CENTRO UNIVERSITÁRIO DE ARARAQUARA
MESTRADO PROFISSIONAL EM ENGENHARIA DE PRODUÇÃO

Erick Eduardo Petrucelli

ANÁLISE DA APLICAÇÃO DO PENSAMENTO ENXUTO NO DESENVOLVIMENTO DE SOFTWARE: ESTUDO DE CASO EM UMA EMPRESA BRASILEIRA DE MÉDIO PORTE

Dissertação apresentada ao Programa de Mestrado Profissional em Engenharia de Produção do Centro Universitário de Araraquara – UNIARA – como parte dos requisitos para obtenção do título de Mestre em Engenharia de Produção, Área de Concentração: Gestão Estratégica e Operacional da Produção.

Prof. Dr. Fábio Ferraz Junior
Orientador

Araraquara, SP – Brasil
2013

P595a Petrucelli, Erick Eduardo
Análise da aplicação do pensamento enxuto no desenvolvimento de software: estudo de caso em uma empresa brasileira de médio porte/ Erick Eduardo Petrucelli. – Araraquara: Centro Universitário de Araraquara, 2013.
133f.

Dissertação - Mestrado Profissional em Engenharia de Produção - Centro Universitário de Araraquara - UNIARA

Prof. Dr. Fábio Ferraz Junior

1. Estudo de caso. 2. Produção enxuta. 3. Pensamento enxuto. 3. Metodologias ágeis. 4. Engenharia de software. I. Título.

CDU 62-1

REFERÊNCIA BIBLIOGRÁFICA

PETRUCELLI, E. E. **Análise da Aplicação do Pensamento Enxuto no Desenvolvimento de Software**: Estudo de Caso em uma Empresa Brasileira de Médio Porte. 2013. 133f. Dissertação de Mestrado em Engenharia de Produção – Centro Universitário de Araraquara, Araraquara-SP.

ATESTADO DE AUTORIA E CESSÃO DE DIREITOS

NOME DO AUTOR: Erick Eduardo Petrucelli
TÍTULO DO TRABALHO: Análise da Aplicação do Pensamento Enxuto no Desenvolvimento de Software: Estudo de Caso em uma Empresa Brasileira de Médio Porte
TIPO DO TRABALHO/ANO: Dissertação / 2013

Erick Eduardo Petrucelli
Rua Voluntários da Pátria, 1309 e 1295 – Centro
14801-320 - Araraquara – SP
erickpetru@gmail.com

Centro Universitário de Araraquara
Rua Voluntários da Pátria, 1309 - Centro - Araraquara - SP
CEP 14801-320 - Caixa Postal 68 - Fone/Fax: (16) 3301-7100
www.uniara.com.br

Dissertação aprovada em sua versão final pela banca examinadora:

Fábio Ferraz Júnior

Prof. Dr. Fábio Ferraz Júnior
Orientador(a) – UNIARA

Prof. Dr. Lucas Baggio Figueira
Centro Universitário Barão do Mauá

Prof. Dr. José Luis Garcia Hermosilla
UNIARA – Araraquara

Araraquara, 11 de dezembro de 2013

Agradecimentos

Em primeiro lugar, a Deus, pois sem a fé e a força concedida para a superação das dificuldades, nada disso seria possível.

Aos meus queridos pais, Tânia Maria França Petrucelli e Adilson Ramiro Petrucelli, que desde os meus primeiros anos de vida me ensinaram a importância do estudo, da dedicação e da responsabilidade para o sucesso profissional e para a realização pessoal.

À minha irmã Ellen Eloá Petrucelli, à minha avó Malvina Jacira Moratta França (*in memoriam*) e ao meu avô Eurides França, pela presença determinante durante minha infância e adolescência, influenciando tão positivamente minha personalidade e educação.

Ao meu orientador, Prof. Dr. Fábio Ferraz Junior, o qual exemplarmente me guiou no desenvolvimento do presente trabalho, me orientando não apenas na pesquisa, mas sendo também um grande direcionador durante os desafios encontrados.

Aos demais professores do programa de mestrado, que tão bem contribuíram para esta pesquisa e, principalmente, para a evolução do meu conhecimento.

A todos os outros familiares e amigos que sempre estiveram por perto de alguma forma, apoiando direta ou indiretamente em meus estudos e minha carreira.

Resumo

O mercado de *software* brasileiro tem crescido nos últimos anos, principalmente o desenvolvimento de *software* sob encomenda. Ao mesmo tempo, este mercado enfrenta há anos problemas com custos altos, não cumprimento de prazos e projetos fracassados. Como tentativa de obtenção de um processo de desenvolvimento de *software* mais adequado e flexível ao cenário de mudanças constantes, metodologias inspiradas ou complementares aos conceitos do pensamento enxuto têm emergido há alguns anos, sendo por vezes também denominadas metodologias ágeis. Com o questionamento sobre quais combinações de abordagens efetivamente apresentam resultados e como se aplicam em uma empresa real, este trabalho apresenta um estudo de caso explorando uma empresa brasileira de médio porte que conciliou abordagens como Lean Software Development e Extreme Programming com a metodologia Scrum. Como base para o estudo, o trabalho apresenta inicialmente revisão bibliográfica acerca da produção enxuta, os conceitos e limitações da engenharia de *software* tradicional prescritiva, as características de conceitos enxutos para o desenvolvimento de *software* e a relação entre metodologias enxutas e ágeis. O estudo de caso foi conduzido através de observação direta, aplicação de entrevistas com gestores e um questionário com desenvolvedores. Demonstrou-se o sucesso da empresa ao conciliar estas abordagens, com resultados positivos qualitativos em produtividade, qualidade, custos e satisfação dos clientes, principalmente quanto à capacidade de oferecer maior valor ao negócio e de atuar com equipes mais coesas e comprometidas. Também foi possível detectar alguns pontos que a empresa pode melhorar quanto à aplicação do pensamento enxuto, principalmente nos princípios de “amplificação de aprendizado”, “valorização da equipe” e “otimização do todo”. Por fim, concluiu-se que a adaptação do pensamento enxuto ao desenvolvimento de *software* é relevante, pode ser conduzida através de combinações de diferentes metodologias ágeis e oferece a possibilidade de gerar bons resultados à empresa.

Palavras-chave: *estudo de caso, produção enxuta, pensamento enxuto, metodologias ágeis, engenharia de software.*

Abstract

The Brazilian software market has been growing in the past few years, especially the development of customized software. At the same time, this market has faced problems along the years with high costs, non-compliance with established deadlines and failed projects. As an attempt to obtain a more flexible software development process, appropriate to the constant changes scenario, methodologies inspired by or complementary to the concepts of lean thinking have emerged for a few years, being sometimes also called agile methodologies. Based on the question about which combinations of approaches effectively present results, and how to apply them to a real company, this dissertation presents a case study exploring a Brazilian midsize company which conciliated approaches such as Lean Software Development and Extreme Programming with the Scrum methodology. As the foundation for the study, this dissertation starts presenting a literature review about lean production, the concepts and limitations of the traditional prescriptive software engineering, the characteristics of lean concepts applied to software development and the relationship between lean and agile methodologies. The case study has been conducted through direct observation, application of interviews with managers and a survey with developers. The study has demonstrated the company's success to conciliate these approaches, with positive results in productivity, quality, costs and customers satisfaction, in particular with regard to the capacity to add more value to business and to act with more cohesive and committed teams. Moreover, it was possible to detect some points in which the company can improve concerning the application of lean thinking, particularly the principles of “amplify learning”, “empower the team” and “see the whole”. Finally, it was concluded that the adaptation of lean thinking to software development is relevant, it can be conducted by combining different agile methodologies and it offers the possibility to generate good results to the company.

Keywords: *case study, lean thinking, lean manufacturing, agile methodologies, software engineering.*

Lista de Figuras

Lista de Quadros e Tabelas

Lista de Abreviaturas e Siglas

ACM – Association for Computer Machinery
CMMI – Capability Maturity Model Integration
CQZD – Controle da Qualidade Zero Defeitos
ESPRIT – European Strategic Program on Research in Information Technology
EUA – Estados Unidos da América
FDD – Feature-Driven Development
IDE – Integrated Development Environment
IEEE – Institute of Electrical and Electronics Engineers
IMVP – International Motor Vehicle Program
JIT – Just-in-Time
LSD – Lean Software Development
MCT – Ministério da Ciência e Tecnologia
MPS.BR – Melhoria de Processos do Software Brasileiro
NATO – North Atlantic Treaty Organization
PDCA – Plan Do Check Act
ROI – Return On Investment
RUP – Rational Unified Process
SQL – Structured Query Language
STP – Sistema Toyota de Produção
SWEBOK – Software Engineering Body of Knowledge
TCO – Total Cost of Ownership
TDD – Test-Driven Development
TFS – Team Foundation Service
TI – Tecnologia da Informação
UML – Unified Modeling Language
VSM – Value Stream Mapping
XP – Extreme Programming

Sumário

1 Introdução

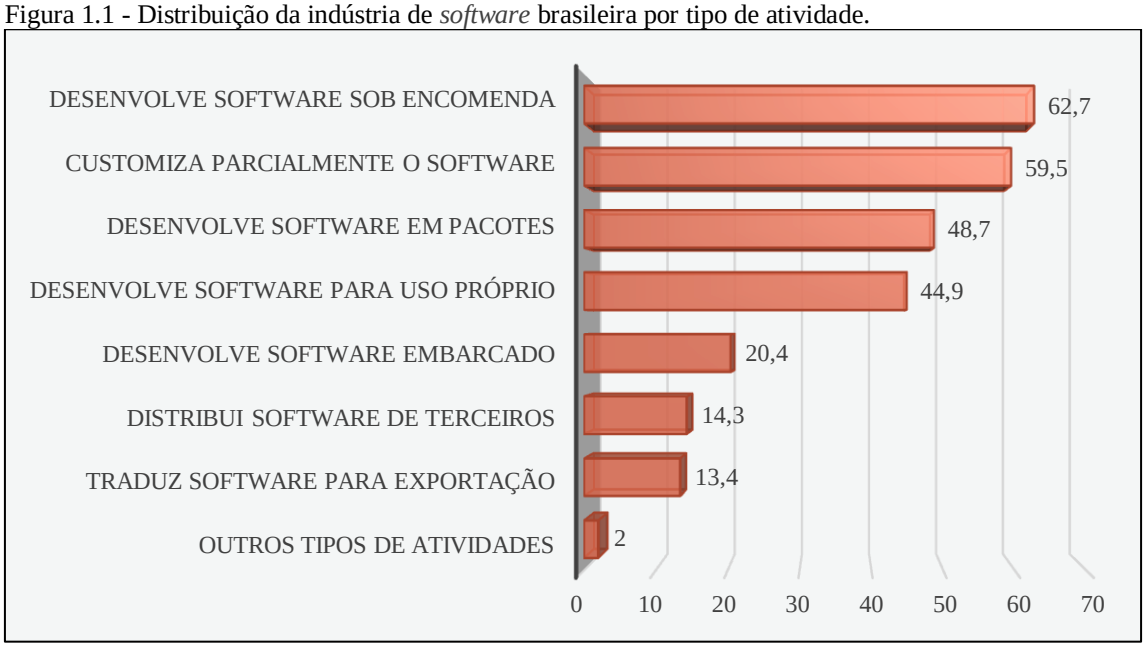
Nesta seção, inicia-se a abordagem ao assunto deste trabalho, apresentando-se o contexto no qual o assunto está inserido com os problemas observados neste, as justificativas para a realização e os objetivos do trabalho, bem como um breve delineamento da metodologia utilizada para classificação e norteio da pesquisa.

1.1 Contextualização

O mercado de *software* brasileiro cresce a cada ano, tendo movimentado por volta de US$ 5,4 bilhões em 2009, ocupando mundialmente a 12ª colocação (ABES, 2010). Destaca-se dentro deste mercado a atividade de desenvolvimento de *software*, visto que, de 8.500 empresas atuantes no setor, 76% trabalham com esta atividade.

Em tais empresas, destaca-se como modelo principal o desenvolvimento de *software* sob encomenda, ou seja, programas desenvolvidos de forma personalizada, a fim de atender requisitos específicos e únicos de cada cliente, atividade também conhecida pelo termo em inglês *software on demand*. Isto se evidencia na Figura 1.1, adaptada de pesquisa realizada pelo Ministério da Ciência e Tecnologia (MCT, 2009), onde 62,7% das empresas respondentes declararam atuar neste seguimento de desenvolvimento de *software*.

Figura 1.1 - Distribuição da indústria de *software* brasileira por tipo de atividade.



Fonte: adaptado de MCT (2009).

Software de computadores continua a ser a tecnologia única mais importante no cenário mundial, afirma Pressman (2011). O autor complementa esta afirmação analisando

que, no mundo atual, o *software* assume um papel duplo: pode ser encarado tanto como um produto quanto como um veículo para distribuição de um produto. Como produto, precisa ser planejado, projetado, construído e entregue de forma organizada, eficaz e eficiente aos interessados. Como veículo, seu papel revela-se como distribuidor do produto mais importante de nossa era, a informação.

No contexto corporativo, dada a importância das informações, sabe-se que a TI indiscutivelmente proporciona fontes de vantagens competitivas para as organizações (PORTER e MILLAR, 1995). Entretanto, nem sempre existe relação direta entre o investimento em TI e o retorno financeiro proporcional, conforme discute Roach (1988), de forma que a obtenção de vantagens competitivas, na verdade, está associada à aderência da TI, incluindo-se aí os *softwares*, às características do negócio.

Sabe-se também que as empresas estão em constante mudança e o mundo globalizado exige que a adequação ocorra de maneira cada vez mais rápida. Sendo assim, os recursos de TI também devem ser constantemente atualizados para manutenção de seu valor, o que torna cada vez mais necessário que as atividades de desenvolvimento e de manutenção de *software* ocorram de maneira produtiva, com menos desperdícios e maior qualidade, sendo de grande valor investigar os problemas encontrados neste processo produtivo (CHARETTE, 2005).

> O mercado atual é altamente dinâmico, tornando a sobrevivência das empresas uma questão de constante atualização. Em função desta constante busca pela atualização, os requisitos de *software* não podem ser estáticos e imóveis, sendo, portanto, necessário um processo de desenvolvimento que possibilite a mudança dinâmica destes requisitos. Em consequência, os processos tradicionais normalmente não aceitam mudanças e demoram a entregar resultados (SILVA, 2011, p. 2).

Segundo Charette (2005), enquanto esta realidade de mudanças se faz constante no mercado, a própria natureza do processo de desenvolvimento de *software* tradicional não é preparada para isto, sendo propícia a falhas e apresentando historicamente baixa satisfação do usuário, atrasos e aumento de custos.

Para Middleton (2001), a utilização de práticas de manufatura enxuta para projetos de *software* pode ser assustadora inicialmente. Porém, pouco tempo depois de compreendido efetivamente os princípios que norteiam o pensamento enxuto, a situação se inverte visivelmente e os ganhos em produtividade, qualidade e sinergia da equipe demonstram claramente a importância de aplicar estes conceitos no desenvolvimento de *software*.

Embora com essa perspectiva promissora, pode-se dizer que a importância dada pelos pesquisadores brasileiros, inclusive em pesquisas oficiais da União, ainda é pequena. A já citada pesquisa sobre o mercado de *software* brasileiro do MCT (2009), apenas informa que

existe alguma presença de metodologias enxutas e/ou ágeis como forma de organização para o desenvolvimento de *software* no Brasil, mas não existem números concretos sobre o percentual destas empresas. O fato deste dado nem ter sido alvo de investigação neste tipo de pesquisa oficial, até a data da pesquisa, demonstra como o assunto ainda é relativamente desconhecido e incompreendido no país.

Percebe-se também um desafio adicional para pesquisar a aplicação de abordagens enxutas na indústria de *software*, uma vez que *softwares* possuem características únicas em comparação a produtos físicos, como: projeto intangível, lógica demasiadamente complexa, alto custo de planejamento em contraste com menor custo de produção (JONSSON, 2012). Conforme apontado por tal autor, tais fatores influenciam para que não seja óbvio como deve ocorrer a adaptação dos princípios do pensamento enxuto à indústria de *software* e, portanto, enfatizam a importância de estudos de caso individuais como a mais valiosa forma de pesquisa empírica para a engenharia de *software* neste contexto.

Com base na contextualização do tema e nos problemas apresentados, a seguinte questão resume a pergunta principal abordada por esta pesquisa:

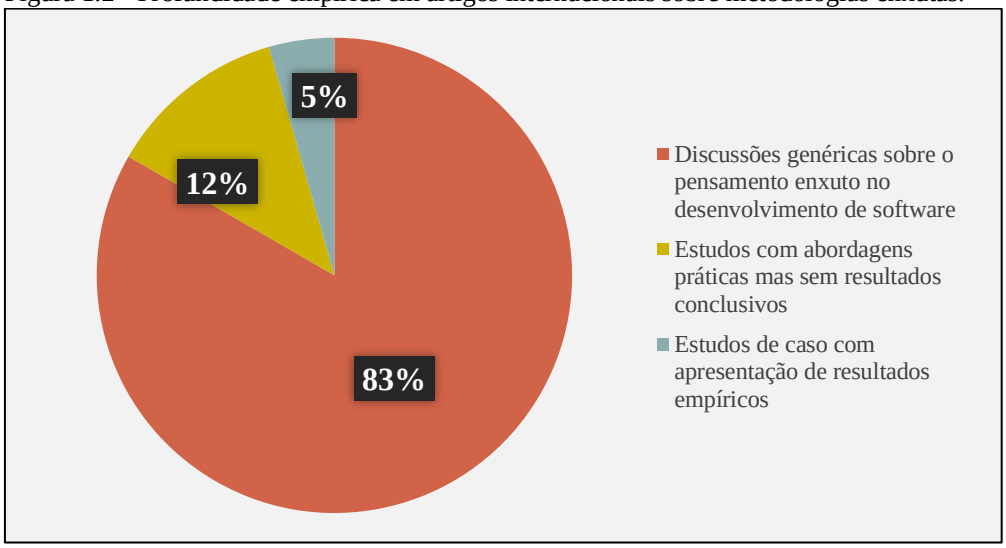
Como os conceitos do pensamento enxuto podem ser empregados no desenvolvimento de *softwares* sob encomenda em uma empresa real?

1.2 Justificativas

Sobre a utilização de práticas e princípios enxutos nas mais diversas áreas, "há uma grande quantidade de histórias de sucesso" (DEGIRMENCI, 2008, p. iii) e, portanto, esta expectativa de sucesso ao aplicar os conceitos também na indústria de *software* é um fator de forte peso ao dedicar uma pesquisa sobre o assunto, por já existirem diversas demonstrações de potencial destas práticas em proporcionar melhorias no processo e no produto, resolvendo problemas das abordagens tradicionais, conforme apresentado ao longo deste trabalho.

A carência de estudos de caso rigorosos sobre o tema também endossa a importância de novos estudos. Em sua revisão sistemática da aplicação de conceitos enxutos ao desenvolvimento de *software*, Jonsson (2012) descreve sua investigação a partir de seis bases internacionais de publicações científicas, apontando que, de 140 resultados encontrados com os termos *lean* e *software*, apenas 30 mostraram discussões realmente relacionadas à aplicação prática de alguma abordagem e apenas 8 resultados contiveram evidência empírica, com resultados qualitativos, quantitativos ou ambos. Na página a seguir, a Figura 1.2 resume graficamente estes números.

Figura 1.2 - Profundidade empírica em artigos internacionais sobre metodologias enxutas.



Fonte: adaptado de Jonsson (2012).

A análise apresentada por Jonsson (2012) corrobora os resultados da investigação apresentada previamente por Dybå e Dingsøyr (2009), onde inicialmente foram investigados 1.996 estudos, mas apenas 36 puderam ser considerados relevantes para os autores por possuírem rigorosidade metodológica, credibilidade e relevância. Destes 36, os autores destacam que "apenas alguns" (sem citar exatamente quantos) focaram os estudos em times maduros que já vinham desenvolvendo *software* anteriormente, enquanto o restante compunha-se apenas de ambientes simulados, criados exclusivamente para estudar as metodologias, principalmente a abordagem Extreme Programming (em 25 dos 36 estudos), a qual será detalhada posteriormente neste trabalho.

Esta carência de estudos empíricos, principalmente que abordem equipes maduras de empresas comerciais consolidadas, que já tivessem anos de experiência com metodologias convencionais antes do investimento em abordagens enxutas, representa forte relevância e incentivo para novas pesquisas neste contexto.

Enquanto o tema ainda passa despercebido por parte do mercado, conforme já discutido na contextualização deste trabalho, é possível encontrar autores exibindo números marcantes sobre o crescimento da adoção por empresas brasileiras, como em Costa (2011), onde 81% de 69 participantes respondentes informaram ter iniciado a utilização de algum tipo de abordagem ágil entre 2008 e 2010. Corroborando com estes dados, resultado similar em relatório técnico da AgilCoop (2012), no qual 466 participantes respondentes apontaram que apenas 10,5% destes nunca haviam tido nenhum contato com metodologias ágeis e/ou enxutas em suas empresas.

Este crescimento na adesão por parte de empresas de todo o país ressalta ainda mais a importância de estudos empíricos que possam agregar conhecimento sobre as formas como diferentes abordagens e ideias enxutas se adaptam ao processo de desenvolvimento de *software* sob encomenda.

Para Silva (2011), embora diferentes abordagens tragam resultados positivos a partir do que pôde ser observado em vários estudos, os conceitos necessários para avaliação de processos produtivos não são alvo de estudo pela área da engenharia de *software* e, portanto, trabalhos provenientes da área de engenharia de produção com este enfoque são efetivamente muito valiosos para a indústria de *software* brasileira.

Por fim, e com base nas diferentes argumentações apresentadas, compete dizer que a presente pesquisa é relevante e justifica-se principalmente por:

- Possibilitar o estudo empírico em um ambiente real, que já tenha obtido sucesso adaptando abordagens ágeis, de forma a explorar como ocorreu esta adaptação e quais vantagens e possíveis desvantagens foram observadas;
- Elucidar de que possível maneira os conceitos do pensamento enxuto podem ser aplicados de forma prática em uma empresa de *software* brasileira através de um estudo de caso;
- Permitir que outras empresas utilizem os sucessos e os fracassos do caso estudado como modelo para suas próprias implementações;
- Elevar a visibilidade do tema e, possivelmente, incentivar trabalhos futuros sobre este assunto no cenário acadêmico brasileiro.

1.3 Objetivos

A partir do contexto e dos problemas apresentados anteriormente, foi definido que o objetivo geral deste trabalho é:

- Explorar quais e como os conceitos do pensamento enxuto, aplicados ao desenvolvimento de *software*, são empregados em uma empresa brasileira de *software* sob encomenda de médio porte.

Para apoiar este objetivo, os seguintes objetivos específicos foram estipulados:

- Explorar as generalidades e as particularidades entre diferentes abordagens enxutas aplicáveis ao desenvolvimento de *software*;

- Analisar quais princípios a empresa alvo do estudo aplica, qual sua relação com cada abordagem e com os conceitos originais do pensamento enxuto;
- Verificar os desafios, sucessos e fracassos da empresa para conciliar as práticas enxutas em relação aos processos tradicionais pré-existentes.

1.4 Metodologia

Conforme aponta Cervo e Bervian (2002), a pesquisa é uma atividade voltada para a solução de problemas com o emprego de processos científicos, partindo de um estado de dúvida ou problema e, com o uso do método científico, buscando uma resposta ou solução. Há na literatura diversos critérios de classificação para tais procedimentos, variando de acordo com o enfoque dado: há classificações do ponto de vista da natureza da pesquisa, dos objetivos de pesquisa e dos procedimentos técnicos de pesquisa (BRYMAN, 1989; GIL, 1999; MARCONI e LAKATOS, 2000; CERVO e BERVIAN, 2002).

A seguir, será apresentada a classificação metodológica do presente trabalho, a partir destes diferentes pontos de vista.

1.4.1 Natureza da pesquisa

Do ponto de vista da natureza da pesquisa, as classificações possíveis são *pesquisa pura ou básica* e *pesquisa aplicada* (CERVO e BERVIAN, 2002). Nesse aspecto, este trabalho é classificado como uma pesquisa aplicada, uma vez que objetiva gerar conhecimentos para aplicação prática dirigida à solução de um problema específico: entender como empresas brasileiras de *software* sob encomenda podem aplicar práticas de produção enxuta em seus processos tradicionais de desenvolvimento de *software*.

