e processos da empresa são aderentes ao Nível de Maturidade 2 do modelo CMMI® o que indica que a empresa estimula a criação e manutenção de uma cultura de desenvolvimento de *software* orientada a processos gerenciados. Para isso, utiliza políticas e práticas organizacionais de Gerenciamento de Projetos. Ela foi selecionada por atender às demandas básicas do estudo, tais como maturidade de processo, capacitação de colaboradores e demanda de projetos e pelo acesso aos dados necessários. Os dados de cada projeto correspondem a uma unidade de análise.

1.3.2 Mercado

Os projetos atendem clientes que fazem uso intensivo de sistemas baseados em *software* para viabilizar seus negócios e exigem negociação e planejamento prescritivos para acordo contratual formal e antecipado. Seus clientes preferem fixar o escopo contratualmente, estimando custo e prazo. Todos os projetos incluídos neste estudo seguiram este modelo contratual. A empresa é experiente em gerenciar requisitos por meio de casos de uso, criação de arquiteturas estáveis nas fases iniciais do ciclo de vida e realização de estimativa baseada na medição de funcionalidades mediante análise de Pontos de Função.

1.3.3 Processo

Algumas considerações acerca do processo utilizado pela empresa são apresentadas a seguir:

- Modelo de ciclo de vida: o modelo de ciclo de vida é fortemente baseado no RUP e foi alterado em alguns pontos para incorporar práticas do Scrum, conforme descrito na seção 2.7. As fases iniciais do ciclo de vida continuaram a ser dirigidas por plano e baseadas na produção de documentação. A fase de construção foi estruturada em termos de sprints de duas semanas de duração para conferir agilidade e entregas rápidas para o cliente.

- Artefatos: produz os mesmos artefatos prescritos pelo RUP com graus similares de detalhe. A exceção é a Realização de Casos de Uso, que é consideravelmente menos detalhada em função da redução da documentação preconizada pelos métodos ágeis. Apenas os casos de uso de alta complexidade são detalhados extensivamente.
- Papéis: considera os mesmos papéis do RUP, com acréscimo do *Product Owner* e *Scrum-Master*. Convém ressaltar que o papel de Gerente de Projeto e *ScrumMaster* por vezes se confundem, porém, o processo identifica estes dois atores separadamente.
- Características: de acordo com o processo híbrido, a comunicação e colaboração entre a equipe é mais frequente, uma vez que os *sprints* são menores. No entanto, o subprocesso Gestão de *Backlog* possui atividades equivalentes no RUP, destinadas à gestão de tarefas.

1.3.4 Coleta de Dados

Este estudo fez uso de diversas fontes de dados quantitativos, o que permite a utilização da técnica de triangulação [Seaman 1999, Yin 2010], para sustentar as proposições, visando analisar a convergência das fontes.

Apesar de dados qualitativos poderem ser utilizados para enriquecer resultados quantitativos com informações explanatórias, neste trabalho eles não foram utilizados porque o escopo da pesquisa não abarca questões complexas envolvendo comportamento humano [Layman et al. 2004]. As fontes de evidências utilizadas neste estudo foram medidas de processo e produto.

A coleta dos dados quantitativos foi feita de maneira semi-automatizada por meio de um sistema informatizado que consolida as informações do modelo de medição. Este sistema é acessível aos desenvolvedores do projeto, de modo que eles mesmos fazem o lançamento dos dados. Assim, por exemplo, um desenvolvedor, ao concluir uma determinada tarefa, registra tanto o instante em que a finalizara quanto as sub-atividades necessárias para sua execução. Algumas informações necessárias ao desenvolvimento desta pesquisa foram obtidas por meio da aplicação de operações lógicas e/ou matemáticas nos dados puros armazenados no banco de dados da empresa.

Exemplos de informações derivadas incluem tempo de *mentoring*, tempo médio de assimilação, tempo médio de contratação e *turnover*. Todas as operações realizadas nos dados seguiram rigorosamente os requisitos do modelo de Stutzke para não comprometer e acurácia das informações. Da forma semelhante, os dados de requisitos de produto são informados pelo profissional responsável pelo seu desenvolvimento e validado pelo coordenador.

1.3.5 Análise de Dados

A análise das evidências de um estudo de caso é um dos aspectos menos desenvolvidos e mais complicados ao realizar estudos de caso [Yin 2010]. A análise de dados deve ser previamente planejada e explicitada no trabalho [Miguel 2007], ou seja, deve ser especificada como parte do protocolo de estudo de caso [Yin 2010]. Yin recomeda algumas estratégias gerais e técnicas analíticas específicas para análise de dados em estudos de caso [Yin 2010]. A estratégia geral escolhida foi a de se basear em proposições teóricas, que determina que a análise de dados seja orientada pelas proposições do estudo de caso, relacionadas à questão de pesquisa.

A técnica analítica de adequação ao padrão foi utilizada neste estudo, a qual consiste em comparar um padrão fundamentalmente experimental com outro de base prognóstica, baseado nas proposições, de modo que, se os padrões coincidirem, os resultados podem ajudar o estudo de caso a reforçar sua validade interna [Alves 2011]. Com base na estratégia geral e técnica analítica escolhida, foi elaborado um procedimento para análise de dados, baseado em [Alves 2011], o qual é ilustrado pela figura 1.1.

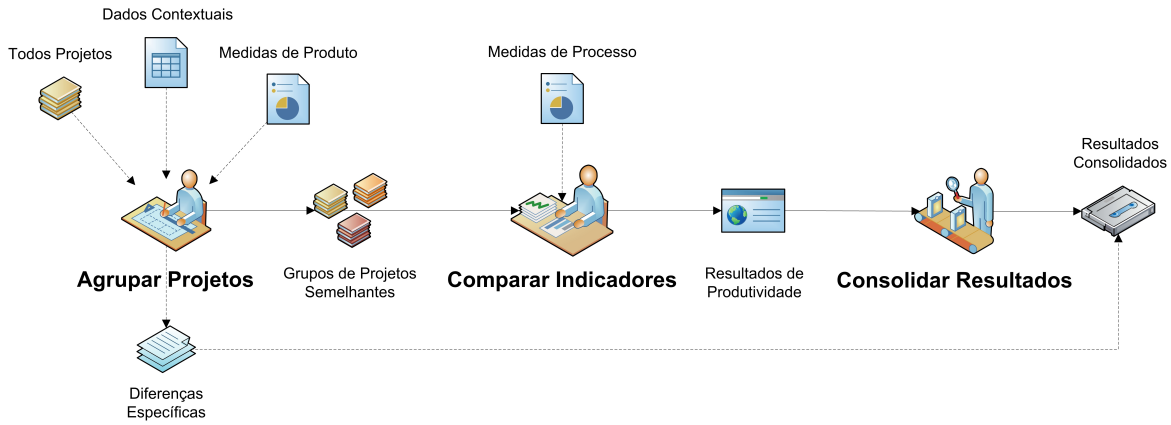

Figura 1.1: Procedimento para análise dos dados [Alves 2011]

As medidas de processo tem o intuito de medir a produtividade para auxiliar a investigar a proposição da pesquisa. Por outro lado, as medidas de produto visam auxiliar a comparação dos projetos e acareação dos resultados obtidos. As macro instruções do algoritmo de análise de dados são detalhadas a seguir.

Agrupar projetos

O agrupamento dos projetos será feito separando-os em conjuntos que possuam caracterísitcas semelhantes com relação aos fatores que influenciam na produtividade. As agregações serão feitas visando isolar estes fatores e selecionar projetos semelhantes para comparação. Entre os projetos considerados, poderá haver diferenças que não sejam significativas a ponto de separá-los em agrupamentos específicos. O uso de ferramentas estatísticas para encontrar diferenças significativas e apoiar o agrupamento dos projetos auxilia na remoção da subjetividade deste processo.

Comparar indicadores

Os indicadores dos projetos RUP e híbridos – derivados do modelo de medição – serão comparados quantitativamente para cada agrupamento de projetos.

Consolidar resultados

De posse dos valores de produtividade, *turnover*, percentual de perda de produtividade e das diferenças detectadas entre os projetos, estes dados serão comparados. Este passo é crucial para identificar a influência dos fatores do *turnover* na produtividade dos projetos, de acordo com as evidências coletadas.

1.4 Ameaças à Validade

Estudos com componentes experimentais possuem ameaças à validade. Entretanto, o sucesso de um estudo experimental pode ser mais bem assegurado se essas ameaças forem identificadas antecipadamente de modo que ações para mitigá-las possam ser realizadas. Há quatro tipos de ameaças a validade em estudos de caso [Yin 2010]: validade do modelo, validade interna, validade externa e validade da conclusão.

1.4.1 Validade do Modelo

Validade do modelo está relacionada com a obtenção das medidas corretas dos conceitos estudados. Uma das ameaças à validade do modelo é em relação às medidas dos fatores moderadores do efeito na produtividade. Com relação à complexidade do produto, embora tenham sido usadas informações contextuais e medidas quantitativas, outras características técnicas não mencionadas podem influenciar na complexidade do produto, além do que a complexidade é relativa aos níveis de capacitação e experiência dos desenvolvedores [Arisholm et al. 2007].

Outra ameaça é com relação aos indicadores usados para medir as pessoas, que podem ser questionados com relação à sua capacidade de identificar corretamente características específicas relevantes quanto à capacitação e experiência da equipe [Wagner e Ruhe 2008]. Por fim, a medida de tamanho do software (que também é o fundamento para se medir a produtividade) utilizada foi baseada somente em pontos de função ajustados, que mede o tamanho do software em termos de funcionalidade e o adequa em função de suas características não funcionais. Além disso, pode-se considerar que a medida do tamanho de um software pode ser melhor obtida por meio da combinação de diversas medidas [Kitchenham et al. 1995b, Kitchenham e Mendes 2004]. Realizou-se o agrupamento de projetos conforme seus indicadores do fator produto, que incluem diversas medidas de complexidade de produto, além de informações contextuais como domínio da aplicação dentre outras.

1.4.2 Validade Interna

Validade interna se refere à relação causal entre os tratamentos e os resultados. Uma das ameaças à validade interna deste estudo refere-se à escolha dos fatores que influenciam na produtividade e, consequentemente, à possível existência de fatores de confusão. Há um risco de que os resultados de produtividade não sejam apenas devido aos fatores determinados no modelo conceitual.

1.4.3 Validade Externa

Validade externa diz respeito a saber se os resultados são generalizáveis além do estudo realizado, o que constitui um grande obstáculo ao se realizar estudos de caso [Yin 2010], devido à sua limitação a um dado contexto. Deve-se então estabelecer o domínio ao qual as descobertas de um estudo de caso podem ser generalizadas. Para isto, buscou-se descrever uma boa descrição do contexto, abarcando todas as facetas identificadas em [Petersen e Wohlin 2009b]. Além disso, fez-se o uso de proposições teóricas de modo a fazer replicações dos resultados existentes de outros estudos. Pesquisadores se tornam mais confiantes em uma teoria quando resultados similares emergem de diferentes contextos [Kitchenham et al. 1995a]. O registro de variáveis contextuais

de vários estudos experimentais permite aos pesquisadores a construção de evidências por meio de uma família de estudos. A replicação de estudos de caso pode tratar algumas ameaças à validade [Basili et al. 1999].

1.4.4 Validade da Conclusão

A validade da conclusão diz respeito a até que ponto um pesquisador pode seguir os mesmos procedimentos do estudo e chegar às mesmas constatações e conclusões. A ameaça identificada aqui é o viés de pesquisador nos procedimentos do estudo. Os procedimentos de coleta de dados sofreram pouca participação do pesquisador, uma vez que os dados quantitativos eram registrados pelos próprios desenvolvedores e as informações contextuais eram obtidas em registros históricos. O procedimento mais ameaçado pelo viés de pesquisador é o de análise de dados, particularmente os procedimentos de comparação dos projetos. Para reduzir vieses na comparação dos projetos, foram usadas ferramentas estatísticas e ferramentas de filtragem e ordenação de dados sempre que aplicáveis.

1.5 Revisão da Literatura

A literatura está repleta de exemplos que denotam favoritismo a um paradigma em detrimento do outro, seja por parte de acadêmicos ou profissionais da indústria [Boehm e Turner 2003a, Ramsin e Paige 2008]. Termos como "agilistas" e "tradicionalistas" foram cunhados e utilizados para identificar os defensores dos paradigmas ágil e tradicional respectivamente [Leffingwell 2011], porém não se encontram referências de pesquisas que se preocuparam em analisar a diferença dos efeitos do *turnover* na produtividade de projetos tradicionais e projetos ágeis.

As décadas de 1990 e 2000 testemunharam o surgimento de várias propostas de processos de desenvolvimento de *software* com as mais diversas características [Ramsin e Paige 2008]. No entanto, existem poucas evidências experimentais referentes a processos de desenvolvimento [Abrahamsson et al. 2003, Rombach e Seelisch 2007, Ramsin e Paige 2008]. Medir o real valor que um processo tem sobre o outro não é trivial, e por vezes encontram-se favoritismos e vieses subjetivos que interferem na qualidade da pesquisa científica [Ramsin e Paige 2008].

Apesar do grande interesse e dos avanços da indústria na utilização de processos, existe um campo vasto a ser explorado, pois os estudos existentes são preliminares e esta área de investigação ainda se encontra em níveis iniciais de maturidade [Rombach e Seelisch 2007, Dybå e Dingsøyr 2008, Dybå e Dingsoyr 2009]. Assim, a realização de pesquisas sobre adoção de processos tradicionais, ágeis ou hibridos é crucial para o desenvolvimento da engenharia de software, além de dar subsídios para as empresas decidirem qual abordagem é mais adequada para seu ambiente corporativo.

Pouco se sabe sobre os reais benefícios dos métodos ágeis ou híbridos pois, até o presente momento, estudos experimentais conduzidos em conformidade com boas práticas de pesquisa em engenharia de *software* são escassos [Dybå e Dingsøyr 2008, Dybå e Dingsoyr 2009]. A maioria das pesquisas sobre abordagens ágeis refere-se ao método XP – *eXtreme Programming* [Beck 2004]. Apesar do Scrum estar sendo amplamente adotado na indústria, sua popularidade na indústria não se reflete no volume de pesquisas a seu respeito [Dybå e Dingsoyr 2009]. A integração dos

métodos ágeis com métodos tradicionais é uma área altamente inexplorada [Galal-Edeen et al. 2007].

A quantificação dos impactos negativos que o *turnover* pode ocasionar em um projeto também é um campo de pesquisa amplo e virtualmente inexplorado. Independentemente de qual processo de desenvolvimento seja utilizado, é importante que os custos associados a ele sejam entendidos e medidos. Programadores, analistas e engenheiros de *software* precisam desenvolver modelos mentais complexos do sistema, entender o ambiente e os processos de negócio, codificar, testar e implantar o *software*. Mesmo os profissionais mais experientes e capacitados precisam de tempo para assimilar a arquitetura e o *design* do sistema, entender o funcionamento da infraestrutura de *hardware*, configurar os *frameworks* e praparar os ambientes de desenvolvimento.

Uma taxa de *turnover* de 10%, por exemplo, diminui o lucro e a produtividade média de um projeto em cerca de 3% [Stutzke 1995]. Roetzheim [Roetzheim 1988] e Abdel-Hamid [Abdel-Hamid e Madnick 1991] quantificaram os custos do *turnover*, mas em geral, os estudos quantitativos referentes a este assunto também são esparsos e inconclusivos.

As próximas seções apresentam uma revisão da literatura relacionada a esta pesquisa, subdividida em seções com propósitos específicos.

1.5.1 Processos Ágeis e Híbridos

Os trabalhos de Nord et. al [Nord e Tomayko 2006] e Boehm e Turner [Boehm e Turner 2003a,Boehm e Turner 2004] são marcos relevantes na discussão sobre a importância de se utilizar práticas ágeis sem abrir mão de princípios de desenvolvimento de *software* tradicional, notadamente a atenção à elaboração de arquitetura fundamentada em princípios e boas práticas de engenharia de *software*, de modo a aumentar a eficácia do gerenciamento de riscos, comunicação entre envolvidos, garantia a sistemas críticos e apoio a decisões técnicas e financeiras.

Complementando esta visão, os resultados de Dybå e Dingsøyr [Dybå e Dingsoyr 2009] sugerem que métodos ágeis podem colaborar com abordagens tradicionais porque utilizam melhores procedimentos para micro planejamento, controle diário do trabalho e relato de progresso. A comunicação entre membros da equipe e desta com o cliente é mais efetiva, pois os métodos ágeis privilegiam entregas amiúde e incremental de *software*. Eles também estimulam a realização de reuniões face-a-face para detalhar requisitos, descrever comportamentos e monitorar parâmetros do projeto, em detrimento da documentação. Recentemente, Petersen, Layman e Wohlin [Layman et al. 2004, Petersen e Wohlin 2009a] apresentaram uma lista consolidada das vantagens e desvantagens dos métodos ágeis.

Os estudos existentes sobre os efeitos em produtividade decorrentes da adoção de processos híbridos ou ágeis monstram que esta área de investigação ainda se encontra imatura e pouco explorada. Os estudos de Dybå e Dingsøyr, por exemplo, mostram alguns resultados intrigantes ou mesmo contraditórios [Dybå e Dingsøyr 2008]. A maioria dos estudos limitou-se ao escopo do método ágil *eXtreme Programming* (XP) e as informações contextuais são relatadas com qualidade variável [Tessem 2003, Layman et al. 2004, Dybå e Dingsøyr 2008, Petersen e Wohlin 2009a].

Outra característica marcante de processos ágeis é a dinamicidade com que profissionais são alocados e desalocados nas equipes. Métodos ágeis são apropriados para ambientes que

apresentam altas taxas de *turnover* [Boehm e Turner 2003a], pois, em tese, apresentam bons resultados por utilizarem profissionais altamente capacitados durante todo o ciclo de vida de desenvolvimento. Isso se deve ao fato de um dos princípios do desenvolvimento ágil ser a utilização de equipes multidisciplinares. Esta característica exige a alocação de profissionais capazes de desempenhar múltiplos papéis no ciclo de desenvolvimento [Cockburn 2000, Elssamadisy 2008, Boehm e Turner 2003a]. Em contrapartida, processos tradicionais necessitam de profissionais altamente capacitados nas fases iniciais do projeto, quando os requisitos são identificados e validados e a arquitetura é elaborada e estabilizada.

1.5.2 Efeitos do *turnover*

Estudos sobre os efeitos do *turnover* no processo de desenvolvimento de *software* têm sido conduzidos desde a década de 1970. Fred Brooks enunciou que "adicionar força de trabalho a um projeto atrasado, torna-o ainda mais atrasado" [Brooks 1974]. Ele reconheceu que a principal razão para este efeito é a necessidade dos profissionais antigos treinarem os novatos, o que diminui, por um período, a quantidade de trabalho produtivo em tarefas do projeto. A razão secundária, no entanto, é o aumento de *overhead* de comunicação causado pela necessidade de treinamento dos novos membros.

Roetzheim [Roetzheim 1988] e Abdel-Hamid [Abdel-Hamid e Madnick 1991] quantificaram os custos do *turnover*. Eles definiram um modelo exponencial para o tempo médio de contratação do profissional e o utilizaram para simular a diminuição da equipe. Em seguida, o modelo de política de pessoal repõe os profissionais e calcula os custos de assimilação. No entanto, os autores não combinam estes dois efeitos em um único modelo, como o modelo de Stutzke.

Stutzke desenvolveu um modelo matemático da Lei de Brooks que quantifica o impacto da alocação de novos profissionais no esforço e duração do projeto [Stutzke 1995], e que será utilizado nesta pesquisa. O modelo de Stutzke mostra que a Lei de Brooks nem sempre é verdadeira: em condições específicas, a alocação adicional de profissionais para trabalhar em projetos atrasados pode fornecer o volume necessário de trabalho para atender aos prazos ou acelerar a data de entrega [Stutzke 1994, Stutzke 1995].

Stutzke desenvolveu seu trabalho com base na premissa de que o esforço gasto na assimilação dos novos profissionais representa o maior custo associado à sua alocação [Stutzke 1994]. Durante o período de assimilação, os novos profissionais não contribuem efetivamente com trabalho útil para o projeto, porém, o esforço gasto nesta fase é debitado do orçamento do projeto. O modelo calcula a quantidade total de esforço útil produzido pela equipe, dado o número original de profissionais, N_i, a fração de aumento no tamanho da equipe, f, o tempo restante até o término do projeto, r, e um parâmetro chamado tempo para assimilação efetiva, a'.

O modelo de Stutzke considera, também, os efeitos da adição de profissionais na equipe do projeto e investiga os custos associados com a perda de um grupo de profissionais. Adicionalmente, a investigação sobre *turnover* permite a quantificação do custo associado à perda de uma pessoa chave da equipe. Devido ao impacto negativo e potencialmente grande que o *turnover* pode ocasionar, é importante entender suas causas e medir os efeitos dos custos associados a ele porque estas informações são essenciais para apoiar investigações sobre as reais implicações da agilidade nas taxas de produtividade.

Gopal *et alli* (2003) também mostraram evidências de que a duração, o tamanho, o tipo de contrato e a estratégia de alocação de membros na equipe interferem na lucratividade de projetos *offshore* ou *in house*. Eles argumentam que, num hipotético cenário de negócios onde as informações fossem corretas, completas e disponíveis desde o início das negociações, não importaria o tipo de contrato que fosse escolhido para que os resultados fossem conhecidos *ex ante*.

No entanto, a maioria dos contextos do mundo real é caracterizada por informações incompletas, em que o *software* é produzido por profissionais que executam tarefas de acordo com um processo. Sendo assim, parâmetros como o momento e sequência em que os profissionais abandonam ou são adicionadas à equipe afetam diretamente os resultados finais em termos de cronograma, produtividade, lucro, esforço e qualidade [Gopal et al. 2003].

A pesquisa qualitativa de Herbsleb sugere que o desenvolvimento que adota equipe com alta rotatividade pode aumentar o tempo de ciclo de desenvolvimento [Herbsleb et al. 2001]. Foram usados dados de pesquisas e do sistema de gerenciamento de versões para modelar a extensão do atraso em projetos em que houve alto *turnover*. Os autores também mediram as diferenças no padrão de comunicação de projetos desenvolvidos por equipes fixas e equipes de alta mobilidade de profissionais, semelhante à dinâmica adotada pelos métodos ágeis, e analisaram a relação destas variávies com atrasos no cronograma. Os resultados mostraram que, comparadas com equipes fixas, equipes de alta rotatividade, levam muito mais tempo, e requerem mais pessoas para desenvolver projetos de tamanho e complexidade semelhantes.

1.6 Estrutura da Dissertação

Para atender aos seus objetivos e facilitar o entendimento, a dissertação é organizada nos seguintes capítulos:

- Capítulo 2: Apresenta os conceitos e fundamentos relacionados ao tema da dissertação. São abordados os paradigmas tradicional e ágil, método RUP e Scrum, modelo CMMI, cálculo de produtividade, ferramentas estatísticas utilizadas no estudo de caso e descrição do processo de desenvolvimento utilizado.
- Capítulo 3: Descreve o modelo de Stutzke para calcular os efeitos do *turnover*. São apresentados os pressupostos do modelo, equações básicas, notações, convenções, coeficientes de produtividde, esforços de assimilação, contratação e *mentoring*, abordagens de substituição de recursos e custo do *turnover*.
- Capítulo 4: Apresenta o estudo experimental realizado para avaliar o impacto do *turnover*. São descritos o modelo de medição utilizado envolvendo projetos, requisitos, *bugs*, riscos e pessoas e também são apresentados os resultados consolidados da análise dos indicadores de produto, pessoas, tecnologia, risco e qualidade.
- Capítulo 5: Apresenta as considerações finais e discute direções futuras para pesquisa, seja por meio da extensão do modelo de Stutzke ou mediante a utilização de amostragens maiores e mais detalhadas de projetos reais para medir os efeitos do *turnover* e outros fatores na produtividade.

Capítulo 2

Conceitos e Fundamentos

O objetivo deste capítulo é apresentar os conceitos relacionados às áreas de conhecimento envolvidas na pesquisa e a revisão da literatura referente ao tema de investigação deste trabalho.

2.1 Paradigma Tradicional e RUP

Não há um nome unificado para designar o paradigma de desenvolvimento de *software* que remete às abordagens não ágeis de processo de desenvolvimento de *software*.

> Termos como "tradicional", "dirigido por plano", "disciplinado" e "pesado" são os mais encontrados na literatura. Com certa frequencia os autores dão preferência à utilização do termo "dirigido por plano", talvez por entenderem que o termo "tradicional" pode ser erroneamente associado a "antiquado"; e que o termo "pesado" pode sugerir que o processo, se não for ágil, necessariamente deva ser extremamente complexo; e que o termo "disciplinado" sugere que não há disciplina em métodos ágeis, o que é criticado por defensores do paradigma ágil [Alves 2011, Leffingwell 2011].

Os métodos tradicionais são focados em planejamento e elaboração de arquiteturas bem estabilizadas [Boehm e Turner 2004]. Seus principais objetivos são previsibilidade, estabilidade e alta garantia. Os métodos tradicionais valorizam a 1) criação e manutenção de artefatos bem definidos, 2) verificação, 3) validação, e 4) *interfaces* documentadas em especificações detalhadas.