1.4.2 Forma de abordagem ao problema

Do ponto de vista da forma de abordagem ao problema, as classificações possíveis são *pesquisa qualitativa* e *pesquisa quantitativa* (BRYMAN, 1989). Considerando que este trabalho de pesquisa não se apoia em modelos estatísticos, pode ser caracterizado como uma abordagem qualitativa.

1.4.3 Objetivos da pesquisa

Do ponto de vista dos objetivos da pesquisa, as classificações possíveis são *exploratória*, *descritiva* e *explicativa* (GIL, 1991). Nesse aspecto, este trabalho é classificado

como pesquisa exploratória, pois visa proporcionar maior familiaridade com os temas abordados com vistas a tornar o cenário explícito. Para Roesch (1999), a pesquisa exploratória por concepção é favorecida pelos métodos utilizados pela abordagem qualitativa.

1.4.4 Procedimentos técnicos

Do ponto de vista dos procedimentos técnicos, as classificações possíveis são *pesquisa bibliográfica*, *pesquisa documental*, *pesquisa experimental*, *levantamento*, *estudo de caso*, *pesquisa-ação* e *pesquisa participante* (GIL, 1991; YIN, 2005). Nesse aspecto, o trabalho se sustenta em dois núcleos metodológicos: pesquisa bibliográfica e estudo de caso.

O primeiro núcleo, por intermédio da revisão bibliográfica pertinente ao tema da pesquisa, tem como escopo proporcionar embasamento teórico, verificar as posições de outros autores sobre os temas e comparar com as ações da empresa estudada, pois a pesquisa bibliográfica é sempre vertente importante de qualquer trabalho acadêmico, conforme pontua Cervo e Bervian (2002, p. 88):

> Praticamente todo o conhecimento humano pode ser encontrado nos livros ou em outros impressos que se encontram nas bibliotecas. A pesquisa bibliográfica tem como objetivo encontrar respostas aos problemas formulados, e o recurso é a consulta dos documentos bibliográficos.

No segundo núcleo, a partir do estudo de caso, busca-se abordar uma situação real de aplicação de produção enxuta de *software* em uma fábrica de *software* brasileira, pois, segundo Yin (2005), o estudo de caso é particularmente adequado à tentativa de explicar nexos causais em situações da vida real que são demasiadamente complexas para serem tratadas por meio de estratégias experimentais. Segundo Gil (1991), a pesquisa como estudo de caso é particularmente útil quando não se dispõe de informação suficiente para fornecer resposta ao problema, ou então em uma situação em que a informação disponível se encontra desordenada de tal maneira que não possa ser relacionada ao problema.

1.4.5 Critério para definição do sujeito

Do ponto de vista do critério utilizado para definir o sujeito do estudo de caso, foi do tipo não probabilístico e intencional por conveniência, uma vez que a seleção da empresa obedeceu a critérios pré-definidos no seguinte aspecto:

- Empresa brasileira de desenvolvimento de *software* sob encomenda;

- Que declarasse publicamente utilizar uma ou mais abordagens do pensamento enxuto em seu processo produtivo.

Também se optou intencionalmente em selecionar uma empresa de pequeno ou médio porte, onde, conforme apresentado anteriormente, a vantagem da utilização do pensamento enxuto em relação às metodologias tradicionais possivelmente seria maior, e também para detectar possíveis desafios dados os menores recursos financeiros para investimento.

1.4.6 Critério para coleta dos dados

O presente estudo foi conduzido a partir de: observação direta na empresa para a visualização de seu funcionamento em todas as fases do processo produtivo de desenvolvimento de *software*; entrevistas semiestruturadas e não estruturadas com gestores; questionários com desenvolvedores, a fim de explorar o quanto o pensamento enxuto é compreendido e aplicado em todas as fases do processo.

Na seção 5 deste trabalho, a coleta de dados será descrita em detalhes, apresentando-se a ordem de execução das atividades e descrevendo-se como as ações ocorreram.

1.5 Estrutura

O presente trabalho está estruturado em seis seções. A presente seção 1 corresponde à introdução e contempla a contextualização, a justificativa, os objetivos, a metodologia e a estrutura do trabalho.

Na seção 2 é tratada a revisão bibliográfica acerca do tema produção enxuta, descrevendo suas origens, seus princípios e a filosofia do pensamento enxuto.

Na seção 3 é tratada a revisão bibliográfica acerca da engenharia de *software* tradicional (ou prescritiva), apresentando seus principais conceitos e limitações.

Na seção 4 é tratada a revisão bibliográfica acerca das metodologias enxutas para o desenvolvimento de *software*, analisando-se as diferentes adaptações da produção enxuta existentes para o setor.

Na seção 5 são apresentadas a coleta e a análise dos dados, expondo-se os resultados encontrados durante o estudo de caso realizado.

Na seção 6 são apresentadas as considerações finais do estudo.

2 Produção enxuta

Conforme comenta Ghinato (2000), produção enxuta é um termo genérico, cunhado originalmente ao final dos anos 80 por pesquisadores do IMVP, para definir de maneira mais genérica os conceitos do Sistema Toyota de Produção (STP), os quais serviram de inspiração para a disseminação da filosofia do pensamento enxuto pelo mundo.

Visando apoiar a compreensão dos conceitos sobre o pensamento enxuto, bem como suas principais vertentes que levaram às adaptações existentes para o desenvolvimento de *software*, esta seção trará uma revisão dos conceitos envolvidos, provenientes da engenharia de produção, que podem melhor embasar esta discussão.

2.1 Histórico

Os conceitos que representam a filosofia do pensamento enxuto foram originalmente desenvolvidos para a manufatura e suas origens remontam mais especificamente à indústria automobilística, em especial à Toyota Motor Corporation (LIKER e HOSEUS, 2008).

> O entusiasmo da família Toyoda pela indústria automobilística começou ainda no início do século, após a primeira viagem de Sakichi Toyoda aos Estados Unidos em 1910. No entanto, o nascimento da Toyota Motor Co. deve-se mesmo a Kiichiro Toyoda, filho do fundador Sakichi, que em 1929 também esteve em visita técnica às fábricas da Ford nos Estados Unidos. Como decorrência deste entusiasmo e da crença de que a indústria automobilística em breve se tornaria o carro-chefe da indústria mundial, Kiichiro Toyoda criou o departamento automobilístico na Toyoda Automatic Loom Works, a grande fabricante de equipamentos e máquinas têxteis pertencente à família Toyoda, para, em 1937, fundar a Toyota (GHINATO, 2000).

Este autor ainda explica que a Toyota entrou na indústria automobilística desde então, mas especializando-se primeiramente em caminhões para as forças armadas, embora com o propósito de entrar na produção em larga escala de carros de passeio e caminhões comerciais com a inspiração causada pela Ford, mas o envolvimento do Japão na Segunda Guerra Mundial adiou as pretensões da Toyota.

> Desde o início da produção de automóveis na empresa Toyota, os recursos financeiros, as máquinas e a própria demanda dos consumidores configuravam um quadro bem diferente da realidade da empresa de Ford. Enquanto empresas ao redor do mundo seguiam o exemplo da Ford, o Japão estava abatido pela Segunda Guerra Mundial, na qual o país sofreu o ataque que dizimou duas de suas maiores cidades, desestruturando sua sociedade e economia (SILVA, 2011, p. 19).

Ghinato (1996) conta que, com o final da Segunda Guerra em 1945, a Toyota retomou os seus planos de tornar-se uma grande montadora de veículos mas que, em qualquer análise, indicava-se uma monstruosa distância que a separava dos grandes competidores americanos.

Costumava-se dizer que a produtividade dos trabalhadores americanos era aproximadamente dez vezes superior à produtividade da mão-de-obra japonesa. Esta constatação não desanimou os japoneses, mas sim serviu para "acordar" e motivar a alcançar a indústria americana, o que de fato aconteceu anos mais tarde.

Conforme argumenta Ohno (1997), a produtividade americana tão superior à japonesa chamou a atenção para a forte diferença de produtividade, a qual só poderia ser explicada pela existência de perdas no sistema de produção japonês. A partir daí, o que se viu foi a estruturação de um processo sistemático de identificação e eliminação das perdas.

> Na Toyota, e em todas as indústrias manufatureiras, o lucro só pode ser obtido com a redução de custos. Quando aplicamos o princípio de custos, preço de venda = lucro + custo real, fazemos o consumidor responsável por todo o custo. Este princípio não tem lugar na atual indústria automotiva competitiva (OHNO, 1997, p. 30).

Silva (2011) explica a frase anterior de Ohno argumentando que, em um cenário de produção em massa como na Ford, a definição do preço de venda desta maneira poderia fazer sentido, afinal o custo real poderia ser abatido pelo grande volume de vendas. Porém, em um cenário de escassez de recursos como o da Toyota, a fórmula lucro = preço de venda - custo real aponta grande bom senso ao retirar do consumidor a responsabilidade por arcar com custos altos, passando à empresa a responsabilidade de lutar contra desperdícios para reduzir seu custo, uma vez que seu lucro depende disso.

Vários autores (SHINGO, 1996; OHNO, 1997; GHINATO, 2000; WOMACK *et al.*, 2004) apontam que, por mais de 20 anos, a Toyota trabalhou em busca da adequação de diversos conceitos ao seu processo produtivo, conceitos estes que serão apresentados brevemente nos próximos tópicos desta seção.

> A Toyota começou a receber o reconhecimento mundial somente a partir da crise do petróleo de 1973, ano em que o aumento vertiginoso do preço do barril de petróleo afetou profundamente toda a economia mundial. Em meio a milhares de empresas que sucumbiam ou enfrentavam pesados prejuízos, a Toyota emergia como uma das pouquíssimas empresas a escaparem praticamente ilesas dos efeitos da crise. Este "fenômeno" despertou a curiosidade de organizações no mundo inteiro: qual o segredo da Toyota? (GHINATO, 2000).

Ikonen (2011) destaca que, embora os valiosos esforços de Ohno e Shingo durante as décadas de 1950 e 1960 tenham formado a base do STP e sido o pontapé inicial para que o mundo começasse a se interessar em como obter resultados tão bons, trabalhos como os de Deming em relação ao Controle Estatístico da Qualidade durante a década de 1950, a Teoria das Restrições de Goldratt na década de 1980 e o fenômeno do Controle Total da Qualidade

disseminado por Deming, Juran e Feigenbaum, dentre outros, entre as décadas de 1970 e 1980, também foram incorporados ao STP.

Para Anders (2004), o que se tornou conhecido como Sistema Toyota de Produção também passou a ser chamado mais recentemente de *The Toyota Way* (eventualmente traduzido como Método-Toyota ou Estilo-Toyota) pela percepção de que os métodos de gerenciamento são aplicáveis também fora da manufatura tradicional. O autor também destaca o entendimento de que o Sistema Toyota de Produção ganhou fama no ocidente como "enxuto" graças a Womack *et. al.* em 1990, com a primeira edição do livro "A Máquina que Mudou o Mundo".

2.2 Conceitos-chave

Durante os anos de evolução do STP, foram desenvolvidas e aplicadas diferentes ideias e ferramentas em busca de melhorias de qualidade e eficiência dos processos da Toyota, descritas brevemente nos tópicos a seguir. Embora o conjunto completo de conceitos do STP seja demasiadamente extenso para o enfoque deste trabalho, uma visão rápida sobre os pontos mais importantes é necessária para o entendimento da filosofia por trás do pensamento enxuto e para melhor compreensão de sua ligação com desenvolvimento de *software* de forma enxuta.

Com exceção do termo Just-in-Time descrito a seguir, composto por palavras da língua inglesa, todos os outros termos são originalmente escritos em japonês, dada a origem da produção enxuta no Japão.

2.2.1 Just-in-Time

O Just-in-Time, abreviado como JIT, é "um sistema para produzir e entregar os itens certos, no tempo certo e na quantidade certa" (WOMACK e JONES, 1998, p. 349).

Através do JIT promove-se a produção "puxada", onde uma unidade de produção não deve "empurrar" nada para o cliente ou para outra unidade de produção ligada a esta. Ao invés disso, este cliente ou unidade "puxa" o produto necessário. Tal política evita superprodução: se o cliente ou unidade "puxa" apenas os produtos realmente necessários, a unidade produtiva só desperdiçará seu tempo e recursos se produzir produtos desnecessários. Além disso, quando os produtos estão prontos no tempo certo, o uso de estoques se torna inútil, uma vez que o cliente "puxa" o produto para si mesmo antes que a operação de estocagem – a qual não agrega valor ao cliente – precise iniciar (SHINGO, 1996).

> Just-In-Time significa que cada processo deve ser suprido com os itens certos, no momento certo, na quantidade certa e no local certo. O objetivo do Just-in-Time é identificar, localizar e eliminar as perdas, garantindo um fluxo contínuo de produção. A viabilização do JIT depende de três fatores intrinsecamente relacionados: fluxo contínuo, *takt time* e produção puxada (GHINATO, 2000).

Ohno (1997) considera o Just-in-Time um dos dois pilares do Sistema Toyota de Produção, sendo o outro pilar o Jidoka, a ser descrito posteriormente nesta seção.

2.2.2 Kanban

Conforme define Ghinato (2000), trata-se de um sistema de sinalização entre um processo-cliente e um processo-fornecedor onde se informa ao processo-fornecedor exatamente o que, quanto e quando produzir.

Kanban é considerada a ferramenta responsável pelo comando no STP, como uma ferramenta de gerenciamento visual, de rápida assimilação, a qual proporciona transparência ao fluxo de execução do processo (OHNO, 1997).

Ainda segundo o autor, é a utilização do cartão Kanban que viabiliza o JIT, flexibilizando a produção, tornando dispensáveis os documentos utilizados anteriormente para controlar o processo produtivo, e tornando dispensável o comando dos gerentes. Inclusive, para Womack *et al.* (2004), Kanban refere-se à própria ferramenta JIT, apenas com um nome diferente no STP.

O papel de gerente, criado originalmente por Taylor, é representado na Toyota como um treinador, orientador, desafiador dos trabalhadores. Os gerentes devem encontrar oportunidades e oferecer desafios técnicos para os trabalhadores que se auto organizam para encontrar as soluções de menor esforço e melhor desempenho (SILVA, 2011).

Observa-se, portanto, que os gerentes deixaram de executar a função controle, e parte das outras funções gerenciais – como planejamento da produção – também foram assimiladas pelos operadores durante o processo.

Segundo Ohno (1997), o uso dos cartões Kanban faz com que todos vejam e entendam melhor o processo. Com isso, os operadores passam a entender melhor sobre o processo e contribuir, tomando decisões próprias, aumentando sua participação, gerando maior desempenho e tornando-se mais satisfeitos com seu trabalho. Após o início do uso dos cartões Kanban, se estabelece uma relação de dependência com os trabalhadores que se tornam essenciais ao processo de maneira positiva.

Silva (2011) descreve as etapas de um processo produtivo utilizando a ferramenta Kanban da seguinte maneira:

- Ao receber uma solicitação de produção através de um cartão Kanban, a operação fornece imediatamente o produto devolvendo junto o cartão Kanban da solicitação;
- Após o fornecimento, este "supermercado" estará desabastecido, podendo ser realizada uma nova produção para reabastecê-lo;
- Para realizar a produção de reabastecimento são utilizados os cartões Kanban locais da operação, referentes aos insumos necessários à produção, que são usados para solicitar estes insumos às operações fornecedoras responsáveis;
- Cada cartão é levado a um fornecedor interno, que deverá devolvê-lo juntamente ao insumo solicitado;
- Após receber os insumos necessários, devidamente acompanhados dos cartões Kanban locais, a produção é iniciada e não deve ser interrompida até o seu término;
- Ao terminar, o produto pode ser usado para repor o "supermercado" ou ser entregue para alguma operação solicitante.

A otimização do processo produtivo graças à identificação de gargalos, demoras e esperas fica mais fácil a partir do monitoramento do uso dos quadros Kanban, pois estes dão visibilidade ao processo como um todo. As soluções de otimização passam a ser vistas por todos, o que gera maior colaboração dos operadores, que se sentem instigados a colaborar com a melhoria do processo e eliminação de desperdícios (WOMACK e JONES, 1998).

As regras de uso do Kanban destacadas por Shingo (1996) são:

- O processo-cliente solicita somente o número de itens indicado no cartão Kanban de movimentação do processo-fornecedor;
- O processo-fornecedor produz e fornece itens somente na quantidade de produção e sequência indicadas pelo cartão Kanban;
- Nenhum item é produzido ou transportado sem um cartão Kanban;
- O cartão Kanban sempre acompanha os próprios produtos;
- Produtos defeituosos nunca podem ser enviados para o processo seguinte. O resultado é mercadorias 100% livres de defeitos;
- Reduzir o número de Kanban aumenta sua sensibilidade aos problemas.

A Figura 2.1 na página a seguir, apresenta um exemplo de como o processo de reposição de materiais ocorre com a utilização dos cartões Kanban. O exemplo, adaptado a partir de explicação disponibilizada por Geric (2011), baseia-se na utilização de vários clipes

de papel em uma linha de produção fictícia, onde o operador retira itens do recipiente à esquerda como insumo para seu produto, até que este se esvazie.

Figura 2.1 - Exemplo da utilização de cartões Kanban.

Fonte: adaptado de Geric (2011).

No evento 1 da figura apresentada, o operador sinaliza através do cartão Kanban o término do material neste recipiente (neste exemplo, a sinalização está informatizada com a utilização de um leitor de código de barras integrado a um sistema de controle, mas poderia ser uma sinalização manual). No evento 2, demonstra-se que o operador continua a produção movendo o outro recipiente da direita para a esquerda. No evento 3, o recipiente vazio recebe novos materiais (esta ação poderia ser realizada por um trabalhador específico para reposição ou até por um fornecedor responsável por repor este material). No evento 4, o cartão Kanban é utilizado para sinalizar a nova condição do recipiente (novamente suprido). No evento 5, demonstra-se como o ciclo permanece em execução, com o novo recipiente da esquerda sendo esvaziado até a situação em que o evento 1 da figura ocorrerá novamente.

2.2.3 Kaizen

Kaizen significa “mudança para melhor”, “melhoria incremental”, “evolução”. As mudanças Kaizen são normalmente pequenas e locais, sugeridas pelos técnicos, que devem ser motivados a encontrar melhorias para o processo. Tais melhorias devem visar a eliminação ou minimização de custos e desperdícios e aumento da satisfação do cliente (SILVA, 2011).

Kaizen pode ser encarado como uma filosofia ou como práticas que enfocam a melhoria contínua do processo (IKONEN, 2011).

A seguir, apresenta-se uma descrição dos passos descritos por Silva (2011) como comportamentos para as mudanças Kaizen:

- Observar uma oportunidade de melhoria;
- Medir os benefícios e desperdícios atuais;
- Estimar os resultados da melhoria (prós e contras);
- Planejar a implantação da melhoria e medidas defensivas;
- Obter o consenso de todos;
- Implementar a melhoria;
- Medir os resultados;
- Comparar com a situação anterior;
- Obter consenso sobre os resultados;
- Prevenir recorrências;
- Padronizar a operação;
- Aplicar nas operações similares;
- Buscar novas oportunidades de melhoria.

2.2.4 Kaikaku

Kaikaku significa “melhoria radical” ou “revolução”. A técnica de Kaikaku visa planejar, implantar, avaliar uma mudança radical no processo que, diferentemente das mudanças Kaizen, são sentidas no processo como um todo, sendo necessário um maior planejamento. Portanto, tais operações são menos frequentes e devem ser conduzidas com o apoio de especialistas e a participação da equipe (SILVA, 2011).

Ikonen (2011) enfatiza que este tipo de mudança radical normalmente ocorre por causa da introdução de novas técnicas de produção, novos equipamentos, novas estratégias corporativas ou novos conhecimentos.

Tipicamente um processo de mudança radical Kaikaku é iniciado pela gestão da empresa, mas pode ser ocasionado também fatores externos, como mudanças nas condições do mercado (WOMACK e JONES, 1998).

2.2.5 Jidoka

Jidoka pode ser encontrado com traduções como “automação inteligente”, “automação com toque humano” ou “autonomação”, sendo considerado por Ohno (1997) o segundo pilar do Sistema Toyota de Produção.

Para Shingo (1996), é importante diferenciar “autonomação” de “automação”, dizendo que o primeiro está muito mais ligado a autonomia e “inteligência com toque humano”. Este autor caracteriza autonomação como uma pré-automação, pois não é limitada a processos automáticos, sendo utilizada mesmo em operações manuais.

Autonomação consiste em facultar à máquina ou ao operador a autonomia de interromper a produção sempre que alguma anormalidade for detectada ou quando a produção requerida for atingida, não sendo, portanto, um conceito restrito às maquinas, mas sim à autonomia oferecida a elas e aos seus operadores em parar a produção (GHINATO, 1996).

Jidoka previne tanto a superprodução quanto a produção de produtos defeituosos. Ao se parar a produção no momento exato de uma anomalia, avaliar os processos e entender as razões por trás do problema, pode-se com maior certeza garantir que o problema nunca ocorrerá novamente (LIKER e HOSEUS, 2008).

2.2.6 Poka-Yoke

Poka-Yoke, termo eventualmente encontrado com a grafia incorreta Poke-Yoke, é um termo que significa “anti-erro” (GHINATO, 1996).

Podendo ser compreendidos como uma extensão do conceito de Jidoka discutido anteriormente, são instrumentos criados por Shingo e Ohno com o objetivo de impedir que insumos fossem introduzidos de forma incorreta nas máquinas e que o resultado do processamento fosse monitorado continuamente (SILVA, 2011).

Os dispositivos Poke-Yoke possibilitam auto inspeção reforçada, que atua sobre 100% da produção através de controles físicos e mecânicos (SHINGO, 1996).

Ghinato (1996) discute que os dispositivos Poka-Yoke estão diretamente relacionados a um conceito maior, que é o CQZD (Controle da Qualidade Zero Defeitos), argumentando que o conceito de "Zero Defeitos" no STP é bem diferente do que foi consagrado no ocidente, uma vez que "na Toyota não é um programa, mas um método racional e científico capaz de eliminar a ocorrência de defeitos através da identificação e controle das causas".

Conforme apresentado por Shingo (1996), enumera-se quatro pontos fundamentais para a sustentação do CQZD:

- Utilização da inspeção na fonte, como um método de inspeção com caráter preventivo, capaz de eliminar completamente a ocorrência de defeitos pois a função controle é aplicada na origem e não sobre os resultados;
- Utilização de inspeção 100% ao invés de inspeção por amostragem;
- Redução do tempo decorrido entre a detecção de um erro e a aplicação da ação corretiva necessária;
- Reconhecimento de que os trabalhadores não são infalíveis, aplicando-se os dispositivos Poka-Yoke para o cumprindo da função controle junto à execução.

2.2.7 Andon

Andon refere-se a um sistema de notificações visuais para que gerenciamento, manutenção e qualquer outro trabalhador apropriado possam ficar sabendo e atuar ao ocorrer um problema de qualidade ou de processo (IKONEN, 2011).

Significando "luz" em japonês, representa relação direta ao conceito de Jidoka, sinalizando problemas na linha de produção que revelem de forma visual e simplificada o estado de máquinas e do processo.

Na Toyota, o primeiro passo para a autonomação é a identificação de problemas, onde cada membro da equipe possui meios para chamar atenção para um problema. Com o Andon, essa "chamada de atenção" torna-se simples para todos os membros da equipe (LIKER e HOSEUS, 2008).

Conforme explica Shingo (1996), nem todo problema precisa interromper a continuidade da produção. Alguns problemas podem ser configurados em dispositivos Andon para avisar ao operador sobre seu acontecimento, mas mantendo o processamento. Alguns problemas também podem ser apontados manualmente pelo operador através de interruptores que acionem o alerta de problemas Andon, normalmente algum tipo de iluminação utilizando-se de diferentes cores para sinalizar o ocorrido.

A Figura 2.2 apresenta um exemplo de Andon simples, concebido apenas com duas lâmpadas de lava, mais conhecidas pelo nome em inglês *lava lamp*. Enquanto a lâmpada verde permanece acesa, não existem problemas naquela estação de trabalho. Conforme a iluminação transita para a lâmpada vermelha, todos na planta podem facilmente visualizar que aquela estação de trabalho está enfrentando problemas que podem potencialmente atrapalhar – ou futuramente parar – a produção.

Figura 2.2 - Exemplo de utilização de *lava lamp* para sinalização Andon.

Fonte: adaptado de Clark (2007).

2.3 Pensamento enxuto

Segundo Womack e Jones (1998), o pensamento enxuto representa um modelo conceitual sobre o que ocorreu no Sistema Toyota de Produção, uma filosofia a ser seguida para se produzir de maneira enxuta, mesmo em cenários onde as ferramentas originais do STP não são aplicáveis de forma direta.