O Processo Unificado de Desenvolvimento de Software, como descrito por Jacobson et al. [Jacobson et al. 1999], é um *framework* de *software* tradicional, dirigido por casos de uso que utiliza a notação da UML. Suas regras e documentação incluem descrições detalhadas de atividades organizadas de forma semi-ordenada, de tal modo sua estrutura possa ser flexibilizada e customizada. Isso o torna altamente customizável e permite que processos baseados nele atendam diversos contextos organizacionais. Assim, diversas atividades e artefatos podem ser desconsiderados ou reduzidos, a fim de deixar o grau de rigidez do processo de desenvolvimento adequado à realidade que o contexto atual exige.

O processo unificado deu origem ao RUP [Kruchten 2003], amplamente utilizado no desenvolvimento de *software*, e que foi definido pela Rational Corporation - atualmente uma divisão da IBM -, a partir de vários métodos e técnicas de análise e projeto orientado a objetos, sendo fortemente baseado em UML [Jacobson et al. 1999]. O RUP utiliza a UML como notação

padrão dos modelos que compõe os principais resultados das atividades do processo [Kruchten 2003, Jacobson et al. 1999].

O RUP é um processo iterativo que consiste de cinco *workflows* principais que são executados ao longo de quatro fases. A figura 2.1 ilustra as fases do ciclo de vida e um exemplo do volume de trabalho demandado pelos *workflows* ao longo de cada uma.

Esta mesma figura também ilustra a representação de dois aspectos RUP. O eixo horizontal apresenta o aspecto dinâmico do processo, enquanto o eixo vertical representa o aspecto estático do *framework*. A dimensão dinâmica representa o tempo e os aspectos do ciclo de vida do *software* na medida em que ele é desenvolvido. A dimensão estática apresenta as disciplinas do RUP que agrupam as atividades do processo em função de sua natureza.

Geralmente existe mais de uma iteração numa fase, assim, uma iteração de fase no RUP corresponde a um ciclo do modelo em espiral. O ciclo de vida do projeto acontece de tal forma que todas as quatro fases são executadas de acordo com a sequência de atividades dos cinco *workflows*.

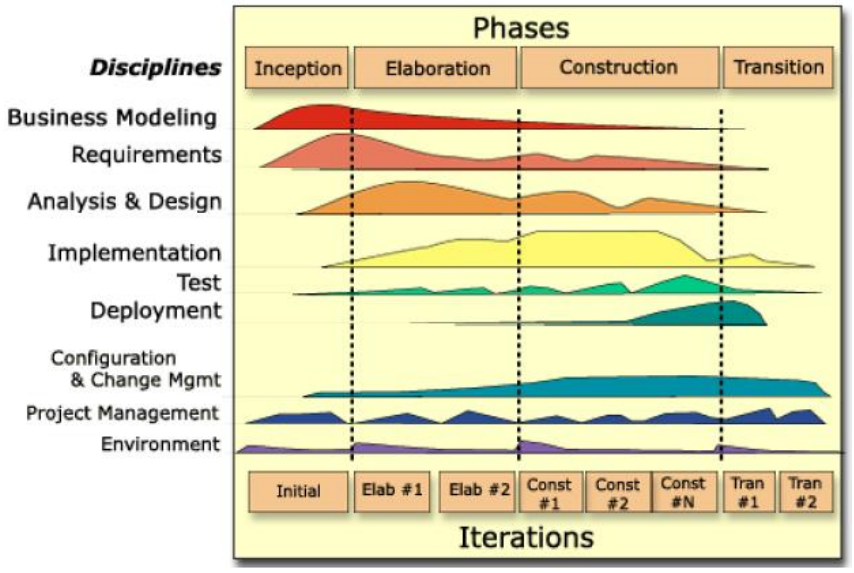

Figura 2.1: Ciclo de vida do RUP conforme Jacobson [Jacobson et al. 1999]

Nos próximos tópicos são descritas as principais características do RUP, bem como seus aspectos dinâmico e estático.

2.1.1 Características Principais

Os RUP e demais processos derivados do Processo Unificado de Desenvolvimento de *Software* apresentam um conjunto de características centrais:

Desenvolvimento Iterativo de *Software*

O desenvolvimento iterativo é uma estratégia de estruturação do ciclo de vida que consiste de várias iterações. Uma iteração incorpora um conjunto quase sequencial de atividades em modelagem de negócios, requisitos, análise e *design*, implementação, teste e implantação. Cada

iteração possui um escopo e objetivos definidos e o *software* é desenvolvido incrementalmente ao longo das iterações.

Gerenciamento de Requisitos

O gerenciamento de requisitos é uma abordagem sistemática para identificar, organizar e documentar os requisitos do sistema, além de estabelecer compromissos e firmar acordos entre os *stakeholders* do projeto.

Dirigido por casos de uso

Casos de uso [Cockburn 2000] são utilizados para capturar os requisitos e correspondem a um conjunto de cenários correlatos que descrevem a interação entre um tipo de usuário (ator) e o sistema. Por terem esta natureza, os casos de uso são adotados ao longo do processo pois ajudarão a assegurar que o sistema se comporte como desejado do ponto de vista do usuário. Os casos de uso servem de base para diversas disciplinas como projeto, implementação e testes.

Centrado na arquitetura

A arquitetura é o coração dos esforços da equipe para se modelar o sistema. Como um único modelo não é capaz de cobrir todos os aspectos de um sistema, o RUP prevê diversos modelos e visões arquiteturais. A arquitetura do sistema deve ser estabilizada ao final da fase de elaboração e servirá de alicerce para o desenvolvimento restante.

Arquitetura baseada em componentes

Componentes possuem *interfaces* e conectores bem definidos e podem ser substituídos conforme a necessidade. Arquiteturas baseadas em componentes tendem a reduzir a complexidade e o tamanho efetivo da solução, sendo mais robustas e flexíveis.

Modelagem Visual

Um modelo é uma visão simplificada de um sistema que mostra os elementos essenciais do sistema de uma perspectiva e oculta os detalhes não essenciais. Modelos visuais elaborados com a UML ajudam na compreensão do sistema, permite a comparação de projetos arquiteturais, servem de base para a implementação, capturam os requisitos com precisão e comunicam decisões sem ambiquidade.

Gerenciamento de Mudanças

O gerenciamento de mudanças permite um controle formal das mudanças nos requisitos, projeto, implementação e testes do sistema. Além disso, permite um controle de versões e *releases* do sistema, assim como das *baselines* dos artefatos criados ou modificados ao longo do desenvolvimento do *software*.

Verificação contínua da qualidade

Ao longo do desenvolmento de software, a qualidade dos produtos intermediários é verificada em marcos ou periodicidade pré-definidos. O RUP prevê planejamento de testes, revisões, auditorias e avaliações da qualidade do *software*.

Focado em risco

O processo visa mitigar os principais riscos nas fases iniciais. Os produtos produzidos em cada iteração, especialmente na fase de elaboração, devem ser selecionados de forma a assegurar que os maiores riscos sejam tratados primeiramente.

2.1.2 Dimensão Dinâmica do RUP

A dimensão dinâmica do RUP divide o ciclo de vida do software em quatro fases, e cada uma delas é subdividida em iterações. Cada iteração representa um ciclo completo de desenvolvimento. Ao final das iterações obtém-se um conjunto de funcionalidades que evolui incrementalmente até que a versão final seja obtida. Uma fase corresponde ao tempo entre dois *milestones* durante o qual um conjunto definido de objetivos é atendido, artefatos são produzidos e decisões sobre quais elementos serão promovidos às fases subsequentes são tomadas [Kruchten 2003, Tamaki 2007]. As quatro fases do RUP são descritas a seguir [Kruchten 2003, Jacobson et al. 1999, Tamaki 2007]:

Iniciação

A meta da fase de Iniciação é obter o compromisso entre os *stakeholders* sobre os objetivos do projeto. Seus principais objetivos envolvem definir o escopo do projeto, delimitar as fronteiras do *software*, conceber algumas propostas arquiteturais, estimar custos, identificar os riscos potenciais, preparar o ambiente de suporte do projeto e estabelecer os principais marcos. Ao final da fase de Iniciação, tem-se o marco com os objetivos do projeto, em que são feitas análises de oportunidade e viabilidade que darão subsídios para a decisão de continuar ou interromper o projeto.

Elaboração

A meta da fase de Elaboração é estabelecer a *baseline* da arquitetura do sistema, a qual evolui a partir da priorização dos requisitos, casos de usos mais significativos e da avaliaçao de riscos. Os objetivos principais da Elaboração são:

- Assegurar que a arquitetura, os requisitos e o planeamento sejam estáveis o suficiente e os riscos sejam adequadamente mapeados e diminuídos.
- Mitigar os riscos significativos do ponto de vista da arquitetura do projeto.
- Estabelecer uma *baseline* da arquitetura, elaborada a partir da realização dos cenários arquiteturais relevantes.
- Estabelecer o ambiente de suporte ao desenvolvimento do projeto.

- Planejar as tarefas de cada fase do ciclo de vida com base na priorização de requisitos, arquitetura e expectativas de negócio.

Ao final desta fase, tem-se o marco arquitetural do ciclo de vida do projeto. Este marco estabelece uma *baseline* da arquitetura e permite a maximização do desempenho da alocação de profissionais na equipe do projeto, sobretudo para a fase de Construção.

Construção

A meta da fase de Construção é detalhar os requisitos restantes e concluir o desenvolvimento do sistema com base na *baseline* da arquitetura. Nesta fase, a ênfase está no gerenciamento de recursos e no controle de operações para obter os melhores parâmetros possíveis para custo, cronograma e qualidade. Os principais objetivos da Construção são:

- Minimizar os custos de desenvolvimento, aperfeiçoando recursos e evitando retrabalhos desnecessários.
- Construir *releases* intermediários do produto para teste – incluindo as versões *alfa*, *beta*, *candidate release* e *final release* –, ou *delivery*.
- Concluir a análise, projeto arquitetural, codificação e teste de todas as funcionalidades necessárias.
- Planejar as atividades de homologação, operação assistida, e treinamento de usuários.

Ao final desta fase, é estabelecido o marco operacional inicial, em que se determina se o produto está pronto para ser homologado, implantado ou integrado ao ambiente de negócio.

Transição

A meta da fase de Transição é assegurar que o software seja integrado ao ambiente do cliente e disponibilizado para os usuários finais. Os objetivos da Transição são:

- Estabelecer a *baseline* de implantação do sistema, contendo a versão funcional mais atualizada do *software*.
- Validar o sistema de acordo com as necessidades e expectativas dos usuários.
- Realizar a migração de dados de outros sistemas para o atual, caso necessário.
- Obter aceite formal dos *stakeholders* de que as *baselines* de implantação estão corretas, completas e consistentes com os critérios de aceitação.

Ao final da fase de Transição estabelece-se o marco de *release* do produto final, e a partir daí, prossegue-se com atividades de operação assistida e manutenção.

2.1.3 Dimensão Estática do RUP

A dimensão estática do RUP estabelece, basicamente, os responsáveis pela realização de cada atividade, quando elas deverão ser feitas e de que forma. Para isso, são utilizados os elementos descritos nos próximos tópicos [Tamaki 2007].

Fluxo de Trabalho

Um fluxo de trabalho é uma sequência de atividades, ou macro atividades, que produz resultado de valor observável quando é executado [Kruchten 2003]. Cada macro atividade representada na figura 2.2 correponde a um primeiro nível de detalhamento, que pode ser expandido de acordo com a necessidade ou complexidade.

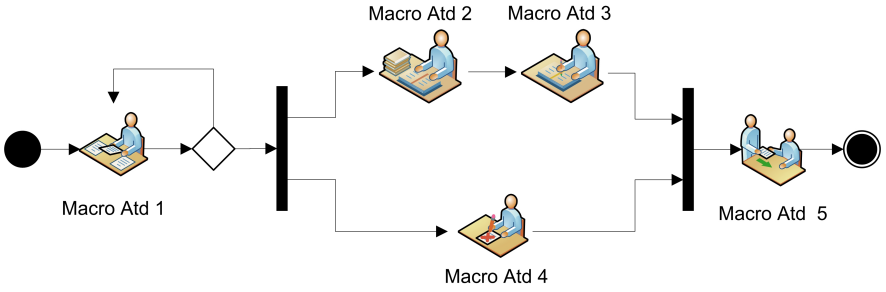

Figura 2.2: Exemplo de fluxo de trabalho do RUP

As macro atividades representam um conjunto de atividades, papéis e artefatos. Cada fluxo é repetido a cada iteração, ou seja, o início e o fim do diagrama representam, respectivamente, o começo e encerramento de uma iteração. Apesar do fluxo ser executado a cada iteração, o esforço gasto em cada atividade pode variar [Davis 1993, Clements et al. 2003, Shaw e Garlan 1996, Ammann e Offutt 2008]. Por exemplo, o esforço de arquitetura geralmente é grande nas iterações iniciais ao passo que, nas iterações finais, ele é mínimo.

Papéis

Um papel define um conjunto de comportamentos ou responsabilidades que um profissional ou equipe deve possuir ao longo do processo de desenvolvimento estabelecido pelo RUP. Um mesmo profissional pode desempenhar diversos papéis durante o desenvolvimento do *software*. Os principais papéis definidos pelo RUP, em função da natureza e da relevância das tarefas designadas a eles, são: analistas, codificadores, testadores e gerentes. A figura 2.3 ilustra um exemplo de detalhamento de fluxo que mostra as atividades que um papel realiza e os respectivos artefatos que elas produzem ou utilizam.

Atividades

As atividades são unidades de trabalho que possuem objetivos claros e são executadas mediante criação ou modificação de artefatos, seguindo procedimentos específicos. Cada atividade representa parte do trabalho que produza algum resultado significativo para o projeto. A cada papel é atribuído um conjunto de atividades, conforme ilustrado na figura 2.3.

Artefatos

Um artefato é um produto de trabalho do processo. Os profissionais produzem ou modificam artefatos ao executar as atividades dos papéis que desempenham, conforme ilustrado na figura 2.3.

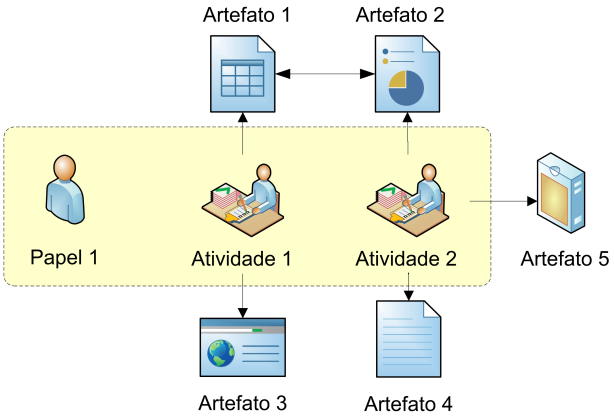

Figura 2.3: Exemplo de detalhamento de fluxo de trabalho do RUP

As disciplinas do RUP

Uma disciplina é um conjunto de atividades relacionadas a uma mesma área de interesse. O objetivo do agrupamento das atividades em disciplinas é facilitar a compreensão do processo. Cada disciplina possui seu próprio fluxo de trabalho, que é repetido a cada iteração. O RUP agrupa as atividades do ciclo de vida em nove disciplinas:

- **Modelagem de Negócio**: responsável pelo entendimento do domínio do problema, pela identificação de falhas no modelo de negócio que será automatizado pelo sistema e identificação de oportunidades de melhoria.
- **Requisitos**: responsável por identificar e manter o acordo entre cliente, desenvolvedores e demais *stakeholders* sobre os objetivos do sistema.
- **Análise e *Design***: responsável pela elaboração da arquitetura, análise e *design* técnico do sistema, de acordo com os requisitos e restrições identificados.
- **Implementação**: responsável pela codificação e testes unitários de componentes do sistema.
- **Testes**: responsável pelo planejamento e execução de testes integrados e de sistema. Caso outros tipos de teste sejam necessários, como testes de desempenho, *stress* ou segurança, suas atividades também são agrupadas nesta disciplina.
- **Implantação**: responsável por assegurar que o sistema seja implantado adequadamente no ambiente dos usuários finais.
- **Gestão de Projeto**: responsável pela elaboração, monitoramento e controle dos planos de execução do projeto.
- **Gestão de Configuração e Mudança**: responsável por controlar as alterações feitas nos artefatos do projeto, manter sua integridade e criar as *baselines* necessárias.
- **Ambiente**: responsável por estabelecer e manter os recursos necessários para desenvolvimento do projeto.

2.2 Paradigma Ágil

Os métodos ágeis surgiram como reação aos processos tradicionais de desenvolvimento [Marçal 2009]. O termo ágil foi cunhado, nesta acepção, em fevereiro de 2001, após um encontro em Utah, nos EUA, em que foi criada a Agile Alliance, uma organização que promove conceitos de agilidade para o desenvolvimento de software [Beck et al. 2001, Alves 2011]. Neste encontro surgiu o "Manifesto para Desenvolvimento Ágil de Software" [Beck et al. 2001], que estabelece quatro principais valores do paradigma ágil. Este documento contém declarações do tipo:

> "Passamos a valorizar:
>
> - Indivíduos e interações mais que processos e ferramentas.
> - Software em funcionamento mais que documentação abrangente.
> - Colaboração com o cliente mais que negociação de contratos.
> - Responder a mudanças mais que seguir um plano."

Segundo Cohen et al. (2003),

> "...os métodos ágeis podem ser considerados uma coletânea de diferentes técnicas e métodos que compartilham os mesmos valores e princípios básicos, alguns dos quais remontando em técnicas introduzidas em meados dos anos de 1970 como o desenvolvimento e melhoria iterativos" [Cohen et al. 2003].

Williams e Cockburn (2003) afirmam que

> "...o desenvolvimento ágil tem tudo a ver com abraçar feedback e alterações e, também, que os métodos ágeis são construídos para aceitar, ao invés de rejeitar, altas taxas de alterações" [Cockburn 2002].

Abrahamsson afirma que os métodos ágeis são caracterizados por serem: incrementais, devido aos ciclos rápidos de desenvolvimento; cooperativos, já que estimulam a proximidade entre o cliente e interação entre os desenvolvedores; diretos, devido à simplicidade de aprendizado e documentação; e finalmente adaptativos, pela habilidade de avaliar e acomodar mudanças ao longo do projeto [Abrahamsson et al. 2003].

Além disso, o desenvolvimento ágil enfatiza a importância do planejamento adaptativo, simplicidade e liberação contínua de software com valor operacional em pequenas iterações com tempo fixado. Em oposição ao paradigma dirigido por plano, os métodos ágeis tratam o desafio de um mundo imprevisível confiando na criatividade das pessoas e não nos processos [Nerur et al. 2005, Dybå 2000].

2.3 Método Scrum

O Scrum é um método ágil criado em 1996 por Schwaber e Sutherland (2001). Seus criadores acreditam que o desenvolvimento de *software* é imprevisível e formaliza a abstração, sendo aplicável a ambientes voláteis. O Scrum diferencia-se dos demais métodos ágeis pela maior ênfase

dada ao gerenciamento do projeto [Udo e Koppensteiner. 2003, Marçal 2009] e reúne atividades de monitoramento e realimentação, em geral, reuniões rápidas e diárias com toda a equipe, visando a eliminação rápida de impedimentos no processo de desenvolvimento.

O Scrum pressupõe que projetos de desenvolvimento de *software* sejam complexos e imprevisíveis a tal ponto que seja inviável planejá-lo totalmente no seu início. Ao invés de planejar detalhadamente o projeto de forma prescritiva, deve-se conduzir as atividades empiricamente[1] para garantir visibilidade, inspeção e adaptação do planejamento [Schwaber 2004]. O Scrum baseia-se em princípios como: equipes pequenas de no máximo 7 pessoas; com requisitos que são pouco estáveis ou desconhecidos; com ciclos curtos em que divide o desenvolvimento em *time-boxes* de no máximo 30 dias, também chamados de *sprints*.

2.3.1 Papéis e Responsabilidades

O Scrum implementa um *framework* iterativo e incremental cujas atividades são assumidas por três papéis principais [Schwaber 2004]:

Product Owner

O *Product Owner* representa os interesses de todos no projeto, sobretudo do cliente. Ele define os fundamentos do projeto definindo requisitos iniciais e gerais (*Product Backlog*), retorno do investimento, objetivos e planos de entregas e prioriza o *Product Backlog* a cada *sprint*, garantindo que as funcionalidades de maior valor sejam construídas prioritariamente.

ScrumMaster

O *ScrumMaster* gerencia o Scrum, ensinando-o a todos os envolvidos no projeto e implementando-o de modo adequado à cultura da organização. Ele garante que todos sigam as regras e práticas do Scrum e é responsável por remover os impedimentos do projeto.

Equipe

A equipe é responsável por desenvolver as funcionalidades do produto, definir as estratégias de implementação do *Product Backlog* e gerenciar seu próprio trabalho numa *sprint*.

2.3.2 Artefatos

De acordo com Schwaber [Schwaber 2004, Marçal 2009], o Scrum introduz os seguintes artefatos principais usados ao longo do seu fluxo de desenvolvimento: *Product Backlog*, *Sprint Backlog* e incremento de funcionalidade do produto.

O *Product Backlog* consiste de uma lista de itens priorizados que incluem os requisitos funcionais e não funcionais do sistema/produto que está sendo desenvolvido no projeto. O *Product Owner* é o responsável pelos conteúdos e priorização do *Product Backlog* que é usado no plano de projeto apenas como uma estimativa inicial dos requisitos. O *Product Backlog* nunca está completo e evolui de acordo com a evolução do produto e do ambiente em que está inserido. O

[1] O modelo experimental de controle de processos proporciona controle através de exercícios frequentes de inspeção e adaptação em processos que não estão totalmente definidos, gerando resultados que não são repetitivos.

gerenciamento constante das mudanças serve para identificar o que o produto precisa para ficar apropriado, competitivo e usável.

O *Sprint Backlog* corresponde à lista de tarefas que a equipe do projeto define para implementar na *sprint* os requisitos selecionados do *Product Backlog*. Apenas a equipe pode mudar o *Sprint Backlog* que representa o planejamento real do time para a *sprint*. A equipe deverá entregar incrementos de funcionalidade a cada *sprint*. Estes incrementos consistem de códigos testados, estruturados e bem escritos, que são executáveis acompanhados da documentação necessária para a sua operação.

2.3.3 Fases do Scrum

O ciclo de vida do Scrum possui quatro fases, como definido em [Cockburn 2002, Layman et al. 2004]:

1. **Planejamento**: objetiva estabelecer a visão do projeto e as expectativas entre os *stakeholders*, além de garantir orçamento para a realização do projeto;
2. **Preparação**: objetiva identificar os requisitos e priorizá-los, pelo menos para a próxima *sprint*. Nesta fase, o *Product Backlog* é dividido em *sprints*, de acordo com a priorização realizada, levando em consideração a produtividade da equipe;
3. **Desenvolvimento**: corresponde à implementação do sistema em uma série de *sprints* de até 30 dias consecutivos, com incrementos funcionais ao final de cada *sprint*;
4. **Entrega**: corresponde à implantação operacional do sistema.

2.3.4 Fluxo de Trabalho

Projetos Scrum iniciam com a elaboração da visão do produto que será desenvolvido [Schwaber 2004]. A visão contém a lista das características do produto estabelecidas pelo cliente, além de algumas premissas e restrições. Em seguida, o *Product Backlog* é criado, listando todos os requisitos conhecidos. O *Product Backlog* é então priorizado e dividido em *releases*. O fluxo de desenvolvimento detalhado do Scrum é mostrado na figura 2.4, adaptada de [Gloger 2007].

Todo o trabalho no Scrum é realizado em iterações chamadas de *sprints*, que se iniciam com uma reunião de planejamento (*Sprint Planning Meeting*), na qual o *Product Owner* e a equipe decidem em conjunto o que deverá ser implementado (*Selected Product Backlog*). A reunião é dividida em duas partes. Na primeira parte (*Sprint Planning 1*), o *Product Owner* apresenta os requisitos de maior valor e prioriza aqueles que devem ser implementados.

A equipe, então, define o que poderá entrar no desenvolvimento da próxima *sprint*, considerando sua capacidade de produção. Na segunda parte (*Sprint Planning 2*), a equipe planeja seu trabalho, definindo o *Sprint Backlog*, que são as tarefas necessárias para implementar as funcionalidades selecionadas a partir do *Product Backlog*. Nas primeiras *sprints*, é realizada a maioria dos trabalhos de arquitetura e de infra-estrutura. A lista de tarefas pode ser modificada ao longo da *sprint* pelo time e as tarefas podem variar entre 4 a 16 horas para a sua conclusão.

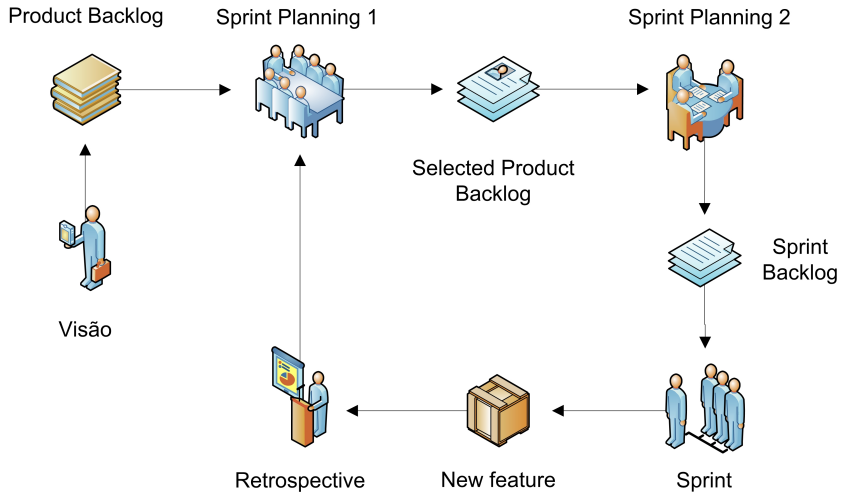

Figura 2.4: Visão geral do processo do Scrum [Cockburn 2002]

Durante a execução das *sprints*, diariamente o time faz uma reunião de 15 minutos para acompanhar o progresso do trabalho e agendar outras reuniões necessárias. Na reunião diária (*Daily Scrum Meeting*), cada membro da equipe responde a três perguntas básicas:

1. O que eu fiz no projeto desde a última reunião?
2. O que irei fazer até a próxima reunião?
3. Quais são os impedimentos?

No final da *sprint*, é realizada a reunião de revisão (*Sprint Review Meeting*) para que a equipe apresente o resultado alcançado na *sprint* ao *Product Owner*. Neste momento as funcionalidades são inspecionadas e adaptações do projeto podem ser realizadas. Em seguida o *ScrumMaster* conduz a reunião de retrospectiva (*Sprint Retrospective Meeting*), com o objetivo de melhorar o processo, equipe e/ou produto para a próxima *sprint*.

2.4 CMMI®

O CMMI® é o modelo de qualidade que consolida boas práticas da indústria de software. Elas visam aprimorar os processos organizacionais de desenvolvimento, gerenciamento e manutenção de produtos e serviços, abrangendo o ciclo de vida do produto desde sua concepção até a entrega e manutenção.

O CMMI® foi desenvolvido pelo *Software Engineering Institute* (SEI), vinculado à Universidade Carnegie Mellon (CMU) e financiado pelo Departamento de Defesa norte-americano (DoD). O modelo foi criado para dar suporte ao desenvolvimento de sistemas em empresas que fornecessem software ao governo daquele país. O CMMI® consolida um conjunto de boas práticas de melhoria de processos que se destinam à promoção da qualidade dos produtos desenvolvidos [Mohagheghi e Conradi 2004]. O SEI considera que organizações que possuam processos maduros sejam melhor preparadas para controlar riscos e resolver problemas graças à visão e entendimento compartilhados, linguagem comum e visibilidade objetiva baseada em indicadores quantitativos [Chrissis et al. 2011].

Assim, a proposta do CMMI® é ser um modelo integrado que pode ser utilizado em várias disciplinas, com foco, sobretudo, em desenvolvimento integrado do produto e do processo, desenvolvimento de sistemas, desenvolvimento de *software* e subcontratação. Este modelo descreve orientações para a definição de implantação de processos, porém não descreve processo algum, deixando isto a cargo das organizações: são orientações definidas pelas práticas especificadas.

Ao desenvolver o CMMI®, o SEI adotou como premissa fundamental que a qualidade do produto final está diretamente relacionada à qualidade do processo utilizado para produzi-lo. Isso representa uma mudança cultural dentro da organização, que modifica, de forma positiva, sua maneira de operar, integrando atividades de administração e controle com as de engenharia de *software* em um todo consistente e coerente.

2.4.1 Evolução

Desde 1991, foram desenvolvidos CMMs para uma gama de disciplinas. Alguns dos mais conhecidos foram os modelos para Engenharia de Sistemas, Engenharia de Software, Aquisição de Software, Gestão e Desenvolvimento de Força de Trabalho, e Desenvolvimento Integrado de Processo e Produto (IPPD).

Embora esses modelos tenham se mostrado úteis para muitas organizações, o uso de múltiplos modelos foi considerado problemático. Muitas organizações gostariam que seus esforços de melhoria pudessem englobar diferentes grupos na organização. Entretanto, as diferenças entre esses modelos específicos orientados a disciplinas e utilizados por cada equipe, quanto à arquitetura, ao conteúdo e à abordagem, têm limitado a capacidade dessas organizações em ampliar com sucesso a abrangência de suas melhorias. Além disso, a aplicação de vários modelos não integrados em uma organização é dispendiosa em termos de treinamento, avaliações e atividades de melhoria. O projeto CMM Integration^{SM} foi constituído para resolver o problema originado com o uso de múltiplos CMMs. A missão inicial da Equipe do Produto CMMI era combinar três modelos:

- *Capability Maturity Model for Software* (SW-CMM) v2.0 draft C [SEI 1997].
- *Systems Engineering Capability Model* (SECM) [EIA 1998].
- *Integrated Product Development Capability Maturity Model* (IPD-CMM) v0.98 [SEI 1997].

A combinação desses modelos em um *framework* único visava permitir sua utilização pelas organizações na sua busca pela melhoria de processo no âmbito da corporação como um todo. Esses três modelos utilizados como base foram escolhidos pela sua popularidade nas comunidades de Software e de Engenharia de Sistemas, e em função de suas diferentes abordagens para a melhoria de processo em uma organização.

Utilizando informações desses modelos populares e bem aceitos como base, foi criado um conjunto coerente de modelos integrados que podem ser adotados tanto por aqueles que já estão utilizando os modelos originários, quanto por aqueles que ainda não conhecem o conceito do CMM. Portanto, o CMMI® é resultado da evolução do SW-CMM, do SECM e do IPD-CMM.

O desenvolvimento de um conjunto integrado de modelos significou a combinação de modelos existentes utilizando processos que promovem o consenso. O CMMI® passou a ser um *fra-*

mework que acomoda múltiplas disciplinas e é suficientemente flexível para apoiar as diferentes abordagens dos modelos que o antecederam [Ahern et al. 2003].

2.4.2 Constelações

A partir da versão 1.2 do CMMI® surgiu o conceito de "constelações CMMI®". Uma constelação é um conjunto de componentes CMMI® concebidos para satisfazer as necessidades de uma área de interesse específica. Uma constelação pode ser utilizada para construir um ou mais modelos CMMI®, documentos de avaliação e materiais de treinamento relacionados. Cada constelação possui suas próprias área de processo, mas compartilham áreas de processo em comum, denominadas *Core Process Areas*. As constelações CMMI® existentes são:

1. CMMI-DEV: Orienta o gerenciamento, avaliação e monitoramento dos processos de desenvolvimento. Envolve Integração, Desenvolvimento de Requisitos, Verificação e Validação.
2. CMMI-SVC: Orienta a prestação de serviços para clientes internos e externos das organizações. Envolve Suporte a Serviços, Gerência de Projeto de Serviço e Estabelecimento e Entrega de Serviço.
3. CMMI-ACQ: Orienta atividades de aquisição de produtos e serviços de outros fornecedores. Envolve Gerência de Acordos, Verificação de Aquisição e Gerência Técnica de Aquisição.

Atualmente o CMMI® está em sua versão 1.3, e neste trabalho sempre que o termo CMMI® for utilizado, ele estará se referindo à constelação CMMI®-DEV.

2.4.3 Componentes

Os componentes do modelo CMMI® estão agrupados em três categorias que informam como devem ser interpretados [Chrissis et al. 2011], a saber:

- **Requeridos**: são aqueles que descrevem o que deve ser implementado nos processos da organização visando satisfazer uma área de processo. Os componentes requeridos são os *Objetivos Específicos* e *Objetivos Genéricos* do modelo.
- **Esperados**: são aqueles que descrevem o que uma organização deve implementar para que os componentes requeridos sejam atendidos. Os componentes esperados são as *Práticas Específicas* e as *Práticas Genéricas* do modelo.
- **Informativos**: são aqueles que ajudam as organizações a pensar em como podem implementar os componentes requeridos e esperados. Os componentes informativos correspondem aos demais componentes prescritos pelo modelo.

Uma Área de Processo é um conjunto de práticas relacionadas em uma determinada área que, quando executadas coletivamente, satisfazem o conjunto dos objetivos considerados importantes para a melhoria desta área. Alguns componentes complementares são adicionados à área de processo com o objetivo de descrevê-la claramente:

- **Propósito**: descreve a finalidade da área de processo.

- **Notas introdutórias**: descrevem os conceitos principais tratados pela área de processo.
- **Lista de áreas relacionadas**: mapeiam os relacionamentos entre as áreas de processo.

Um objetivo específico descreve as características que o processo organizacional deve possuir para que a área de processo seja satisfeita. O objetivo específico é composto por várias práticas específicas que descrevem as atividades esperadas para atingí-lo. As práticas específicas relacionam uma lista de produtos típicos de trabalho, usualmente produzidos para atendê-la.

Objetivos genéricos descrevem características que devem estar presentes durante a institucionalização do processo. São genéricos porque se aplicam a todas as áreas de processo. Tanto os objetivos genéricos quanto os específicos são componentes requeridos do modelo CMMI® e são usadas nas avaliações oficiais para determinar se uma PA foi satisfeita ou não.

As sub-práticas, por sua vez, são descrições detalhadas que fornecem orientação para a interpretação e implementação de uma prática específica ou genérica. São componentes informativos do modelo com o propósito de apresentar idéias úteis para a melhoria dos processos.

2.4.4 Representações

O CMMI® é composto por 22 áreas de processo e oferece duas abordagens para melhoria de processos, conforme ilustrado na figura 2.5.

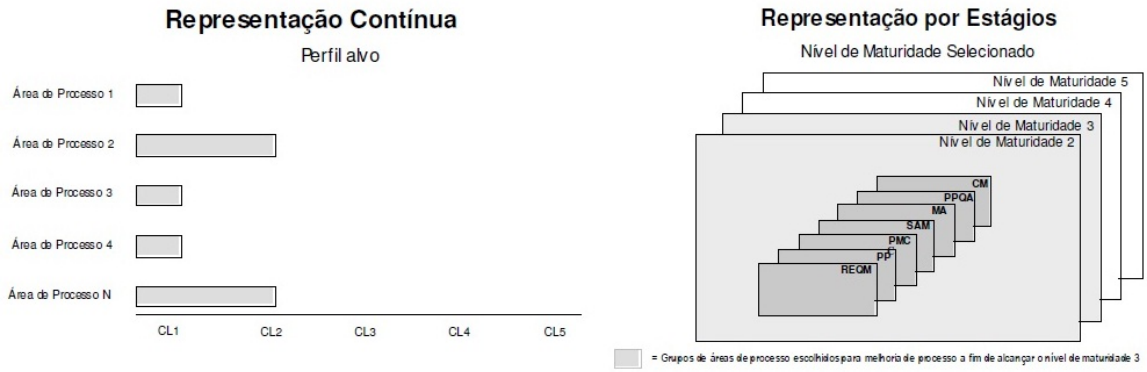

Figura 2.5: Áreas de Processo na Representação Contínua e na Representação por Estágios

A representação contínua define níveis de capacidade por área de processo e a representação em estágios define 5 níveis de maturidade organizacionais, sendo que cada nível reúne um conjunto de áreas de processo a serem implementadas em níveis evolutivos de maturidade. Cada área de processo possui objetivos que representam o resultado efetivo da aplicação das práticas recomendadas e relacionadas ao seu contexto específico.

No entanto, se a organização optar pela representação contínua, ela poderá melhorar o desempenho de uma única área de processo – relacionada a um determinado problema – ou pode trabalhar em diversas áreas independentes que estejam alinhadas aos seus objetivos estratégicos [Chrissis et al. 2011]. Permite também que uma organização melhore diferentes processos em capacidades distintas, limitadas, entretanto, pelas dependências existentes entre algumas áreas de processo. Na representação contínua, as áreas de processos são avaliadas em seis níveis de capacidade numerados de 0 a 5.

As organizações utilizam a representação por estágios com maior frequência que a representação contínua porque aquela oferece um *roadmap* estruturado para a melhoria dos processos da organização baseado em um estágio de cada vez.

As áreas de processo são estruturadas em níveis de maturidade. Quando uma organização atinge um nível de maturidade, considera-se que seus processos alcançaram uma determinada capacidade, ou seja, possuem mecanismos que garantem a repetição sucessiva de bons resultados. A melhoria contínua dos processos da organização é obtida por meio de passos evolutivos entre os cinco níveis de maturidade do modelo, definidos e numerados de 1 a 5 conforme descritos a seguir.

Nível de Maturidade 1 - Inicial

Os processos são informais e caóticos. A organização normalmente não possui um ambiente estável. O sucesso destas organizações depende da competência e heroísmo das pessoas e não do uso de processos comprovados. Apesar deste ambiente informal e caótico, organizações muitas vezes produzem produtos e serviços que funcionam; entretanto, elas freqüentemente excedem o orçamento e o cronograma de seus projetos.

Nível de Maturidade 2 - Gerenciado

Os requisitos, processos, produtos de trabalho e serviços são gerenciados. A situação dos produtos de trabalho e a entrega dos serviços são visíveis para o gerenciamento em marcos definidos. Compromissos são estabelecidos entre os *stakeholders* relevantes e são revistos conforme necessário. Os produtos de trabalho são revisados com os *stakeholders* e controlados. Os produtos de trabalho e serviços satisfazem seus requisitos, padrões e objetivos definidos.

Nível de Maturidade 3 - Definido

O conjunto de processos padrão da organização é estabelecido e melhorado ao longo do tempo. Os projetos estabelecem seus processos definidos adaptando o conjunto de processos padrão da organização. A alta gestão da organização estabelece os objetivos dos processos e assegura que estes objetivos estão sendo tratados de forma adequada. Neste nível, os processos são gerenciados de forma mais pró-ativa, utilizando um entendimento dos inter-relacionamentos das atividades de processos e medidas detalhadas do processo, seus produtos de trabalho e seus serviços.

Nível de Maturidade 4 - Gerenciado Quantitativamente

São selecionados os subprocessos que contribuem significativamente para o desempenho geral do processo. A qualidade e o desempenho do processo são entendidos em termos estatísticos e são gerenciados durante toda a vida dos processos. Medidas de qualidade e desempenho de processos são incorporadas ao repositório de medições da organização para dar suporte a futuras decisões baseadas em fatos ocorridos.

Nível de Maturidade 5 - Otimizado

Os processos são continuamente melhorados com base em um entendimento quantitativo das causas comuns de variações inerentes aos processos.

2.4.5 Categorias das Áreas de Processo

Além da divisão tradicional do CMMI em níveis de maturidade com áreas de processo que perseguem metas genéricas e específicas, as áreas de processo definidas podem ser agrupadas em quatro categorias:

- **Gerenciamento de Projetos**: cobrem as atividades de gerenciamento de projetos relacionadas ao planejamento, monitoramento e controle do projeto;
- **Engenharia**: cobrem as atividades de desenvolvimento e manutenção que são compartilhadas entre as disciplinas de engenharia;
- **Suporte** cobrem as atividades que suportam o desenvolvimento e manutenção de produtos. Tratam os processos que são utilizados no contexto da execução de outros processos;
- **Gerenciamento de Processos**: contêm atividades que percorrem todo o projeto, relativas à definição, planejamento, distribuição de recursos, aplicação, implementação, monitoramento, controle, avaliação, medição e melhoria de processos.

2.5 Medição e Análise

O objetivo das atividades de medição e análise é fornecer subsídios para desenvolver e manter os elementos de medição utilizados para dar suporte às necessidades de informação da empresa. Neste trabalho são utilizadas, sobretudo, medidas associadas ao projeto, como produtividade, tamanho funcional, esforço, *turnover*, número de requisitos e quantidade de *bugs*. Nas próximas seções são definidos e descritos os parâmetros de medição de tamanho funcional, esforço, produtividade, e *turnover* considerados neste trabalho.

2.5.1 Tamanho Funcional

Como explicitado na seção anterior, conhecer o tamanho funcional do *software* é imprescindível para calcular a produtividade dos projetos. Algumas métricas usadas para medir o tamanho do *software* são brevemente descritas a seguir:

1. **Linhas de Código**: É o número de linhas do código fonte, sendo comumente chamado de SLOC (*Source Lines of Code)* [Fenton e Pfleeger 1997]. Há várias formas de contagem de SLOC, algumas delas voltadas às linhas de código propriamente ditas excluindo-se os comentários e linhas em branco, outras voltadas aos *statements* e comandos de programação. A quantidade de SLOCs é considerada uma medida física do tamanho do *software* por medir apenas o volume de código fonte.
2. ***Software Science***: Proposto por Halstead [Halstead 1977], de modo a medir o tamanho do código, através do número de ocorrências de operadores e operandos, e o volume, que

corresponde à quantidade requerida de espaço em disco. Foi uma das primeiras métricas que se preocuparam em considerar, de alguma forma, as funcionalidades e regras do sistema.

3. **Pontos de Função**: Baseado na funcionalidade do programa, sendo o número total de pontos de função influenciados por cinco classes: quantidade de entradas, quantidade de saídas, quantidade de entradas interativas, quantidade de arquivos internos ao sistema e quantidade de arquivos externos ao sistema [Albrecht e Gaffney 1983].
4. ***Feature Points***: extende os pontos de função de modo a incluir algoritmos como uma nova classe [Jones 1996].
5. ***Full Function Point***: é uma outra extensão dos pontos de função de modo a medir aplicações de tempo real [St-Pierre et al. 1997]. Outros métodos de medição de pontos de função são IFPUG, Mark II, 3D, Asset-R, Experience e Cosmic [Maya et al. 1998, Rehesaar 1998, COSMIC-FFP 2001, Laturi 1996].
6. ***Object Points***: Enquanto que feature points e FFP ampliam o escopo de medição dos pontos de função, os *object points* são baseados na quantidade e complexidade de telas, relatórios e componentes 3GL. Apesar de não ser muito popular, *object points* são fáceis de serem usados nas fases iniciais do ciclo de desenvolvimento, sendo usados nos maiores modelos de estimativa de custo, como o COCOMO II [Boehm et al. 1996].
7. ***Use Case Points***: A técnica de estimar o tamanho de um sistema por casos de uso foi proposta por Gustav Kamer em 1993 para ser utilizada na fase de levantamento de Casos de Uso e é baseada no método de Pontos por Função [Saleh 2009]. Este método considera como os usuários utilizarão o sistema de acordo com a complexidade das ações.

Os seguintes critérios foram utilizados para selecfionar pontos de função como unidade de medida funcional:

- A técnica de análise de pontos de função é mantida por uma organizaççao internacional sem fins lucrativos, desde 1986, o *International Function Points Users Group* – IFPUG.
- Os pontos de função possuem suporte no Brasil por meio do *Brazilian Function Point Users Group* – BFPUG, além de empresas especializadas.
- O IFPUG mantém um programa mundial de certificação profissional em Pontos de Função, que confere aos aprovados o título de *Certified Function Point Specialist* – CFPS. No Brasil este programa é mantido pelo BFPUG.
- Os pontos de função são padronizados internacionalmente pela ISO, através da norma ISO/IEC 20926, possibilitando a uniformidade na aplicação.
- Os pontos de função consideram os requisitos em um nível de abstração mais elevado e independente dos artefatos do que os UCP, podendo ser utilizados com qualquer representação de requisitos.
- A existência de acervo de dados sobre pontos de função armazenados por diversas organizações possibilita a realização de estudos e comparações.
- A utilização de pontos de função em contratos e licitações é uma realidade no Brasil, tendo surgido a partir da iniciativa de órgãos públicos.

De acordo com a técnica Análise de Pontos por Função, uma aplicação de software, vista sob a ótica do usuário, é um conjunto de funções ou atividades do negócio que o beneficiam na realização de suas tarefas [Castro 1998]. O manual do IFPUG classifica os seguintes tipos de elementos funcionais [Hastings 1995]:

1. **Entrada Externa (EI)**: transações lógicas onde dados entram na aplicação e mantém dados internos.
2. **Saída Externa (EO)**: transações lógicas onde dados saem da aplicação para fornecer informações para usuários da aplicação.
3. **Consulta Externa (EQ)**: transações lógicas onde uma entrada solicita uma resposta da aplicação.
4. **Arquivos Lógicos Internos (ILF)**: grupo lógico de dados mantido pela aplicação.
5. **Arquivos de Interface Externa (EIF)**: grupo lógico de dados referenciado pela aplicação, mas mantido por outra aplicação.

O manual do IFPUG fornece tabelas e diretrizes para determinar a complexidade de cada elemento funcional [IFPUG 2000]. A complexidade dos ILF e EIF é baseada no número de registros lógicos e no número de itens de dados referenciados e a complexidade das transações é baseada no número de Arquivos Referenciados e no número de Itens de Dados Referenciados.

Os elementos funcionais identificados são totalizados para calcular obtenção dos Pontos por Função não Ajustados. Então é calculado a partir de 14 características gerais dos projetos, que permitem avaliar aspectos não funcionais da aplicação[2].

A cada característica é atribuído um peso de 0 a 5, de acordo com o Nível de Influência na aplicação. O Nível de Influência Geral, NI é obtido pelo somatório do nível de influência de cada característica, NI_c. Assim, o Fator de Ajuste, FA é obtido pela expressão:

$$FA = 0,65 + \sum_{c=1}^{14} NI_c.0,01$$

O total de Pontos por Função, PFA, da aplicação é encontrado multiplicando-se o número de Pontos por Função Brutos, PFB, pelo Fator de Ajuste, FA. O FA pode variar entre 0,65 a 1,35 e é uma proposta para ajustar a medida do tamanho do *software* às suas características não funcionais [Lokan 2000, Calazans et al. 2005]. Na prática, o tamanho final do *software* pode variar em $\pm$ 35% em função do Nível de Influência Geral. Maiores detalhes sobre a técnica podem ser obtidos em [IFPUG 2000].

No entanto, apenas conhecer o número de pontos de função de um projeto não é suficiente para estimar o esforço de desenvolvimento. Para isso, é preciso considerar a produtividade média em determinada tecnologia. Com base nestas informações, foi desenvolvida um artefato de apoio à contagem que leva em consideração as regras do manual de contagem e outros *drivers* de estimativa, como risco, *skill* dos profissionais e tecnologia.

[2]Comunicação de Dados, Processamento Distribuído, Atualização de Dados Online, Entrada de Dados Online, Volume de Transações, Eficiência do Usuário Final, Medição de Pontos por Função a Partir de Especificação de Requisitos, Complexidade do Processamento, Facilidade de Implantação, Multiplicidade de Locais, Facilidade de Mudanças, Facilidade Operacional, Desempenho, Utilização do equipamento, Reutilização de Código

O acúmulo de dados dados históricos referentes à produtividade permite estimar o esforço necessário para o atendimento de pedidos na mesma tecnologia. Com base neles, o mecanismo de contagem se transformou num artefato de estimativa, onde são derivados os esforços de diversas atividades com base no número de pontos de função e percentual gasto em fases e disciplinas do RUP.

2.5.2 Esforço

O esforço é um parâmetro notavelmente difícil de ser medido com acurácia. Shepperd et al. descreveram a experiência de acompanhar uma organização nas suas práticas de estimativa de esforço [Shepperd e Schofield 1997]. Os dados referentes ao esforço de um mesmo projeto, de três diferentes fontes, possuíam discrepância de até 30% pois os responsáveis pelas medições consideravam elementos não padronizados. Algumas questões exploratórias que ajudam a delimitar o escopo da medição do esforço são:

1. O projeto será medido em horas ou meses?
2. Será considerado o tempo alocado para atividades gerenciais, de suporte à equipe ou apenas o tempo de desenvolvimento propriamente dito?
3. Serão consideradas as horas extras não pagas?
4. Caso o cliente atue ativamente no projeto, isso contará como uma ajuda no esforço total?

Assim, embora exista uma equação padrão para medição da produtividade, sua aplicação pode gerar resultados divergentes, dependendo das variáveis utilizadas na medição do tamanho do *software* e esforço gasto. Além disso, observa-se que o valor de produtividade calculado dessa forma não reflete as circunstâncias nas quais o software foi desenvolvido.

2.5.3 Produtividade

A medição da produtividade é vital para a melhoria da eficiência e eficácia do desenvolvimento de *software*. Atualmente, as medidas tradicionais de produtividade, embora sejam úteis, não são plenamente capazes de apontar possíveis variações referentes a parâmetros externos, apresentando somente as relações históricas entre entradas e saídas.

Segundo [Gold 1973], os indicadores de produtividade traduzem em números a situação da empresa em um dado instante considerado, mostrando-a exatamente como ela está naquele instante. Para que os indicadores sejam confiáveis, é preciso que eles possuam algumas características, tais como:

- Meçam corretamente e com precisão o estado verdadeiro do fenômeno.
- Mostrem exatamente aquilo que se deseja medir.
- Abranjam todas as partes importantes do processo.
- Sejam quantificáveis, simples, controláveis e inteligíveis.
- Apresentar resultados confiáveis e passíveis de ações corretivas.

- Permitam rastrear o problema até seu foco.