Conforme comenta Pardal *et al.* (2011), esta filosofia enxuta japonesa só começou a ganhar destaque no mercado americano por volta dos anos de 1980, quando as companhias japonesas chegaram ao mercado americano, principalmente dos setores eletrônico e automotivo, impulsionando os primeiros estudos sobre o assunto no ocidente.

> O próprio STP é provavelmente o mais famoso exemplo de como obter sucesso com a abordagem do pensamento enxuto. Um conjunto de conceitos enxutos, entretanto, não são o suficiente. O necessário é a construção de uma cultura enxuta em toda a corporação. Os princípios para se fazer as coisas na Toyota, chamados de princípios do Método-Toyota, constituem as crenças e os valores praticados pela cultura da Toyota. Esta cultura pode ser encapsulada em quatro itens: (1) filosofia de longo prazo (o propósito e as razões de existência da Toyota); (2) processos enxutos (os quais direcionam à excelência operacional) com ênfase na continua eliminação de desperdício; (3) desenvolver e desafiar pessoas e parceiros através de relações de longa duração; (4) aprendizado organizacional dirigido à solução de problemas e à melhoria contínua para geração de valor. Apesar de seu sucesso, o modo de fazer as coisas na Toyota está sempre mudando (LIKER e HOSEUS, 2008, p. 11).

Em empresas que aplicam o pensamento enxuto, o trabalho é realizado em fluxo, com uma gestão horizontal ao invés de funcional e vertical. Neste contexto, o trabalhador assume um sentimento de equipe e precisa conhecer o fluxo de trabalho como um todo, não apenas suas funções. Desta forma, se ocorre um gargalo no fluxo de trabalho em determinada função, outros trabalhadores podem se unir para ajudar o trabalhador sobrecarregado (SILVA, 2011).

Womack e Jones (1998) descrevem o conceito de valor como a "capacidade provida ao cliente no momento certo a um preço apropriado, conforme definido em cada caso específico pelo próprio cliente". Para eles, o grande diferencial do pensamento enxuto está justamente na importância dada à criação de valor e de como isto é enfatizado através do combate aos problemas que geram desperdícios.

2.3.1 Geradores de desperdício

Os três grandes problemas à geração de valor, que a Toyota chama de 3M's, são normalmente apresentados com seus termos originalmente japoneses, Muda, Mura e Muri, os quais serão descritos brevemente a seguir.

2.3.1.1 Muda

A palavra japonesa Muda pode ser traduzida literalmente como "completamente inútil", mas é entendida no contexto do STP como "desperdício", sendo o principal problema combatido pelo pensamento enxuto. "Identificação e eliminação de desperdícios é o principal foco do Kaizen (melhoria contínua), pois colabora para a redução de custos" (TERA PRUDENT SOLUTIONS, 2012, p. 1).

Conforme analisa Silva (2011), no início da produção de automóveis pela Toyota, os recursos financeiros, as máquinas e até a demanda dos consumidores configuravam um quadro bem diferente da realidade de empresas americanas como a Ford, o que levou a Toyota a considerar a utilização de hábitos e operações diferentes dos padrões praticados no

ocidente. Para resolver as dificuldades então encontradas, tanto os responsáveis pela produção quanto os próprios operadores, buscaram formas de eliminar os desperdícios, encontrando através de experimentos com tentativas e erros, padrões operacionais mais eficientes.

Para Ohno (1997), existem três tipos de trabalho executas pela organização: trabalho efetivo que agrega valor, trabalho que não agrega valor mas suporta o trabalho efetivo e trabalho desnecessário que causa perdas. O autor enfatiza que o trabalho efetivo deve ser otimizado, as operações que não adicionam valor mas que não podem ser eliminadas por suportar este trabalho efetivo devem ser alvo de melhorias e automatizações para consumir o mínimo de recursos possível, e os desperdícios de trabalhos desnecessários devem ser identificados e completamente eliminados.

Já Shingo (1996) foi mais radical ao considerar que todas as operações que não agregam valor ao cliente são consideradas desperdício e deve-se buscar sua erradicação. Para auxiliar na detecção dos desperdícios, o autor documentou uma relação com sete tipos de desperdício, sendo eles apresentados a seguir:

- **Superprodução:** produção além do demandado pelo cliente.
- **Espera:** refere-se ao tempo no qual uma operação aguarda o resultado de outra.
- **Transporte:** excesso de movimentação de produtos ou insumos pela planta.
- **Processamento:** realização de atividades a mais ou a menos do que o cliente deseja.
- **Estoque:** inadequação no estoque de insumos, produtos semiacabados ou acabados.
- **Movimentação:** excesso de movimentação dos trabalhadores.
- **Defeitos:** produtos com qualidade inadequada ou produzidos com falhas.

Esta relação de desperdícios tornou-se referência para adaptações do pensamento enxuto em outras áreas, inclusive no desenvolvimento de *softwares*, conforme será apresentado em detalhes na seção 4.4.

2.3.1.2 Mura

A palavra japonesa Mura pode ser traduzida como “desnível, irregularidade, falta de uniformidade”, sendo entendida no contexto do STP como desequilíbrio por excesso de variação na aplicação dos recursos, na carga de trabalho ou nos intervalos de execução das operações (TERA PRUDENT SOLUTIONS, 2012).

Para Morgan e Liker (2008), sempre ocorrem variações durante as operações. Estas devem ser estudadas para que ocorra sua compreensão e se dimensionem os recursos. Os recursos necessários devem estar disponíveis independente de variações, visando não baixar a qualidade do produto.

2.3.1.3 Muri

A palavra japonesa Muri pode ser traduzida como "irracionalmente, forçosamente, excessivamente", sendo entendida no STP como sobrecarga do trabalho de pessoas, de máquinas ou do processo como um todo (TERA PRUDENT SOLUTIONS, 2012).

Para Morgan e Liker (2008), este conceito refere-se a forçar processos, operadores e máquinas além dos seus limites naturais. Os autores exemplificam que processos sobrecarregados ocasionam "gargalos", aumentando o tempo de ciclo, gerando erros e tornando-se, portanto, imprevisíveis. Sobre pessoas sobrecarregadas, eles exemplificam que estas produzem trabalhos imprecisos, imperfeitos, com problemas de qualidade e riscos de segurança. E sobre equipamentos sobrecarregados, enfatizam que evidentemente estes causam defeitos e acabam tendo sua vida útil reduzida desnecessariamente.

Sobre o encadeamento entre os três termos, Womack (2006) reflete: "em resumo, Mura e Muri são atualmente as causas raízes do desperdício em muitas organizações". Ele prossegue argumentando que, embora originalmente tenha-se pensado que a sequência lógica do pensamento enxuto seria combater Muda, para então atuar sobre Mura e Muri, percebeu-se posteriormente "que Mura cria Muri, o qual enfraquece os esforços anteriores para eliminar Muda" e, portanto, a ordem de compreensão e atuação deveria ser Mura, Muri e Muda.

2.3.2 Princípios enxutos

> O pensamento enxuto pode ser explicado como a identificação do valor real, o alinhamento da melhor sequência de operações que criam o valor, a realização desta cadeia de operações sem interrupção ou demoras, somente diante da solicitação do cliente, buscando a melhoria contínua do processo através da eliminação de desperdícios e a redução dos custos de produção (SILVA, 2011, p. 22).

Womack e Jones (1998) disseminaram o termo "pensamento enxuto" como a solução para evitar os desperdícios citados, provendo uma maneira de oferecer aos *stakeholders*[1] o

[1] Stakeholders: indivíduos ou organizações ativamente envolvidos no projeto, ou cujos interesses podem ser afetados pela execução ou finalização do projeto, ou cujas ações podem afetar a condução ou os insumos entregues durante o projeto (PROJECT MANAGEMENT INSTITUTE, 2013).

que eles desejam, para isso necessitando de menos esforços, equipamentos, tempo e espaço. Para chegar a este objetivo, tais autores elencaram os cinco princípios descritos a seguir.

2.3.2.1 Identificação de valor

Este princípio é o primeiro passo considerado ao buscar a consolidação de uma filosofia enxuta nas organizações. Valor é o produto ou serviço que o cliente deseja ou precisa segundo seu ponto de vista, na quantidade, hora e local em que precisa e a um custo aceitável (WOMACK e JONES, 1998).

Para Pardal *et al.* (2011), esta definição deixa evidente que compete inteiramente ao cliente estabelecer as características específicas do produto ou do serviço que são esperadas, não devendo a organização tentar imputar em seus produtos coisas que considera importantes se não existem clientes que as justifiquem.

Silva (2011) reflete que essa especificidade é determinante não só para a concepção de produtos adequados a um cliente, mas também para a compreensão de que a qualidade apontada por um cliente pode ser irrelevante para outro – ou mesmo algo ruim. Este autor ainda cita que em produtos personalizados e que demandam um bom tempo de produção, como é o caso dos *softwares* sob encomenda, a questão temporal existente na definição de valor é ainda mais importante, pois as necessidades atuais do cliente podem mudar completamente até que o produto esteja pronto para utilização.

2.3.2.2 Mapeamento da cadeia de valor

Este princípio, segundo passo para a organização enxuta, trata-se de "traçar todas as ações, processos e as funções necessárias para transformar entradas em saídas, de modo a identificar e a eliminar os desperdícios" (PARDAL *et al.*, 2011, p. 4).

Para Silva (2011), a cadeia de valor representa o conjunto de operações do processo produtivo sendo analisado, sendo que o objetivo inicial da identificação destas operações é a obtenção de um maior conhecimento sobre o processo produtivo, mapeando-se:

- As informações necessárias de entrada e saída;
- Os materiais necessários e produzidos;
- O relacionamento entre as operações;
- O tempo de espera até o início da execução da operação;
- O tempo médio de execução de cada operação;
- Quanto cada operação agrega de valor ao produto sob o ponto de vista do cliente.

Para a realização de um mapeamento da cadeia de valor, conforme apresentado por Rother e Shook (2008), segue-se o caminho da produção desde os fornecedores até os clientes, cuidadosamente desenhando-se uma representação visual de cada processo no fluxo material e/ou informacional. Para os autores, "repetir este processo várias e várias vezes é o caminho mais simples – e o melhor que conhecemos – para ensinar a si mesmo e a seus colegas como enxergar o valor e, especialmente, as fontes de desperdício".

2.3.2.3 Fluxo contínuo de valor

Através da eliminação dos desperdícios, o fluxo remanescente – realmente essencial ao processo – deve fluir continuamente, sem as variações ou sobrecargas já citadas.

Segundo Womack e Jones (1998), estabelecer um fluxo contínuo de valor é fazer com que as atividades que agregam valor para o cliente sejam executadas em um fluxo constante, desde o início até o fim do processo. Inclusive, Ohno (1997) afirmar que tudo o que o STP faz em suma visa diminuir o tempo e os recursos investidos entre a solicitação do cliente e a entrega do valor esperado.

A definição de um fluxo contínuo tem como objetivo, portanto, eliminar as demoras e esperas entre operações, tornando o fluxo produtivo contínuo no sentido de ininterrupto, buscando-se assim eliminar Mura do processo, tornando o processo contínuo e estável, aumentando-se assim sua previsibilidade (SILVA, 2011).

> Em primeiro lugar, o fluxo contínuo oferece o maior retorno para os investimentos em termos de eliminação de desperdícios e redução do tempo de aprovisionamento. É também a área mais simples para começar a trabalhar. Nem é necessário estabelecer um sistema puxado se você puder criar fluxo contínuo (ROTHER e SHOOK, 2008).

Apesar da frase citada acima, estabelecer um sistema puxado faz parte dos princípios enxutos e refere-se ao próximo passo para sua aplicação.

2.3.2.4 Clientes puxam o fluxo

Em termos simples, um sistema de produção utiliza uma abordagem "puxada" quando o produtor não gera nenhum serviço ou produto até que um pedido seja feito pelo cliente, não havendo atividades antes de ser explicitamente necessário (WOMACK *et al.*, 2004).

Este princípio enxuto relaciona-se diretamente com o conceito do JIT, o qual segundo Ohno (1997) representa o sistema de produção puxada estabelecido para a Toyota, onde

qualquer tipo de produção só ocorre quando existe a demanda por um cliente, seja este interno (outro setor da empresa) ou externo (consumidor final).

Destaca-se sobre isso, ainda, que a comunicação entre os clientes e os produtores deve ser direta, transparente e o mais simples possível. No cenário ideal do JIT, a demanda deve chegar diretamente aos produtores, os quais devem gerar exatamente o necessário para atendê-la (PARDAL *et al.*, 2011).

2.3.2.5 Busca pela perfeição

Conforme apontam Womack e Jones (1998, p. 308), definem a busca pela perfeição como a busca pela "completa eliminação de Muda, para que todas as atividades ao longo da cadeia de valor efetivamente criem valor".

> A perfeição é um alvo móvel, nunca atingido, porém eternamente perseguido. Os passos anteriores não precisam ser atingidos de uma só vez. Ao contrário, pode-se crescer na medida do entendimento da filosofia e do aprendizado sobre o próprio processo. O pensamento enxuto dá suporte ao bom-senso, que deve ser estimulado a quebrar as barreiras do senso coletivo e das verdades inabaláveis (SILVA, 2011).

A remoção de tempo e esforço desperdiçados representa a maior oportunidade para melhoria de desempenho e direcionamento de foco para a real criação de valor. Estabelecer um fluxo contínuo e "puxado" inicia uma reorganização radical de processos individuais, mas os ganhos tornam-se efetivamente significativos quando todos os princípios ligam-se entre si. Quando isto ocorre, mais e mais camadas de desperdícios tornam-se visíveis, então o processo pode evoluir em busca do teórico ponto da perfeição, onde cada pequena ação efetivamente adiciona valor para o consumidor final. Desta forma, o pensamento enxuto representa um caminho para melhoria de desempenho sustentável e não um simples programa pontual de melhoria de processo (WOMACK e JONES, 1998).

O núcleo da busca por mudanças contínuas positivas é a atitude das pessoas: uma atitude de autorreflexão e autocrítica, com um desejo contínuo de melhorar. O melhor incentivo que uma organização pode oferecer para que as pessoas aprendam é disponibilizar um ambiente no qual os trabalhadores possam discutir abertamente seus problemas, chamar a responsabilidade de solução e propor contramedidas para prevenir que os problemas ocorram novamente (LIKER e HOSEUS, 2008).

3 Engenharia de software tradicional

Conforme definido pela IEEE Standards Association (1993), engenharia de *software* refere-se à aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de *software*, isto é, a aplicação de engenharia ao *software*, bem como o estudo de possíveis abordagens para tal.

Nesta seção, será apresentada a evolução histórica da área, seus métodos e processos aplicados tradicionalmente na indústria de *software*, bem como algumas reflexões sobre as limitações e os problemas frequentemente atribuídos a estes métodos, a fim de facilitar a comparação com as metodologias enxutas que serão apresentadas posteriormente.

3.1 Histórico

O termo engenharia de *software* foi empregado pela primeira vez, inclusive como título do evento, na NATO Software Engineering Conference de 1968, conferência organizada para discutir e propor reações ao que foi chamado na época de "a crise do *software*" (NAUR e RANDELL, 1969; SOMMERVILLE, 2007).

Como apontado posteriormente por Randell (2001), tornou-se visível a relevância da discussão e a concordância de todos participantes sobre a importância em divulgar ao mundo a seriedade dos problemas ali discutidos e a importância de conduzir o desenvolvimento de *software* como uma atividade de engenharia. Desta forma, cópias do relatório logo se espalharam rapidamente e chamaram grande atenção, ajudando não só a perpetuar o termo, como também fortalecer a estruturação da área.

Durante os anos que se seguiram, várias abordagens e subáreas foram propostas e aplicadas mundo a fora, transformando a engenharia de *software* de algo abstrato em uma disciplina fundamentada e sólida (SOMMERVILLE, 2007).

Destacam-se, ao final dos anos de 1980 e durante os anos de 1990, os trabalhos dos autores já citados Ian Sommerville e Roger Pressman com o lançamento das primeiras edições de seus livros, considerados a base teórica da área, documentando de forma aprofundada todas as facetas relacionadas à engenharia de *software*.

Em 2001, surge a primeira versão do SWEBOK, o Guia do Conhecimento em Engenharia de *Software*, como um trabalho em conjunto entre grandes órgãos relacionados ao setor, dentre eles ACM e IEEE, material que viria a se tornar, em 2005, o padrão internacional ISO/IEC 19759:2005 (ABRAN *et al.*, 2004). Desde então, tal padrão exerce o papel de guia de referência oficial para as abordagens tradicionais de engenharia de *software*.

3.2 Conceitos principais

Conforme define Sommerville (2007), a engenharia de *software* é uma disciplina de engenharia relacionada a todos os aspectos da produção de *software*, considerando-se um modelo de processo de desenvolvimento de *software* um conjunto de atividades cujo objetivo é o desenvolvimento ou a evolução do *software*, apresentadas em uma perspectiva específica como uma representação simplificada destas tarefas.

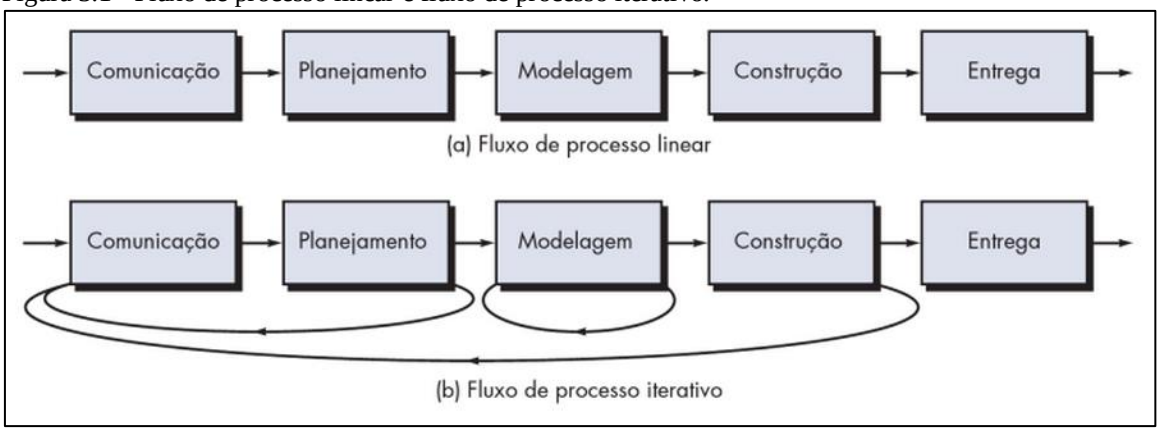
> Modelos são, de modo geral, representações simplificadas da realidade. Os modelos são vastamente utilizados na engenharia para definir características de produtos que serão posteriormente implementados, de acordo com cada modelo. A maior função de modelos de processo é desenvolver, de forma rápida, simplificada e objetiva, algo que é muito mais complexo em sua forma real (LOBO, 2009).

Sommerville (2007) enfatiza que modelos de processo não podem ser considerados descrição definitivas para a execução do processo de *software*, sendo abstrações do processo que podem ser usadas para explicar diferentes abordagens para o desenvolvimento de *software*. O autor considera que estes modelos podem ser encarados como *frameworks* de processo que podem ser aplicados e adaptados para criar processos mais específicos de engenharia de *software*.

Para Pressman (2011, p. 54), um aspecto importante de qualquer processo de *software* é o chamado fluxo do processo, que "descreve como são organizadas as atividades metodológicas, bem como as ações e tarefas que ocorrem dentro de cada atividade em relação à sequência e ao tempo".

O autor citado acima também destaca que modelos de processos prescritivos utilizam-se geralmente de fluxos lineares (Figura 3.1a) ou de fluxos iterativos (Figura 3.1b).

Figura 3.1 - Fluxo de processo linear e fluxo de processo iterativo.



Fonte: Pressman (2011, p. 54).

Em fluxos lineares, as atividades são executadas sequencialmente, culminando num *software* teoricamente concluído ao final da última atividade. Em fluxos iterativos, repete-se uma ou mais atividades antes de se prosseguir à próxima.

Conforme também observado através da Figura 3.1 apresentada, Pressman (2011) define que uma metodologia genérica para a engenharia de *software* estabelece estas cinco atividades metodológicas: **comunicação**, **planejamento**, **modelagem**, **construção** e **entrega**. Já para Sommerville (2007), quatro atividades fundamentais podem ser utilizadas para representar uma metodologia genérica para a engenharia de *software*, as quais são: **especificação**, **projeto e implementação**, **validação** e **evolução**.

A seguir, serão apresentados os principais modelos de processo prescritivos e um resumo de suas concepções.

3.2.1 Modelos de processo prescritivos

Modelos de processo prescritivos foram a primeira proposta efetiva para trazer ordem ao caos existente na área de desenvolvimento de *software*, tendo sido historicamente uma considerável contribuição para a estruturação da engenharia de *software*, fornecendo um roteiro razoavelmente eficaz para as equipes (PRESSMAN, 2011).

Ainda segundo o autor citado acima, todos os modelos de processo de software podem acomodar as cinco atividades metodológicas genéricas citadas anteriormente, porém com uma ênfase diferente a essas atividades e com um fluxo de processo distinto que invoca cada atividade metodológica (bem com tarefas e ações dentro dessas).

Por fim, Pressman (2011) explica que estes modelos denominam-se "prescritivos" por prescreverem um conjunto de elementos de processo: desde as atividades metodológicas genéricas e o fluxo do processo já citados, até as ações, as tarefas, os insumos e os produtos de cada etapa, bem como mecanismos de controle para garantia da qualidade e gestão de mudanças. Outra característica marcante citada por ele sobre modelos de processo prescritivos é o fato de que a rigidez quanto à ordem e à consistência do projeto é dominante.

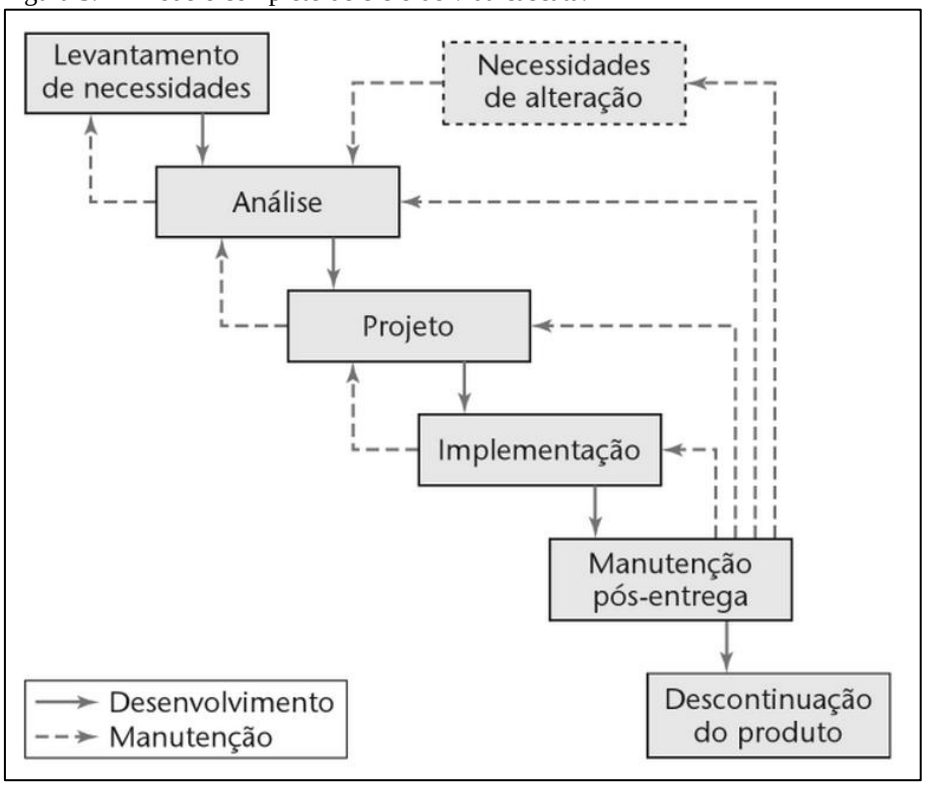
3.2.1.1 Modelo cascata

Dentre os modelos de processo prescritivos, o **modelo cascata** (em inglês, *waterfall*), algumas vezes chamado de "ciclo de vida clássico", foi o precursor na definição de uma abordagem sequencial e sistemática para o desenvolvimento de *software*, considerando as

atividades fundamentais do processo como fases separadas (SOMMERVILLE, 2007; SCHACH, 2010; PRESSMAN, 2011).