A maioria dos estudos que analisam a produtividade de *software* se baseiam apenas na perspectiva quantitativa da produção, de acordo com a Equação 2.5.3.

$$Produtividade = \frac{Tamanho\,do\,Software}{Esforco\,Necessario}$$

Elementos qualitativos que podem ser considerados são Experiência da Equipe, Ambiente de Trabalho, Capacidade do Projeto, Processo, Complexidade do Projeto e Cliente [da Cunha 2008]. Apesar de reconhecer a importância de utilizar critérios qualitativos e ambientais para aumentar a acurácia da medição, como os descritos em [da Cunha 2008], neste trabalho será usada uma métrica puramente quantitativa conforme equação 2.5.3, uma vez que o modelo de métricas utilizado não considera variáveis qualitativas de medição.

2.5.4 *Turnover*

O *turnover*, T_p, ou rotatividade de pessoal, refere-se à relação entre alocações e substituição de profissionais antigos por novos da equipe de desenvolvimento de um projeto. Esta relação é expressa em termos percentuais e a expressão matemática utilizada para calculá-la é a seguinte:

$$T_p = \left(\frac{D}{N_i + N_f}\right).100$$

onde:

- T_p = *turnover* do projeto.
- D = número de profissionais que deixaram a equipe.
- N_i = tamanho da equipe no início do projeto.
- N_f = tamanho da equipe no final do projeto.

É importante que os gestores acompanhem os índices de rotatividade dos projetos, pois ele é um indicador da existência de problemas organizacionais. Como não existe um índice ideal de *turnover*, pois há variações de mercado, projetos, tecnologias, é interessante que cada empresa procure estabelecer baselines internas para comparação.

2.6 Ferramentas estatísticas

O uso de ferramentas estatísticas para estudar a correlação de dados em Engenharia de *Software* permite realizar diversas análises que possibilitam compreender o comportamento dos dados coletados e gerar conhecimento a partir deles. Em análise estatística, os dados coletados merecem especial atenção.

De acordo com [Mirer 1983], dados são fatos quantitativos ou pedaços de informações que se quer trabalhar estatisticamente. Ainda segundo este autor, qualquer processo de engenharia que faça estudos experimentais, os dados são coletados de um conjunto de casos ou ocorrências de um processo [de Lucca Ramos 1999]. Estes casos são chamados *observações*, e a natureza destes casos ou instantes define a unidade da observação.

Algumas das medidas estatísticas mais comuns dos dados são a medida da tendência central e as medidas de dispersão:

- Medidas de tendência central são caracterizadas pela média dos dados e preocupa-se com a medida do típico valor que a variável assume nos dados analisados.
- Medidas de dispersão dos dados preocupam-se em determinar a faixa em que os valores se encontram, assim como procura quantificar o tamanho desta faixa.

Algumas das medidas mais comuns de dispersão são a amplitude total, a variância e o desvio padrão. Outras medidas mais avançadas para avaliar os dados são os coeficientes de assimetria, γ_3, e curtose, γ_4. Como o uso da medida do coeficiente de assimetria e curtose são utilizadas ao longo do trabalho, será dada ênfase nestas medidas.

Os próximos tópicos apresentam as medidas estatísticas que possibilitam obter melhor entendimento dos procedimentos utilizados para análise dos indicadores e dados de Engenharia de *Software* neste trabalho.

2.6.1 Tendência central

A tendência central é quantificada pela média dos dados. Esta média é geralmente encontrada somando-se todos os dados da variável analisada e dividindo-se esta soma pela quantidade de observações. A fórmula é apresentada como:

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

- X_i é o valor da i-ésima variável analisada.
- n é o número de observações.

2.6.2 Desvio padrão

O desvio padrão é a medida mais comum de dispersão e é geralmente estimada como:

$$\gamma_2 = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \bar{X})^2}{n-1}}$$

- X_i é o valor da i-ésima variável analisada.
- $\bar{X}$ é a média da distribuição.
- n é o número de observações.

2.6.3 Assimetria

A medida de assimetria mede o grau de assimetria de uma distribuição (conjunto de dados) em torno de sua média. A distribuição é dita com assimetria positiva quando a maioria dos dados assumem valores positivos em comparação a média da distribuição [de Lucca Ramos 1999]. Da mesma forma, a distribuição é dita assimetricamente negativa quando a maioria dos dados

possuem valores negativos quando comparados com a média da distribuição. O valor da medida de assimetria de uma amostra de tamanho n é dado pela fórmula:

$$\gamma_3 = \frac{n}{(n-1).(n-2)} . \sum_{i=1}^{n} \left(\frac{X_i - \bar{X}}{\gamma_2} \right)^3$$

- X_i é o valor da i-ésima variável analisada.
- $\bar{X}$ é a média da distribuição.
- n é o número de observações.
- γ_2 é o desvio padrão da distribuição.

Interpretação:

- Valores de assimetria próximos a 0 indicam dados simétricos.
- Valores negativos implicam assimetria à esquerda.
- Valores positivos indicam assimetria à direita.

Após a determinação do coeficiente de assimetria, e antes que venhamos a determinar o coeficiente de curtose, cabe definir o que é uma distribuição normal.

2.6.4 Distribuição Normal

A distribuição de um conjunto de dados é dita normal quando a média dos seus desvios $(X_i - \bar{X})$ é igual a zero e a variância dos erros da distribuição é constante. Para uma distribuição normal o coeficiente de assimetria é igual a zero, pois os dados analisados encontram-se simétricos com relação a média, uma vez que para a distribuição dita normal a média dos desvios é nulo.

2.6.5 Curtose

O teste de curtose é geralmente analisado para avaliar a normalidade da distribuição, e compara a distribuição amostrada com a distribuição caracterizada como normal. O valor da medida de curtose de uma amostra de tamanho n é dado pela fórmula:

$$\gamma_4 = \left(\frac{n.(n+1)}{(n-1).(n-2).(n-3)} . \sum_{i=1}^{n} \frac{X_i - \bar{X}}{\gamma_2} \right)^4 - \frac{3.(n-1)^2}{(n-2).(n-3)}$$

- X_i é o valor da i-ésima variável analisada.
- $\bar{X}$ é a média da distribuição.
- n é o número de observações.
- γ_2 é o desvio padrão da distribuição.

Para uma distribuição normal o coeficiente de curtose é nulo. Uma regra conveniente para suspeitar de não normalidade da distribuição é encontrar coeficientes de curtose $\gamma_2 < -1$ ou $\gamma_4 > +1$ [Foster 1986]. A interpretação para os valores de γ_4 é:

- $\gamma_4 \approx 0$: pico da curva tem inclinação normal (curva mesocúrtica).
- $\gamma_4 < 0$: pico da curva da distribuição é mais achatado que o usual (curva platicúrtica).
- $\gamma_4 > 0$: pico da curva de distribuição é mais agudo que o usual (curva leptocúrtica).

2.6.6 Correlação

Correlação, também chamada de coeficiente de correlação, indica a força e a direção do relacionamento linear entre duas variáveis aleatórias. A correlação se refere a medida da relação entre duas variáveis, embora correlação não implique causalidade. O coeficiente indica o grau de intensidade da correlação entre duas variáveis e, ainda, o seu sentido, positivo ou negativo. O coeficiente de correlação mais conhecido é o de Pearson, o qual é obtido dividindo a covariância de duas variáveis pelo produto de seus desvios padrão.

$$r = \frac{\sum_{i=1}^{n} x_i.y_i - \frac{\sum_{i=1}^{n} x_i . \sum_{i=1}^{n} y_i}{n}}{\sqrt{\left(\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}\right) . \left(\sum_{i=1}^{n} y_i^2 - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n}\right)}}$$

Os valores de r pertencem ao intervalo $(-1, +1)$. Valores de r iguais a -1 ou $+1$ indicam que a correlação é perfeita. Valores de r próximos de -1 ou $+1$ indicam uma correlação forte e valores de r próximos de zero indicam correlação fraca. O sinal de r indica se a correlação é **positiva** ou **negativa**. De maneira prática, para tirar conclusões significativas sobre o comportamento simultâneo de variáveis analisadas, é necessário que $0,6 \leq |r| \leq 1$.

- Se $0,3 \leq |r| \leq 0,6$ há uma correlação relativamente fraca entre as variáveis.
- Se $0 \leq |r| \leq 0,3$, a correlação é muito fraca e, praticamente nada se pode concluir sobre a relação entre as variáveis em estudo.

2.7 Processo de Desenvolvimento

Nesta seção é apresentado o processo híbrido de desenvolvimento adotado pela empresa, o qual incorpora características de ambos os paradigmas: ágil e tradicional. Nas próximas seções, será apresentado o *framework* utilizado para integrar as práticas do Scrum ao processo baseado no RUP. O termo *framework* é utilizado para denotar que o processo descrito não possui caráter rígido, podendo sofrer pequenas adaptações de acordo com as exigências do contexto no qual for aplicado, desde que seja respeitada sua estrutura geral. Os detalhes e as principais decisões de projeto são apresentados e discutidos e cada tópico aborda um aspecto do processo.

2.7.1 Modelo de ciclo de vida

O modelo de ciclo de vida incorpora características do modelo de ciclo de vida quase-espiral. Na sequência, um conjunto de subprocessos é executado em estilo parcialmente iterativo, entrega evolutiva e *time boxed* [Karlstrom e Runeson 2005, Pikkarainen et al. 2008].

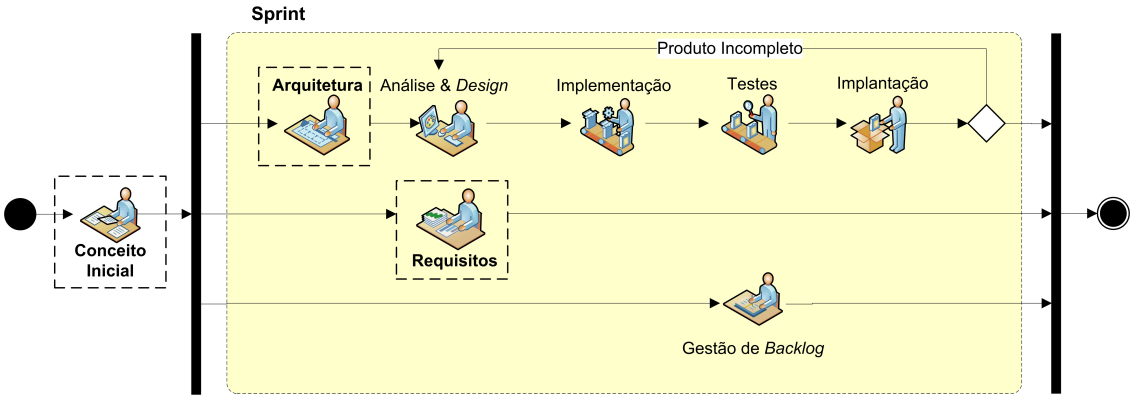

Figura 2.6: Ciclo de vida dos projetos de acordo com processo híbrido

A figura 2.6 ilustra o modelo de ciclo de vida utilizado pelo processo híbrido. As atividades destacadas são as que seguem as práticas do paradigma tradicional e as demais seguem os preceitos ágeis. A seguir é feita uma descrição breve de cada subprocesso indicado.

- **Conceito Inicial**: Responsável pela identificação dos requisitos do sistema, documentação de aspectos arquiteturais relevantes e estimativa utilizando análise de pontos de função e elaboração da proposta comercial.
- **Arquitetura**: Ocupa-se em elaborar a arquitetura do sistema e mitigar os principais riscos do projeto.
- **Requisitos**: Compreende fundamentalmente atividades de especificação de casos de uso [Cockburn 2000] e validação dos mesmos frente ao cliente.
- **Análise & *Design***: Corresponde às atividades de produção de todos os tipos de artefatos de design não arquitetônico, como realizações de casos de uso e protótipos de *interface* com usuário.
- **Implementação**: Destinado a realizar atividades de codificação, testes de unidade, integração e empacotamento.
- **Testes**: Corresponde a atividades de planejamento e execução de testes.
- **Implantação**: Atividades de implantação em ambientes de homologação e produção, bem como produção de documentação.
- **Gestão de *Backlog***: Corresponde ao trabalho de manter o *Product Backlog* atualizado gerando e priorizando *user stories* [Cohn 2004].

2.7.2 Características

O ciclo de vida do processo utilizado é parcialmente aderente ao modelo quase-espiral pois é executado o subprocesso `Conceito Inicial` responsável pelo levantamento inicial dos requisitos e questões arquitetônicas mais importantes, obtenção de recursos e infraestrutura para o projeto. Ele também incorpora caracteríticas do modelo entrega evolutiva porque, diferentemente do que acontece em desenvolvimento puramente ágil, a elaboração da arquitetura é antecipada para os momentos iniciais do projeto, como no RUP [Kruchten 2003]. O desenvolvimento, assim como

no Scrum, é subdividido em iterações de tempo fixo com possibilidade de entrega de incremento do produto ao final de cada uma [Alves 2011].

Arquitetura

A elaboração da arquitetura é finalizada e estabilizada o quanto antes no ciclo de vida para subsidiar a realização das atividades de análise e *design* detalhados. É ela que delineia as linhas mestras do *design* detalhado, mitiga os riscos inerentes ao *refactoring* e evita a tomada de decisões tardias de projeto [McBreen 2003, Petersen e Wohlin 2009a, Clements et al. 2003].

A elaboração e estabilização da arquitetura, gestão formal de requisitos e especificação de casos de teste podem contribuir para reduzir a necessidade de profissionais altamente qualificados, geralmente requeridos para o sucesso da utilização de métodos ágeis [Merisalo-Rantanen et al. 2005, Boehm e Turner 2003a]. Isso ocorre porque a documentação produzida como saída destes subprocessos pode ser utilizada como guia para os membros menos experientes da equipe.

Requisitos

As atividades de especificação e validação podem ocorrer por toda a "macro etapa" pontilhada do ciclo de vida, porém tal esforço deve se concentrar nas iterações iniciais do projeto, especialmente no sentido de priorizar os requisitos que tenham impacto na elaboração da arquitetura. Os requisitos funcionais são especificados como casos de uso, seguindo o princípio do RUP.

Testes

De modo a contemplar o princípio de o projeto ser guiado por casos de uso, a especificação de casos de testes se faz necessária para que se possa assegurar que os casos de uso estão sendo devidamente atendidos, assim como para assegurar níveis aceitáveis de qualidade e formalidade na validação do produto, o que não impede a utilização de princípios ágeis.

Gestão de *Backlog*

Este subprocesso ocorre ao longo da "macro etapa" pontilhada indicada na figura anterior. Para manter o *backlog* atualizado, este subprocesso deve utilizar informações geradas pelos subprocessos de Requisitos, Arquitetura e *Design* e Testes. O subprocesso Gestão de *Backolog* é a *interface* de comunicação entre os elementos ágeis e tradicionais do processo.

Particionamento ágil e tradicional

Os subprocessos Conceito Inicial, Requisitos, Arquitetura e Design devem ser executados com o rigor prescrito no paradigma tradicional. O subprocesso Teste se situa em uma zona de predominância indefinida. Os demais subprocessos são predominantemente ágeis pois os objetivos primários do paradigma tradicional são melhor atendidos se a arquitetura for elaborada nas etapas iniciais do processo e os requisitos forem especificados e validados formalmente.

Time Boxing

O processo possui caráter híbrido pois, embora ciclos de tempo fixo para potencial entrega de incrementos no produto sejam estabelecidos, a delimitação do escopo e a definição de parâmetros do projeto é feita previamente, como resultado do subprocesso Conceito Inicial. Além disso, todas as "macro etapas" do ciclo de vida devem ser executadas na forma de *sprints*.

2.7.3 Artefatos, papéis e Atividades

A tabela 2.1 descreve o subconjunto de artefatos encontrados no processo.

Artefato	**Descrição**	**Subprocesso**
SR1	A Especificação de Requisitos contém um conjunto de requisitos que define o comportamento externo do sistema a ser desenvolvido. Ao elaborá-lo os requisitos já terão sido identificados e descritos com detalhes suficientes para estimar o escopo, esforço e prazo do projeto.	Conceito Inicial
SR2	Evolução do SR1, contendo detalhamento de requisitos e especificação dos casos de uso para validação.	`Requisitos`
Architeture	Abrange decisões significativas sobre a organização do sistema, seleção de elementos estruturais e suas *interfaces*. Especifica questões de usabilidade, funcionalidade, performance, reuso e tecnologia.	`Arquitetura`
Data Model	Modelo de dados do sistema.	`Arquitetura`
Product Backlog	Lista de requisitos na forma de *user stories* [Schwaber e Sutherland 2010]. Em uma *Sprint* tem-se o *Sprint Backlog*, que é a lista de estórias, priorizadas a partir do *Product Backlog*, para a tal *Sprint*.	`Gestão de Backlog`
Release Burndown	Representação gráfica dos esforço restante, ao longo do tempo, para execução do projeto [Schwaber e Sutherland 2010]. Em nível de *Sprint* existe o artefato equivalente *Sprint Burndown*.	`Gestão de Backlog`
Implementation Element	Partes físicas que compõe a implementação do sistema, tais como arquivos de código, arquivos de dados, diretórios e ajuda online.	`Implementação`
Build	Versão operacional do sistema ou parte do sistema que demonstra um subconjunto das capacidades a serem providas para produto final.	`Implementação`
Test Case	Conjunto de cenários de teste e seus elementos com propósito de verificar algum aspecto do item a ser testado.	`Testes`
Test Evidence	Relato quantitativo e/ou qualitativo da execução de um teste, com o propósito de evidenciar o resultado obtido no mesmo.	`Testes`
User Manual	Auxilia o usuário a aprender, utilizar e operar o produto.	`Implantação`
Instalation Guide	Documento que provê instruções requeridas para instalar o produto em ambiente de produção.	`Implantação`

Tabela 2.1: Principais Artefatos do Processo Utilizado

Capítulo 3

Quantificação dos Efeitos do Turnover

O objetivo deste capítulo é apresentar o modelo matemático desenvolvido por Stutzke [Stutzke 1994, Stutzke 1995] com base na Lei de Brooks, cujo enunciado afirma que "adicionar força de trabalho a um projeto atrasado, torna-o ainda mais atrasado" [Brooks 1974]. Brooks reconheceu que a principal razão para isso é a necessidade dos profissionais antigos treinarem os novatos, o que diminui momentaneamente a quantidade de esforço dedicado às tarefas do projeto, e ainda o aumento de esforço de comunicação. Brooks não criou mecanismos para computar os efeitos quantitativos da rotatividade de profissionais, mas Stutzke desenvolveu expressões matemáticas para a Lei de Brooks que quantifica este impacto [Stutzke 1994].

A maioria dos projetos que dura mais que poucas semanas enfrenta *turnover*, que é a saída de profissionais da equipe antes da finalização das atividades em que estavam envolvidos. Apesar deste fenômeno ser potencialmente danoso a qualquer tipo de projeto, seus efeitos são mais visíveis naqueles que envolvem grande volume de esforço e longa duração. Além disso, projetos de desenvolvimento de *software* são mais sensíveis aos efeitos do *turnover* do que operações repetitivas de manufatura porque desenvolvimento de *software* é um trabalho altamente baseado em conhecimento.

O modelo de Stutzke é utilizado como referencial teórico para analisar os efeitos do *turnover* na produtividade em projetos de desenvolvimento reais, construídos por uma empresa de tamanho médio especializada em soluções customizadas para clientes de diversos segmentos. Os projetos considerados neste estudo foram desenvolvidos ao longo de dois anos utilizando um processo híbrido de desenvolvimento.

Uma característica marcante de projetos desenvolvidos de acordo com processos ágeis ou híbridos é a dinamicidade com que os analistas são alocados e desalocados nas equipes. Métodos ágeis são apropriados para ambientes que apresentam altas taxas de *turnover* [Boehm e Turner 2003a], pois, em tese, apresentam bons resultados por utilizarem profissionais altamente capacitados durante todo o ciclo de vida [Boehm e Turner 2003a, Qumer e Henderson-Sellers 2008]. Um dos princípios do desenvolvimento ágil é a utilização de equipes multidisciplinares, o que exige a alocação de profissionais aptos a desempenhar múltiplos papéis do ciclo de vida. Em contrapartida, processos de desenvolvimento de *software* tradicionais demandam profissionais altamente capacitados sobretudo nas fases iniciais do ciclo de vida. Estes profissionais geralmente deixam a equipe após o término das fases iniciais do projeto em que os requisitos foram validados e a arquitetura elaborada [Clements et al. 2003].

O modelo de Stutzke pressupõe que o esforço gasto na assimilação dos novos profissionais representa o maior custo associado à alocação dos novos membros na equipe. Durante o período de assimilação, os novos profissionais não contribuem com esforço efetivo para o projeto, no entanto, o trabalho realizado é debitado do orçamento do projeto. O modelo permite calcular a quantidade total de esforço efetivo realizado pela equipe em função do número original de profissionais, N_i, a fração de aumento no tamanho da equipe, f, o tempo restante até o término do projeto, r, e o tempo de assimilação, a'. Estes parâmetros são utilizados para estimar os efeitos da adoção de diferentes estratégias de alocação de profissionais, e dão suporte ao cálculo dos custos associados à rotatividade de profissionais.

Gerentes de projeto podem utilizar este modelo para auxiliar na definição de estratégias de gestão de equipe, como, por exemplo, compressão do cronograma para mitigar os efeitos do *turnover*. Além disso, quando são utilizados dados históricos da organização, o modelo pode ser empregado para planejar e contingenciar o efeito do *turnover* nos custos e prazo de entrega do projeto.

3.1 Pressupostos do Modelo

O principal pressuposto do modelo é que o esforço despendido pelos membros da equipe durante a assimilação de novos profissionais representa o maior custo associado à alocação destes. Este custo é responsável direto pelo decréscimo momentâneo da produtividade da equipe porque o treinamento dos novos profissionais consome esforço dos demais. Além disso, as seguintes condições são estabelecidas:

1. A contratação é feita exclusivamente pelo corpo administrativo, sem envolvimento do corpo técnico. Isto não é estritamente verdade porque os técnicos geralmente precisam selecionar currículos, corrigir testes e realizar entrevistas.
2. Os novos profissionais possuem as habilidades necessárias à sua função e todos recebem o mesmo treinamento específico necessário ao entendimento do projeto. O Este treinamento inclui instalações, políticas organizacionais, processos, padrões de desenvolvimento, ferramentas, *frameworks*. Em alguns casos, o treinamento é fornecido em cursos formais, mas na maioria das vezes, isto é feito na forma de *mentoring*: os profissionais mais antigos e experientes da equipe são designados para treinar e acompanhar os novatos.
3. As tarefas de *mentoring* são distribuídas uniformemente pelos profissionais experientes. o que permite que o restante da equipe continue trabalhando em tarefas do projeto.
4. O esforço de assimilação é realizado num mesmo intervalo de tempo, o que significa que os novos membros da equipe aprendem numa taxa uniforme.
5. Os novos profissionais são alocados na equipe ao mesmo tempo. Esta é uma simplificação útil para fins de planejamento e modelagem pois a quantidade de esforço realizado passa a ser diretamente proporcional ao tamanho da equipe. Como o processo de desenvolvimento considerado é baseado em Scrum, pressupõe-se que os profissionais estarão disponíveis no início de cada *sprint*.

6. A equipe original e a nova equipe apresentam as mesmas taxas de produtividade, pois considera-se que todos os profissionais tenham a mesma capacitação e que os recursos necessários ao desempenho de suas funções lhes tenham sido fornecidos.
7. Desconsideram-se quaisquer perdas de produtividade associadas ao aumento do tamanho da equipe, como por exemplo, aumento da dificuldade de comunicação.
8. O esforço e a duração necessários para realizar o projeto são conhecidos e bem definidos.
9. A equipe não possui, de antemão, todo o conhecimento necessário à execução do trabalho, ou seja, todos os membros passarão por treinamento e assimilação.
10. Os profissionais récem alocados na equipe praticamente não contribuem com trabalho efetivo até que a fase de *mentoring* seja concluída.

O modelo calcula tanto a quantidade total de esforço produzido quanto a quantidade total de esforço realizado pela equipe. Esforço produzido e esforço realizado não são iguais porque é preciso utilizar alguma quantidade de esforço para treinar os novos membros da equipe.

3.2 Modelo de Stutzke

Considerando que a data prevista para finalização de um projeto de desenvolvimento de *software* esteja comprometida por algum motivo, assume-se que o trabalho remanescente a ser feito, E_r, e o esforço necessário para finalizá-lo sejam conhecidos. O modelo lida com a adição de um número específico de profissionais à equipe do projeto e duas fases são identificadas: assimilação e esforço pleno. Durante a assimilação, alguns membros experientes da equipe são designados para treinar e dar suporte aos novatos, impendindo-os de realizar, momentaneamente, parte das tarefas do projeto. Considera-se que os novos profissionais praticamente não contribuem com trabalho efetivo durante este período.

Entretanto, tanto os profissionais novatos quanto os experientes consomem horas do orçamento do projeto. Uma vez finalizada a assimilação, toda a equipe passa a realizar trabalho efetivo. O modelo calcula as taxas de trabalho realizado e de trabalho efetivo para cada fase. Estas taxas são expressas em termos de esforço por período de tempo. O esforço é medido em pessoas-dia e a duração é medida em dias úteis para eliminar os efeitos de finais de semana e feriados.

A quantidade de esforço efetivo realizado por uma equipe de N_i profissionais é N_i pessoas por dias úteis. Geralmente cada profissional trabalha uma pessoa-dia por dia útil à taxa R_i, o que a torna numericamente idêntica ao número de pessoas, N_i.

3.3 Equações Básicas

Com base nas considerações e pressupostos anteriores, espera-se que o *turnover* na equipe cause, inicialmente, queda na produtividade. Isto se deve à redução do volume de trabalho efetivo que é destinado às tarefas do projeto. No entanto, após os novos membros da equipe terem sido assimilados, a produtividade voltará a aumentar até os limites permitidos pela estratégia

de substituição de profissionais adotada. Nestes termos, o esforço total gasto em um projeto é [Stutzke 1994]:

$$E_e = (1 + f).N_i.d_r$$

Sendo que f é a fração de aumento no tamanho da equipe. O volume total de trabalho efetivo realizado pela equipe aumentada é:

$$E_u = N_i.[(1 + f).d_r - f.d'_a]$$

Onde d'_a é chamado de tempo para assimilação efetiva e é igual a $d_a.(1 + m)$. O ganho adicional de esforço é dado pela expressão:

$$\begin{aligned} E_{gain} &= E_u - (N_i.d_r) \\ &= f.N_i.(r - a') \end{aligned}$$

Este resultado mostra que a adição de pessoas à equipe produz um ganho líquido de esforço com $f > 0$ e $d_r > d'_a$. Isto significa que, para cumprir o planejamento inicial do projeto, é necessário:

- Adicionar pessoas à equipe utilizando alguma estratégia de substituição de profissionais.
- Assegurar que o prazo restante necessário para realizar o trabalho é maior do que o tempo de assimilação.

O modelo provê uma forma de computar d'_a em termos da "dificuldade de assimilação", D, que depende de três fatores:

1. O esforço útil total produzido por um número específico de pessoas adicionadas ao projeto
2. O número máximo de pessoas que podem ser adicionadas ao projeto
3. Quão tarde no desenvolvimento do projeto as pessoas podem ser adicionadas e ainda gerar ganho líquido de esforço

O modelo também considera os seguintes parâmetros:

1. Fração da redução do trabalho efetivo durante o período compreendido entre a saída dos profissionais e a conclusão da assimilação dos novos membros da equipe ; e,
2. Esforço dispendido para contratar os profissionais que serão adicionados à equipe.

3.4 Notações e Convenções

Neste trabalho são utilizadas as seguintes notações referentes aos diversos parâmetros do projeto:

3.4.1 Duração, Esforço e Equipe

O modelo da Lei de Brooks utiliza os seguintes parâmetros de prazo:

- d_r = duração restante para finalizar o projeto.
- d_h = duração do período de substituição (contratação) dos novos trabalhadores.
- d_a = duração do período de assimilação.
- d_f = duração do período de máxima alocação de profissionais no projeto.

A duração medida por d_r se refere ao período compreendido entre a saída dos profissionais até a finalização do projeto. Assim, tem-se que $d_r = d_h + d_a + d_f$. A duração pode ser extendida para compensar perdas decorrentes da substituição dos trabalhadores. Quando necessário, distingue-se a duração original da nova duração utilizando $d_r(orig)$ e $d_r(novo)$ respectivamente.

Para remover o efeito de finais de semana e feriados, os prazos são medidos em dias úteis, e não em dias calendário [Stutzke 1994]. Todas as variáveis de esforço descritas a seguir são expressas em pessoas/dia.

- E_a = Esforço de assimilação.
- E_e = Esforço gasto total.
- E_u = Esforço útil realizado em tarefas do projeto.
- E_r = Esforço necessário para finalizar as tarefas do projeto.

O esforço necessário para completar o projeto, E_r, pode variar se os profissionais deixarem trabalho não concluído ou quando este precisa ser totalmente refeito. Para distinguir estas duas situações, definem-se os seguintes valores:

$$E_r(new) = E_r(orig) + E_{add}$$

onde:

- $E_r(orig)$ = esforço necessário antes da saída de profissionais da equipe.
- $E_r(new)$ = esforço necessário depois da saída dos profissionais.
- $E_(add)$ = esforço adicional necessário para repor trabalho perdido ou incompleto.

Na prática, sempre que acontece *turnover* na equipe, algum trabalho ou conhecimento são perdidos, assim $E_r(new)$ sempre será maior que $E_r(orig)$. Ao finalizar as tarefas, $E_r(new) = E_u$.

Com relação à equipe, são considerados os seguintes elementos de medição:

- N_i = tamanho original da equipe.
- f_l = fração de profissionais que deixaram a equipe original.
- f_a = fração de profissionais que ingressaram na equipe.

3.4.2 Coeficientes de Produtividade

Stutzke definiu um fator de *mentoring*, m, para representar a quantidade de esforço gasto pelos mentores para treinar cada novo trabalhador [Stutzke 1994]. O valor típico de m é 0,25. O esforço de *mentoring* envolve tanto os novos trabalhadores quanto seus mentores. Pressupõe-se que o *mentoring* ocorra de forma constante e contínua durante todo o período de assimilação, d_a, e seu respectivo esforço é dado pela fórmula:

$$N_i.f_a.d_a + N_i.f_a.m.d_a = f_a.N_i.a'$$

Onde a' é o tempo de assimilação efetiva, e é igual a $a(1+m)$

3.4.3 Taxa de Realização de Esforço

Os N trabalhadores realizam esforço efetivo para o projeto numa taxa R. Embora, na prática, o tempo diário efetivamente trabalhado seja menor que oito horas[1], o modelo pressupõe que cada profissional realiza este volume de esforço por dia de trabalho. Qualquer quantidade de horas extras faz com que a taxa de realização de esforço seja $R = N(1+OT)$.

O tamanho da equipe é medido em pessoas alocadas *full time* no projeto. O esforço é medido em pessoas-dia. A taxa de realizaçção de esforço é medida em pessoas-dia por dias úteis [Stutzke 1994]. Se for necessário considerar horas extras, deve-se substituir N por R nas equações.

O modelo assume que o esforço necessário, E_r, para completar o projeto[2], o tamanho da equipe, N_i, e a fração da equipe que foi perdida, f_l, são conhecidos, da mesma forma que parâmetros do projeto referem-se aos custos de contratação e *mentoring*.

3.4.4 Esforço de Contratação

O esforço efetivo total realizado pela equipe pode ser obtido descontando-se a fração de tempo destinado à contratação ou seleção de novos membros para o projeto. Este processo pode envolver desde uma simples consulta ao banco de dados de competências da empresa até a triagem de currículos, realização de entrevistas ou ainda, esperar que o profissional finalize outros projetos. Neste cálculo, considera-se apenas a parte do esforço realizado pela equipe do projeto, sendo ignorados os esforços despendidos pela equipe administrativa ou gerencial. Esta abordagem visa assegurar que apenas o esforço gasto pelo pessoal diretamente responsável pelo projeto seja computado. O esforço de contratação, $E_u(hire)$, pode ser expresso por:

$$E_u(hire) = N_i.d_h.(1 - f_l - f_a.h)$$

Para recuperar atrasos no cronograma, pode ser necessário adicionar um número maior de trabalhadores do que foi originalmente perdido. Esta é a razão pela qual distingue-se f_l de f_a.

[1] Vários eventos podem consumir tempo útil de trabalho, como por exemplo, ligações telefônica, idas ao banheiro e pausas para lanche.

[2] Este volume inclui algum esforço adicional para compesar retrabalho ou perda de conhecimento

3.4.5 Esforço de Assimilação

Durante o período de assimilação, os novos profissionais contribuem com pouco trabalho efetivo, mas consomem esforço que deve ser pago. Também é perdido o trabalho que os mentores poderiam ter realizado em tarefas do projeto caso não precisassem treinar os novos membros da equipe. Como os analistas que deixaram a equipe ainda não foram substituídos, o projeto continua a perder trabalho efetivo. Isto resulta em assimetria entre os valores de f_l e f_a nas equações que calculam o esforço de assimilação. O trabalho efetivo perdido depende apenas de f_l enquanto que o esforço total despendido depende de f_a e f_l. Dessa forma:

$$\begin{aligned} E_u(assim) &= N_i.d_a.(1 - f_l - f_a.m) \\ &= N_i.d_a.(1 - f_l + f_a) \end{aligned}$$

Estes resultados se verificam uma vez que os analistas que deixaram a equipe não debitam seus custos do orçamento do projeto e os novos profissionais passam a fazê-lo. Uma vez terminado o período de assimilação, a equipe passa a realizar uma quantidade de esforço efetivo que pode ser calculada pela seguinte equação:

$$\begin{aligned} E_e(full) &= E_u(full) \\ &= N_i.d_f.(1 - f_l + f_a) \end{aligned}$$

O tempo de assimilação pode ser reduzido mediante a utilização de programas efetivos de treinamento e de disseminação de conhecimento.

3.4.6 Equações Combinadas

Esforço normalizado é igual ao esforço realizado dividido por N_i, e sua unidade é dias úteis desde que N_i corresponda à taxa de realização de esforço efetivo. Ao somar as equações de cada fase, obtém-se o esforço total efetivo realizado pela equipe ao longo do projeto.

$$\begin{aligned} E'_u &= d_h.(1 - f_l - f_a.h) + d_a.(1 - f_l - f_a.m) + d_f.(1 - f_l + f_a) \\ &= d_r.(1 - f_l) + f_a.(-d_h.h - d_a.m + d_f)) \end{aligned}$$

Isto significa que o esforço efetivo é igual ao esforço realizado pela equipe reduzida menos o esforço para contratação e *mentoring* mais o esforço realizado pela nova equipe após o período de assimilação. Desde que $d_r = d_h + d_a + d_f$ os valores de E'_u são calculados pelas expressões:

$$\begin{aligned} E'_u &= d_r.(1 - f_l + f_a) - f_a.(d_h.(1 + h) + d_a.(1 + m)) \\ &= d_r.(1 + f_n) - f_a.(d'_h + d'_a) \end{aligned}$$

Onde:

- f_n = fração de incremento no tamanho da equipe = $f_a - f_l$

- d'_h = tempo de reposição de profissionais = $d_h.(1 + h)$
- d'_a = tempo efetivo de assimilação = $d_a.(1 + m)$

Somando-se as equações de cada fase, tem-se o esforço total normalizado gasto pelo projeto durante todo o período de recuperação (d_r):

$$\begin{aligned} E'_e &= d_h.(1 - f_l) + d_a.(1 - f_l + f_a) + d_f.(1 - f_l + f_a) \\ &= d_r.(1 - f_l) + f_a.(d_a + d_f) \end{aligned}$$

Isso significa que o esforço total despendido é igual ao esforço total da equipe reduzida mais o esforço que a nova equipe gastou durante e após o período de assimilação. Expandindo d_f temos:

$$\begin{aligned} E'_e &= d_r.(1 - f_l + f_a) - f_a.d_h \\ &= d_r.f_n - f_a.d_h \end{aligned}$$

3.5 Condições de Término do Projeto

Para finalizar as tarefas do projeto a equipe deve realizar, pelo menos, o esforço, $E_r(orig)$, que restava no orçamento no momento em que profissionais saíram do projeto. No entanto, o projeto pode incorrer em custos adicionais caso ocorram algumas situações peculiares, como por exemplo:

- O trabalho incompleto deve ser atribuído a outro analista, que levará algum tempo para aprender como finalizá-lo.
- O produto incompleto precisa ser descartado e o trabalho totalmente refeito.
- Saída de um profissional essencial ao projeto.

Seja qual for a situação, quando um profissional deixa parte do trabalho por fazer ao sair da equipe, o esforço adicional necessário para finalizá-lo deve ser estimado e adicionado ao $E_r(orig)$ para obter a nova previsão de conclusão do projeto. Entretanto, existe uma perda de esforço que é mais difícil de mensurar, que é o conhecimento especializado e a experiência de uma pessoa chave que deixa a equipe. Isto também representa custos adicionais para o projeto que devem ser adicionados a $E_r(orig)$. O esforço total necessário para finalizar o projeto, $E_r(new)$, após a ocorrência de turnover é $E_r(new) = E_r(orig) + E_{add}$ onde E_{add} denota o esforço necessário para suprir trabalho perdido ou incompleto. Para calcular a projeção de conclusão do trabalho, faz-se $E_r(new)$ igual a E_u, obtendo:

$$E'_r(new) = d_r.(1 + f_n) - f_a.(d'_h + d'_a)$$

3.5.1 Ponto de Equilíbrio

O ponto de equilíbrio, ou *break-even*, do projeto é o ponto em que o esforço produzido e o esforço realizado são iguais, ou seja, não há perda ou ganho líquido de esforço. O projeto tem

ganho líquido de esforço efetivo somente se d_r for grande o suficiente para permitir que a fase de assimilação seja concluída, o que é expresso por $E_{gain} = E_u(new) - E_u(orig) > 0$, em que o esforço útil é computado para o intervalo, d_r, tanto para a estimativa original quanto para a nova previsão.

Quando E_{gain} é igual a zero, tem-se o menor valor possível de d_r para obtenção do ponto de equilíbrio. Resolvendo as equações, obtém-se:

$$\begin{aligned} 0 &= d_r.(1 + f_n) - f_a.(d'_h + d'_a) - d_r \\ &= f_n.d_r - f_a.(d'_h + d'_a) - d_r \end{aligned}$$

Assim,

$$d_r(break - even) = f_a.(\frac{d'_h + d'_a}{f_a})$$

Considerar o ponto de equilíbrio faz sentido para o caso em que há aumento da equipe para acelerar o cronograma. No caso de *turnover*, o projeto incorre em custos adicionais, e assim, comparar a nova equipe com a original é irrelevante.

3.5.2 Cálculo da nova Duração

O modelo permite calcular tanto a data de finalização (duração), d_r, dada uma quantidade específica de aumento de pessoal, f_a, ou a quantidade de aumento de pessoal, f_a, necessária para alcançar uma data específica de conclusão, d_s.

Assumindo que d_h e d_a sejam conhecidos, pode-se resolver a equação (3) para $d_r(new)$, que é o prazo necessário para obter a nova quantidade de esforço necessário, o qual é igual a $E_r(old) + E_{add}$:

$$d_r = \frac{E'_r + f_a.(d'_h + d'_a)}{1 + f_n}$$

e

$$f_a = \frac{E'_r - (1 - f_n).d_s}{d'_s - d'_h - d'_a}$$

O que nos permite isolar R:

$$R = \frac{d_r(orig)}{d_r(orig) - d'_h - d'_a}$$

Onde o $d_r(new)$ denota esforço normalizado, definido como um esforço dividido pela taxa de realização de esforço N_i.

$$\begin{aligned} d_r(new) &= \frac{E'_r(new) + f_a.(d'_h + d'_a)}{1 + f_n} \\ &= \frac{E'_r(old) + E'_{add} + f_a.(d'_h + d'_a)}{1 + f_n} \\ &= \frac{d_r(old) + E'_{add} + f_a.(d'_h + d'_a)}{1 + f_n} \end{aligned}$$

Nota-se que $d_r(new)$ não é igual a $E'_r(new)$ porque o esforço normalizado, E' é obtido dividindo-se o esforço por N_i. Uma vez que N_i não é igual ao número médio de pessoas no restante no projeto, $d_r(new)$, esta normalização não fornece a duração atual. A duração anterior é $E'_r(orig)$, então a duração final é aumentada.

3.6 Recomposição da equipe

Há três possíveis estratégias de recomposição de equipe disponíveis para o gerente quando ele enfrenta *turnover* na equipe:

3.6.1 Substituição Exata

Se $f_l = f_a$ acontece a substituição do pessoal que deixou o projeto, e o tamanho da equipe passa a ser N_i trabalhadores. O valor de f_n é zero. Mesmo que o projeto não demande esforço adicional, ainda assim, d_r será maior que $d_r(orig)$ e as datas do cronograma serão adiadas. Esta abordagem envolve substituir os profissionais que deixaram o projeto pelo número exato de novos trabalhadores. Isso envolve aceitar que o novo cronograma do projeto tenha duração maior que o original. Esta política de recomposição da equipe é recomendada quando:

- Não existem restrições irreconciliáveis do cliente pela data de entrega.
- Profissionais adequados para se juntar à equipe estarão disponíveis apenas em momentos posteriores do processo. Por exemplo, o responsável pela arquitetura estará disponível apenas após um ou dois *sprints*.

O adiamento será tanto maior quanto for a necessidade de recompor trabalho e conhecimento.

3.6.2 Cronograma programado

Pode-se calcular o aumento da equipe, f_a, necessário para cumprir restrições de prazo, d_s:

$$f_a = \frac{E'_r(new) - (1 - f_l).ds}{ds - dh' - da'}$$

$E'_r(new)$ é igual a $d_r(orig) + E'_{add}$. Disto resulta a fração de aumento no tamanho da equipe necessária para compensar a perda de f_l profissionais e ainda assim cumprir a meta de duração

d_s. Como d_s se aproxima de $(d'_h + d'_a)$ pela direita, o aumento de profissionais requeridos se torna infinito.

Essa abordagem é adotada quando é necessário recompor a equipe com trabalhadores em número suficiente para cumprir a duração prevista originalmente. A nova duração será maior que a original porque existem custos decorrentes do esforço de contratação e assimilação. Esta estratégia é recomendada quando:

- Existem restrições irreconciliáveis do cliente com relação à data de entrega do projeto.
- Profissionais adequados para se juntar à equipe estarão disponíveis apenas em momentos posteriores do *pipeline*. Por exemplo, o responsável pela arquitetura estará disponível apenas após um ou dois *sprints*.
- O orçamento do projeto permite que sejam alocados profissionais mais experientes.

3.6.3 Cronograma variável

Pode-se calcular o aumento de pessoal necessário para completar o projeto sem adiamento das tarefas do cronograma. O resultado do tópico anterior pode ser usado se for definido que d_s seja igual a $d_r(orig)$, considerando que $d_r(orig) = E'_r(orig)$.

$$f_a(no\ slip) = \frac{E'_{add} + f_l.d_r(orig)}{d_r(orig) - d'_h - d'_a}$$

Se nenhum profissional for perdido, entao $f_l = 0$ e $E_{add} = 0$, dado que $f_a = 0$. Se $E_{add} = 0$, define-se o limite inferior de f_a necessário para recuperar o prazo final do cronograma original:

$$f_a(min) = \frac{f_l.d_r(orig)}{d_r(orig) - d'_h - d'_a} = f_l.RR \qquad = \frac{d_r(orig)}{d_r(orig) - d'_h - d'_a}$$

R é chamado de *fator de recuperação de cronograma*. O parâmetro $f_a(min)$ indica a fração mínima necessária de aumento da equipe para que o planejamento do cronograma seja mantido, se nenhum esforço adicional for necessário. A quantidade de aumento de pessoal necessária para manter o cronograma dada a perda dos trabalhadores é $f_a = f_l.R$ onde R é o fator de recuperação de cronograma.

Considerando um projeto que apresente valores de $d_r(orig) = 100$ dias úteis, $d'_h = 11$ dias úteis e $d_a = 25$ dias úteis, $R = 1,56$. Isso significa que é necessário adicionar 56% mais profissionais ao número original dos que saíram para compensar a perda.

Esta estratégia consiste em adicionar um número de trabalhadores que não produza exatamente o efeito de manter o cronograma origial, mas que também não produza atrasos irreconciliáveis. Ela é recomendada quando:

- As restrições do cliente quanto à data de entrega existem, mas podem ser negociadas sem gerar atritos desmesurados.
- Profissionais adequados para se juntar à equipe não estão imediatamente disponíveis para se juntar à equipe.

- O orçamento do projeto não permite que sejam alocados profissionais mais experientes que possam recuperar o trabalho perdido em menos tempo.

3.6.4 Custo do *Turnover*

O custo adicional em que o projeto incorre devido ao *turnover* é $E_e - E_r(orig)$, o que resulta em:

$$E_{turnover} = E_e - E_r(orig)$$

ou

$$E'_{turnover} = d_r(new).f_n - f_a.d_h - d_r(orig)$$

A fração de aumento de custo é:

$$F_{cost} = \frac{E_{turnover}}{E_r(orig)}$$

Isso resulta no seguinte resultado:

$$F_{cost} = \frac{d_r(new).f_n - f_a.d_h}{d_r(orig) - 1}$$

As variáveis f_a e $d_r(new)$ são acopladas. Pode-se eliminar tanto $d_r(new)$ quanto f_a fazendo as devidas substituições de variáveis, o que resulta em:

$$F_{cost} = -1 + \frac{f_n.(E'_{add} + d_r(orig) - f_a.(d'_h + d'_a)}{(1 + f_n).d_r(orig)}$$

3.7 Rotatividade nos Projetos

Neste estudo considera-se que os profissionais são alocados ou remanejados das equipes de projeto numa taxa constante. Um pressuposto para adotar o modelo de [Stutzke 1994] é assumir a existência de uma condição de estado estacionário, em que o número de trabalhadores que deixam a equipe do projeto é exatamente igual ao número de profissionais que serão selecionados e alocados para substitui-los. Sendo assim, em qualquer momento existem vários funcionários em pipelines para a contratação, assimilação e trabalho. Existem duas interpretações possíveis para o estado estacionário:

1. **Substituição reativa**: a contratação para realocação de trabalhadores substitutos começa após os funcionários atuais deixarem o projeto.
2. **Substituição proativa**: a contratação é mantida continuamente, mesmo que o projeto esteja com a equipe completa porque se sabe que alguns trabalhadores deixarão o projeto e não se deseja atrasos decorrentes de sua substituição.

Ao adotar a política de substituição reativa, o gerente de projetos deve estar ciente que a equipe sempre realizará menos esforço que o valor desejado pois a saída dos profissionais acontece numa taxa uniforme. Por outro lado, a substituição proativa exige que custos adicionais sejam feitos para contratar e treinar novos profissionais.

Dessa forma, o gerente de projetos gasta recursos financeiros adicionais para contratar e treinar novos trabalhadores antes que isso seja efetivamente necessário, o que é uma estratégia de prevenção de riscos. A substituição proativa é razoável em ambientes que empregam um grande número de trabalhadores com habilidades semelhantes.

Uma vez que o paradigma ágil preconiza que as equipes devem ser multidisciplinares [Cockburn 2002], isso pressupõe que as funções, atribuições e responsabilidades dos profissionais que compõem a equipe sejam análogas. Assim, a política de substituição proativa é a mais recomendada para lidar com a rotatividade dos profissionais em projetos que adotem princípios ágeis.

3.7.1 Abordagem e Definições

Seja T_p o percentual de *turnover* de cada projeto, definida como a fração da equipe que foi alterada[3] ao longo do prazo de duração do projeto e seja L a quantidade de esforço perdido por dia útil no projeto.

Para calcular L, o valor de T_p deve ser convertido na quantidade de dias úteis perdidos em função das alterações havidas na composição da equipe durante o desenvolvimento do projeto. Considerando 50 semanas de trabalho por ano e cinco dias úteis por semana, o valor de L é:

$$L = T_p . \left(\frac{365,25}{250^2} \right) = 0,0058 . T_p$$

Por exemplo, se a taxa de *turnover* do projeto for 10% por ano, então L é aproximadamente igual a 0,06% por dia útil. Assume-se que os efeitos da realização de horas-extra são equivalentes para todos os profissionais: experientes, novatos e mentores. Esta estratégia elimina a necessidade de adotar fatores diferentes para cada papel.

3.7.2 Pressupostos para substituição proativa

Neste trabalho, considera-se as taxas de realização de esforço para calcular as perdas de produtividade decorrentes da adoção da política de substituição proativa de profissionais na equipe. Seja R_u a taxa de esforço útil produzido pela equipe e R_u a taxa na qual o esforço é consumido[4].

Em qualquer momento do projeto, existem $N_i.L.d_a$ pessoas em processo de assimilação. Na equação anterior, o parâmetro d_a e não d'_a é utilizado, pois aquele é o tempo de aprendizagem real, e que determina quanto tempo um indivíduo permanece no *pipeline*. Estas pessoas contribuem com nenhum esforço útil para o projeto durante este período. Entretanto, seus mentores consomem esforço igual a $m.N_i.L.d_a$.

[3] Entende-se como alteração qualquer acréscimo ou redução de pessoas na equipe.
[4] R_e também é chamada de *burn rate*.

Como o *turnover* e o *mentoring* acontecem continuamente ao longo do projeto – considerando a política de substituição proativa –, a taxa de realização de esforço útil no projeto reduz-se devido à alocação de mentores e profissionais novatos.