Embora originalmente proposto como um modelo onde existe a capacidade de retroação considerando-se o *feedback* entre as fases, como é possível observar na Figura 3.2, é visto como um modelo de processo linear, conforme explica Pressman (2011).

Figura 3.2 - Modelo completo de ciclo de vida cascata.



Fonte: Schach (2010, p. 51).

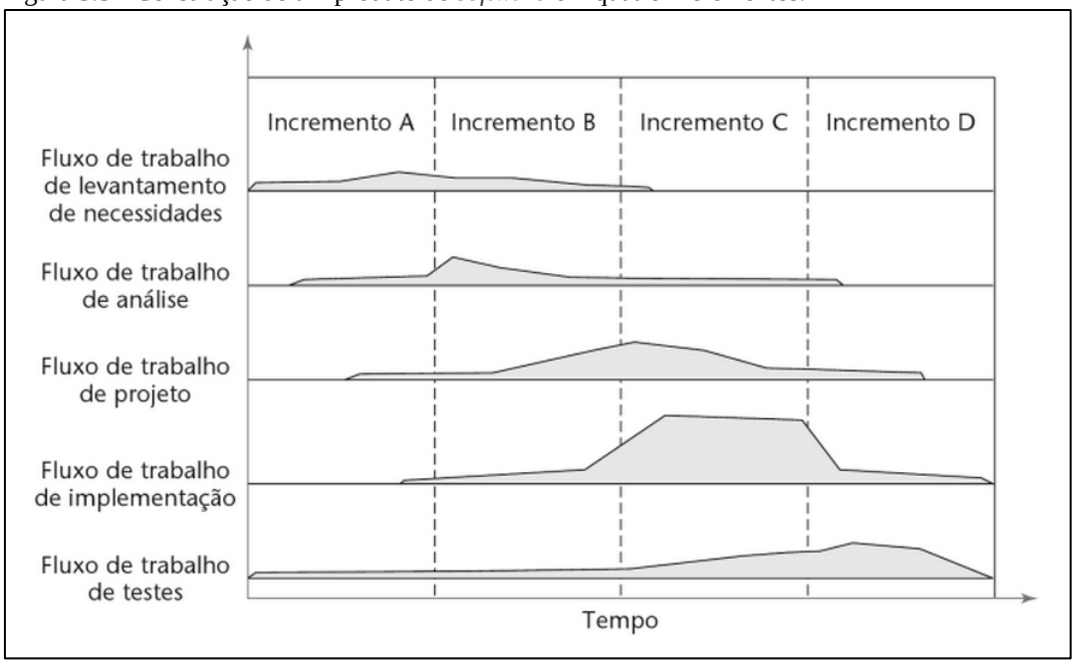
3.2.1.2 Modelo iterativo incremental

Outro modelo prescritivo historicamente conhecido e empregado é o chamado **modelo incremental** (PRESSMAN, 2011) ou **modelo iterativo-incremental** (SOMMERVILLE, 2007; SCHACH, 2010). Sua principal característica é combinar os fluxos de processo linear e paralelo, passando-se linearmente por todas as atividades metodológicas como ocorre no modelo cascata, porém repetindo-se esse fluxo inúmeras vezes, conforme necessário, de forma iterativa, e ao final de cada iteração gerando-se um incremento dos artefatos.

> Considere sucessivas versões de um artefato, por exemplo, o documento de especificação ou um módulo de código. Sob esse ponto de vista, o processo básico é iterativo, ou seja, produz-se a primeira versão do artefato, em seguida ele é revisado, então se produz a segunda versão, e assim por diante. O intento é que cada versão esteja mais próxima de nossa meta que sua predecessora e, finalmente, construamos uma versão que seja satisfatória (SCHACH, 2010).

A Figura 3.3, a seguir, apresenta um exemplo de como um fluxo de trabalho com cinco atividades poderia transcorrer ao longo do tempo se dividindo em quatro incrementos.

Figura 3.3 - Construção de um produto de *software* em quatro incrementos.



Fonte: Schach (2010, p. 43).

É interessante observar como Pressman (2011, p. 62) destaca que, embora originalmente descrito e classificado como um modelo prescritivo, um processo incremental tem seu foco voltado para a entrega de um produto operacional em cada um dos incrementos. “Os primeiros incrementos são versões seccionadas do produto final, mas eles realmente possuem capacidade para atender ao usuário e também oferecem uma plataforma para avaliação do usuário”. O autor então conclui que uma filosofia de fluxo de processo incremental também é utilizada em todas as abordagens enxutas/ágeis, que serão tratadas em detalhes neste trabalho durante a seção 4.

3.2.1.3 Modelo evolucionário

Conforme explica Sommerville (2007, p. 43), este modelo intercala as atividades de especificação, desenvolvimento e validação. “Um sistema inicial é desenvolvido rapidamente baseado em especificações abstratas. Este sistema é, então, refinado com as entradas do cliente para produzir um sistema que satisfaça as necessidades do cliente”.

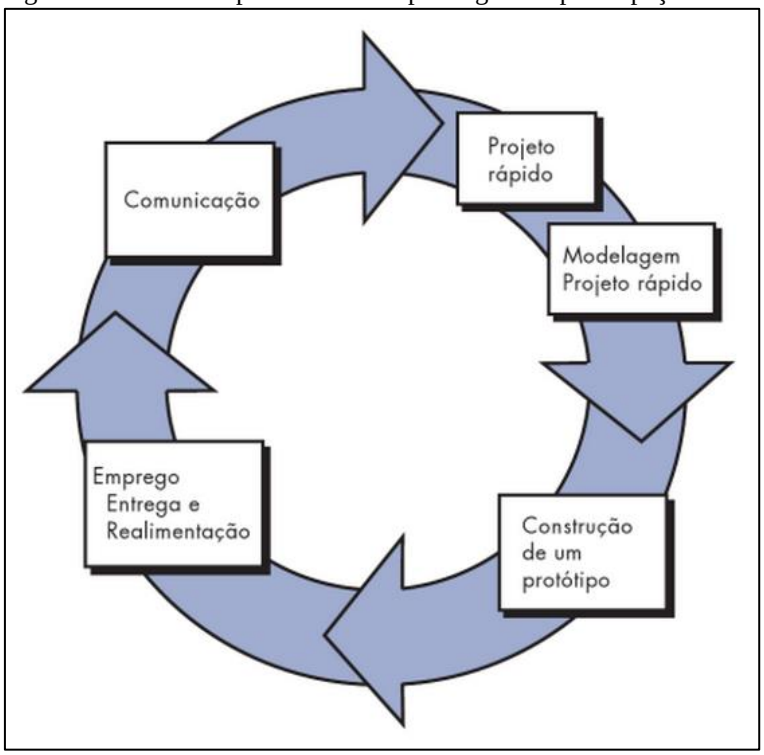
Quanto às maneiras de conduzir o processo de forma evolucionária, Pressman (2011) destaca a utilização do **paradigma da prototipação** e do **modelo espiral**.

Sobre o **paradigma da prototipação**, este autor descreve que frequentemente um cliente define uma série de objetivos gerais para o *software*, mas não identifica detalhadamente os requisitos, ou ainda um desenvolvedor sente-se inseguro quanto ao entendimento ou a eficiência de uma proposta de solução. Nestes cenários pode ser relevante o uso de prototipação, entendida como uma espécie de "projeto rápido" para demonstração da proposta de solução, para posteriormente ser refinado até tornar-se um produto real.

Pressman (2011) ainda destaca que prototipação não é necessariamente um modelo de processo isolado (embora possa ser utilizada dessa forma), mas sendo mais comumente utilizada como uma técnica passível de ser implementada como parte do processo de qualquer um dos modelos citados. Sommerville (2007) corrobora com esta visão e acrescenta que o objetivo é compreender os requisitos do cliente e, a partir disso, desenvolver melhor definição dos requisitos para o sistema, concentrando o protótipo na experimentação dos requisitos que estão mal compreendidos.

Abaixo, a Figura 3.4 demonstra graficamente um modelo do fluxo de processo seguindo-se o paradigma da prototipação para conduzir as atividades do projeto.

Figura 3.4 - Fluxo do processo com o paradigma da prototipação.



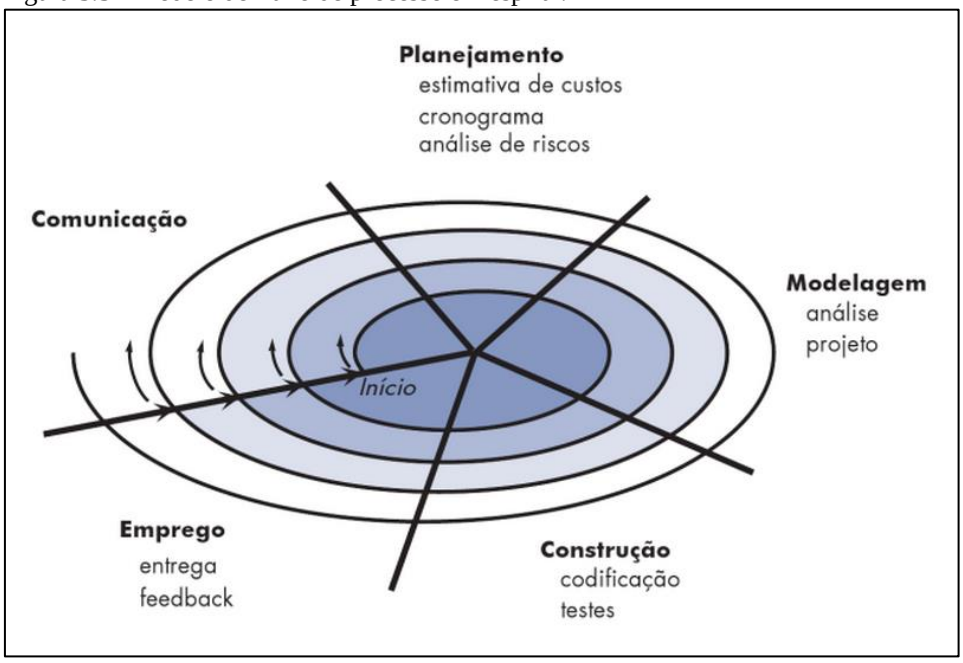
Fonte: Pressman (2011, p. 63).

Já sobre o **modelo espiral**, Pressman (2011) descreve como sendo um modelo de processo de *software* evolucionário que acopla a natureza iterativa da prototipação com os

aspectos sistemáticos e controlados do modelo cascata, fornecendo potencial para o rápido desenvolvimento de versões cada vez mais completas do *software*.

Abaixo, a Figura 3.5 apresenta graficamente a representação visual do modelo espiral típico, a qual inclusive justifica o nome do modelo, onde o incremento do *software* surge de forma evolutiva na medida em que as atividades metodológicas do processo vão se repetindo.

Figura 3.5 - Modelo de fluxo de processo em espiral.



Fonte: Pressman (2011, p. 65).

Conforme explica Pressman (2011), o primeiro circuito em volta da espiral normalmente resulta no desenvolvimento de uma especificação do produto, com as passagens seguintes usadas para desenvolver inicialmente um protótipo e depois o produto em si. Progressivamente, versões mais sofisticadas do *software* vão surgindo conforme as atividades metodológicas são repetidas com o contorno da espiral.

O autor citado acima ainda conclui que modelos evolucionários, de maneira geral, possuem como objetivo desenvolver software de alta qualidade, não apenas no conceito de garantir a satisfação do cliente, mas de garantir também o cumprimento de uma série de critérios técnicos. Para Sommerville (2007), observa-se características de modelos evolucionários em diversas vertentes, inclusive no Processo Unificado, descrito a seguir.

3.2.2 Processo unificado

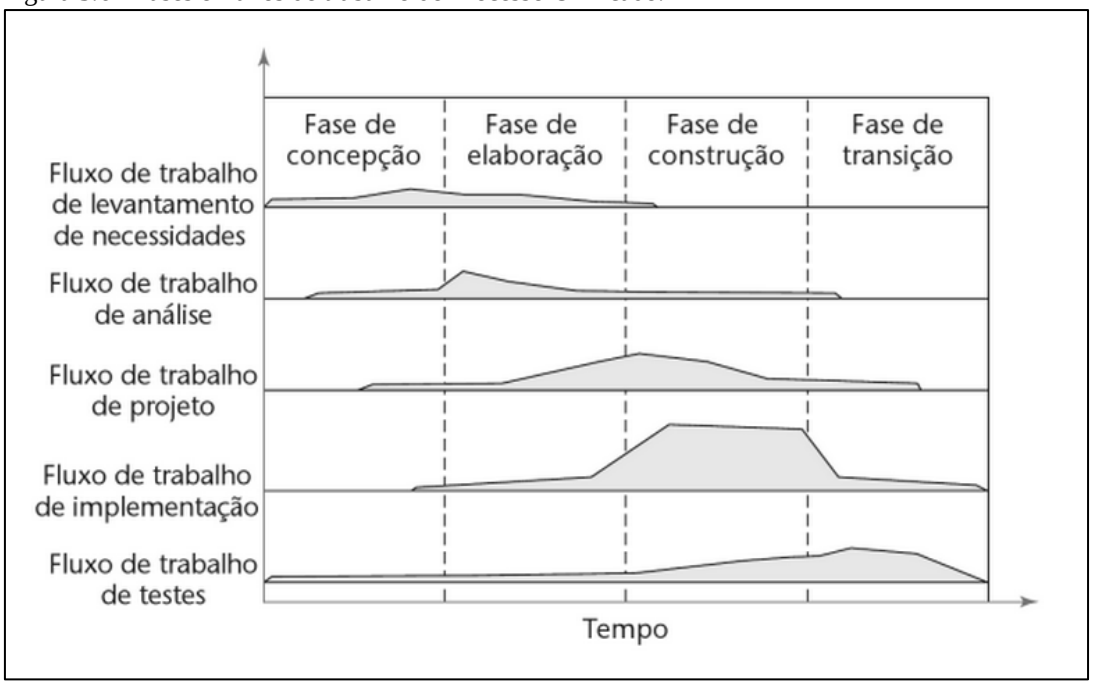
Conforme explica Schach (2010), o Processo Unificado também possui o nome Processo Unificado Racional, ou no inglês, **Rational Unified Process** (RUP), não só por que

seus três autores, cuja publicação original encontra-se em Jacobson *et al.* (1999), consideravam irracionais as outras metodologias conceituadas da época, mas principalmente porque os três eram executivos de alto escalão na empresa Rational Inc., especializada no desenvolvimento de *softwares* para modelagem de sistemas, a qual foi posteriormente adquirida pela empresa IBM em 2003.

Segundo os próprios Jacobson *et al.* (1999), o RUP surgiu da necessidade de um processo de *software* dirigido a casos de uso, centrado na arquitetura, iterativo e incremental, em um cenário onde a demanda por *softwares* cada vez mais complexos, enquanto ao mesmo tempo mais adaptados às nossas necessidades pessoais, torna-se mais enfático.

Conforme explica Schach (2010), o fluxo de processo do RUP estende-se por quatro fases, as quais podem ser entendidas como quatro iterações, cada qual gerando um incremento. Em cada uma destas fases, ocorrem fluxos de trabalho que correspondem às atividades genéricas que devem ser executadas para que o *software* seja criado. Como é possível observar na Figura 3.6, as atividades dos fluxos de trabalho são realizadas com maior ênfase em algumas fases em detrimento de outros, mas podem ocorrer e maneira paralela e se estender por fases subsequentes.

Figura 3.6 - Fases e fluxos de trabalho do Processo Unificado.



Fonte: Schach (2010, p. 86).

Enfatiza-se também que, ao final de cada fase, espera-se que artefatos tenham sido gerados. Existe grande quantidade de material oficial, anteriormente comercializado pela

Rational e atualmente disponível *online* pela IBM (IBM, 2013), descrevendo detalhadamente os fluxos e artefatos e oferecendo-se modelos de documentação eles.

Em Sommerville (2007), Schach (2010) e Pressman (2011) encontram-se descrições mais detalhadas de cada uma das fases, as quais estão resumidas a seguir:

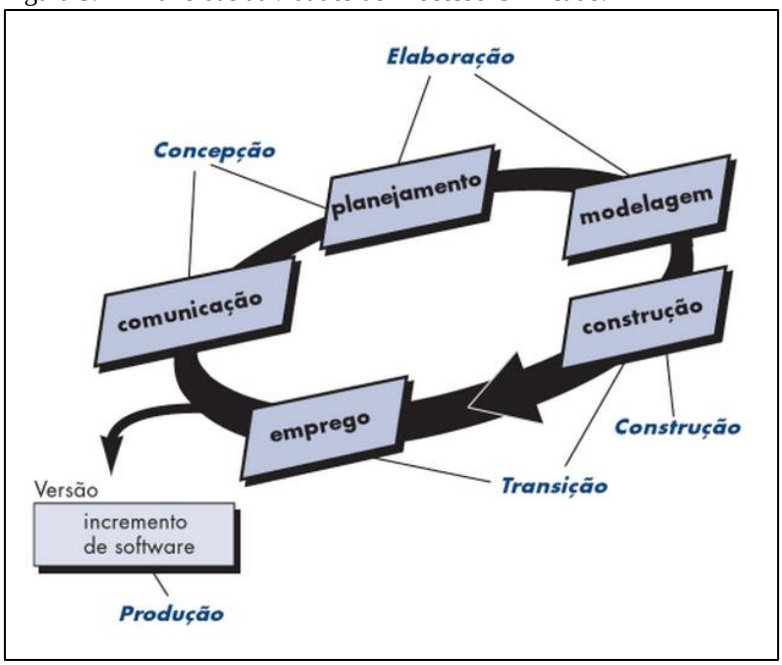
- **Concepção:** envolve comunicação com o cliente, identificação de entidades externas envolvidas (pessoas e sistemas), utilizando-se estas informações para avaliar a contribuição do sistema com o negócio. Colaborando-se com os *stakeholders*, identificam-se as necessidades de negócio iniciais e pode-se até se iniciar atividades de modelagem e de definição de arquitetura rudimentares. Se detectar-se que a contribuição do proposto *software* para o negócio não é adequada, o projeto pode ser cancelado ainda nesta fase sem ser considerado fracassado.
- **Elaboração:** o principal objetivo desta fase é desenvolver um entendimento do domínio do problema, elaborar um plano oficial para o projeto, elaborando-se em seguida os modelos usando diagramas em formato UML[2]. No auge desta fase, todos estes planos e documentos são revisados cuidadosamente para assegurar que escopo, riscos e datas estabelecidas permaneçam razoáveis.
- **Construção:** corresponde à produção efetiva do *software*, utilizando todos os insumos documentais gerados anteriormente como base para o desenvolvimento (ou aquisição) de componentes de *software* que tornem operacional os casos de uso especificados durante a elaboração do projeto. Os testes para garantia da qualidade do *software* desenvolvido também se enquadram prioritariamente nesta fase.
- **Transição:** refere-se à transferência do produto desenvolvido, do ambiente de desenvolvimento para o ambiente final onde será utilizado. Ignorado por boa parte dos modelos de processos, é uma atividade relevante e que pode se tornar onerosa e problemática. Também compreende a elaboração de materiais de apoio, como: manuais, guias para resolução de problemas e procedimentos de instalação.

Além destas fases já apresentadas, Pressman (2011) entende que também deve ser considerada uma quinta fase, denominada **Produção**. Nesta fase, monitora-se o uso contínuo do software, disponibilizando-se suporte para a infraestrutura operacional, realizando-se e avaliando-se relatórios de defeitos e solicitações de mudanças.

[2] UML: Unified Modeling Language, linguagem de modelagem unificada que contém uma notação robusta para a modelagem e o desenvolvimento de sistemas, a qual segue o paradigma arquitetural da orientação a objetos, amplamente difundido na indústria de *software* (PRESSMAN, 2011).

Desta forma, para o autor citado, o fluxo de processo do RUP poderia ser apresentado graficamente conforme a Figura 3.7.

Figura 3.7 - Fluxo das atividades do Processo Unificado.



Fonte: Pressman (2011, p. 73).

Embora visualmente diferente do apresentado anteriormente por Schach (2010), inclusive lembrando o fluxo de processo de modelos evolucionários, efetivamente só diferente pelo entendimento de que existe esta quinta fase adicional.

3.3 Principais problemas e limitações

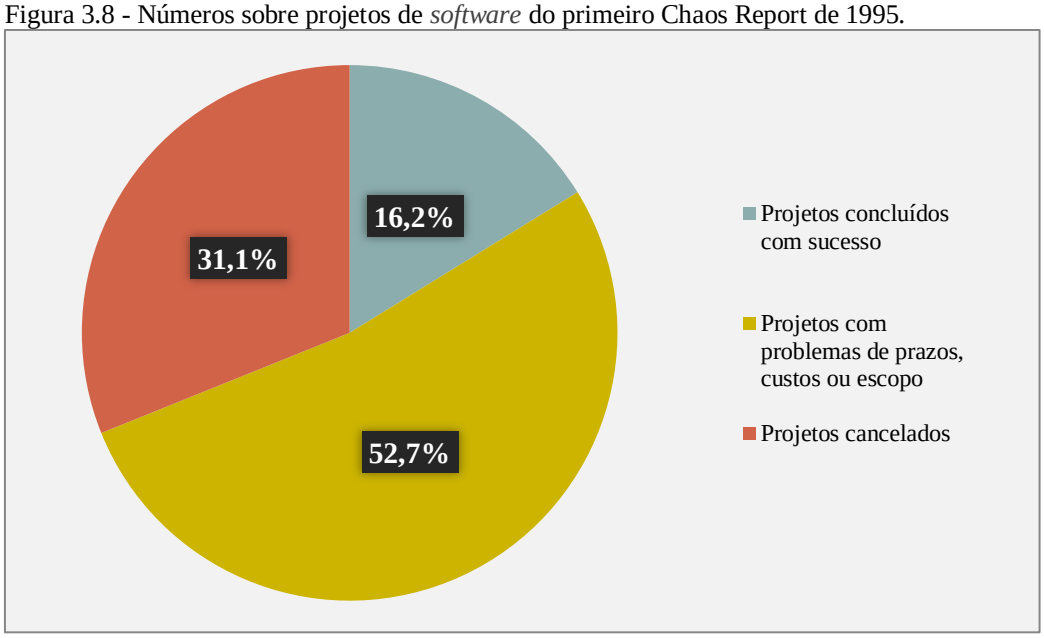
Pressman (2011) aponta que questionamentos à engenharia de *software* prescritiva não são recentes, encontrando-se argumentações contrapondo suas vantagens desde meados da década de 1990. A seguinte reflexão de Nogueira *et al.* (2000), após analise de várias pesquisas da década citada, discute como o excesso de ordem pode ser prejudicial.

> A fronteira do caos é definida como um estado natural entre ordem e caos, um grande compromisso entre estrutura e surpresa. Esta fronteira pode ser visualizada como um estado instável, parcialmente estruturado... Instável porque é constantemente atraído para o caos ou para a ordem absoluta. Temos uma tendência de pensar que ordem é o estado ideal da natureza. Isso pode ser um erro. Pesquisas em teoria organizacional, gerenciamento de projetos e economia suportam a teoria de que a operação longe do equilíbrio gera criatividade, processos auto-organizados e lucros crescentes. Ordem absoluta implica ausência de variabilidade, o que poderia ser uma vantagem em ambientes previsíveis. A mudança ocorre quando existe uma estrutura que permita que a mudança possa ser organizada, mas tal estrutura não deve ser tão rígida a ponto de impedir que a mudança ocorra. Por outro lado, caos em demasia pode tornar impossível coordenação e coerência. Mas a falta de estrutura nem sempre implica em desordem (NOGUEIRA *et al.*, 2000, p. 8).

Os primeiros números concretos demonstrando quantidades preocupantes de projetos de *software* problemáticos foram apresentados por Standish Group (1995), após questionário conduzido no ano de 1994, respondido por gerentes executivos de TI de 365 empresas americanas de diversos portes em diversos segmentos da indústria, totalizando dados sobre 8.380 projetos de *softwares* que haviam sido geridos por estes respondentes.

Como resultado desta pesquisa, visualiza-se que apenas 16,2% dos projetos haviam sido concluídos de forma completamente bem sucedida: sem atrasos, sem custos maiores que o estimado e sem problemas de escopo. Por outro lado, somando-se os projetos concluídos com estes problemas citados com aqueles que fracassaram completamente e foram cancelados, apresenta-se o número assustador de 83,8% dos projetos. A Figura 3.8 apresenta graficamente estes números citados.

Figura 3.8 - Números sobre projetos de *software* do primeiro Chaos Report de 1995.



Fonte: adaptado de Standish Group (1995, p. 3).

Desde então, bianualmente o Standish Group realiza novo questionário e publica relatórios atualizando os números citados. Em Standish Group (2010), o número de projetos bem sucedidos é maior, compreendendo 26% dos projetos analisados. O número de projetos completamente fracassados caiu consideravelmente para 15%. Contudo, o número de projetos concluídos de forma desafiadora com os problemas já citados subiu para 59%.

Conforme avaliação do próprio grupo no relatório citado, inegável que ouve uma evolução na capacidade de utilização dos modelos prescritivos por parte das organizações e esta evolução apresenta-se em forma de resultados melhores do que os apresentados durante os primeiros anos da realização da pesquisa.

Contudo, a critério comparativo, o grupo também passou a avaliar o percentual de sucesso em projetos ágeis. Os dados do mesmo relatório apresentam 43% de projetos bem sucedidos, 45% de projetos desafiadores e apenas 12% de projetos fracassados. Ainda segundo Standish Group (2010), estes números deixam evidente como a complexidade gerada pelos métodos prescritivos, afeta negativamente de forma enfática os resultados e que métodos mais leves, ao invés de desorganizados, são mais eficientes.

É possível também encontrar discussões apontando que as abordagens tradicionais da engenharia de *software* são limitadas não apenas como processo de desenvolvimento, mas também limitadas para o gerenciamento do projeto como um todo.

Esta visão é apoiada por Wysocki (2011), quando argumenta que projetos geridos de maneira tradicional seguem um plano altamente detalhado elaborado antes de qualquer trabalho ser executado. O plano é baseado na concepção que o objetivo (ou seja, a solução desejada) é claramente conhecido e especificado previamente por completo. O sucesso é totalmente dependente da correta especificação de todos os objetivos durante a definição do projeto e durante as atividades iniciais de escopo, o que raramente é possível na realidade.

Em produtos altamente mutáveis ao longo do projeto, como é o caso dos *softwares*, este tipo de abordagem tende a limitar a flexibilidade na condução do projeto. Como já apresentado, mesmo autores clássicos da engenharia de *software* tradicional concordam que flexibilidade é algo inerente à natureza necessária a qualquer tipo de projeto de *software*. Evidencia-se portanto que, embora esta natureza seja conhecida, as abordagens tradicionais acabam agindo de maneira inversa e representando barreiras à flexibilidade do projeto em detrimento de planejar e seguir fielmente o plano estipulado.

Outra discussão interessante é conduzida por Cockburn (2002) sobre uma falha essencial de metodologias prescritivas em ignorar a fragilidade das pessoas. O autor destaca que engenheiros de *software* não são robôs, ou seja, por mais que existam prescrições altamente detalhadas (e teoricamente corretas) a se seguir, existe grande variação nos estilos de trabalho e diferenças significativas no nível de habilidade, criatividade, organização, consistência e espontaneidade. Afirma ainda que diferentes modelos de processo podem "lidar com as fraquezas comuns das pessoas com disciplina e/ou tolerância", sendo que processos prescritivos optam totalmente por disciplina e, como "consistência nas ações é uma fraqueza humana, as metodologias com disciplina elevada são frágeis".

4 Metodologias enxutas para software

Os negócios atuais operam em um ambiente global e suscetível a mudanças rápidas. Sendo o *software* parte de praticamente todas as operações de negócio, é essencial que todo projeto de um novo *software*, desde sua concepção até o termino de seu desenvolvimento, seja flexível e possa acompanhar tal cenário de mudanças constantes (SOMMERVILLE, 2007; PRESSMAN, 2011).

Em busca desta maior flexibilidade e a fim de enfrentar os diversos fracassos em projetos de engenharia de *software* vistos ao longo dos anos, conforme já apresentado anteriormente neste trabalho, muitas novas abordagens de desenvolvimento de *software* surgiram, principalmente na última década. Ainda segundo Pressman (2011), grande parte dos defensores destas abordagens convencionaram chama-las de metodologias ágeis, enquanto outros autores as nomeiam como metodologias enxutas, sendo apenas variações de nomenclatura, embora com os mesmos conceitos.

4.1 Metodologias enxutas ou ágeis?

Mais do que diferença de nomenclatura, Poppendieck e Poppendieck (2009) argumentam que, embora ideologicamente metodologias enxutas e metodologias ágeis sejam similares e com ideias que se suportam entre si, o que se discute quanto a agilidade é algo tendenciosamente americano, mas o que se discute quanto a conceitos enxutos é originalmente japonês. Ou seja, prosseguem os autores, determinados conceitos enxutos podem parecer contra intuitivos para ocidentais, onde o enfoque em curto prazo é muito mais forte. Talvez por isso, muita gente classifique desenvolvimento ágil de *software* como algo diferente, e até mais simples de aplicar, do que desenvolvimento enxuto, quando, na verdade, todos são estudos baseados em filosofias parecidas e com objetivos muito similares.

Já segundo Ikonen (2011), enquanto metodologias ágeis emergiram como solução para muitos dos problemas das metodologias convencionais, o pensamento enxuto surge com abordagens que complementam as abordagens ágeis, trazendo maior valor ao desenvolvimento de *software*. Nesta visão, as metodologias ágeis surgiram como resposta direta às metodologias convencionais e somente posteriormente conceitos do pensamento enxuto surgem de forma a complementar estas.

Já para Sillitti e Succi (2007) os termos são compreendidos de maneira inversa, onde as metodologias ágeis implementam em suas práticas os conceitos básicos do pensamento enxuto no desenvolvimento de *software*, enfatizando-se a satisfação do cliente e a melhoria

contínua do processo de desenvolvimento. Nesta visão, as metodologias ágeis seriam a implementação de práticas baseadas nos princípios enxutos, ou seja, uma evolução de quais técnicas aplicar a partir da filosofia enxuta.

Para Anderson (2012), esta confusão se iniciou justamente quando o Manifesto Ágil foi concebido, o qual será descrito no tópico 4.2 desta seção, uma vez que Robert Charette (um dos primeiros a estudar de maneira aprofundada o pensamento enxuto para a indústria de *software*) não pôde comparecer ao evento de sua concepção, perdendo assim a oportunidade de participar dessa reunião histórica e de oficialmente consolidar o pensamento enxuto como parte (ou como princípio) da filosofia ágil que estava ali emergindo.

Contudo, este autor ainda argumenta que no início do século 21, os princípios enxutos foram utilizados como a explicação para a adoção de metodologias ágeis, ou seja, sendo estes encarados como o argumento teórico da adoção de práticas ágeis. Somente mais recentemente o conceito de desenvolvimento enxuto de *software* alçou o reconhecimento como uma disciplina própria ao invés de um simples recorte teórico das metodologias ágeis. Finalmente, este autor conclui que, ao mesmo tempo em que o pensamento enxuto está lado a lado com as metodologias ágeis em seu surgimento e em seus objetivos, atualmente representa uma disciplina específica.

O próprio Charette (2010, p. 2), argumenta que tanto o conceito de "enxuto" quanto o conceito de "ágil" surgiram com a premissa de desafiar o pensamento convencional sobre o desenvolvimento de *software*. "Ao desafiar-nos nossas próprias certezas, abrimo-nos para novas fontes de descobertas e inovação", disse o autor.

Entretanto, ele também ressalta que, aparentemente, as metodologias ágeis vêm perdendo justamente essa característica, tornando-se cada vez mais "receitas prontas" sobre como conduzir seu modelo de processo, ou seja, estão se tornando prescritivas. Neste cenário, o pensamento enxuto se destaca e se diferencia, por naturalmente entender que não existe um modelo perfeito a ser seguido, mas uma mudança constante, uma busca incessante pela perfeição inalcançável, pela eliminação de todos os desperdícios imagináveis, pela entrega do valor absoluto. Na visão de Charette, apenas o pensamento enxuto possui este poder de continuar nos incentivando a desafiar como as coisas estão sendo feitas, enquanto as metodologias ágeis estariam se tornando os novos modelos prescritivos da moda.

Uma análise ainda mais aprofundada desta relação entre "ágil" e "enxuto" pode ser vista em Petersen (2010), onde o autor relaciona 26 princípios em 7 categorias, sendo elas: engenharia de requisitos, design e implementação, garantia da qualidade, versionamento e entregas, planejamento do projeto, gerenciamento da equipe e fluxo ponta-a-ponta. A partir

desta categorização, detecta quais princípios são citados em materiais que utilizam o termo "metodologias ágeis" e quais princípios são citados em materiais que utilizam o termo "metodologias enxutas". Conforme pode ser observado no Quadro 4.1 adaptado a partir deste estudo, destacam-se 15 princípios como recorrentes em ambas nomenclaturas, 6 princípios como exclusivamente ágeis e 5 princípios como exclusivamente enxutos. Desta maneira, o autor aponta que é possível afirmar que os termos são complementares, mas não é possível afirmar que são idênticos.

Quadro 4.1 - Comparativo entre metodologias ágeis e metodologias enxutas.

Grupos e princípios	**Existente em abordagens ágeis**	**Existente em abordagens enxutas**
Engenharia de requisitos		
Cliente como parte da equipe	X	
Requisitos em forma de metáforas/histórias	X	X
Design e implementação		
Refatoração/refabricação	X	X
Padrões de codificação	X	
Equipe proprietária do código	X	
Baixa dependência arquitetural		X
Garantia da qualidade		
Automação de testes	X	X
Programação em pares	X	X
Integração contínua	X	X
Revisões e inspeções	X	X
Gerenciamento de configuração	X	X
Versionamento e entregas		
Entrega incremental ao cliente	X	X
Separação entre versões internas e externas	X	X
Planejamento do projeto		
Iterações curtas	X	X
Planejamento adaptável às prioridades do cliente	X	X
Fatias de tempo pré-fixadas	X	X
Jogo de planejamento	X	
Gerenciamento da equipe		
Equipe agrupada localmente	X	X
Papéis multifuncionais e intercambiáveis	X	X
Semana de 40 horas	X	
Reuniões em pé	X	
Equipe organiza suas próprias tarefas	X	X
Fluxo ponta-a-ponta		
Mapeamento da cadeia de valor		X
Gerenciar estoque com teoria de filas e restrições		X
Existência do papel de engenheiro chefe		X
Sinalização Kanban em *software*		X

Fonte: adaptado agrupando-se diversos quadros de Petersen (2010).

Esta visão de que os termos são complementares é compartilhada por Fowler (2008), um dos idealizadores do Manifesto Ágil (o qual será apresentado posteriormente neste

trabalho), em seu *blog* pessoal. Para ele, independentemente das origens de cada metodologia existente atualmente, todas compartilham princípios e valores que enfocam principalmente em adaptabilidade a mudanças e enfoque nas pessoas.

> Os conceitos Lean e Agile estão profundamente interligados no mundo do *software*. Você realmente não pode falar sobre eles como sendo alternativas. Se você está fazendo algo de forma ágil, você está fazendo também de forma enxuta, e vice-versa. Agile sempre foi concebido como um conceito muito amplo, um conjunto básico de valores e princípios, que foram compartilhados por processos que podem parecer superficialmente diferentes embora com as mesmas bases. Você não utiliza Agile ou Lean, você utiliza Agile e Lean. (FOWLER, 2008).

Para Lane *et al.* (2012), em uma perspectiva industrial, um conjunto de valores enxutos para desenvolvimento de *software* podem ser aplicados a projetos diversos como forma de descobrir possibilidades enxutas do método que estiver sendo executado, sugerindo que conceitos enxutos podem estar "escondidos" em outras metodologias (inclusive prescritivas) e que muitas vezes já são utilizados, mesmo que não por completo, sem que as equipes saibam disso conscientemente.

É possível também encontrar argumentação dizendo que a ligação entre ser enxuto e ser ágil não é sempre bem entendida, porém quando se consegue eliminar as atividades que não agregam valor, a agilidade de resposta é consequência. "Em uma empresa enxuta, eliminar atividades que não agregam valor é muito mais importante que acelerar um processo ou atividade individual" (PARDAL *et al.*, 2011, p. 4).

O paradoxo fica ainda mais evidente ao se comparar visões como a dos autores Dybå e Dingsøyr (2009), para os quais desenvolvimento enxuto de *software* é mais uma dentre tantas outras metodologias ágeis que emergiram na última década, com visões de outros autores como Hibbs *et al.* (2009), Peterson e Wohlin (2010) e Middleton e Joyce (2012), os quais entendem o pensamento enxuto aplicado a *software* como algo independente de metodologias ágeis, que até podem servir como base teórica para a aplicação destas, mas não sendo necessariamente relacionados ou dependentes.

Visto que tantas visões díspares podem levar a compreensões divergentes, optou-se por seguir o conceito proposto por Pressman (2011) e, portanto, os termos "metodologias ágeis" e "metodologias enxutas" podem ser compreendidos como sinônimos no presente trabalho e, portanto, as abordagens apresentadas são provenientes de autores que utilizam ambos os termos ou qualquer um destes termos separadamente.

Nos próximos tópicos, será apresentada a evolução histórica dos estudos destas metodologias e do surgimento do Manifesto Ágil, as características genéricas que são

compartilhadas entre as abordagens existentes, bem como serão detalhadas de maneira mais aprofundada aquelas que ganharam maior destaque ao longo dos últimos anos na literatura da área, para elucidar suas técnicas e compreender suas particularidades a fim de permitir aprofundar a exploração das aplicações práticas discutidas no estudo de caso.

4.2 A concepção do Manifesto Ágil

Conforme explica Bassi Filho (2008), durante a segunda metade da década de 1990 surgiram iniciativas para a proposição de metodologias que foram na contramão do grande fluxo da indústria de *software* na época, que eram os modelos prescritivos. Estas metodologias concentraram-se mais nos fatores humanos e em entregar maior valor ao cliente, tendo sido inicialmente chamadas de "métodos leves".

Em fevereiro de 2001, um grupo formado por 17 desenvolvedores experientes, consultores e líderes de renome na comunidade internacional de desenvolvimento de *software* se reuniu em Utah, nos EUA, para discutir ideias e procurar alternativas aos processos burocráticos e pesados adotados pelas abordagens tradicionais da engenharia de *software* e da gerência de projetos, surgindo a partir desta reunião o Manifesto do Desenvolvimento Ágil de Software, também conhecido simplesmente como Manifesto Ágil (BECK *et al.*, 2001).

Ainda conforme os autores citados acima, o Manifesto Ágil foi concebido baseando-se em quatro valores básicos, reproduzidos a seguir:

- **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.

Estes valores foram reproduzidos propositalmente destacando-se as palavras à esquerda, pois estão grafados desta maneira no Manifesto Ágil. Segundo os autores, o entendimento desejado é que, mesmo havendo valor nos itens à direita, valoriza-se mais os itens à esquerda ao atuar com agilidade.

> Normalmente, um manifesto é associado a um movimento político emergente: atacando a velha guarda e sugerindo uma mudança revolucionária (espera-se que para melhor). De certa forma, é exatamente do que trata o desenvolvimento ágil (PRESSMAN, 2011, p. 81).

4.3 Características comuns entre as metodologias

Conforme já discutido, durante a evolução dos estudos de metodologias enxutas para o desenvolvimento de *software*, várias abordagens têm sido propostas com algumas variações conceituais ou quanto às técnicas para sua aplicação. Nos próximos tópicos, serão apresentadas as principais abordagens encontradas na literatura, porém antes convém destacar algumas características relevantes que são compartilhadas por todas.

Embora existam variações nas práticas empregadas por cada abordagem durante as etapas do ciclo de vida do *software*, algumas características gerais são compartilhadas por todas (ABRAHAMSSON *et al.*, 2003; MESO e JAIN, 2006), sendo elas:

- Incrementais (entregas pequenas com ciclos rápidos de desenvolvimento).
- Cooperativas (grande aproximação entre clientes e desenvolvedores).
- Simples (fáceis de aprender e modificar, documentadas suficientemente).
- Adaptáveis (possuem habilidade de reagir a mudanças de última hora).

Sobre o modelo de desenvolvimento incremental, convém definir que os incrementos são pequenos lotes de *software* implementado e, normalmente, cada uma destas pequenas versões do sistema são projetadas, desenvolvidas e disponibilizadas aos clientes a cada duas ou três semanas (SOMMERVILLE, 2007).

Sobre o envolvimento dos clientes em todo o processo, destaca-se a importância desta característica para obtenção de *feedback* rápido sobre a evolução dos requisitos. Os clientes podem se sentir parte integrante do projeto, pois participam efetivamente das decisões sobre como o *software* será desenvolvido, não somente sobre os requisitos do produto, mas também para todas as decisões tomadas sobre o produto e sobre o processo.

Esta característica de cooperação apoia e reforça a característica de simplicidade, pois assim minimiza-se a documentação, visto que se utiliza mais a comunicação informal do que reuniões formais com documentos escritos (MESO e JAIN, 2006), ao mesmo tempo em que documentações vastas de planejamento tornam-se desnecessárias, uma vez que o planejamento se restringe de maneira geral apenas ao lote do incremento corrente.

Além das características citadas, diversos autores, como Lindstrom e Jeffries (2004), Ceschi *et al.* (2005), Sato (2007), Dybå e Dingsøyr (2009), Abrahamsson *et al.* (2010) e Conboy *et al.* (2011), apontam que tais abordagens também compartilham como maior desafio o fator humano, ou seja, problemas relacionados a recrutamento, treinamento, motivação, gerenciamento e avaliação de produtividade representam fortes barreiras à adoção

de qualquer abordagem enxuta se não forem adequadamente compreendidos por todos os envolvidos. Os autores citam que habilidades de comunicação, de aprendizado rápido, de motivação e de autogerenciamento, em cada indivíduo da equipe, são determinantes para o sucesso de projetos com abordagens enxutas. Utilizar alguma destas metodologias normalmente significa "profundas mudanças culturais", principalmente em empresas já consolidadas, tarefa que pode demorar de meses a anos, conforme analisa Cukier (2010).

Tal perspectiva é abordada de forma aprofundada por Conboy *et al.* (2011), onde se exploram tais desafios relacionados às pessoas em estudo envolvendo 17 organizações com equipes de Alemanha, China, Estados Unidos da América, Finlândia, Índia, Irlanda, Malásia, Reino Unido e Suíça, concluindo-se a forte relevância de fatores humanos no sucesso da utilização de todas as abordagens encontradas em tais equipes, as quais foram Lean Software Development, Extreme Programming, Scrum e Crystal.

A partir de tal estudo, apresentam-se também os principais contrastes observados entre metodologias convencionais e as metodologias ágeis estudadas, agrupados no Quadro 4.2.

Quadro 4.2 - Contrastes entre metodologias tradicionais prescritivas e metodologias ágeis.

Componente do projeto	**Tradicionais**	**Ágeis**
Controle	Centrado nos processos	Centrado nas pessoas
Estilo de gerenciamento	Comando e controle	Liderança e colaboração
Gestão do conhecimento	Explícito	Tácito
Definição de papéis	Individual – favorecimento quanto à especialização	Equipes auto-organizáveis – intercambialidade de papéis
Comunicação	Formal e somente se necessário	Informal e contínua
Envolvimento do cliente	Usualmente importante apenas durante etapa de análise	Crítico e contínuo
Ciclo do projeto	Guiado por tarefas e atividades	Guiado por funcionalidades do produto
Modelo de desenvolvimento	Modelo cascata, modelo espiral ou variações destes	Modelo de entrega evolucionária
Estrutura organizacional	Mecanicista (burocrática com alta formalização)	Orgânica (flexível, participativa e encorajando-se cooperação)
Tecnologia	Sem restrição	Favorece orientação a objetos
Localização da equipe	Predominantemente distribuído	Predominantemente agrupado
Tamanho da equipe	Frequentemente superior a 10	Usualmente menor que 10
Aprendizado contínuo	Pouco encorajado	Frequente
Cultura de gerenciamento	Comando e controle	Responsivo
Participação da equipe	Não compulsório	Necessário
Planejamento do projeto	Antecipado	Contínuo
Mecanismo de retroalimentação	Não se obtém facilmente	Numerosos mecanismos disponíveis
Documentação	Substancial	Mínima

Fonte: adaptado de Conboy et al. (2011).

Temprado e Bendito (2010) generalizam ao afirmar que a filosofia do pensamento enxuto é boa para se aplicar a qualquer empresa da área de TI, independentemente dos métodos de trabalho utilizados atualmente por esta. Tais autores, após realizarem estudo em uma pequena empresa de desenvolvimento de *software* da Suécia, afirmam que as combinações entre diferentes abordagens (as quais foram Lean Software Development, Extreme Programming, Scrum e Test-Driven Development) se mostraram vantajosas ao:

- Proporcionar melhoria da qualidade dos *softwares* produzidos pela empresa.
- Melhorar a imagem da empresa no mercado e a satisfação de seus clientes.
- Permitir acompanhamento mais efetivo por parte dos gestores.
- Eliminar desperdícios sem a necessidade de grandes investimentos.

Uma vez que as características gerais compartilhadas entre as diversas abordagens enxutas foram apresentadas, a seguir serão discutidas de forma mais aprofundada as principais abordagens discutidas na literatura nos últimos anos, as quais foram detectadas como aplicadas (totalmente ou parcialmente) na empresa estudada, conforme será apresentado na seção 5 deste trabalho.

4.4 Lean Software Development

A abordagem de Desenvolvimento de *Software* Enxuto, do inglês Lean Software Development (LSD) deriva do Sistema de Desenvolvimento de Produtos do STP, agregando a capacidade de adaptação rápida e efetiva a um grande conjunto de demandas do cliente, combinada à habilidade de produção regular e escalável, que continuamente melhora o processo interno e, ao mesmo tempo, é flexível para a produção de vários tipos de produtos e, portanto, sua adaptação para *softwares* se mostra muito viável, conforme conceitua Bassi Filho (2008). O autor ainda ressalta:

> A implantação desta abordagem implica mudanças estruturais na instituição e, por isso, a sua utilização depende muito da aceitação de suas ideias pelas esferas de comando da empresa. Isto porque o impacto das mudanças estará além do escopo das atividades individuais dos desenvolvedores e demais integrantes da equipe do projeto, ou seja, abrangerão o processo como um todo, sendo portanto impreterível que os principais envolvidos na tomada de decisão tenham seus ideais alinhados ao pensamento enxuto (BASSI FILHO, 2008, p. 71).

Segundo Anderson (2012), o termo Lean Software Development apareceu oficialmente, pela primeira vez, como o título de uma conferência organizada pela iniciativa ESPRIT da União Europeia, que ocorreu na Alemanha em 1992. Este autor ainda cita que,

possivelmente de maneira independente e sem relação a esta conferência, o autor Robert Charette sugeriu este termo em 1993, como parte de seus estudos explorando melhores maneiras de gerenciar riscos em projetos de *software*.

Para Highsmith (2003), embora as discussões neste sentido tenham iniciado nos primeiros anos da década de 1990, ideias enxutas para desenvolvimento de *software* só ganharam força como movimento a partir do início de 2003, e certamente enfrentando grande descrédito por parte dos engenheiros de *software*.

Desta forma, prossegue ele, foi muito importante Poppendieck e Poppendieck (2003) terem estruturado e oficializado esta abordagem, baseando-se em suas experiências pessoais com agilidade e pensamento enxuto no desenvolvimento de produtos industriais, proporcionando assim credibilidade adicional ao movimento ágil de *software* e, principalmente, provendo boas doses de ideias que podem fortalecer de várias formas todas as abordagens existentes, ou mesmo tentativas específicas de utilização híbrida entre elas.

Embora os créditos da concepção do LSD sejam, desta forma, comumente atribuídos a Mary e Tom Poppendieck, a ideia de estruturar os princípios do pensamento enxuto para o desenvolvimento de *software* já havia sido exposta também por Middleton (1995), em artigo onde o autor apresenta diversos casos de sucesso do STP, com destaque para a ferramenta JIT, em indústrias de manufatura de vários locais do mundo, argumentando que o processo de manutenção de *software* poderia efetivamente ser melhorado com aplicação do pensamento enxuto, sugerindo a necessidade de concepção mais detalhada de uma abordagem de desenvolvimento de *software* enxuto.

Posteriormente, Middleton (2001) utiliza o nome LSD em artigo homônimo onde inicialmente apresenta exemplos de adaptações do STP bem sucedidas ao longo da década de 1990 em corporações de diferentes áreas, prosseguindo com um estudo de caso conduzido com duas pequenas equipes de desenvolvedores na tentativa de aplicar os conceitos do pensamento enxuto nas atividades de manutenção de um *software* do setor financeiro diariamente realizadas por tais desenvolvedores. Com o resultado positivo obtido em ambas as equipes, principalmente em relação à motivação dos envolvidos, o autor conclui afirmando que "da mesma maneira que as técnicas enxutas transformaram partes da indústria de manufatura, elas tem potencial para transformar o desenvolvimento de *software*".