3.7.3 Substituição Reativa

Para contabilizar os atrasos decorrentes de contratações, define-se seu respectivo atraso médio, d_h (em dias úteis). A redução da taxa de realização de esforço decorrente da existência de posições não preenchidas é $N_i.d_h.L$. Essas vagas ociosas também afetam a taxa com que o esforço é consumido, uma vez que não originam débitos no orçamento. Dessa forma, tem-se que:

$$\begin{aligned} R_e &= N_i.(1 + L.d_h) \\ R_u &= R_e - R_{hiring} - R_{assim} \\ &= N_i.(1 - L.d_h - L.d_h.h - L.d_a - L.d_a.m) \\ &= N_i.(1 - L.(d'_h - d'_a)) \end{aligned}$$

Onde $d'_h = d_h.(1+h)$. A Fração de Perda de Produtividade, FLP, pode ser calculada com relação ao esforço gasto, R_e ou com relação à taxa planejada de realização de esforço, $R_i = N_i$

$$\begin{aligned} FLP_{reativo,\,gasto} &= \frac{R_e - R_u}{R_e} \\ &= \frac{L.(d_h.h + d'_a)}{1 - L.d_h} \\ &\approx L.[d_h.h + d'_a + L.d_h.(L.d_h + L.d'_a) + \dots \\ FLP_{reativo,\,planejado} &= \frac{R_i - R_u}{R_i} \\ &= L.(d'_h + d'_a) \end{aligned}$$

Como exemplo, assuma-se uma taxa de *turnover*, T_p, anual de 10%, com $L = 0.06\%$ por dia útil. Sejam também, $h = 0,1$, $m = 0,25$, $d_a = 40$ dias úteis e $d_h = 10$ dias úteis. Obtém-se $a' = 50$ e $d'_h = 11$ dias úteis. Com base nestes dados, $FLP_{reativo,\,gasto} = 3,06\%$ e $FLP_{reativo,\,planejado} = 3,66\%$. Essa redução se refere ao aumento de custo e atraso do cronograma.

Grosseiramente falando, a perda de produtividade é de cerca de um quarto da taxa de *turnover*. Abdel-Hamid e Madnick [Abdel-Hamid e Madnick 1991] citam estudos que reportam taxas de *turnover* de 25%, 30% e até 34%. Se uma taxa de turnover de 30% for utilizada, os valores acima serão triplicados, resultando numa redução de 9% na produtividade.

É importante substituir rapidamente os profissionais que deixaram o projeto por outros com competências e habilidades semelhantes. Se o *turnover* for de 10% e o tempo de contratação aumentar de 10 para 40 dias úteis, a perda de produtividade dobra, atingindo cerca de 6%.

3.7.4 Substituição Proativa

Neste caso, despende-se esforço adicional com contratação antecipada, de tal forma que não haja perda de durante esta fase. No entanto, a perda decorrente da fase de assimilação não deixa de acontecer.

$$\begin{aligned} R_e &= N_i.(1 + L.d_h.h) \\ R_u &= R_e - R_{hiring} - R_{assim} \\ &= N_i.(1 - L.d_a - L.d_a.m) \\ &= N_i.(1 - L.d'_a) \end{aligned}$$

A partir daí, pode-se calcular a FLP, fazendo:

$$\begin{aligned} FLP_{proativo,\,gasto} &= \frac{R_e - R_u}{R_e} \\ &= \frac{L.(d_h.h + d'_a)}{1 + L.d_h.h} \\ &\approx L.[d_h.h + d'_a - L.d_h.h.(d_h.h + d'_a) + \ldots \\ FLP_{proativo,\,planejado} &= \frac{R_i - R_u}{R_i} \\ &= L.d'_a \end{aligned}$$

Para exemplificar, assumam-se os mesmos valores utilizados no cálculo da FLP para a política de substituição reativa. Neste caso, $FLP_{proativo,\,gasto} = 3,06\%$ e $FLP_{proativo,\,planejado} = 3,00\%$. Comparada à política de substituição reativa, a substituição proativa resulta em perda menor de produtividade, conforme esperado. Assim, a substituição proativa é a política mais adequada, pelo menos de acordo com os pressupostos do modelo.

3.8 Discussão

A rotatividade de profissionais reduz a produtividade média da equipe por um valor constante. A produtividade aparente das duas políticas de pessoal são essencialmente idênticas. No entanto, comparada com a política de substituição reativa, os resultados da política de substituição proativas mostram diminuição menor da produtividade o que significa que a probabilidade do projeto atingir as metas estabelecidas sejam maiores nesta política. Os custos reais das duas políticas também é diferente, sendo que a substituição pró-ativa custa a $L.d'_h$ mais que a reativa. Uma vez que a substituição pró-ativa reduz a quantidade de adiamentos no cronograma, ela é mais adequada dadas as nossas hipóteses, mesmo que seus custos sejam ligeiramente maiores. No próximo capítulo é apresentado o estudo experimental realizado para identificar quantitativamente os efeitos do *turnover* na produtividade de projetos de *software*.

Capítulo 4

Estudo Experimental

Este capítulo apresenta o estudo experimental realizado para cumprir o objetivo geral da pesquisa, que é investigar o impacto do *turnover* na produtividade de projetos que utilizam RUP e processos híbridos conforme abordagens descritas nas seções 2.1 e 2.7, respectivamente.

4.1 Modelo de Medição

Para responder à questão de pesquisa, foi criado um modelo conceitual de medição que congrega as variáveis que interferem no cálculo do *turnover* e da fração de produtividade perdida em decorrência dele. Medir a fração de produtividade perdida em função do *turnover* não é tarefa trivial, pois há vários fatores propostos na literatura que interferem no cálculo da produtividade e do *turnover* [Wagner e Ruhe 2008].

Para balizar a construção do modelo de informações da pesquisa foram utilizadas as meta-análises feitas por [Cunha et al. 2008] e [Stutzke 1995], em que os principais grupos de fatores que influenciam a produtividade e turnover são identificados: tecnologias, ferramentas, requisitos, pessoas, processos, tempo de assimilação, tempo de contratação, tempo de *mentoring* e produto.

As informações do modelo conceitual podem ser utilizadas em trabalhos futuros para subsidiar a criação de sistemas de informações de gerenciamento de projetos, que envolvam, por exemplo:

- Ferramentas automatizadas para elaboração de cronogramas e planos de projeto integrados ao sistema de gerenciamento de configuração.
- Sistema de coleta, distribuição de informações ou integrações com outros sistemas de apoio organizacional.
- Painel dinâmico de monitoramento dos fatores de exposição ao risco do projeto, índices de qualidade e outros indicadores de desempenho.
- Mecanismo de apoio à estimativa de custos do projeto com base em dados históricos referentes ao *turnover* e seus efeitos.

As meta-informações foram agrupadas em projeto, requisitos, profissionais, risco e *bugs*, e com base nelas foi elaborado um diagrama de metaclasses UML com os elementos mais relevantes para a análise dos efeitos do *turnover* na produtividade. Este diagrama é ilustrado pela figura 4.1.

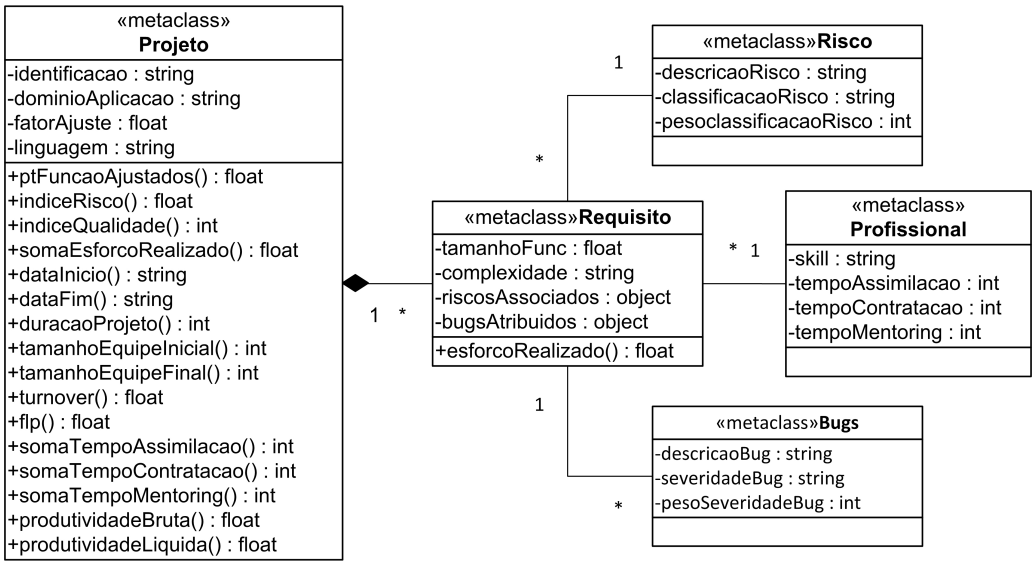

Figura 4.1: Modelo conceitual dos parâmetros relacionados à produtividade e *turnover*

Como o entendimento do modelo de medição é essencial para o desenvolvimento da pesquisa os próximos tópicos descrevem as metaclasses e seus respectivos meta-atributos e operações.

4.1.1 Projeto

A meta-classe `Projeto` contém os meta-atributos e operações que caracterizam um projeto, o qual deve ser entendido como um esforço temporário empreendido para criar o *software* contratado pelo cliente [PMI 2009]. As seguintes informações são utilizadas para caracterizar um projeto:

- `identificacao`: identificador único que referencia os projetos.
- `dominioAplicacao`: nicho de mercado atendido pelo *software* desenvolvido pelo projeto, por exemplo, financeiro, comércio eletrônico e infraestrutura.
- `fatorAjuste`: índice que ajusta o número de pontos de função brutos aos requisitos não funcionais do projeto [IFPUG 2000].
- `linguagem`: linguagem de programação na qual o *software* foi desenvolvido.
- `ptFuncaoAjustados`: tamanho funcional do projeto expresso em pontos de função.
- `indiceRisco`: índice de exposição ao risco do projeto, calculado com base na classificação dos riscos do projeto.
- `indiceQualidade`: índice de qualidade do projeto, calculado com base na quantidade e severidade dos *bugs* identificados no projeto.
- `somaEsforcoRealizado`: esforço, em $homens.horas^{-1}$, efetivamente gasto para desenvolver o projeto.
- `dataInicio`: data em que as atividades do projeto foram iniciadas.
- `dataFim`: data de finalização das atividades do projeto.
- `duracaoProjeto`: duração do projeto em dias úteis, contados desde `dataInicio` até `dataFim`.
- `tamanhoEquipeInicial`: quantidade de pessoas da equipe que iniciou o projeto.

- `tamanhoEquipeFinal`: quantidade de pessoas da equipe que finalizou o projeto.
- *`turnover`*: taxa de rotatividade de pessoas da equipe, considerando tanto aquelas que deixaram a empresa quando as que foram alocadas em outros projetos.
- `flp`: a FLP, ou *fractional loss of productivity*, corresponde ao percentual da produtividade do projeto que é perdido em função do *turnover* [Stutzke 1994].
- `somaTempoAssimilacao`: soma da quantidade de dias úteis necessários para os novos membros da equipe assimilar as informações necessárias para desenvolver suas atividades [Stutzke 1994].
- `somaTempoContratacao`: soma da quantidade de dias úteis necessários para substituir um profissional que deixou a equipe [Stutzke 1994].
- `somaTempoMentoring`: soma da quantidade de dias úteis que os atuais membros da equipe gastaram para treinar os novos profissionais [Stutzke 1994].
- `produtividadeBruta`: expressa a velocidade de desenvolvimento do projeto, em $hh.pf^{-1}$, equivalendo à razão do tamanho funcional pelo esforço gasto.
- `produtividadeLiquida`: expressa a velocidade de desenvolvimento do projeto, em $hh.pf^{-1}$, levando em consideração a FLP.

4.1.2 Requisito

A meta-classe `Requisito` contém os meta-atributos e operações que caracterizam os requisitos funcionais e não-funcionais do projeto. Os requisitos correspondem a uma função, restrição ou outra propriedade que precisa ser fornecida, encontrada ou atendida para satisfazer às necessidades do usuário do futuro sistema [IEEE 1990, Pressman 2011, Sommerville 2011]. Eles são enunciados em lingugem natural, de forma descritiva. O modelo de medição considera, indistintamente, tanto requisitos funcionais quanto requisitos não funcionais.

- **Requisitos Funcionais**: são declarações de funções ou tarefas que deverão ser disponibilizadas pelo sistema, envolvendo tanto a reação à entradas específicas quanto o comportanto esperado diante destas entradas [Pressman 2011, Sommerville 2011].
  Exemplo: *O sistema deverá disponibilizar relatórios de vendas de todas as filiais, agrupando as informações por mês, quinzena e semana.*
- **Requisitos Não Funcionais**: são os requisitos relacionados ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenibilidade e tecnologias envolvidas. Em geral, requisitos não-funcionais são restrições aplicáveis aos requisitos funcionais e se referem a parâmetros de qualidade e/ou peculiaridades que os serviços ou funções do sistema devem possuir.
  Exemplo: *O sistema deverá ser capaz de processar pelo menos 40 transações por segundo em horários de pico.*

As seguintes informações são utilizadas para caracterizar os requisitos que compõem o projeto:

- `tamanhoFunc`: tamanho funcional do requisito, calculado de acordo com o manual de contagem da IFPUG versão 4.2.1 e expresso em pontos de função não ajustados.

- `complexidade`: denota o nível de complexidade do requisito. Este valor é calculado pelo algoritmo de análise de pontos de função. Pode assumir os valores: `Alta`, `Média` e `Baixa`.
- `fatorRisco`: corresponde ao nível de risco atribuído ao requisito, podendo assumir os valores: `Alto`, `Médio` e `Baixo`.
- `bugsAssociados`: lista de *bugs* associados ao requisito.
- `esforcoRealizado`: horas efetivamente gastas para desenvolver o requisito.

4.1.3 Profissional

A meta-classe `Profissional` modela as informações referentes aos membros da equipe do projeto. Embora os papéis e responsabilidades específicas para os membros da equipe do projeto sejam designados, neste trabalho as informações relevantes dizem respeito, sobretudo, à duração dos esforços em assimilação, contratação e *mentoring* [Stutzke 1995]. As informações utilizadas para caracterizar os esforços relevantes para a pesquisa feitos pelos profissionais são descritas a seguir:

- `skill`: representa o nível de habilidade do profissional. Pode assumir os valores `Júnior`, `Pleno` e `Sênior`.
- `tempoAssimilacao`: dias úteis gastos pelo profissional para assimilar as informações necessárias ao desenvolvimento de suas atividades.
- `tempoContratacao`: dias úteis necessários que o profissional demorou para substituir alguém que deixou a equipe.
- `tempoMentoring`: tempo que o profissional dedicou para treinar ou acompanhar os novos membros da equipe.

4.1.4 Risco

Esta meta-classe abstrai as informações referentes aos riscos que ameaçam o projeto e suas respectivas qualificação e quantificação de impacto. Um risco é um evento de ocorrência incerta que possui potencial para causar efeitos nocivos ao projeto [PMI 2009]. As informações utilizadas para calcular o nível de exposição ao risco, no final do projeto, são descritas a seguir:

- `descricaoRisco`: descrição de alto nível do risco identificado.
- `classificacaoRisco`: classificação atribuída ao risco. Pode assumir os valores `Baixo`, `Médio` e `Alto`.
- `pesoClassificacaoRisco`: valor que representa o peso do risco de acordo com sua classificação, de acordo com a seguinte conveção:
  - $classificacao(risco) = \texttt{Baixo} \Rightarrow pesoClassificacao(risco) = 1$
  - $classificacao(risco) = \texttt{Médio} \Rightarrow pesoClassificacao(risco) = 2$
  - $classificacao(risco) = \texttt{Alto} \Rightarrow pesoClassificacao(risco) = 3$

4.1.5 *Bug*

A meta-classe `bug` trata as informações referentes aos *bugs* associados aos requisitos do projeto. Um *bug* é um erro no funcionamento do *software* produzido pelo projeto [IEEE 2004]. A quantidade e/ou severidade dos *bugs* encontrados é diretamente proporcional à percepção da qualidade da aplicação. As informações utilizadas para calcular o índice de qualidade associado aos *bugs* são descritas a seguir:

- `descricaoBug`: descrição em alto nível do *bug* associado ao requisito.
- `severidadeBug`: severidade atribuída ao *bug* em função dos impactos que podem ocasionar. Pode assumir os valores `Baixo`, `Médio` e `Alto`.
- `pesoSeveridadeBug`: valor que representa o peso do *bug* de acordo com sua severidade, respeitando a seguinte conveção:
  - $classificacao(bug) = \texttt{Baixo} \Rightarrow pesoSeveridade(risco) = \texttt{1}$
  - $classificacao(bug) = \texttt{Médio} \Rightarrow pesoSeveridade(risco) = \texttt{2}$
  - $classificacao(bug) = \texttt{Alto} \Rightarrow pesoSeveridade(risco) = \texttt{3}$

4.2 Dados Consolidados

Foram utilizados dados de 27 projetos desenvolvidos ao londo de três anos, entre julho de 2008 e julho de 2011, sendo que 10 utilizaram RUP e 17 utilizaram o processo híbrido. Os projetos foram selecionados por terem sido totalmente desenvolvidos durante o período considerado.

Algumas observações referentes às características da amostragem são pertinentes e necessárias ao entendimento e garantia de validade do estudo de caso:

- Apesar do tamanho da amostra de projetos híbridos ser 70% maior que projetos RUP, a diferença da média do tamanho funcional é de 24,47% e do esforço de desenvolvimento é 10,09%.
- Os projetos RUP foram desenvolvidos entre Julho de 2008 e Março de 2010 e os projetos hídridos aconteceram desde Setembro de 2008 a Julho de 2011. A partir de Março de 2010 todos os projetos passaram a ser desenvolvidos de acordo com o processo híbrido.
- Entre Setembro de 2008 até Março de 2010 – aproxidamente 18 meses – houve desenvolvimento simultâneo de projetos RUP e híbridos, o que é relevante para a pesquisa porque neste período de transição os projetos compartilharam o mesmo *pool* de profissionais.
- Apesar de alguns projetos apresentarem desvios significativos de produtividade em relação à média, eles não foram descartados do estudo para não interferir no resultado da análise estatística das variáveis.

Para calcular e analisar a correlação dos indicadores com o percentual de produtividade perdida em função do *turnover* os dados do modelo de medição foram consolidados na tabela 4.1. Estes dados serão utilizados tanto na análise estatística quanto na análise comparativa descritas na seção 4.3.

Tabela 4.1: Sumarização dos dados dos Projetos Analisados

identificacao	*duracaoProjeto*	*esforcoRealizado*	*linguagem*	*numeroRequisitos*	*ptFuncaoAjustados*	*indiceRisco*	*indiceQualidade*	*turnover*	*prodPerdida*	*prodBruta*	*prodLiquida*	*trocaSeniores*	*dominioSolucao*
R_1	65	2220	JEE	39	148,7	1,98	2,01	12	4,4	14,92	15,57	1	CE
R_2	52	1180	JEE	27	104	1,96	1,77	22	7,27	11,34	12,16	2	FN
R_3	131	3511	.NET	60	300,9	1,15	1,98	31	3,87	11,66	12,11	1	FN
R_4	76	1919	JEE	28	148,7	1,67	2,06	10	3,55	12,9	13,35	1	TC
R_5	72	1880	.NET	29	145,7	1,81	1,92	11	3,75	12,9	13,38	1	FN
R_6	77	1876	C++	21	105,2	2,06	2	23	7,58	17,83	19,18	2	TC
R_7	86	2188	C++	24	145,9	1,98	1,93	22	7,35	14,99	16,09	1	FN
R_8	67	2069	JEE	34	160,1	2	1,92	17	4,25	12,92	13,46	2	IE
R_9	148	3546	C++	24	125,1	1,97	1,88	22	5,5	28,34	29,89	2	IE
R_{10}	73	1043	JEE	23	112,8	2,08	2,03	22	7,39	9,24	9,92	1	TC
H_1	65	858	JEE	18	84	2,12	1,95	31	10,06	10,21	11,23	3	CE
H_2	67	1372	JEE	25	115,6	2,24	2,02	42	13,36	11,86	13,44	1	TC
H_3	76	1329	.NET	24	125,4	2,34	1,93	35	11,42	10,59	11,79	2	CE
H_4	98	1944	JEE	40	175,7	1,81	2,15	34	11,11	11,06	12,28	2	CE
H_5	61	1881	JEE	32	149,4	2,45	2,06	26	13,05	12,59	14,23	2	TC
H_6	109	2138	JEE	40	231,6	2,01	2,03	41	16,64	9,23	10,76	3	CE
H_7	97	2666	JEE	48	236,5	2	1,91	42	12,85	11,27	12,71	2	FN
H_8	102	1558	JEE	28	144,4	1,93	2,2	35	13,05	10,78	12,18	3	CE
H_9	102	950	JEE	12	71,2	2,08	1,96	36	11,34	13,34	14,85	3	FN
H_{10}	80	3192	JEE	34	206,4	2,5	1,5	42	10,64	15,46	17,1	4	FN
H_{11}	106	2449	JEE	27	176	2,87	2,33	35	10,03	13,91	15,3	2	TC
H_{12}	108	2562	JEE	28	188,6	1,13	2	42	10,5	13,58	15	4	TC
H_{13}	93	1254	.NET	18	130,5	2,75	2,5	28	13,52	9,6	10,89	1	VJ
H_{14}	106	3846	JEE	43	303,8	2,11	2,25	31	12,45	12,65	14,22	2	VJ
H_{15}	177	4897	JEE	54	364,5	2,87	1,87	26	11,23	13,43	14,93	4	TC
H_{16}	131	3842	JEE	43	298,3	2,89	2	20	13,45	12,87	14,6	3	CE
H_{17}	111	3375	C++	27	166,1	2,19	2,25	43	11,6	20,31	22,66	3	IE

Tabela 4.2: Identificação do Projeto ($identificacao$)

Definição	$identificacao(prj) :: \{R_n, H_m\}$
Descrição	Projetos identificados com R_n foram desenvolvidos usando RUP e aqueles com H_m seguiram o processo hídrido.
Cálculo	Consulta Simples.
Exemplo	$identificacao(R_1) = R_1$

Tabela 4.3: Duração do Projeto ($duracaoProjeto$)

Definição	$duracaoProjeto(prj) :: dias\ uteis$
Descrição	Duração do projeto, em dias úteis, incluindo todas as fases do ciclo de vida, seja RUP ou híbrido.
Cálculo	$duracaoProjeto(prj) = dataFim(prj) - dataInicio(prj)$
Exemplo	$duracaoProjeto(R_2) = 20/12/2008 - 10/10/2008 = 52\ dias\ uteis$

Tabela 4.4: Esforço Realizado no Projeto ($esforcoRealizado$)

Definição	$esforcoRealizado(prj) :: homem.hora^{-1}(hh)$
Descrição	Volume total de horas de esforço realizado para desenvolver o projeto. Corresponde ao somatório do esforço gasto nos requisitos.
Cálculo	$$esforcoRealizado(prj) = \sum_{req=1}^{n} esforcoRealizado(req_n)$$
Exemplo	$esforcoRealizado(R_4) = 1919\ hh$

Tabela 4.5: Linguagem ($linguagem$)

Definição	$linguagem(prj) :: \{\texttt{JEE}, \texttt{.NET}, \texttt{C++}\}$
Descrição	Linguagem em que o sistema foi majoritariamente codificado.
Cálculo	Consulta Simples.
Exemplo	$linguagem(R_1) = \texttt{JEE}$

Tabela 4.6: Número de Requisitos ($numeroRequisitos$)

Definição	$numeroRequisitos(prj) :: numero\ inteiro$
Descrição	Quantidade total de requisitos funcionais e não funcionais do projeto, de acordo com a definição descrita na subseção4.1.2
Cálculo	$$numeroRequisitos(prj) = \texttt{count}(req[1..n])$$
Exemplo	$numeroRequisitos(R_4) = 28$

Tabela 4.7: Fator de Ajuste ($fatorAjuste$)

Definição	$fatorAjuste(prj) :: numero\,real$
Descrição	Valor utilizado para ajustar o tamanho funcional do *software* às suas características não funcionais.
Cálculo	$$fatorAjuste(prj) = 0,65 + \sum_{c=1}^{14} NI_c.0,01$$ onde: NI_c: Nível de Influência das características gerais do sistema segundo o manual de contagem da IFPUG.
Exemplo	$fatorAjuste(R_2) = 1,04$

Tabela 4.8: Pontos de Função Ajustados ($ptFuncaoAjustados$)

Definição	$ptFuncaoAjustados(prj) :: numero\,real$
Descrição	Pontos de função ajustados dos requisitos que compõe o projeto.
Cálculo	$$ptFuncaoAjustados(prj) = \sum_{req=1}^{n} tamanhoFunc(req_n).fatorAjuste(prj)$$
Exemplo	$ptFuncaoAjustados(R_4) = 169$

Tabela 4.9: Índice de Risco ($indiceRisco$)

Definição	$indiceRisco(prj) :: numero\,real$
Descrição	Quantificação do risco global do projeto. Corresponde à média aritmética do fator de risco dos requisitos. O impacto dos riscos é avaliado de acordo com os valores `baixo`, `médio` ou `alto`, sendo que cada valor possui um peso associado a ele.
Cálculo	$$indiceRisco(prj) = \left(\sum_{req=1}^{n} fatorRisco(risco(req_n)) \right) .\frac{1}{n}$$. sendo que: Se $risco(req) = baixo \Rightarrow fatorRisco(req) = 1$ Se $risco(req) = medio \Rightarrow fatorRisco(req) = 2$ Se $risco(req) = alto \Rightarrow fatorRisco(req) = 3$
Exemplo	$indiceRisco(R_3) = 1,15$

Tabela 4.10: Índice de Qualidade (*indiceQualidade*)

Definição	$indiceQualidade(prj) :: numero\,real$
Descrição	Quantificação da qualidade global do projeto. Corresponde à média aritmética do índice de qualidade dos requisitos. Os *bugs* associados a cada requisito são classificados de acordo com os valores `baixo`, `médio` ou `alto`, sendo que cada valor possui um peso associado a ele.
Cálculo	$$indiceQualidade(prj) = \left(\sum_{req=1}^{n} \sum_{bug=1}^{m} peso(bug_m).\frac{1}{m} \right).\frac{1}{n}$$. sendo que: Se $bug(req) = baixo \Rightarrow peso(baixo) = 1$ Se $bug(req) = medio \Rightarrow peso(medio) = 2$ Se $bug(req) = alto \Rightarrow peso(alto) = 3$
Exemplo	$indiceQualidade(R_3) = 1,98$

Tabela 4.11: Turnover (*turnover*)

Definição	$turnover(prj) :: numero\,real$
Descrição	Taxa de rotatividade da equipe do projeto, considerando os profissionais que deixaram a empresa ou foram alocados em outros projetos.
Cálculo	$$turnover = \left(\frac{D}{N_i + N_f} \right).100$$ onde: D = número de profissionais que deixaram a equipe. N_i = tamanho da equipe no início do projeto. N_f = tamanho da equipe no final do projeto.
Exemplo	$turnover(H_2) = 0,42$

Tabela 4.12: Percentual de Produtividade Perdida (*prodPerdida*)

Definição	$prodPerdida(prj) :: numero\,real$
Descrição	Percental da produtividade que foi perdida em função do *turnover*.
Cálculo	Calculado de acordo com o modelo de Stutzke descrito no capítulo 3.
Exemplo	$prodPerdida(R_4) = 3,55\%$

Tabela 4.13: Produtividade Bruta do Projeto (*prodBruta*)

Definição	$prodBruta(prj) :: horas.(pontos\ funcao)^{-1}$
Descrição	Produtividade do projeto considerando o esforço realizado e a quantidade de pontos de função ajustados.
Cálculo	$prodBruta(prj) = \frac{somaEsforcoRealizado}{ptFuncaoAjustados}$
Exemplo	$produtividadeBruta(prj) = (R_4) = 12,9\ hs/pf$

Tabela 4.14: Produtividade Líquida do Projeto (*prodLiquida*)

Definição	$prodLiquida(prj) :: horas.(pontos\ funcao)^{-1}$
Descrição	Produtividade do projeto considerando o esforço realizado e a quantidade de pontos de função ajustados.
Cálculo	$prodBruta(prj) = \frac{somaEsforcoRealizado}{ptFuncaoAjustados} + prodPerdida$
Exemplo	$prodLiquida(prj) = (R_4) = 13,35\ hs/pf$

Tabela 4.15: Domínio da Aplicação (*dominioAplicacao*)

Definição	$dominioAplicacao(prj) :: \{\texttt{CE}, \texttt{FN}, \texttt{TC}, \texttt{IE}, \texttt{VJ}\}$
Descrição	Domínio de negócio atendido pela implementação do sistema.
Cálculo	Consulta Simples. sendo que: `CE` = Conteúdo e e-commerce `FN` = Financeiro `TC` = Telecomunicação `IE` = Infraestrutura `VJ` = Varejo
Exemplo	$dominioAplicacao(R_1) = \texttt{CE}$

Tabela 4.16: Troca de Sêniores na Equipe (*trocaSeniores*)

Definição	$trocaSeniores(prj) :: numero\ inteiro$
Descrição	Quantidade de trocas de profissionais sêniores havidas ao longo do projeto.
Cálculo	Consulta Simples.
Exemplo	$trocaSeniores(R_2) = 2$

4.3 Análise dos Indicadores

Esta seção apresenta a análise dos indicadores – derivados do modelo conceitual apresentado na seção 4.1 – com relação às suas características e desempenho. Os valores dos indicadores são analisados para determinar sua correlação em relação ao percentual de produtividade perdido em função do *turnover*.

A análise estatística foi feita utilizando-se os testes detalhados na seção 2.6. Após calcular os coeficientes de assimetria e curtose das variáveis, calcula-se a correlaçao daquelas que passarem no teste de normalidade com a variável *prodPerdida*. A análise comparativa foi feita agrupando-se os projetos de acordo com o processo de desenvolvimento utilizado – RUP ou híbrido –, e comparando o comportamento dos valores de *prodPerdida* e *prodLiquida* em relação às variáveis dos indicadores cujos dados tenham passado no teste de normalidade.

É imporante ressaltar que variações significativas nos valores das variáveis nem sempre foram descartadas porque estes pontos podem lançar luz ao entendimento de aspectos que interfiram na produtividade e que esteja além dos limites do escopo da pesquisa. Nestes casos, serão feitos apontamentos para que não se perca a riqueza das informações coletadas, mesmo que não sejam exaustivamente tratadas.

A comparação objetiva dos projetos foi feita com base em dados estatisticamente significativos para assegurar o nível de significância dos resultados encontrados. As próximas seções apresentam os resultados das análises estatística e comparativa realizados para identificar a correlação dos indicadores com a perda de produtividade decorrente do *turnover*.

4.3.1 Produto

As características e o desempenho do desenvolvimento do produto foram medidos qualitativamente pelo domínio da aplicação e quantitativamente pelo número de requisitos e tamanho funcional. Para isso, as variáveis do modelo de medição consideradas foram domínio da solução (*dominioSolucao*), número de requisitos (*numeroRequisitos*) e pontos de função ajustados ($ptFuncaoAjustados$).

Análise Estatística

Considerando os projetos $\{R_1 \dots R_{10}\}$ e $\{H_1 \dots H_{17}\}$, o teste de normalidade das distribuições de dados destas variáveis resultou nos seguintes valores:

- P-Value para *numeroRequisitos*= 0,077
- P-Value para *dominioSolucao*= 0,010
- P-Value para $ptFuncaoAjustados < 0{,}005$

Os valores da média e desvio padrão de *numeroRequisitos* são, respectivamente, 31,48 e 11,28, considerando o conjunto de projetos $\{R_1 \dots R_{10}\}$ e $\{H_1 \dots H_{17}\}$. A figura 4.2 ilustra a distribuição de frequências de *numeroRequisitos*.

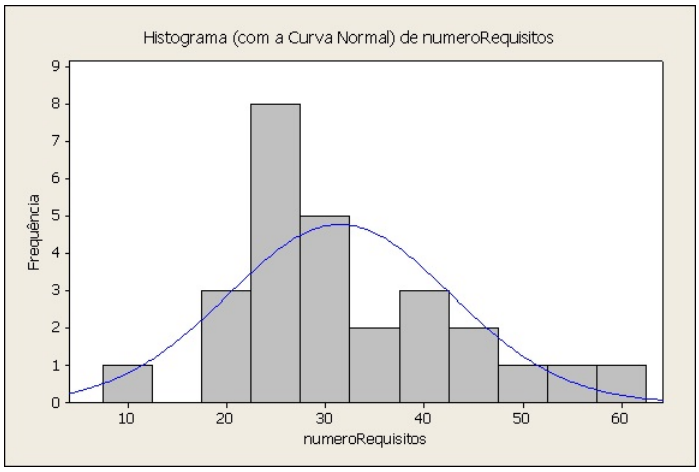

Figura 4.2: Histograma da distribuição de frequências de *numeroRequisitos*

Os valores das medidas de assimetria e curtose do número de requisitos do projeto, respectivamente, 0,81 e 0,45. O valor do coeficiente de correlação de Pearson para *prodPerdida* e *numeroRequisitos* é 0,005 e $P - Value = 0,981$.

Para calcular as medidas de tendência, dispersão e testar a normalidade da distribuição de dados de domínio de aplicação, os valores `Conteúdo e e-commerce`, `Financeiro`, `Telecomunicação`, `Infraestrutura` e `Varejo` da coluna *dominioSolucao* , da tabela 4.1 foram substituídos, respectivamente pelos valores numéricos `1`, `2`, `3`, `4` e `5`. A figura 4.3 mostra o histograma da distribuição de frequências de *dominioSolucao*.

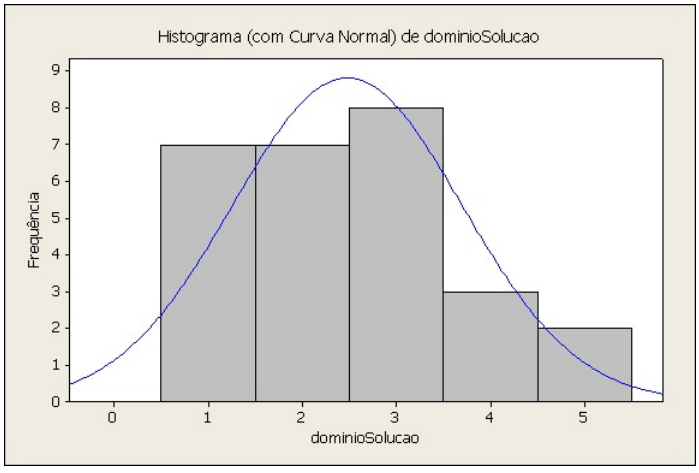

Figura 4.3: Histograma da distribuição de frequências de *dominioSolucao*

Os valores das medidas de assimetria e curtose de *dominioSolucao* são, respectivamente, 0,46 e -0,52. O valor do coeficiente de correlação de Pearson para *prodPerdida* e *dominioSolucao* é 0,005 e $P-Value = 0,010$. O coeficiente de correlação entre *prodPerdida* e *ptFuncaoAjustados* não foi calculado porque ele não é estatisticamente significativo pois a distribuição de dados de *ptFuncaoAjustados* não passou no teste de normalidade, pois $P - Value < 0,005$.

Análise Comparativa

O conjunto de projetos $\{H_{13}, H_{14}\}$ do domínio `Varejo` foi excluído desta comparação porque nenhum projeto deste segmento foi desenvolvido utilizando RUP. O percentual médio de perda de produtividade dos projetos RUP e híbridos foi calculado para cada domínio de aplicação e resultou nos seguintes valores:

- Projetos RUP
  - Conteúdo e *e-commerce*: perda média de 4,40% na produtividade
  - Financeiro: perda média de 5,56% na produtividade
  - Infraestrutura: perda média de 4,88% na produtividade
  - Telecomunicação: perda média de 6,17% na produtividade
- Projetos Híbridos
  - Conteúdo e *e-commerce*: perda média de 12,62% na produtividade
  - Financeiro: perda média de 11,61% na produtividade
  - Infraestrutura: perda média de 11,60% na produtividade
  - Telecomunicação: perda média de 11,63% na produtividade

Ao analisar os resultados anteriores constata-se que:

- Houve perda de produtividade substancial para os projetos de todos os domínios de aplicação. A produtividade média perdida em função do *turnover* para projetos RUP foi de 5,25%, enquanto que, para projetos híbridos esta perda foi de 11,86%.
- Esta variação equivale a uma redução de 11,95% da produtividade em projetos híbridos, referentes à média de perda de produtividade dos domínios de aplicação considerados.

Além disso, a análise do valor do desvio padrão dos agrupamentos por domínio de aplicação sugere que a perda de produtividade dos projetos híbridos seja mais homogênea – e por conseguinte, mais previsível – que aquela encontrada no conjunto de projetos RUP, por apresentarem valores 0,75 e 0,50 respectivamente.

4.3.2 Pessoas

Este fator é medido utilizado como referencial o *turnover*, a produtividade líquida e a quantidade de troca de sêniores ao longo do projeto. A senioridade dos profissionais é definida de acordo com o seu enquadramento funcional. Considerando os projetos $\{R_1 \dots R_{10}\}$ e $\{H_1 \dots H_{17}\}$, o teste de normalidade da distribuições de dados destas variáveis resultou nos seguintes valores:

- $P - Value$ do Teste de Normalidade para $turnover = 0,216$
- $P - Value$ do Teste de Normalidade para $trocaSeniores < 0,005$

Análise Estatística

Os valores das medidas de tendência central e desvio padrão de *turnover* são, respectivamente, 28,93 e 10,11, considerando os projetos $\{R_1 \dots R_{10}\}$ e $\{H_1 \dots H_{17}\}$. A figura 4.4 mostra o histograma da distribuição de frequências de *turnover*.

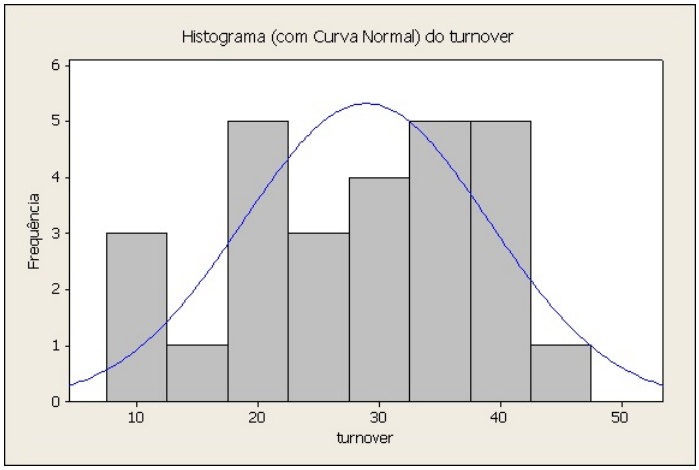

Figura 4.4: Histograma da distribuição de frequências de *turnover*

Os valores das medidas de assimetria e curtose de *turnover* são, respectivamente, -0,27 e -0,90. O valor do coeficiente de correlação de Pearson para *prodPerdida* e *turnover* é 0,711 e $P - Value = 0,000$.

O coeficiente de correlação entre *prodPerdida* e *trocaSeniores* não foi calculado porque ele não é estatisticamente significativo pois a distribuição de dados de *trocaSeniores* não passou no teste de normalidade, pois $P - Value < 0,005$.

Análise Comparativa

O *turnover* médio dos projetos $\{R_1 \dots R_{10}\}$ foi de 19% e dos projetos $\{H_1 \dots H_{17}\}$ foi de 35%, ou seja, foi verificado um aumento de 84,21% na taxa de rotatividade dos profissionais das equipes híbridas frente às equipe RUP. O aumento do *turnover* em equipes de projetos híbridos era esperado pois as características e a dinâmica dos métodos ágeis privilegia este comportamento [Nord e Tomayko 2006, Petersen e Wohlin 2009a].

No que se refere à variação da produtividade associada ao número de troca de sêniores havida ao longo do projeto, a tabulação de dados mostra que nos projetos RUP houve, em média, 1,40 troca de profissional sênior por projeto, ao passo que este número é de 2,35 em projetos híbridos. Esta variação é condizente com o resultado anterior que apontou a elevação do *turnover* da equipe. Ao agrupar os projetos RUP e híbridos por número de troca de sênior, foram encontrados os seguintes resultados de perda média de produtividade:

- Projetos RUP:
  - 1 troca: perda média de 5,05% na produtividade
  - 2 trocas: perda média de 6,15% na produtividade

- Projetos Híbridos:
  - 1 troca: perda média de 13,44% na produtividade
  - 2 trocas: perda média de 11,69% na produtividade
  - 3 trocas: perda média de 13,26% na produtividade
  - 4 trocas: perda média de 10,57% na produtividade

Ao analisar o comportamento das variáveis que interferem no indicador de pessoas percebe-se que o *turnover* da equipe é a causa preponderante da perda de produtividade. Entretanto, a quantidade de profissionais sêniores que tenham sido substituídos não é significativa, sobretudo nos projetos híbridos.

4.3.3 Tecnologia

O indicador que aponta a influência da tecnologia no desempenho de um projeto é representado pela linguagem de programação utilizada em sua construção.

Análise Estatística

Para calcular as medidas de tendência, dispersão e testar a normalidade de *linguagem*, os valores `Java`, `.NET` e `C++` da coluna *linguagem*, da tabela 4.1 foram substituídos, respectivamente pelos valores numéricos `1`, `2` e `3`. O teste de normalidade da variável linguagem resultou em $P - Value < 0,005$ considerando os projetos $\{R_1 - R_{10}\}$ e $\{H_1 - H_{17}\}$. Como a distribuição de dados de *linguagem* não passou no teste de normalidade, o valor do coeficiente de correlação entre *prodPerdida* e *linguagem* não é estatisticamente significativo.

Análise Comparativa

Ao agrupar os projetos RUP e híbridos por linguagem de desenvolvimento foram encontrados os seguintes resultados de perda média de produtividade:

- Projetos RUP:
  - `JEE`: perda média de 5,37% na produtividade
  - `.NET`: perda média de 3,81% na produtividade
  - `C++`: perda média de 7,47% na produtividade
- Projetos Híbridos:
  - `JEE`: perda média de 12,13% na produtividade
  - `.NET`: perda média de 12,47% na produtividade
  - `C++`: perda média de 11,60% na produtividade

Ao analisar os resultados anteriores constata-se que:

- A produtividade média perdida para projetos RUP foi de 5,55%, enquanto que, para projetos híbridos a perda foi de 12,06%.

- Houve perda substancial de produtividade para os projetos desenvolvidos em todas as linguagens, no entanto, chama a atenção que a diferença de produtividade dos projetos `C++` foi significantemente menor que a das demais, conforme resultados a seguir:
  - Diferença projetos `JEE`: 6,76%
  - Diferença projetos `.NET`: 9,66%
  - Diferença projetos `C++`: 4,13%
- Apesar deste dado ser sugestivo de que a produtividade em projetos C++ seja mais homogênea não é possível demonstrar esta afirmação empiricamente. Todavia, estudos futuros podem se aprofundar neste tema.

Além disso, a análise do valor do desvio padrão dos agrupamentos por linguagem sugere, mais uma vez, que a perda de produtividade dos projetos híbridos seja mais homogênea – e por consequinte, mais previsível – que aquela encontrada no conjunto de projetos RUP, por apresentarem valores 1,84 e 0,44 respectivamente.

4.3.4 Qualidade

A qualidade percebida de um *software* é diretamente proporcional à quantidade, severidade e impacto dos *bugs* associados a ele [IEEE 2004]. Este indicador permite comparar a qualidade dos *softwares* desenvolvidos pela empresa ao considerar tanto a quantidade quanto a severidade dos *bugs* detectados.

Análise Estatística

O teste de normalidade de *indiceQualidade* foi positivo porque resultou em $P - Value = 0,016$, considerando os projetos $\{R_1 \dots R_{10}\}$ e $\{H_1 \dots H_{17}\}$.

Os valores das medidas de tendência central e desvio padrão de *indiceQualidade* são, respectivamente, 2,015 e 0,1880 considerando os projetos $\{R_1 \dots R_{10}\}$ e $\{H_1 \dots H_{17}\}$. A figura 4.5 mostra o histograma da distribuição de frequências de *indiceQualidade*.

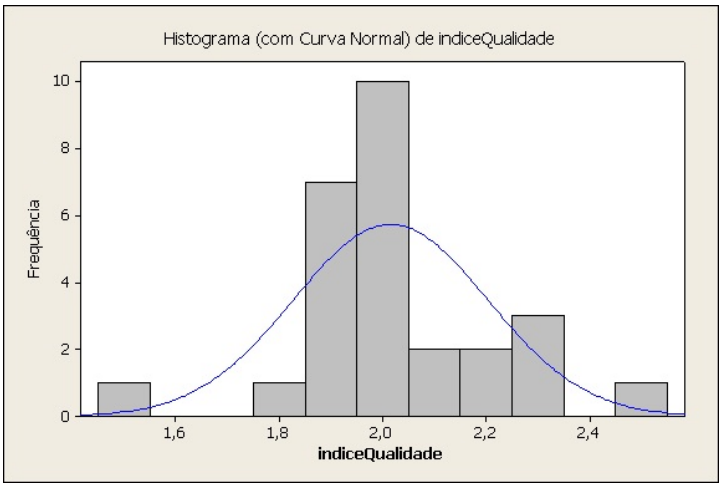

Figura 4.5: Histograma da distribuição de frequências de *indiceQualidade*

Os valores das medidas de assimetria e curtose de *indiceQualidade* são, respectivamente, -0,27 e -0,90. O valor do coeficiente de correlação de Pearson para as variáveis *prodPerdida* e *indiceQualidade* é 0,288 e $P - Value = 0,145$.

Análise Comparativa

Para comparar a variação do indicador de qualidade foram calculados os valores médios da variável *indiceQualidade* para os agrupamentos de projeto, conforme a lista a seguir:

- Projetos $\{R_1 - R_{10}\}$ e $\{H_1 - H_{17}\}$: 2,02
- Projetos $\{R_1 - R_{10}\}$: 1,95
- Projetos $\{H_1 - H_{17}\}$: 2,05

A variação do índice de qualidade denota uma piora de 5,12% na qualidade dos projetos desenvolvidos de acordo com o processo híbrido, sendo que os projetos desenvolvidos em `.NET` foram os que apresentaram o pior índice médio de qualidade, com o valor de 2,50.

4.3.5 Risco

O nível de exposição ao risco do projeto é utilizado como indicador considerando a avaliação do seu nível de impacto e respectivo peso associado.

Análise Estatística

O teste de normalidade de *indiceRisco* foi positivo porque resultou em $P - Value = 0,015$, considerando os projetos $R_1 - R_{10}$ e $H_1 - H_{17}$.

Os valores das medidas de tendência central e desvio padrão de *indiceRisco* são, respectivamente, 2,109 e 0,4350 considerando os projetos $R_1 - R_{10}$ e $H_1 - H_{17}$. A figura 4.6 mostra o histograma da distribuição de frequências de *indiceRisco*.

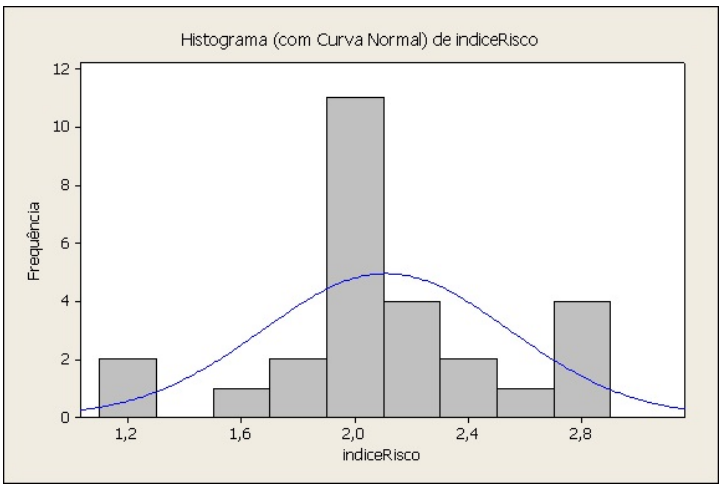

Figura 4.6: Histograma da distribuição de frequências de *indiceeRisco*

Os valores das medidas de assimetria e curtose de *indiceRisco* são, respectivamente, -0,12 e 0,78. O valor do coeficiente de correlação de Pearson para os dados de *prodPerdida* e *indiceRisco* é 0,464 e $P - Value = 0,015$.

Análise Comparativa

Para comparar a variação do indicador de risco foram calculados os valores médios da variável *indiceRisco* para os agrupamentos de projeto, conforme a lista a seguir:

- Projetos $\{R_1 - R_{10}\}$ e $\{H_1 - H_{17}\}$: 2,11
- Projetos $\{R_1 - R_{10}\}$: 1,87
- Projetos $\{H_1 - H_{17}\}$: 2,25

A variação do índice de risco denota um aumento de 20,32% no fator de exposicao ao risco dos projetos desenvolvidos de acordo com o processo híbrido, sendo que os projetos desenvolvidos em `.NET` foram os que apresentaram o índice médio de exposição ao risco mais significativo, com um valor de 2,55.

4.3.6 Discussão dos Resultados

Segundo Stutzke, uma taxa de *turnover* de 10% diminuiu o lucro e a produtividade média de um projeto em cerca de 3% [Stutzke 1995]. Considerando que estes valores se aproximem aos dos efeitos do *turnover* em processos RUP, e dessa forma espera-se encontrar perdas maiores de produtividade em projetos baseados em métodos híbridos. Os resultados encontrados pela investigação mostram uma perda média de produtividade 5,49% em projetos baseados em RUP e de 12,14% para projetos híbridos.

Os dados apresentados neste estudo de caso mostram que a produtividade de projetos híbridos é mais afetada pelo *turnover* do que os projetos RUP, o que corrobora com a proposição da pesquisa de que projetos híbridos são mais afetados pelo *turnover* do que projetos RUP e sugerem que novas investigações sejam feitas para determinar este impacto em outros tipos de processo, como Scrum ou eXtreme Programming.

Os projetos foram analisados utilizando um modelo de medição único, o que permitiu que eles fossem comparados sem a interferência de viezes metodológicos diferenciados. Os resultados mais relevantes da pesquisa apontam que:

- Os projetos mais sujeitos aos efeitos do *turnover* na produtividade são os percententes ao domínio de aplicação `Conteúdo e e-commerce`, desenvolvidos em `.NET` e nos quais houve a troca de um profissional sênior ao longo do ciclo de desenvolvimento.
- Por outro lado, os projetos menos afetados pelo *turnover*, no que se refere à redução da produtividade, são os desenvolvidos em `C++` para o domínio de aplicação `Infraestrutura` e nos quais houve a troca de quatro profissionais ao longo do desenvolvimento.