De qualquer maneira, aponta Highsmith (2003), o esforço de Mary e Tom em congressos, palestras e eventos diversos de engenharia de *software* foi determinante para modificar a forma como esta abordagem – e também as outras metodologias enxutas para *software* – são encaradas pelos profissionais e pelo mercado.

Embora o empenho do casal Poppendieck na divulgação do pensamento enxuto aplicado ao desenvolvimento de *software* seja marcante, esta metodologia foi influencia por diversas fontes, as quais são justificadas por Lane *et al.* (2012) da seguinte forma: sua fonte primária é a gestão de operações, área original onde o STP foi concebido e a partir da qual os princípios filosóficos básicos foram extraídos e adaptados. A área de desenvolvimento de produtos também pode ser destacada, uma vez que o desenvolvimento de *software* é muito próximo ao projeto de novos produtos e, portanto, *frameworks* genéricos de princípios de gerenciamento enxutos sintetizados para esta área representaram forte contribuição para a estruturação da abordagem LSD. E por fim, a própria área de desenvolvimento de *software* é uma fonte marcante, graças a muitos trabalhos publicados durante os anos de 1990 com diferentes iniciativas de adaptação de conceitos enxutos às abordagens existentes da engenharia de *software*, os quais foram discutidos principalmente por Charette e Middleton em suas propostas de concepção formal do LSD.

4.4.1 Princípios enxutos

> Princípios são verdades implícitas que não mudam ao longo do tempo ou do espaço, enquanto práticas são a aplicação de princípios a uma situação em particular. Práticas podem e devem variar conforme você se move de um ambiente para o próximo, e também variar conforme uma situação evolui (POPPENDIECK e POPPENDIECK, 2006, p. 19).

A base da abordagem LSD são os sete princípios enxutos, descritos originalmente em Poppendieck e Poppendieck (2003) e posteriormente expandidos e explicados mais detalhadamente em Poppendieck e Poppendieck (2006), baseados em uma adaptação dos princípios originais do pensamento enxuto do STP. Tais princípios apontam meios para aumentar a qualidade do processo e do produto, aplicando-se práticas que afetam o modo de executar tarefas e o fluxo de trabalho com foco na otimização dos resultados gerais do processo de produção (POPPENDIECK e POPPENDIECK, 2006). Tais princípios elaborados por eles serão apresentados de forma sucinta, a seguir.

4.4.1.1 Elimine o desperdício

Desperdícios são quaisquer atividades realizadas no processo que não acrescentam valor ao produto na percepção do cliente (OHNO, 1997). O próprio autor declara o STP um "sistema de gerenciamento para a eliminação absoluta de desperdícios", da seguinte maneira:

> Tudo o que estamos fazendo é olhar a linha do tempo do momento que o freguês nos entrega um pedido até o ponto em que recebemos o dinheiro. E estamos reduzindo essa linha do tempo removendo os desperdícios que não agregam valor (...) o que se refere a eliminar todos os elementos de produção que só aumentam os custos sem criar valor adicional – por exemplo, excesso de pessoas, excesso de estoques e excesso de equipamento (OHNO, 1997, p. 11 e 71).

Dada a importância deste conceito no STP, tal princípio demonstra-se o mais importante e de maior amplitude dentre todos os princípios do LSD, refletindo mudanças em diversos conceitos do ciclo de vida do *software* e na maneira como se encaram tanto o processo quanto o produto, incluindo-se aí todos os artefatos intermediários.

Este princípio pode ser resumido na afirmação: "cada etapa e atividade realizada no processo deve necessariamente contribuir para que o produto seja construído mais rapidamente, com mais qualidade ou a um custo mais baixo (BASSI FILHO, 2008, p. 72)".

A seguir, exemplos de desperdício no processo de desenvolvimento de *software*:

Funcionalidades incompletas representam desperdício, pois necessitam de esforços para serem iniciadas e conduzidas, embora não adicionam valor ao *software* por não serem encerradas. Normalmente trechos de código incompletos tendem a se tornar obsoletos, tornando-se mais difíceis de serem integrados ao restante do código.

Para Poppendieck e Poppendieck (2006), trabalho parcialmente realizado no desenvolvimento de *softwares* é o grande vilão entre as formas de desperdício, fazendo-se uma analogia com o problema de estoques inadequados da produção enxuta: pode ser perdido, cresce de forma obsoleta, esconde problemas de qualidade e "rouba" dinheiro desnecessariamente.

Quanto maior o tempo entre o início da codificação e o seu término, maior a chance de que os programadores lembrem-se menos a respeito da intenção inicial do código. Mesmo se concluídas posteriormente, não deixarão de representar desperdício, pois o esforço prévio poderia ter sido realizado mais tarde em troca da implementação de outras funcionalidades que entrariam em produção primeiro e, portanto, agregariam valor desde o princípio (BASSI FILHO, 2008).

Antecipação de funcionalidades representa desperdício porque aumenta a complexidade do *software* desnecessariamente com mais código, mais esforços com testes e mais integrações. Este cenário também pode ser descrito como excesso de tentativas de antecipação de funcionalidades importantes para o futuro e não para o momento.

Mais funcionalidades criam mais pontos de defeito em potencial (prejudiciais para a qualidade do produto), sem a certeza de que alguém realmente usará os recursos extras,

determinando-se assim investimento de esforço injustificado (prejudicial para os custos). Conforme aponta Johnson (2002), cerca de 64% das funcionalidades são raramente ou nunca utilizadas pelos usuários dos *software* desenvolvidos, demonstrando-se assim a enorme quantidade de esforço que poderia ter sido evitado.

Excesso de processos representa desperdício, pois estes demandam recursos e aumentam o tempo para a conclusão das tarefas, principalmente em virtude de um possível excesso de documentação, pois se consome tempo para produzi-la, sem garantias de que alguém irá lê-la. Inclusive, mesmo se forem lidas, o próprio tempo empregado para ler vastas documentações representa maior geração de desperdícios no processo.

> Excesso de processos é uma das maiores causas de desperdício. Não gere documentação se não for realmente necessária. Se você precisar produzir algo que adicione pouco valor ao cliente, lembre-se de duas regras: mantenha isso pequeno, mantenha isso em alto nível. Para avaliar quando um processo é interessante, simplesmente verifique se alguém está esperando por isso (TEMPRADO e BENDITO, 2010, p. 13).

Documentos tornam-se desatualizados rapidamente no cenário de mudanças constantes em um produto como *software*, conforme já apresentado anteriormente. Além disso, tornam a comunicação mais lenta e reduzem o poder comunicativo, pois são um meio de comunicação de via única ao qual não é possível que escritor e leitor interajam em tempo real. Muitas vezes, documentos representam apenas formalismos burocráticos que não acrescentam valor ao *software*.

Bassi Filho (2008) resume a questão do desperdício com processos afirmando que estes demandam comunicação e atividades de gerenciamento e, quanto mais simples e objetivos eles forem, menos pessoas serão necessárias, menos etapas precisarão ser cumpridas até a conclusão de um ciclo e, portanto, o processo inteiro será mais rápido e barato.

Alternação entre tarefas representa desperdício, ocorrendo principalmente quando um desenvolvedor é associado a vários projetos ao mesmo tempo. Como consequência, um significante tempo para a alternância incorre até que o desenvolvedor reorganize seus pensamentos e consiga entrar no fluxo da nova tarefa a ser realizada e, desta maneira, as tarefas demoram mais do que o necessário para serem concluídas (POPPENDIECK e POPPENDIECK, 2006).

Tempo em espera por parte dos desenvolvedores representa desperdício, e pode ser causado por muitos fatores, como os citados por Silva (2011): tempo no início do projeto esperando os insumos iniciais para começar o desenvolvimento, tempo desprendido em

reuniões, tempo com leitura e atualização de documentos (conforme já discutido anteriormente), tempo aguardando resultado de revisões e testes, dentre outros.

Esperas durante as atividades são constantes e comuns em boa parte dos processos de desenvolvimento de *software*, e por isso Poppendieck e Poppendieck (2009) refletem que até pode parecer contra-intuitivo encará-las como desperdício. Porém, no sentido de que o objetivo do desenvolvimento de *softwares* é que o cliente veja o *software* desejado o mais rápido possível, qualquer espera deve ser encarada como desperdício e combatida com todo empenho pelos envolvidos.

Movimentações de pessoas, de documentos e de artefatos também representam desperdício, pois diminuem o tempo aplicado na produção do *software*. Poppendieck e Poppendieck (2006) exemplificam diversas situações de movimentações que causam desperdício, como: um desenvolvedor com um questionamento técnico e que precise se movimentar por muito tempo para encontrar uma resposta; cliente não acessível para responder questões sobre funcionalidades; resultados de revisões e testes dificilmente acessíveis ou localmente distantes dos desenvolvedores.

No contexto do desenvolvimento de *software*, não só problemas de localização e de excesso de deslocamento físico podem ser considerados movimentações desnecessárias, mas também restrições lógicas, ou seja, inacessibilidade de documentos, arquivos, *softwares* de apoio e até membros da equipe ou representantes do cliente por meios digitais (como comunicadores instantâneos) podem forçar a equipe a se deslocar fisicamente ou aguardar desnecessariamente para contornar estas situações.

Defeitos, por fim, representam desperdício considerável. Conforme aponta Abrahamsson *et al.* (2010), a correção de defeitos existentes é uma forma de desperdício principalmente pelo fato de que corrigi-los não evita que novos defeitos apareceram, o que pode causar um ciclo vicioso de consumo de tempo e recursos do projeto. Tão importante quanto corrigir defeitos rapidamente é detectar as causas, evitando-se que novos erros ocorram pelos mesmos motivos, sejam falhas de comunicação ou problemas no processo.

4.4.1.2 Amplifique o aprendizado

Este princípio refere-se à criação de conhecimento a partir de experiência adquirida. Extrair lições das experiências vividas pela equipe e incorporá-las ao processo, transformando dificuldades em fonte de conhecimento representa uma forma inteligente de evitar repetição de erros, contribuindo para o amadurecimento da equipe e do processo. Conforme comenta

Bassi Filho (2008), um processo com definições rígidas engessa o aprendizado, por isso é essencial que existam possibilidades para que o processo seja melhorado continuamente.

Para Schwaber e Beedle (2001), desenvolvimento de *software* é similar ao desenvolvimento de novos produtos, por ser uma atividade que gera algo único para o cliente, ao contrário do cenário numa linha de produção em massa onde o produto é previsível e imune a mudanças. Esta visão é endossada na abordagem LSD, onde destaca-se que desenvolver *software* é como criar uma receita, um processo de aprendizado, que envolve tentativas e erros (POPPENDIECK e POPPENDIECK, 2003).

A partir dessa analogia, torna-se relevante observar como organizações que têm se destacado no desenvolvimento de produtos compartilham um traço comum: elas criam conhecimento e o tornam acessível a toda a organização de uma forma concisa, não apenas tornando acessível o conhecimento explícito, como também encontrando formas de compartilhar o conhecimento tácito (NONAKA e TAKEUCHI, 2004).

O processo de aprendizado sugerido por Poppendieck e Poppendieck (2003) possui forte relação com o ciclo de melhoria contínua proposto por Deming (1986), também conhecido como modelo PDCA. O modelo propõe um ciclo onde se identifica o problema, localiza-se a sua causa, cria-se uma solução e a implementa, verifica-se os resultados, adaptando-se à nova realidade, em um ciclo contínuo que se inicia novamente com a identificação de novos problemas.

4.4.1.3 Adie comprometimentos

Tomar de decisões prematuramente é um modo falho de planejamento, visto que restringe o aprendizado, agrava o impacto de defeitos, limita a utilidade do produto e aumenta o custo de mudanças (THIMBLEBY, 2002).

Adiar comprometimentos no contexto de LSD significa manter a flexibilidade de adaptação em relação às mudanças, uma vez que ambientes de incerteza como o desenvolvimento de *software* dificultam previsões precisas, ou seja, ao se adiar decisões permite-se obter maior conhecimento sobre as necessidades, basear as escolhas em certezas ao invés de previsões e evitar planejar antes da hora.

Bassi Filho (2008) acrescenta a este princípio a sugestão de que, para retardar decisões durante a construção de sistemas, é importante que a equipe crie a capacidade de absorver mudanças, tratando os planejamentos como estratégias para atingir um objetivo e não como comprometimentos imutáveis. Assim, mudanças serão vistas como oportunidades para aprender e atingir as metas.

Este princípio tem ligação direta com a eliminação de desperdícios por antecipação de funcionalidades. Conforme citado anteriormente segundo Johnson (2002), cerca de 64% das funcionalidades implementadas em *softwares* são utilizadas raramente ou nunca, ou seja, as decisões por seu desenvolvimento foram antecipadas erroneamente e se tivessem sido adiadas, poderiam ter sido evitadas até que um maior conhecimento sobre as necessidades tivesse sido adquirido pelas equipes. Conforme citam Poppendieck e Poppendieck (2006), a melhor estratégia é evitar generalizações desnecessárias e fazer com que o *software* seja flexível apenas nas áreas mais propícias à mudança.

Dentre as práticas para se alcançar este princípio, Katayama (2010) destaca o uso da **Refatoração,** uma técnica sistemática para reestruturação do código existente, alterando sua estrutura interna sem que seu comportamento externo seja impactado, a fim de melhorá-lo de alguma forma.

4.4.1.4 Entregue rápido

O contexto de evolução tecnológica rápida do mercado atual, conforme já abordado na primeira seção do presente trabalho, modifica o enfoque necessário por parte dos fabricantes de *software*. Não são as maiores empresas que possuem vantagem para sobreviver neste cenário, mas sim as empresas mais rápidas.

Quanto mais cedo o produto final é entregue sem defeitos consideráveis, mais cedo pode acontecer a realimentação ao processo, incorporando-se mudanças rapidamente na iteração seguinte. Quanto menor o tamanho das iterações, mais favorecida a aprendizagem e a comunicação dentro da equipe. Velocidade proporciona também garantia do cumprimento das necessidades atuais do cliente, e não de necessidades antigas – e por vezes já desnecessárias. Os clientes certamente valorizam a entrega rápida de um produto de qualidade, ganhando a oportunidade de adiar a tomada de decisões até o momento mais propício (POPPENDIECK e POPPENDIECK, 2003; POPPENDIECK e POPPENDIECK, 2009).

Dentre as práticas que apoiam este princípio, Katayama (2010) destaca:

- **Iteração:** trata-se de um ciclo completo que engloba as etapas de projeto, codificação, testes e entrega de funcionalidades do produto. A cada nova iteração, a equipe se reúne para refletir sobre o progresso realizado, planejar novas funcionalidades a serem criadas, dividir funcionalidades grandes em pequenas tarefas. O desenvolvimento em iterações proporciona às equipes a auto-organização de suas tarefas e a comunicação do progresso ao restante da organização, além de

combater um risco comum nos projetos de *software*: a tendência do trabalho levar mais tempo do que o esperado (SHORE e WARDEN, 2007).

- **Folga:** trata-se de um tempo incluído propositalmente no planejamento da iteração, para que eventuais atrasos não atrapalhem a entrega. Além de permitir maior flexibilidade para possível replanejamento das tarefas, Selby (2007) possuir folga é fundamental para evitar que ocorra o efeito conhecido como *thrashing*, quando uma organização entrega menos valor nos momentos onde estão utilizando mais recursos que o necessário.
- **Kanban:** trata-se da criação de um fluxo contínuo de trabalho através utilização dos pequenos cartões sinalizadores, tornando visível o andamento das tarefas da iteração e sua distribuição nas etapas do ciclo de vida do projeto de *software*. Conforme aponta Poppendieck e Poppendieck (2009), sistemas Kanban foram idealizados de forma a limitar o trabalho em progresso, pois quanto maior esta quantidade, mais lento é o fluxo do processo.

4.4.1.5 Valorize a equipe

Uma empresa que respeita às pessoas, valorizando suas equipes, desenvolve bons líderes e certifica-se de que eles auxiliam a equipe, motivando, apoiando e direcionando para que elas alcancem os resultados esperados. A valorização da equipe significa oferecer este líder motivador, e não diretivo, ou seja, significa respeitar o intelecto e a capacidade das pessoas confiando a elas o cumprimento dos objetivos planejados através da auto-organização da equipe (ANDERSON *et al.*, 2003).

Poppendieck e Poppendieck (2006) reforçam que, em cenários de mudanças muito rápidas, é necessário trocar a liderança diretiva – onde o líder determina as providências e as técnicas para execução das tarefas – por liderança liberal – onde as pessoas possuem maior liberdade para decidir como executar as tarefas. Para que a equipe se mantenha coesa é necessário que o ambiente ofereça o suporte necessário às pessoas, permitindo que elas descubram qual a próxima tarefa que deve ser executada, sem a necessidade de um direcionamento do líder, bem como descubram facilmente a ocorrência de problemas em outros pontos da equipe que poderão causar impacto em suas atividades.

"No desenvolvimento de *software*, o valor também é construído ao longo de um fluxo que transcende os limites departamentais da TI e, portanto é preciso engajar todas as funções organizacionais envolvidas da concepção à entrega (KISTE e MIYAKE, 2013)".

Em Poppendieck e Poppendieck (2003) podemos encontrar a seguinte reflexão: respeitar as pessoas refere-se a respeitar seu intelecto e sua capacidade, oferecendo objetivos justos e confiando que a equipe pode se auto-organizar para atingi-los. Uma empresa que respeita as pessoas desenvolve bons líderes para que auxiliem a equipe com motivação, apoio e direcionamento, a invés de simplesmente ficarem tomando decisões técnicas no lugar de quem efetivamente possui competência, experiência e formação para isso.

É relevante, por fim, também citar o fato de que o balanceamento entre a vida pessoal e a vida profissional aumenta a performance do colaborador e da equipe, além de criar um ambiente de trabalho mais leve, criativo e interessante à equipe. Em um momento de emergência pessoal, é importante que a pessoa possa contar com o apoio da administração para se ausentar se necessário, e que a equipe se una em dedicação para resolver as atividades que este colaborador ficou impedido de concluir (TEMPRADO e BENDITO, 2010).

4.4.1.6 Adicione segurança

> O *software* só faz sentido se agrega valor ao cliente. Portanto, a identificação do valor desejado pelo cliente é essencial para o processo de desenvolvimento de *software*. Este conjunto de valores desejados faz sentido para o cliente dentro de um domínio de problema onde as funcionalidades interagem, os dados colaboram e o cliente sente-se confortável (SILVA, 2011).

Este princípio refere-se a aplicar medidas que ofereçam maior segurança ao processo, quanto a garantir a agregação de valor dentro do desejado pelo cliente. Inclusive, os conceitos estudados na disciplina de Garantia da Qualidade de *software*, como técnicas de Verificação & Validação, são relevantes neste sentido e já são discutidos há vários anos, inclusive em metodologias prescritivas, conforme é possível encontrar descrito de maneira aprofundada em Pressman (2011), principalmente entre os capítulos 14 a 16 deste autor.

Contudo, estas práticas tornam-se ainda mais fortes e enfáticas no pensamento enxuto, uma vez que todo o processo é direcionado para a geração de valor ao cliente.

Conforme explicado por Shingo (1996), existem basicamente dois tipos de inspeção: inspecionar após a ocorrência de defeitos e inspecionar para a prevenção destes. A ideia é corrigir problemas imediatamente quando ocorrem, ou até antes que ocorram, impedindo que se propaguem para outras etapas do processo e, o pior, que cheguem até o cliente.

Para Temprado e Bendito (2010), o famoso *slogan* “faça certo da primeira vez” é aplicado no desenvolvimento de software no sentido do código produzido se comportar exatamente como desejado desde o princípio e, para isto, práticas como **Integração Contínua**

e **Desenvolvimento Baseado em Testes** (a ser definido em mais detalhes no tópico 4.7) são convenientes e muito relevantes.

4.4.1.7 Otimize o todo

A maioria das teorias de gerenciamento de projetos de *software* são baseadas em teorias de decomposições: decomponha o problema em pequenas partes e melhore cada uma delas isoladamente (KATAYAMA, 2010).

Esta visão é combatida através deste princípio enxuto, conforme afirmam Poppendieck e Poppendieck (2006) que uma organização enxuta deve buscar otimização em toda a cadeia de valor, desde o momento em que recebe um pedido para atender a necessidade do cliente até o momento em que se entrega o *software*. Estes autores ainda afirmam que se a organização se concentra somente em otimizar as partes, sem favorecer uma visão global, é comum que a cadeia de valor seja prejudicada, com base em suas experiências como consultores em várias empresas de desenvolvimento de *software* dos EUA apresentadas no livro citado.

Curiosamente, conforme aponta Madison (2010), um problema recorrente em projetos ágeis é a falta de visão do todo, criando-se uma falsa visão de agilidade nas primeiras iterações do projeto, posteriormente resultando-se em constantes quedas de produtividade por aumentos consecutivos da complexidade para refatoração. O autor sugere que manter o papel de arquitetos de *software*, profissionais mais experientes, generalistas e com visão abrangente sobre toda a cadeia de valor do produto pode ser determinante para o sucesso da implantação de qualquer prática ágil no processo.

Esta afirmação é corroborada pelo já citado estudo de Petersen (2010), onde se demonstrou como as práticas relacionadas ao fluxo ponta-a-ponta, como o mapeamento da cadeia de valor, são técnicas exclusivas de abordagens baseadas na manufatura enxuta e, de maneira geral, não existentes dentre as práticas sugeridas em abordagens ágeis. O autor ainda sugere que estas práticas exclusivas são a maneira mais interessante de estender as metodologias ágeis para a obtenção de uma organização realmente enxuta.

Dentre as práticas que apoiam este princípio, Katayama (2010) destaca a utilização da técnica de **Mapeamento da Cadeia de Valor**, onde o fluxo do processo é minuciosamente analisado e os tempos totais das atividades (e entre as atividades) é medido e classificado entre tempo empregado necessariamente (o qual agrega valor) e tempo desperdiçado (o qual não agrega valor), para que então se busque a máxima eliminação deste desperdício entre atividades. O autor ainda cita a técnica de **Mapeamento de Histórias**, uma espécie de

adaptação do VSM para o mapeamento da relação entre as Histórias de Usuário existentes em diversas metodologias ágeis.

4.5 Extreme Programming

A abordagem conhecida como Programação Extrema, do inglês Extreme Programming (XP), foi formalizada com a ampliação do artigo original de Beck (1999) transformando-o no livro seminal lançado em 2000.

Segundo o próprio Beck (2000), sua concepção ocorreu a partir da consolidação de técnicas que vinham sendo discutidas desde o final da década de 1980.

Representa uma das propostas ágeis mais discutidas nos últimos dez anos e, conforme afirma Pressman (2011), é a abordagem enxuta mais amplamente utilizada no mundo para o desenvolvimento de *softwares*.

No cenário brasileiro, pesquisas recentes (COSTA, 2011; AGILCOOP, 2012) apontam que a XP se encontraria na verdade em segundo lugar, enquanto a abordagem Scrum, a ser apresentada posteriormente neste trabalho, detém a maior utilização no mercado nacional. Tais pesquisas ainda apontam uma quantidade considerável de aplicações híbridas entre as duas abordagens dentre as empresas respondentes.

Enquanto Pressman (2011) não apresenta dados para embasar sua afirmação sobre o uso de XP no mundo, as amostras das duas pesquisas citadas somam menos de 500 empresas nacionais de desenvolvimento de *software*, ou seja, afirmar qual abordagem é a mais utilizada atualmente torna-se algo inconclusivo e, inclusive, não é relevante para a presente pesquisa.

Segundo a definição de Beck (2004), a Programação Extrema é uma maneira leve, eficiente, de baixo risco, flexível, previsível, científica e divertida de desenvolver *softwares*, distinguindo-se de outras metodologias por:

- Seu rápido *feedback*, concreto e continuado a partir de ciclos curtos.
- Sua abordagem de planejamento incremental, que rapidamente surge com um plano geral que deverá evoluir ao longo do ciclo de vida do projeto.
- Sua flexibilidade para adotar a implementação de novas funcionalidades, respondendo às mudanças constantes das necessidades de negócio.
- Sua confiança em testes automatizados escritos por programadores e clientes para monitorar o progresso do desenvolvimento, para permitir a evolução do sistema, e para detectar defeitos mais cedo.