Este último resultado contraria o senso comum porque seria esperado que a troca de quatro profissionais fosse mais impactante que a troca de quantidades menores de analistas. Não é possível generalizar este achado, mesmo porque o tamanho da amostra é insuficiente. Uma hipótese que pode ser investigada em trabalhos futuros é que projetos *C++* sofrem menos com a rotatividade porque a capacitação dos profissionais que trabalham com esta tecnologia seja maior.

Apesar de não haver dados experimentais suficientes para estabelecer uma relação de causalidade direta, é provável que o aumento em 20,32% do fator de exposição ao risco nos projetos híbridos seja uma das razões da perda de produtividade. O processo de detalhamento dos requisitos deste processo é sujeito a maiores índices de retrabalho do que seus correspondentes desenvolvidos em RUP.

Além disso, a variação do índice de qualidade denota uma piora de 5,12% na qualidade dos projetos desenvolvidos de acordo com o processo híbrido. Uma hipótese para ter sido encontrada uma pequena variação neste índice é que a aderência dos processos da empresa às práticas do modelo CMMI contribui para que o gerenciamento dos projetos seja feito de forma padronizada e previsível, o que se reflete positivamente na qualidade dos projetos.

Capítulo 5

Considerações Finais

Neste trabalho foi apresentada uma investigação sobre os efeitos do *turnover* na produtividade de projetos de desenvolvimento de *software* construídos de acordo com processos tradicionais e híbridos. O referencial teórico sobre *turnover*, produtividade e processos de desenvolvimento foi estabelecido com base nas discussões encontradas na literatura e em resultados de pesquisas anteriores.

Um estudo de caso industrial multi-projeto foi realizado para buscar respostas para a questão de pesquisa, para verificar a diferença do efeito do *turnover* sobre a produtividade de equipes RUP e de equipes híbridas. Seria desejável responder aos questionamentos sob os pontos de vista quantitativo e causal. No entanto, neste estudo, ela foi respondida apenas quantitativamente, o que abre novos campos de investigação para trabalhos futuros.

Apesar do objetivo da pesquisa ser condizente com a realização de um estudo que poderia fornecer resultados conclusivos para generalização estatística [Alves 2011], os dados dos projetos são reais e por isso não há controle sobre as variáveis independentes, o que caracteriza os dados como sendo observacionais e não experimentais. Além disso, trabalhos futuros poderão utilizar um número maior de projetos para configurar uma amostra significativa e aleatória para análise quantitativa.

Ao longo da pesquisa foram selecionados os indicadores que apontam a influência da perda de produtividade decorrente do turnover, o método e o modelo de medição utillizado. Foram coletados e analisados dados de projetos desenvolvidos nos últimos três anos, envolvendo variáveis como linguagem, risco, *turnover*, tamanho funcional, produtividade e percentual de perda decorrente da rotatividade de membros da equipe.

5.1 Publicações

Além dos resultados diretos da dissertação, os seguintes artigos foram submetidos e aceitos para publicação ao longo do período de realização da pesquisa.

- ALVES, N., CARVALHO, W., LAMOUNIER, E. *Scrum and Plan-driven Process Integration and its Impact on Effort Estimation.* International Conference on Software Engineering and Knowledge Engineering, ISBN 1-891706-26-8, 2010.
- CARVALHO, W. C. S., ROSA, P. F., SOARES, M. S., CUNHA, M. A. T., BUIATTE,

L. C. *A comparative Analysis of the Agile and Traditional Software Development Processes Productivity*. XXX International Conference of the Chilean Computer Science Society, ISBN 978-0-7695-4689-6, 2011.

- ALVES, N. M. M., CARVALHO, W. C. S., CARDOSO, A., LAMOUNIER, E. A. Um estudo de caso industrial sobre integração de práticas ágeis no RUP. Revista Ciência e Tecnologia, Vol. 14, No 24/25, ISSN 2236-6733, 2011.

Outros trabalhos estão sendo elaborados para publicação em conferências de forma a disseminar os resultados obtidos com a dissertação para a comunidade acadêmica de engenharia de *software* experimental.

5.2 Trabalhos Futuros

Este trabalho apresentou um estudo de caso para descrever as observações e medições realizadas para compreender um fenômeno bem determinado, que são os efeitos do *turnover* da produtividade de projetos de *software*. Esta abordagem situa a pesquisa no escopo de atuação da engenharia de *software* experimental.

A engenharia de *software* experimental baseia-se na observação e análise de dados provenientes de processos reais de desenvolvimento de *software*. Estudos de caso são apenas um dos diversos métodos de pesquisa experimental que podem ser utilizados em pesquisas experimentais de engenharia de *software*. A questão de pesquisa desta investigação foi respondida apenas qualitativamente, e trabalhos futuros de engenharia de *software* experimental podem investigar as causas raízes dos efeitos do *turnover* na produtividade.

A utilização de outros métodos experimentais em trabalhos futuros – como experimentos em ambiente controlado, pesquisa etnográfica, análise do discurso e entrevistas – pode enriquecer o conjunto de informações geradas a partir dos dados de projetos reais. Isso possibilita que o universo de pesquisa seja explorado sob perspectivas inéditas expandindo-o para abarcar um número maior de projetos e organizações, e, se possível, em regiões geográficas distintas.

A ampliação do universo de pesquisa e a adoção de ferramentas estatísticas apropriadas permitirá analisar os dados de forma a aumentar o intervalo de confiança dos resultados. Além disso, como sugestão para trabalhos futuros, pode-se estudar detalhadamente cada um dos indicadores tratados neste trabalho de forma independente – e propor outros mais –, para compreender e modelar matematicamente o comportamento particular de cada variável que interfira na produtividade de projetos.

Além disso, os dados experimentais coletados permitem que estudos envolvendo outras disciplinas sejam desenvolvidos, como por exmeplo:

- Parametrização das condições de término do projeto e cálculo da nova duração em função do turnover calculado ao longo do projeto, e não apenas ao seu final, como foi feito nesta pesquisa.
- Criação de estratégias de recomposição da equipe do projeto com base em heurísticas definidas a partir da análise de dados de projetos de várias empresas de desenvolvimento.

- Definição das condições em que as abordagens de substituição reativa e proativa são mais adequadas em função do domínio de aplicação, linguagem ou outros parâmetros de projeto.
- Extensão do modelo de Stutzke [Stutzke 1994, Stutzke 1995] para estimar, ainda fases iniciais do ciclo de vida, os efeitos do *turnover* sobre o projeto. Isso permitiria, também, que indicadores de tendência de desvio fossem analisados em marcos intermediários para subsidiar o corpo gerencial na tomada de decisões.

O aumento do universo de pesquisa, da granularidade da análise e a utilização da expansão do modelo de Stutzke são condições estratégias e basilares para, por exemplo, a criação de um *framework* de gerenciamento de projetos e processos, que poderia ser automatizado e transformado numa ferramenta de apoio às unidades organizacionais produtores de *software*. O desenvolvimento de tal produto de *software* seria, também, uma contribuição palpável para a comunidade de tecnologia de informação e sociedade em geral.

Referências Bibliográficas

[Abdel-Hamid e Madnick 1991] Abdel-Hamid, T. e Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. Prentice Hall, Upper Saddle River, New Jersey.

[Abrahamsson et al. 2003] Abrahamsson, P., Warsta, J., Siponen, M., e Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. In *25th International Conference on Software Engineering*, pp. 244–254.

[Ahern et al. 2003] Ahern, D. M., Clouse, A., e Turner, R. (2003). *CMMI® Distilled: A Practical Introduction to Integrated Process Improvement*. Addison-Wesley Professional, Boston, 2nd edition.

[Albrecht e Gaffney 1983] Albrecht, A. J. e Gaffney, J. E. (1983). Software Function, Source Lines of codes, and Development Effort Prediction: A Software Science Validation. In *IEEE Transactions on Software Engineering*, volume SE-9, pp. 639–648.

[Alves 2011] Alves, N. M. M. (2011). *Integração de Princípios de Desenvolvimento Ágil de Software ao RUP: Um Estudo Empírico*. PhD thesis, Universidade Federal de Uberlândia – Faculdade de Engenharia Elétrica, Uberlândia, MG, Brasil.

[Ammann e Offutt 2008] Ammann, P. e Offutt, J. (2008). *Introduction to Software Testing*. Cambridge University Press, New York, USA.

[Arisholm et al. 2007] Arisholm, E., Gallis, H., Dybå, T., e Sjøberg, D. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. In *IEEE Transactions on Software Engineering*, volume 33, pp. 65–86.

[Basili et al. 1999] Basili, V., Shull, F., e Lanubile, F. (1999). Building Knowledge Through Families of Experiments. In *IEEE Transactions on Software Engineering*.

[Beck 2004] Beck, K. (2004). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, Indianapolis, Indiana, 2nd edition.

[Beck et al. 2001] Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., e Thomas, D. (2001). Manifesto for Agile Software Development.

[Becker e Whittaker 1997] Becker, S. e Whittaker, J. (1997). *Cleanroom Software Engineering Practices*. Idea Group Publishing, Hershey.

[Boehm e Turner 2003a] Boehm, B. e Turner, R. (2003a). *Balancing Agility and Discipline: a Guide for the Perplexed*. Addison-Wesley Professional, Boston.

[Boehm e Turner 2003b] Boehm, B. e Turner, R. (2003b). Observations on Balancing Discipline and Agility. In *Proceedings of Agile Development Conference ADC 2003*, pp. 32–39.

[Boehm e Turner 2004] Boehm, B. e Turner, R. (2004). Balancing agility and discipline: evaluating and integrating agile and plan-driven methods. In *26th International Conference on Software Engineering ICSE 2004*, pp. 718–719.

[Boehm et al. 1996] Boehm, B. W., Clark, B., Horowitz, E., Westland, C., Madachy, R., e Selby, R. (1996). The COCOMO 2.0 Software Cost Estimation Model. In *American Programmer*, pp. 2–17.

[Brooks 1974] Brooks, F. P. (1974). *The Mythical Man-Month.* Datamation, Los Angeles, California.

[Calazans et al. 2005] Calazans, A., Lisboa, I., e Oliveira, M. (2005). Avaliação das Características Gerais de Sistemas na Análise de Pontos de Função (APF) por Meio da Aplicação do Goal, Questions, Metrics (GQM). In *Simpósio Internacional de Melhoria de Processos de Software*, São Paulo, SP, Brazil.

[Castro 1998] Castro, J. F. (1998). Introdução à Medição de Software através de Ponto de Função. In *XII Simpósio Brasileiro de Engenharia de Software*, Maringá, PR, 1998.

[Chrissis et al. 2011] Chrissis, M. B., Konrad, M., e Shrum, S. (2011). *CMMI® for Development: Guidelines for Process Integration and Product Improvement.* SEI Series in Software Engineering. Addison-Wesley Professional, Pittsburgh, PA, 3rd edition.

[Clements et al. 2003] Clements, P., Bass, L., e Kazman, R. (2003). *Software Architecture in Practice.* SEI Series in Software Engineering. Addison-Wesley Professional, Boston, MA, 2nd edition.

[Cockburn 2000] Cockburn, A. (2000). *Writting Effective Use Cases.* Addison-Wesley Professional, Boston, 1st edition.

[Cockburn 2002] Cockburn, A. (2002). *Agile Software Development : Software Through People.* Boston, 1st edition.

[Cohen et al. 2003] Cohen, D., Lindvall, M., e Costa, P. (2003). Agile software development: a DACS state of art report. Technical report, Fraunhofer Center Maryland, New York.

[Cohen et al. 2004] Cohen, D., Lindvall, M., e Costa, P. (2004). An Introduction to Agile Methods. *Advances in Computers*, 62:2–67.

[Cohn 2004] Cohn, M. (2004). *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional, Boston, 1st edition.

[COSMIC-FFP 2001] COSMIC-FFP (2001). Measurement Manual Version 2.1. Technical report, Software Engineering Management Research Laboratory - University of Quebec, Montreal, Quebec, Canada.

[Cunha et al. 2008] Cunha, J., Thomaz, J., e Moura, H. (2008). Um Modelo de Avaliação de Produtividade de Projetos de Software Baseado em uma Abordagem Multicritério. In *Simpósio Brasileiro de Qualidade de Software*, Florianópolis. SBC.

[da Cunha 2008] da Cunha, J. A. O. G. (2008). Decisius: Um Processo de Apoio à Decisão e sua Aplicação na Definição de um Índice de Produtividade para Projetos de Software. Master's thesis, Centro de Informática - Universidade Federal de Pernambuco, Recife, PE, Brasil.

[Davis 1993] Davis, A. (1993). *Software Requirements: Objects, Functions, and States.* Prentice Hall, Upper Saddle River, NJ, 2nd edition.

[de Lucca Ramos 1999] de Lucca Ramos, J. P. (1999). O Uso de Índices Financeiros: uma análise empírica. Master's thesis, Universidade Federal de Santa Catarina, Florianópolis.

[Dybå 2000] Dybå, T. (2000). Improvisation in Small Software Organizations. In *IEEE Software*.

[Dybå e Dingsoyr 2009] Dybå, T. e Dingsoyr, T. (2009). What Do We Know about Agile Software Development? 26(5):6–9.

[Dybå e Dingsøyr 2008] Dybå, T. e Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology. IEEE Software*, 50.

[EIA 1998] EIA (1998). Systems Engineering Capability Model. Technical Report EIA/IS-731, Electronic Industries Alliance, Washington, DC, USA.

[Elssamadisy 2008] Elssamadisy, A. (2008). *Agile Adoption Patterns: A Roadmap to Organizational Success*. Addison-Wesley Professional, Boston, 1st edition.

[Fenton e Pfleeger 1997] Fenton, N. E. e Pfleeger, S. L. (1997). *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Company, Boston, 1st edition.

[Foster 1986] Foster, G. (1986). *Financial Statement Analysis*. Prentice-Hall, New Jersey, USA, 2nd edition.

[Galal-Edeen et al. 2007] Galal-Edeen, G., Riad, A., e Seyam, M. (2007). Agility versus Discipline: Is Reconciliation Possible? In *International Conference on Computational & Experimental Engineering and Sciences*. IEEE Computer Society.

[Gloger 2007] Gloger, B. (2007). The Zen of Scrum.

[Gold 1973] Gold, B. (1973). Technology Productivity and Economic Analysis. Technical report, Case Werstern Reserve University, Ohio, USA.

[Gopal et al. 2003] Gopal, A., Sivaramakrishnan, K., Krishnan, M. S., e Mukhopadhyay, T. (2003). Contracts in Offshore Software Development: An Empirical Analysis. In Informs (editor), *Management Science*, volume 49, pp. 1671–1683.

[Halstead 1977] Halstead, M. H. (1977). *Elements of Software Science*. Elsevier, New York, NY.

[Hastings 1995] Hastings, T. (1995). Adapting Function Points to contemporary software systems: A review of proposals. In *Second Australian Conference on Software Metrics*, Sydney, Australia. University of New South Wales.

[Herbsleb et al. 2001] Herbsleb, J. D., Mockus, A., Finholt, T. A., e Grinter, R. E. (2001). An empirical study of global software development: distance and speed. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pp. 81–90, Toronto, Ontario, Canada. IEEE Computer Society.

[Humphrey 1989] Humphrey, W. (1989). *Managing the Software Process*. Addison-Wesley Professional, Boston, 1st edition.

[Humphrey 1997] Humphrey, W. (1997). *Introduction to the Personal Software Process*. Addison-Wesley Professional, Boston, 1st edition.

[Humphrey 2000] Humphrey, W. (2000). *Introduction to Team Software Process*. Addison-Wesley Professional, Boston, 1st edition.

[Humphrey 2010] Humphrey, W. (2010). *Reflections on Management: How to Manage Your Software Projects, Your Teams, Your Boss, and Yourself*. Addison-Wesley Professional, Boston, 1st edition.

[IEEE 1990] IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology.

[IEEE 2004] IEEE (2004). Guide to the Software Engineering Body of Knowledge (SWEBOK).

[IFPUG 2000] IFPUG (2000). *Function Point Counting Practices Manual, Versão 4.1.1.* IFPUG.

[Ilieva et al. 2004] Ilieva, S., Ivanov, P., e Stefanova, E. (2004). Analyses of an Agile Methodology Implementation. In *Euromicro Conference*, Rennes, France. IEEE Computer Society Press.

[Jacobson et al. 1999] Jacobson, I., Booch, G., e Rumbaugh, J. (1999). *Unified Software Development Process.* Addison-Wesley Professional, Boston.

[Jones 1996] Jones, C. (1996). *Applied Software Measurement: Assuring Productivity and Quality.* McGraw-Hill, Los Angeles, 2nd edition.

[Jones 2007] Jones, C. (2007). *Estimating Software Costs: Bringing Realism to Estimating.* McGraw-Hill Osborne Media, Los Angeles, 2nd edition.

[Karlstrom e Runeson 2005] Karlstrom, D. e Runeson, P. (2005). Combining agile methods with stage-gate project management. 22(3):43–49.

[Kitchenham e Mendes 2004] Kitchenham, B. e Mendes, E. (2004). Software Productivity Measurement Using Multiple Size Measures. In *IEEE Transactions on Software Engineering.*

[Kitchenham et al. 1995a] Kitchenham, B., Pickard, L., e Pfleeger, S. (1995a). Case Studies for Method and Tool Evaluation.

[Kitchenham et al. 1995b] Kitchenham, B., Pickard, L., e Pfleeger, S. (1995b). Case Studies for Method and Tool Evaluation. In *IEEE Software.*

[Kruchten 2003] Kruchten, P. (2003). *Rational Unified Process: An Introduction.* Addison-Wesley Professional, Boston, 3 edition.

[Laturi 1996] Laturi (1996). System Product Manual - Version 2.0. Technical report, Information Technology Development Center, Helsinki, Finland.

[Layman et al. 2004] Layman, L., Williams, L., e Cunningham, L. (2004). Exploring extreme programming in context: an industrial case study. In *Proc. Agile Development Conference*, pp. 32–41, Salt Lake City, USA. IEEE Computer Society.

[Leffingwell 2011] Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Agile Software Development Series. Addison-Wesley Professional, Boston, 1st edition.

[Lokan 2000] Lokan, C. (2000). An Empirical Analysis of Function Point Adjustment Factors. In *Information and Software Technology.*

[Marçal 2009] Marçal, A. S. C. (2009). SCRUMMI: Um processo de gestão ágil baseado no SCRUM e aderente ao CMMI. Master's thesis, Universidade de Fortaleza, Fortaleza.

[Maya et al. 1998] Maya, M., Abran, A., Oligny, S., St-Pierre, D., e Desharnais, J.-M. (1998). Measuring the Functional Size of Real-Time Software. In *European Software Control and Metrics Conf.*, pp. 191–199, Maastricht, The Netherlands. Shaker Publishing BV.

[McBreen 2003] McBreen, P. (2003). *Questioning Extreme Programming.* Pearson Education., Boston, 1st edition.

[Merisalo-Rantanen et al. 2005] Merisalo-Rantanen, H., Tuure, T., e Matti, R. (2005). Is Extreme Programming Just Old Wine in New Bottles: a Comparison of Two Cases. *Journal of Database Management*.

[Miguel 2007] Miguel, P. (2007). *Estudo de caso na engenharia de produção: estruturação e recomendações para sua condução*. Produção, Porto Alegre.

[Mirer 1983] Mirer, T. W. (1983). *Economic Statistics e Econometrics*. Macmillan Publishing, New York, 1st edition.

[Mohagheghi e Conradi 2004] Mohagheghi, P. e Conradi, R. (2004). An empirical study of software change: origin, acceptance rate, and functionality vs. quality attributes. In *Proc. Int. Symp. Empirical Software Engineering ISESE '04*, pp. 7–16.

[Nerur et al. 2005] Nerur, S., Mahapatra, R., e Mangalaraj, G. (2005). Challenges of Migrating to Agile Methodologies. In *Communications of the ACM*.

[Nord e Tomayko 2006] Nord, R. L. e Tomayko, J. E. (2006). Software architecture-centric methods and agile development. 23(2):47–53.

[Paulk e Chrissis 2002] Paulk, M. C. e Chrissis, M. B. (2002). The 2001 High Maturity Workshop. Technical Report CMU/SEI-2001-SR-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA. http://www.sei.cmu.edu/library/abstracts/reports/01sr014.cfm.

[Petersen e Wohlin 2009a] Petersen, K. e Wohlin, C. (2009a). A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, 82:1479–1490.

[Petersen e Wohlin 2009b] Petersen, K. e Wohlin, C. (2009b). Context in Industrial Software Engineering. *Empirical Software Engineering and Measurement*.

[Pikkarainen et al. 2008] Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., e Still, J. (2008). The Impact of Agile Practices on Communication in Software Development. *Empirical Software Engineering*.

[PMI 2009] PMI (2009). *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK)*. Project Management Institute, Inc.

[Pressman 2011] Pressman, R. S. (2011). *Engenharia de Software: Uma Abordagem Profissinal*. McGraw-Hill - Artmed, Porto Alegre, 7 ed. edition.

[Prowell et al. 1999] Prowell, S., Trammell, C., Linger, R., e Poore, J. (1999). *Cleanroom Software Engineering: Technology and Process*. Addison-Wesley Professional.

[Qumer e Henderson-Sellers 2008] Qumer, A. e Henderson-Sellers, B. (2008). A Framework to Support the Evaluation, Adoption and Improvement of Agile Methods in Practice. *Journal of Systems and Software*.

[Ramsin e Paige 2008] Ramsin, R. e Paige, R. F. (2008). Process-centered review of object oriented software development methodologies. *ACM Computing Surveys (CSUR)*, pp. 3:1–3:89.

[Rehesaar 1998] Rehesaar, H. (1998). Software Size: The Past and the Future. In *European Software Control and Metrics Conf.*, pp. 200–208, Maastricht, The Netherlands. Shaker Publishing BV.

[Roetzheim 1988] Roetzheim, W. H. (1988). *Structured Computer Project Management*. Prentice Hall.

[Rombach e Seelisch 2007] Rombach, D. e Seelisch, F. (2007). Formalisms in Software Engineering: Myths Versus Empirical Facts. In *Central and East European Conference on Software Engineering Techniques*, pp. 13–25, Poznan.

[Saleh 2009] Saleh, K. A. (2009). *Software Engineering*. Roundhouse.

[Schwaber 1995] Schwaber, K. (1995). SCRUM development process. In *Conference on Object-Oriented Programming Systems, Languagens and Applications*, Austin, TX, USA. Springer.

[Schwaber 2004] Schwaber, K. (2004). *Agile Project Management with Scrum*. Microsoft.

[Schwaber e Beedle 2001] Schwaber, K. e Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall.

[Schwaber e Sutherland 2010] Schwaber, K. e Sutherland, J. (2010). Scrum Guide, 2010.

[Seaman 1999] Seaman, C. (1999). Qualitative Methods in Empirical Studies on Software Engineering. *IEEE Transactions on Software Engineering*.

[SEI 1997] SEI (1997). Integrated Product Development Capability Maturity Model. Technical Report Draft Version 0.98., Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

[Shaw e Garlan 1996] Shaw, M. e Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River, NJ.

[Shepperd e Schofield 1997] Shepperd, M. e Schofield, C. (1997). Estimating Software Project Effort Using Analogy. In *IEEE Trans. Soft. Eng.*, volume SE-23:12, pp. 736–743.

[Sommerville 2011] Sommerville, I. (2011). *Engenharia de Software*. Pearson Brasil, São Paulo, 9 ed. edition.

[St-Pierre et al. 1997] St-Pierre, D., Maya, M., Abran, A., Desharnais, J., e Bourque, P. (1997). Full Function Points: Counting Practice Manual. Technical Report 1997-04, University of Quebec at Montreal.

[Stutzke 1994] Stutzke, R. D. (1994). A Mathematical Expression of Brooks' Law. In *Ninth International Forum on COCOMO and Cost Modeling*.

[Stutzke 1995] Stutzke, R. D. (1995). Quantifying the Effects of Staff Turnover. In *European Software Cost Measurement Conference*, Rolduc, the Netherlands.

[Tamaki 2007] Tamaki, P. A. O. (2007). Uma Extensão do RUP com ênfase no Gerenciamento de Projetos do PMBOK baseada em *Process Patterns*. Master's thesis, Universidade de São Paulo.

[Tessem 2003] Tessem, B. (2003). Experiences in Learning XP Practices: A Qualitative Study. *Extreme Programming and Agile Processes in Software Engineering*.

[Udo e Koppensteiner. 2003] Udo, N. e Koppensteiner., S. (2003). *Will agile change the way we manage software projects? Agile from a PMBOK guide perspective*. Projectway.

[Wagner e Ruhe 2008] Wagner, S. e Ruhe, M. (2008). A Systematic Review of Productivity Factors in Software Development. In *International Workshop on Software Productivity Analysis and Cost Estimation*, Beijing. IEEE Computer Society.

[Williams e Cockburn 2003] Williams, L. e Cockburn, A. (2003). Agile software development: it's about feedback and change. *Computer*, 36(6):39–43.

[Yin 2010] Yin, R. K. (2010). *Estudo de Caso - Planejamento e Métodos*. Bookman Companhia Editorial.