- Sua confiança na comunicação oral, testes e código-fonte para comunicar a estrutura do sistema e sua intenção.
- Sua confiança em um processo evolutivo que dura enquanto dura o sistema.
- Sua confiança na estreita colaboração de programadores com habilidades comuns.
- Sua confiança em práticas que funcionam tanto com os instintos de curto prazo dos programadores quanto com os interesses de longo prazo do projeto.

O autor ainda destaca que a XP é uma disciplina, por englobar não apenas princípios e filosofia, mas também determinar práticas que precisam ser aplicadas para que a abordagem seja efetivamente utilizada, e que tais práticas não são novidades criadas para esta abordagem mas sim o inverso: a abordagem é um trabalho de entrelaçar muitas ideias preexistentes (algumas tão antigas quanto a própria atividade de programação de *software*) e, neste sentido, seria possível dizer até que XP é uma abordagem conservadora, tradicional. De maneira antagônica, pessoas que se deparam com esta abordagem pela primeira vez costumam lutar contra ou, pelo menos, demonstrar desconfiança.

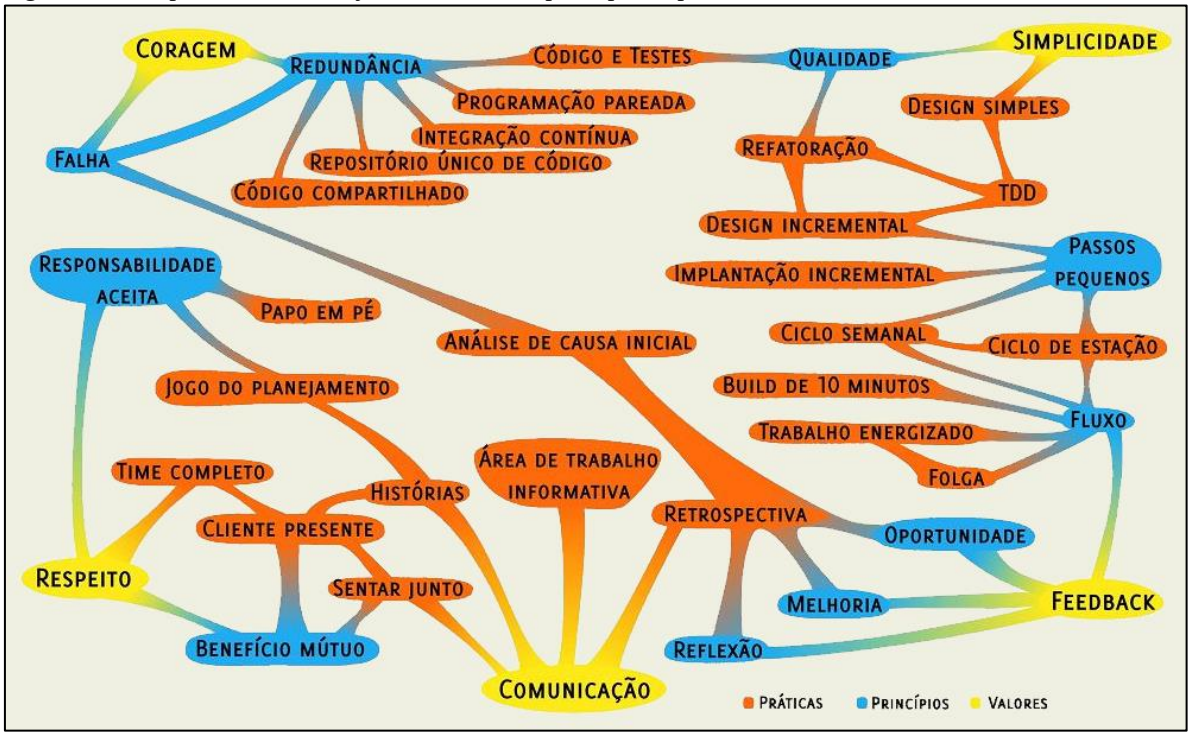
É possível encontrar autores (LINDSTROM e JEFFRIES, 2004; SATO, 2007; DYBÅ e DINGSøYR, 2009) discutindo que a adaptabilidade da abordagem XP está relacionada principalmente a este desafio cultural e social para as equipes e para os clientes, mas não a fatores explícitos como duração e tamanho do projeto, tamanho da equipe, linguagens e tecnologias escolhidas, dentre outros.

Porém, o próprio Beck (2004, p. xviii) afirma o inverso, ao restringir que esta abordagem foi "desenvolvida para funcionar com projetos que possam ser desenvolvidos por times de dois a dez programadores", ou seja, definindo-se um limitador sobre o tamanho da equipe de programação. Esta restrição de tamanho da equipe inclusive é constante também em outras abordagens enxutas, como será apresentado posteriormente.

Bassi Filho (2008), após estudar 4 cenários distintos no mercado brasileiro (na universidade, no setor público, no setor privado em uma empresa de pequeno porte passando por uma reestruturação e em uma *startup* buscando formas de inovação), amplia um pouco o número proposto por Beck, afirmando que as práticas ágeis demonstraram resultado efetivo em equipes com até 14 indivíduos.

A abordagem XP está baseada em cinco valores básicos, alguns princípios relacionados a estes valores e várias práticas que abrangem várias fases dos projetos de *software*. A Figura 4.1 na página a seguir, apresenta um mapeamento da relação entre estes conceitos, resumindo de forma concisa a estrutura da XP.

Figura 4.1 - Mapeamento da relação entre valores, princípios e práticas da XP.



Fonte: Cukier (2008).

Conforme pode ser observado na figura, uma prática pode estar relacionada a vários valores e princípios. Além disso, as práticas relacionam-se entre si, ou seja, existe for dependência entre elas, o que ajuda a compreender por que Beck (2004) afirma que não existe implementação parcial de Programação Extrema em uma organização: ou todos os valores, princípios e práticas são compreendidos ou não se trata de uma organização XP.

4.5.1 Valores

Conforme citado, a abordagem XP baseia-se em cinco valores que formam os pilares sobre os quais todos os princípios e práticas se estendem. Tais valores serão apresentados brevemente a seguir.

4.5.1.1 Comunicação

Para desenvolver um produto, inevitavelmente existe muita interação entre pessoas, como desenvolvedores, arquitetos, líderes, clientes etc. o que exige que os canais de comunicação sejam estabelecidos com qualidade para o sucesso do projeto.

Cockburn (2002) argumenta sobre a relevância de uma comunicação rápida para qualquer metodologia que se proponha a ser ágil, afirmando que “o ritmo de progresso de um projeto está ligado ao tempo necessário para transmitir uma informação da mente de uma

pessoa à outra" e também que "os custos de um projeto crescem proporcionalmente em relação ao tempo necessário para as pessoas se compreenderem".

> Quanto mais intensificada for a comunicação, mais facilmente serão aproveitadas as contribuições de cada membro e tarefas individualizadas poderão contar com a colaboração de outros participantes, fazendo com que críticas tornem-se contribuições que melhoram as soluções antes mesmo de serem implementadas (BASSI FILHO, 2008, p. 63).

Para conseguir comunicação efetiva, a XP enfatiza a colaboração estreita entre clientes e desenvolvedores, inclusive com a presenta de (pelo menos) um representante do cliente, de forma constante, no mesmo local da equipe de desenvolvimento, facilitando a troca de informações necessárias para que os desenvolvedores absorvam com maior facilidade os conceitos do negócio (TELES, 2005).

A comunicação entre os diferentes perfis envolvidos no projeto deve ser colaborativa e informal (verbal), evitando-se documentação volumosa como meio de comunicação. O estabelecimento de metáforas eficazes também é importante, ou seja, permitir que qualquer comunicação sobre os requisitos não sejam realizadas nem com excesso de jargões específicos do negócio e nem com termos técnicos de programação e TI de maneira geral, a menos que estritamente necessário (TELES, 2005; PRESSMAN, 2011).

Além disso, Beck (2004) também destaca que, independentemente da qualidade dos canais de comunicação, a quantidade de pessoas envolvidas sempre influencia. É inevitável que um maior número de pessoas eleve a dificuldade de saber o que os outros estão fazendo para não sobrepor, duplicar ou interferir no trabalho alheio. Por este motivo, o autor recomenda que projetos XP tenham número reduzido de participantes na equipe, conforme já citado anteriormente neste trabalho.

4.5.1.2 Simplicidade

Este valor baseia-se em buscar soluções simples para evitar esforços desnecessários. Paradoxalmente, conforme já citado em vários momentos neste trabalho, muitos projetos de *software* investem recursos em esforços desnecessários, seja por antecipação de funcionalidades que acabam não sendo utilizadas, seja por excesso de pragmatismo metodológico ao seguir padrões de codificação e arquiteturas, seja por duplicação de códigos similares para funcionalidades distintas. Teles (2005) destaca três fenômenos que ajudam a esclarecer as razões desses esforços desnecessários serem tão recorrentes, os quais são apresentados a seguir:

- Os projetos iniciam com um escopo fixo (e pretensiosamente supõe-se que é o escopo completo e correto), sendo que alterações neste são evitadas a todo custo durante o projeto;
- Os desenvolvedores criam soluções genéricas para "facilitar" possíveis alterações que possam ocorrer no escopo;
- Os desenvolvedores produzem funcionalidades adicionais na tentativa de antecipar o que o usuário "certamente" irá solicitar no futuro.

Desta forma, o valor da Simplicidade proposto pela XP está fortemente relacionado com os princípios enxutos já abordados de forma aprofundada na seção 4.4.1 acima, nos tópicos "Elimine o desperdício" e "Adie comprometimentos".

> Simplicidade e comunicação possuem uma maravilhosa relação de apoio mútuo. Quanto mais você comunica, mais claramente você é capaz de ver o que precisa ser feito e mais confiança você tem sobre o que realmente não precisa ser feito. Quanto mais simples é o seu sistema, menos você precisa comunicar sobre, o que leva à comunicação mais completa, especialmente se você for capaz de simplificar o sistema suficientemente a ponto de necessitar de menos programadores (BECK, 2004, p. 31).

Sommerville (2007) destaca que a XP anula, desta forma, um preceito fundamental da engenharia de *software* tradicional, que é "projetar para a mudança", onde a antecipação de mudanças e a implementação de um *software* projetado com o intuito de facilitar futuras modificações é parte constante de metodologias convencionais prescritivas.

Conforme resume Pressman (2011), para alcançar a simplicidade, a XP restringe os desenvolvedores a projetar apenas para as necessidades mais imediatas, criando um projeto simples que possa ser facilmente implementado em código. Se o projeto precisar ser melhorado para acomodar novas funcionalidades ou reduzir defeitos, ele sempre poderá ser "refabricado" mais tarde ou, no termo mais comum em materiais sobre XP, "refatorado".

4.5.1.3 Feedback

A palavra inglesa *feedback* normalmente é traduzida como "realimentação", quando no contexto de avaliação de resultados de processos produtivos, contudo como valor da XP não aparece traduzida nos materiais em português que discutem o assunto, como Teles (2004), Sommerville (2007), Pressman (2011), dentre outros diversos trabalhos observados durante a elaboração desta dissertação.

Conforme reflete Bassi Filho (2008, p. 48), quanto mais cedo impedimentos são encontrados e removidos, por menos tempo eles irão atrapalhar. “Quanto mais frequente forem as avaliações, do produto e do processo de desenvolvimento, mais rapidamente serão identificados os problemas e as soluções”.

Sato (2007) complementa que a XP promove ciclos curtos e constantes de Feedback em variados aspectos do desenvolvimento de *software* e que os valores se complementam, por isso o Feedback é parte importante da Comunicação e da Simplicidade.

4.5.1.4 Coragem

Para Beck (2004), se uma equipe não possui coragem para arriscar, para tentar diferentes soluções, para refatorar códigos que estão funcionando em busca de melhorias, etc. certamente não está alcançando o máximo de sua agilidade.

“Diante de uma dúvida entre três diferentes soluções, tentar todas parece ser um desperdício, porém esta pode ser a melhor forma de descobrir qual solução é mais simples e mais fácil de lidar” (SATO, 2007, p. 17).

Silva (2011) resume o valor da coragem na XP dizendo que, se algum código estiver fora do controle, a equipe deve ter coragem de jogar fora e refazer. Muitas vezes, é melhor jogar fora do que tentar corrigir, o que seria maior desperdício. E se alguma ideia promissora parece valer a pena, a equipe deve ter coragem para investir esforço nela.

Todas essas considerações podem parecer contra-intuitivas, mas Beck (2004) enfatiza que a Coragem por si só é um ótimo meio para o caos, mas a Coragem em conjunto com os outros valores é o diferencial entre as equipes reativas e as equipes verdadeiramente proativas.

> Ter coragem em XP significa ter confiança nos mecanismos de segurança utilizados para proteger o projeto. Ao invés de acreditar que os problemas não ocorrerão e fazer com que a coragem se fundamente nesta crença, projetos XP partem do princípio de que problemas irão ocorrer, inclusive aqueles mais temidos. Entretanto, a equipe utiliza redes de proteção que possam ajudar a reduzir ou eliminar as consequências destes problemas (TELES, 2004, p. 67).

4.5.1.5 Respeito

Na primeira edição de seu livro seminal, Beck (2000) não inclui Respeito como um dos valores da XP, limitando-os aos quatro citados anteriormente. Porém, na segunda edição do livro, Beck (2004, p. 35) amplia acrescentando este quinto valor que, segundo o próprio autor, é “um valor mais profundo, que se encontra abaixo da superfície dos outros quatro”.

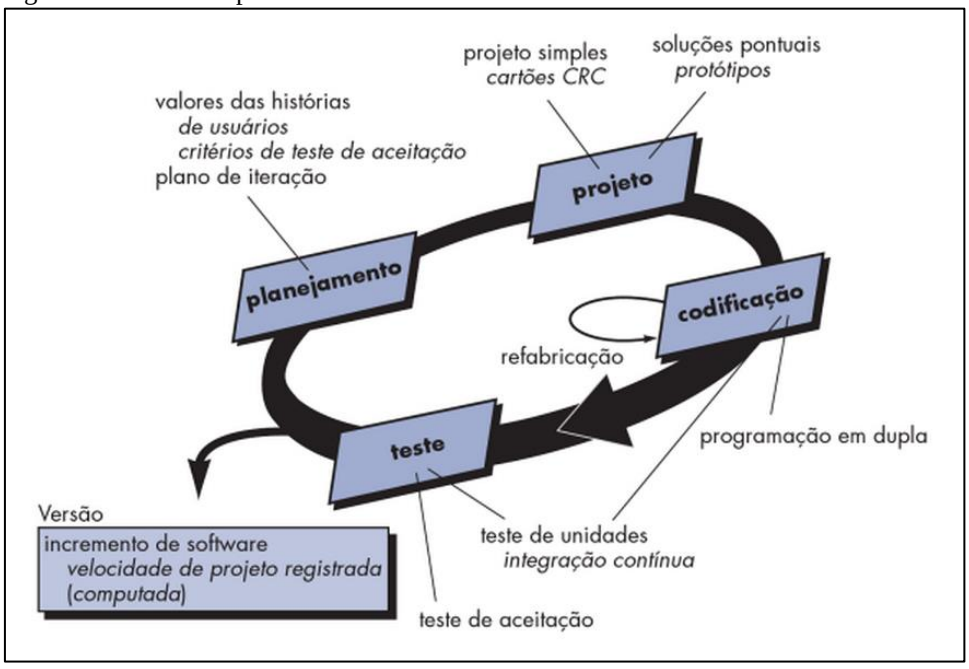
O autor destaca que os membros da equipe precisam se preocupar uns com os outros e com o que estão fazendo para que a XP não fracasse e que, apesar dessa característica ser compartilhada provavelmente pela maior parte das abordagens de desenvolvimento de *software*, a Programação Extrema é extremamente sensível a este valor.

É inevitável compreender que a excelência no desenvolvimento de *software* depende das pessoas, e elas devem se respeitar para conseguir extrair o máximo de seu potencial. A falta de Respeito pode influenciar negativamente vários pontos da adoção de XP, conforme exemplifica Sato (2007): comunicação sem respeito criará atritos internos; coragem sem respeito trará atitudes individualistas que vão contra o bem estar da equipe; a Programação Pareada (uma das práticas utilizadas na XP) é um exercício contínuo de respeito entre dois programadores; horas-extras excessivas irão impactar o ritmo sustentável da equipe; a colaboração entre a equipe e o cliente também exige uma comunicação aberta e respeitosa.

4.5.2 Fluxo do processo

A Figura 4.2 demonstra graficamente o fluxo do processo na XP.

Figura 4.2 - Fluxo do processo na XP.



Fonte: Pressman (2011, p. 89).

Conforme é possível observa na figura exibida, as fases do ciclo de vida de desenvolvimento de *software* são parecidas com as fases já preconizadas em modelos de processo prescritivos como o RUP, com uma diferença enfática de que todas as fases ocorrem sequencialmente e rapidamente durante uma iteração, repetindo-se novamente na próxima.

É importante destacar, apesar das similaridades de nomenclatura, que o peso de cada uma das fases e os artefatos produzidos nelas são definitivamente diferentes do encontrado tradicionalmente. É possível encontrar em Pressman (2011), de maneira detalhada, a explicação deste fluxo de processo.

De forma resumida, durante o Planejamento, definem-se as Histórias de usuário (basicamente como em um levantamento de requisitos tradicional), pontuando-se cada uma delas (estimativas e priorização), planejando-se as atividades a serem conduzidas durante a iteração e os critérios para aceitação (por parte do cliente) para estas Histórias. Na fase de Projeto, elaboram-se descrições simples (normalmente na forma de ações do usuário) para definir o comportamento das Histórias, complementando possivelmente com a elaboração de um rápido protótipo das interfaces gráficas para *feedback* antecipado do cliente. Durante a Codificação, práticas como programação em dupla (ou programação em pares) e refabricação (ou refatoração) são relevantes para agilizar o desenvolvimento ao mesmo tempo que permitindo maior segurança sobre a qualidade do código gerado. A fase de Teste não inicia necessariamente ao final da Codificação, mas sim de maneira concomitante, visto que os desenvolvedores codificam e aplicam testes unitários (e eventualmente até desenvolvimento baseado em testes, o qual será mais bem definido na seção 4.7).

4.6 Scrum

A abordagem Scrum começou a tomar forma através de conceitos discutidos por Nonaka e Takeuchi (1986) para gestão de equipes em projetos de desenvolvimento de novos produtos, sendo inicialmente discutida como um processo para aplicação em produção de *softwares* por Sutherland e Schwaber (1995). Porém, só foi formalizada após a concepção do Manifesto Ágil, com a publicação do livro seminal do tema por Schwaber e Beedle (2001).

O nome Scrum da jogada homônima existente no esporte Rúgbi (em inglês, *Rugby*), onde um grupo de jogadores monta uma formação em torno da bola para avançar em direção ao fundo do campo com maior força (às vezes, até de forma violenta), graças à combinação do esforço da equipe em detrimento do esforço individual (PRESSMAN, 2011).

A abordagem Scrum concentra-se principalmente em aspectos de gerenciamento de projetos (SALGADO *et al.*, 2010), com menos ênfase a aspectos técnicos do desenvolvimento de *software*, sendo portanto geralmente combinada com práticas propostas em outras abordagens, principalmente Programação Extrema, inclusive encontrando-se algumas vezes na literatura o termo ScrumXP para nomear esta combinação de abordagens. Schiel (2009, p.

21) inclusive recomenda o modelo representado pela combinação entre Scrum e XP para “praticamente todas as organizações”.

A natureza gerencial do Scrum também é relevante por lhe tornar compatível com normas técnicas e modelos de qualidade como ISO 9001, CMMI e MPS.BR (VRIENS, 2003; SATO, 2007; SALGADO *et al.*, 2010), o que pode ajudar a entender porque vem crescendo em utilização na indústria de *software* brasileira em comparação a outras abordagens, como discutido anteriormente neste trabalho.

> Scrum é um *framework* dentro do qual pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível. (...) é um *framework* estrutural que está sendo usado para gerenciar o desenvolvimento de produtos complexos desde o início de 1990. Não é um processo ou uma técnica para construir produtos; em vez disso, é um *framework* dentro do qual você pode empregar vários processos ou técnicas. O Scrum deixa claro a eficácia relativa das práticas de gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las (SCHWABER e SUTHERLAND, 2011, p. 3).

A abordagem Scrum fundamenta-se em três pilares teóricos que apoiam a implementação de controle de processo empírico, ou seja, controle de processo fundamento da tomada de decisões baseadas no conhecimento a partir da experiência. Tais pilares, conforme Schwaber e Beedle (2001), são:

- **Transparência:** aspectos significativos do processo devem estar visíveis constantemente aos responsáveis pelo resultado, para tanto sendo importante a definição de conceitos padronizados para que todos os observadores compartilhem um mesmo entendimento do que está sendo visto;
- **Inspeção:** frequentemente os artefatos e o progresso em direção ao objetivos devem ser inspecionados para detectar variações indesejáveis, mas não de forma tão frequente a ponto de atrapalhar a própria execução das atividades;
- **Adaptação:** se durante uma inspeção visualiza-se desvios além de limites aceitáveis, ou seja, que comprometam o resultado aceitável para o produto, o processo e/ou o material produzido deve ser ajustado. O próprio fluxo do processo Scrum fornece momentos considerados ideais para adaptação após inspeção.

4.6.1 Artefatos

> Os artefatos do Scrum representam o trabalho ou o valor das várias maneiras que são úteis no fornecimento de transparência e oportunidades para inspeção e adaptação. Os artefatos definidos para o Scrum são especificamente projetados para maximizar a transparência das informações chave necessárias para assegurar que a equipe Scrum tenha sucesso na entrega do incremento "pronto" (SCHWABER e SUTHERLAND, 2011, p. 12).

Os artefatos no Scrum, conforme detalhados em Schwaber e Beedle (2001) e resumidos em Pressman (2011), são apresentados a seguir:

- **Product Backlog:** eventualmente traduzido como Registro Pendente de Trabalhos, é uma lista com prioridades dos requisitos ou funcionalidades do projeto que fornecem valor comercial ao cliente. A própria lista é incremental, assim como o *software* criado, onde nas primeira iterações apenas requisitos mais genéricos e essenciais são referenciados e, continuamente, novos requisitos surgem, são modificados ou melhor compreendidos e expandidos;
- **Sprint Backlog:** eventualmente traduzido como Registro de Trabalhos de Urgência, são as tarefas selecionadas a partir o Product Backlog para compor a iteração atual, previstas para serem considerados requisitos "prontos" ao final desta. É um plano com detalhes suficientes para que o desenvolvimento do produto seja conduzido durante a iteração, sendo construído e modificado continuamente durante este período conforme a equipe aprende mais sobre o trabalho necessário para alcançar os objetivos do Sprint;
- **Incremento:** é a soma de todos os itens do Product Backlog completados durante a iteração atual com tudo que já existia pronto das iterações anteriores;
- **Definição de "Pronto":** esta definição pode variar drasticamente entre times Scrum diferentes, mas é uma definição que deve ser entendida da mesma maneira por todos os participantes do projeto, assegurando a transparência. Geralmente a definição de "pronto" compete dizer que o incremento entregue é utilizável, ou seja, funciona para as necessidades que já foram atendidas nas iterações realizadas. Enfatiza-se que considerar um item do Backlog como "pronto" não necessariamente significa que o cliente optará por começar a utilizá-lo efetivamente, mas que isso seja possível se desejado;
- **Produto:** no contexto do desenvolvimento de *software*, é comum imaginar que o produto é o *software* completo. Entretanto, é importante compreender que

> normalmente as equipes Scrum não têm como meta terminar efetivamente o produto, mas sim desenvolver um produto incremental e adaptável às mudanças ao longo do tempo, pelo tempo em que o cliente julgar relevante para seu negócio. Como amplamente discutem Beedle *et al.* (1999, p. 638), "Scrum pressupõe a existência do caos" oferecendo "uma abordagem mais adaptativa", capacitando a equipe de *software* a "trabalhar com sucesso em um mundo onde a eliminação de incerteza é impossível", de forma que o escopo do projeto é reconhecidamente um alvo em movimento e não um objetivo imutável.

4.6.2 Composição da equipe

Em equipes Scrum, conforme é observado por Schiel (2009), tipicamente utiliza-se a definição genérica Time de Desenvolvedores para designar todas as pessoas que estão envolvidas diretamente na criação do produto. Esta definição inclui os Programadores que usualmente associa-se ao termo Desenvolvedores, mas também inclui Analistas, Testadores, *Designers*, Arquitetos de Sistemas, Arquitetos de Bancos de Dados, etc. Ou seja, a própria nomenclatura genérica Desenvolvedor já demonstra o enfoque no perfil multifuncional das equipes, onde a mesma pessoa pode assumir diferentes atribuições durante o ciclo de vida de desenvolvimento do *software* e, mesmo quando cada papel é assumido por pessoas diferentes, todos os papéis existem dentro da própria equipe, não sendo necessário depender de pessoas externas que não estejam diretamente ligadas à entrega da iteração.

Para Schwaber e Sutherland (2011), equipes auto-organizáveis e multifuncionais são os aspectos mais importantes do Scrum, permitindo aperfeiçoar flexibilidade, criatividade e produtividade da organização. Os autores ainda analisam que o tamanho ideal para equipes com estas características limita-se de 3 a 9 integrantes. Com menos do que 3 indivíduos, a capacidade produtiva da equipe não é adequada para atender uma quantidade viável de tarefas em cada iteração. Já com mais de 9 indivíduos, a dificuldade de coordenação das pessoas aumentaria demais, gerando desperdícios gerenciais.

Mendonça (2013) resume a discussão sobre o Time de Desenvolvedores no Scrum, enumerando tais características como constantes:

- **Times auto-organizáveis**, onde ninguém (independentemente de seu nível hierárquico na organização), diz aos desenvolvedores como devem transformar requisitos esperados pelo cliente em incrementos de *software*;

- **Times multifuncionais**, onde o time possui todas as habilidades necessárias para criar um incremento do produto, sem depender de habilidades externas;
- **Sem títulos distintivos entre os membros**, independentemente do trabalho que esteja sendo realizado por uma pessoa, o único título empregado para todos é o termo Desenvolvedor, sem exceções;
- **Membros individuais podem ser especializados em determinada habilidade**, contudo a responsabilidade por "saber fazer" pertence ao time como um todo e não à uma pessoa específica;
- **Um time não possui "sub-times"**, como times dedicados a domínios específicos como testes ou análise de negócio. Todas as pessoas estão cientes e acompanham todas as etapas do ciclo de vida de desenvolvimento do *software*.

Além dos desenvolvedores, um papel específico é o Product Owner, eventualmente traduzido como Dono do Produto ou Gerente do Produto, tipicamente algum profissional com atribuições da área de *marketing* ou um usuário-chave do produto que se torna participante interno no projeto (SCHWABER e BEEDLE, 2001). Sua principal função é conduzir a priorização das atividades existentes no Product Backlog, garantindo que os incrementos do produto trarão as funcionalidades mais importantes primeiro (SCHIEL, 2009).

Os autores citados ainda complementam que o Product Owner é a única pessoa que toma decisões sobre o Product Backlog, sendo também a única pessoa que pode ser procurada pelo time de desenvolvimento para questões de priorização de atividades. Mesmo que esta pessoa represente um grupo de interessados (como um comitê do cliente), as decisões relativas ao Product Backlog são centralizadas e de total responsabilidade deste indivíduo.

O outro papel essencial no Scrum é o Scrum Master, que poderia ser comparado de maneira genérica ao perfil de Coordenador de Projetos em equipes tradicionais. A principal função deste indivíduo é certificar-se de que a equipe vive continuamente os valores e práticas do Scrum, protegendo a equipe ao certificar-se que não se sobrecarreguem, mediando e facilitando as reuniões diárias (SCHIEL, 2009). Porém, diferente de um Coordenador de Projetos tradicional, não compete a este indivíduo controlar a condução das atividades da equipe (por exemplo, decidindo a alocação de pessoas nas tarefas), visto que o conceito de times auto-organizáveis discutido anteriormente torna desnecessário – e incorreto – este tipo de atribuição a uma única pessoa.

Schwaber e Sutherland (2011, p. 7) utilizam o termo "servo-líder" para explicar este papel, argumentando que, ao mesmo tempo em que este indivíduo trabalha para a equipe em

busca de remoção de quaisquer impedimentos que surjam, como um servo, também lidera a equipe guiando e ajudando a entender como maximizar o valor criado por todos.

Tais autores ainda complementam que o Scrum Master é a pessoa com a maior visão do todo sobre o produto sendo criado, entendendo todo o Product Backlog a longo-prazo, apoiando o Product Owner no planejamento enquanto apoia os desenvolvedores comunicando claramente visão, objetivo e itens do Product Backlog.

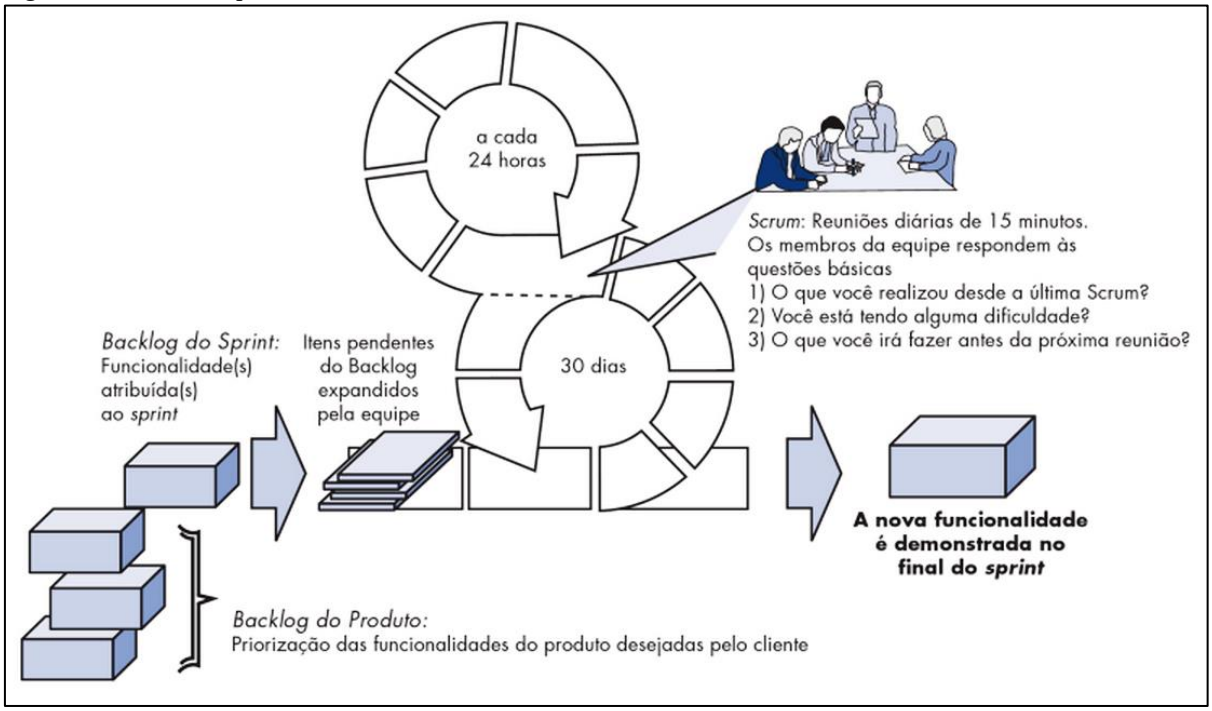
4.6.3 Fluxo do processo

O fluxo do processo ocorre baseado em iterações com fatias de tempo fixas, denominadas Sprint, onde ao seu final espera-se que um incremento "pronto" do produto seja disponibilizado. Tais fatias de tempo limitam-se entre duas semanas a um mês, não sendo comuns Sprints com menor ou maior duração do que este intervalo (KATAYAMA, 2010).

Conforme Schwaber e Sutherland (2011), o processo dentro de um Sprint é composto por uma Reunião de Planejamento, Reuniões Diárias, o trabalho de desenvolvimento em si, Revisão do Sprint e Reunião de Retrospectiva. Também se destaca que durante um Sprint não são feitas mudanças que afetem o objetivo do Sprint, não se modifica a composição da equipe, não se diminuem metas de qualidade, é permitido clarear e renegociar o escopo entre a equipe e o Product Owner quanto mais for aprendido sobre os objetivos. Em resumo, é possível renegociar estimativas sobre tarefas e até removê-las do Sprint de volta ao Product Backlog para que sejam recolocadas em iterações futuras, mas não é permitido mudar totalmente o objetivo de um Sprint durante seu andamento.

A Figura 4.3, na página a seguir, apresenta graficamente o fluxo do processo no Scrum, demonstrando como ocorrem as etapas citadas do início ao fim do Sprint, bem como a relação entre os artefatos no processo.

Figura 4.3 - Fluxo do processo no Scrum.



Fonte: Pressman (2011, p. 96).

No início de cada Sprint realiza-se uma Reunião de Planejamento, que serve para estimar, definir o objetivo do Sprint e as tarefas necessárias para atingi-lo, ou seja, quais tarefas serão retiradas do Product Backlog e passam a fazer parte do Sprint Backlog (BASSI FILHO, 2008).

Durante o fluxo de desenvolvimento, no dia-a-dia do Sprint, destaca-se a realização das Reuniões Diárias, as quais possuem curto tempo de duração (tipicamente de 15 minutos) e, preferencialmente, realizadas com os participantes em pé, visando serem objetivas e evitando-se que gerem desperdício de tempo para a equipe. Conforme demonstrado na figura apresentada, três perguntas-chave são respondidas por cada membro (NOYES, 2002):

- O que você realizou desde a última reunião?
- Quais obstáculos está encontrando?
- O que planeja realizar até a próxima reunião da equipe?

Ao final do Sprint, um incremento de *software* dentro da definição de "pronto" do projeto é liberado. Conforme já citado, compete ao Product Owner confirmar a aceitação do incremento e decidir se efetivamente entrará em uso pelo cliente e, mesmo que o incremento não comece a ser utilizado ainda pelo cliente, seu aceite já representa o sucesso deste Sprint.

Uma Reunião de Retrospectiva é conduzida em seguida, fornecendo a oportunidade de a equipe avaliar seu próprio desempenho, consolidar o aprendizado sobre erros e acertos ocorridos durante o Sprint e, desta forma, agregar melhorias ao processo (SCHWABER e SUTHERLAND, 2011). Ao final deste fluxo, recomeça o planejamento para o início de uma nova iteração, através da escolha e priorização dos requisitos que entrarão no novo Sprint.

4.7 Test-Driven Development

A abordagem Test-Driven Development (TDD), eventualmente traduzida como Desenvolvimento Baseado em Testes, Desenvolvimento Dirigido a Testes ou Desenvolvimento Orientado a Testes, é uma técnica de desenvolvimento de *software* que consiste de pequenas iterações onde novos casos de teste cobrindo uma nova funcionalidade desejada são escritos primeiro, antes de o código existir. Com os testes estruturados, o código necessário é produzido para que estes testes escritos passem com sucesso, seguindo-se este ciclo continuamente por todo o desenvolvimento do *software*. Posteriormente, durante novas codificações, o código é refatorado para acomodar mudanças e melhorar o reaproveitamento.

Trata-se de uma ideia existente desde os primórdios da programação de *software*, onde diversas vezes, antes de escrever uma nova rotina, o programador anotava os dados de entrada da função e os dados esperados para a saída, para então criar a função que fizesse esse processamento de dados, conforme apresentado por Martin Fowler em entrevista concedida a Venners (2002). Este prossegue explicando que tal abordagem sempre esteve presente, com maior ou menor força, durante as décadas que se passaram, mas começou a ser encarada com maior seriedade no final da década de 1990, quando foi incorporada por Kent Beck como uma das práticas da Programação Extrema, vindo a ser formalizada nos anos seguintes em um livro seminal deste autor (BECK, 2003).

TDD destaca-se não somente como uma prática sugerida em diversas abordagens ágeis, como a já citada XP, mas também como uma nova filosofia sobre como conduzir as etapas do ciclo de vida de desenvolvimento de *software*, visto que historicamente a etapa de testes sempre apareceu após a etapa de codificação, mas tal abordagem modifica esta ordem, interferindo inclusive na concepção da arquitetura do *software*, ou seja, ao contrário do que seu nome possa sugerir, é uma abordagem que influencia quando e como ocorrem as etapas de análise, codificação e testes.

“Mesmo se a Programação Extrema perdesse sua popularidade, o Desenvolvimento Baseado em Testes persistiria” (JANZEN e SAIEDIAN, 2005, p. 50).

Silva (2011) destaca que, apesar de inicialmente ter sido concebida como uma técnica visando o processo de teste de *software*, aos poucos quem a aplicou acabou percebendo "efeitos colaterais" desejáveis por causa de sua utilização, conforme descrito a seguir:

- Diminui o tempo de desenvolvimento;
- Auxilia na documentação do *software*;
- Auxilia no planejamento da arquitetura do *software*;
- Aumenta a qualidade do *software* pela prevenção de erros;
- Proporciona confiança ao desenvolvedor, pois gera *feedback* imediato a cada anormalidade encontrada;
- Suporta a manutenção, melhoria e refatoração do código;
- Pode ser usado como uma métrica de qualidade de *software*.

Sobre estes efeitos positivos da aplicação do TDD, Janzen e Saiedian (2005) apresentam em sua revisão bibliográfica resultados coletados em diversos trabalhos analisados, tentando demonstrar numericamente os ganhos em qualidade e/ou produtividade pela adoção da abordagem. Os trabalhos citados compunham-se de 3 experimentos controlados e 2 estudos de caso em empresas, além de 5 estudos com resultados empíricos conduzidos em ambiente acadêmico. Sobre efeitos na qualidade, 2 estudos não apresentaram mudanças pela adoção do TDD, enquanto nos outros 6 apresentaram melhorias de até 54% na redução de defeitos no produto. Sobre os efeitos na produtividade, 4 estudos não apresentaram mudanças consideráveis pela adoção do TDD, um estudo não abordou este fator, 2 estudos apresentaram ganho de produtividade de até 50%, resultado dissonante de um estudo que apresentou 16% de queda de produtividade após a adoção do TDD.

Beck (2003) realmente argumenta que a adoção desta abordagem inicialmente pode parecer improdutiva, visto que equipes que nunca trabalham desta forma podem se sentir perdidas e demorar mais do que o normal para concluir o mesmo tipo de código. Este último resultado citado por Janzen e Saiedian (2005), portanto, pode ter sido reflexo deste efeito.

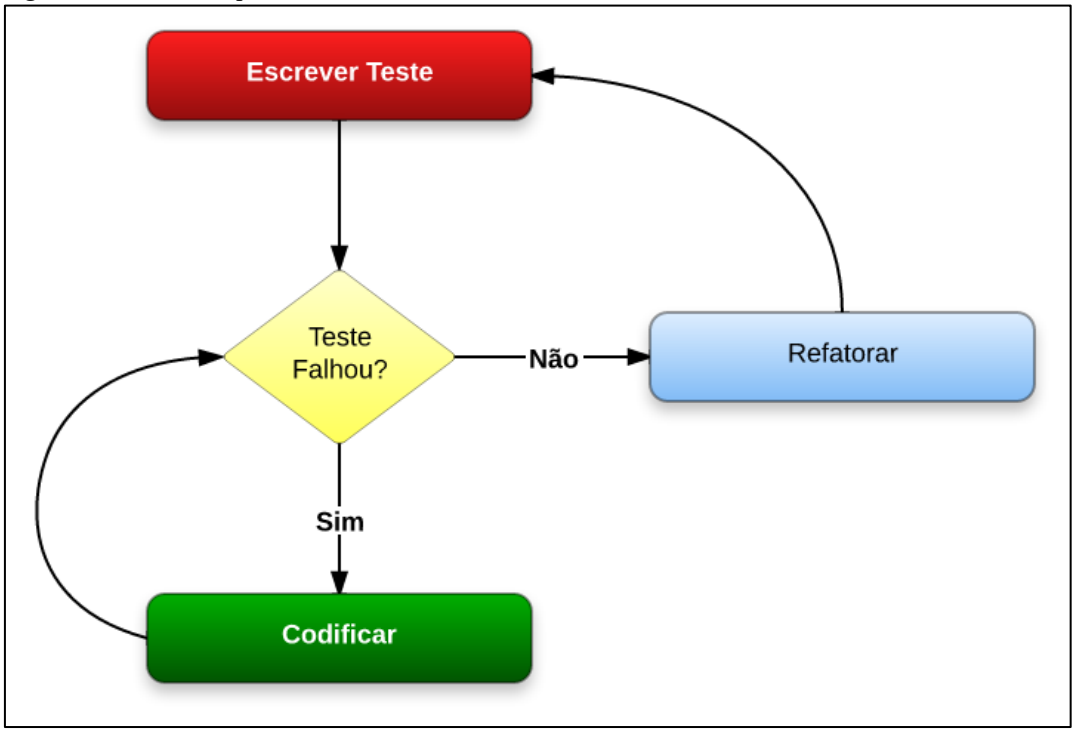
4.7.1 Fluxo do processo

O fluxo do processo no TDD baseia-se em três etapas, comumente denominadas *Red* (Sinal Vermelho), *Green* (Sinal Verde) e *Refactoring* (Refatoração). A seguir estas etapas serão explicadas em mais detalhes, com base principalmente em Beck (2003) e Silva (2011).

- **Sinal Vermelho:** o processo inicia com a elaboração de casos de teste unitários automatizados dentro da IDE (*software* de ambiente integrado de desenvolvimento) escolhida para o projeto. Os testes unitários devem cobrir a verificação de uma pequena unidade de *software* que será codificada futuramente. Em projetos que seguem o paradigma de Programação Orientada a Objetos (a maioria atualmente), esta unidade costuma ser representada por um método ou alguns poucos métodos em uma classe de código. O código do teste deve automatizar as checagens necessárias em relação às entradas e às saídas esperadas para o(s) método(s) em questão, garantindo que aquela unidade do *software* efetivamente cumprirá o requisito necessário. Após a codificação de alguns testes, estes devem ser executados. Irão falhar em suas verificações visto que o código que as executa ainda não existe, o que geralmente é representado por um sinal gráfico vermelho na IDE, por isso o nome atribuído a esta etapa do processo.
- **Sinal Verde:** em seguida, os códigos necessários para que os testes que falharam passem devem ser escritos. O propósito desta etapa é simplesmente e efetivamente apenas passar nos testes e, portanto, o código a ser escrito deve ser o mais enxuto e direto o possível para isso. Evidentemente, nem sempre a solução mais rápida gerará o código mais elegante e/ou mais reaproveitável, mas isso será solucionado posteriormente. Após a escrita dos códigos, os testes devem ser reexecutados para nova verificação. Quando os testes passam, geralmente a IDE representa com um sinal gráfico verde, mantendo em vermelho testes que ainda não foram contemplados. Esta etapa permanece se repetindo enquanto existirem testes em vermelho, ajudando inclusive a guiar o trabalho do desenvolvedor, que sabe que só pode mudar de atividade quando todos os seus testes estiverem "verdes".
- **Refatoração:** assim como sugerido em outras abordagens enxutas, como LSD e XP, a atividade de refatoração (ou refabricação) é determinante para se obter maior qualidade no código criado. No caso do TDD, essa atividade é essencial e parte fundamental do processo, uma vez que, durante o objetivo de passar nos testes a qualquer custo, códigos mal estruturados eventualmente são escritos e, durante a refatoração, devem ser analisados e reorganizados buscando melhor coesão, abstração e reaproveitamento entre unidades do *software*.

Um fluxograma representando graficamente este fluxo de processo pode ser observado a seguir, na Figura 4.4, na página a seguir, onde se destaca também como este fluxo é cíclico.

Figura 4.4 - Fluxo do processo no TDD.



Fonte: adaptado de Silva (2011).

4.8 Feature-Driven Development

A abordagem Feature-Driven Development (FDD), eventualmente traduzida como Desenvolvimento Baseado em Funcionalidades, Desenvolvimento Dirigido a Funcionalidades ou Desenvolvimento Orientado a Funcionalidades, é um processo de desenvolvimento de *software* ágil adaptativo, que pode ser aplicado a projetos de *software* de porte moderado e a projetos maiores (PRESSMAN, 2011).

Esta abordagem foi descrita formalmente pela primeira vez em Coad *et al.* (1999), como uma oficialização do então conhecido como Método Coad (metodologia elaborada por Peter Coad entre as décadas 1980 e 1990 para análise, projeto e programação no paradigma orientado a objetos), com o acréscimo de técnicas de gerenciamento de projetos iterativo, incremental e enxuto utilizadas por Jeff De Luca quando este atuava como gerente de projetos em uma empresa australiana. Contudo, somente com Palmer e Felsing (2002) este trabalho foi estendido e aperfeiçoado, sendo publicado como um livro tratando especificamente deste assunto, tornando-o um processo ágil adaptativo.

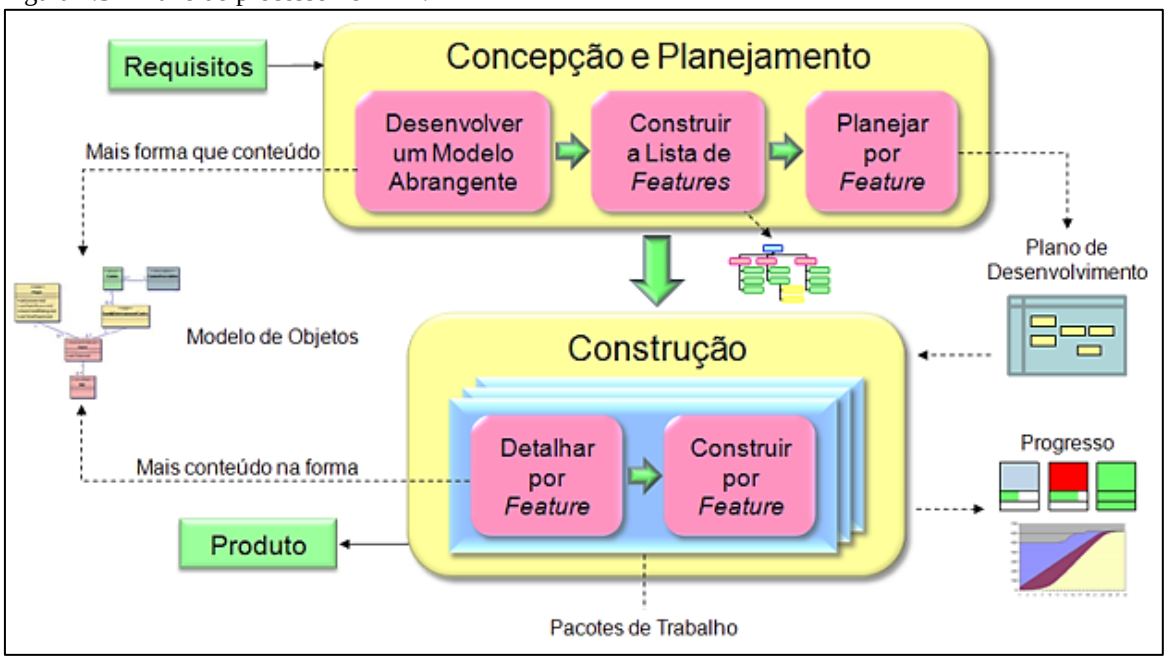
Uma funcionalidade entende-se por uma função valorizada pelo cliente, passível de ser implementada em duas semanas ou menos (COAD *et al.*, 1999). Esta afirmação demonstra como o FDD também enfatiza iterações pequenas, assim como as outras abordagens enxutas já apresentadas anteriormente.

Pressman (2011) resume a filosofia por trás do FDD da seguinte maneira:

- Enfatiza a colaboração entre pessoas da equipe;
- Gerencia problemas e complexidade de projetos utilizando decomposição baseada em funcionalidades, seguida pela integração dos incrementos de *software*;
- Comunicação de detalhes técnicos usando meios verbais, gráficos e de texto.

A seguir é apresentam-se as atividades que compõem o FDD na Figura 4.5:

Figura 4.5 - Fluxo do processo no FDD.



Fonte: Retamal (2007).

O Desenvolvimento Baseado em Funcionalidades apresenta um fluxo de processos inicialmente estranho, por separar o processo apenas em duas fases: Concepção e Construção. Mas a verdadeira relevância desta abordagem é mais filosófica do que estrutural: as Funcionalidades desejadas pelo cliente (que podemos comparar com as Histórias de outras metodologias ágeis) são ao mesmo tempo a matéria-prima, o plano, o roteiro e o grande objetivo a ser atingido para o produto em questão (CONBOY *et al.*, 2011).

5 Estudo de caso

Nesta seção, será apresentada a condução do estudo de caso realizado em uma das unidades, doravante denominada apenas Unidade C, de uma empresa de desenvolvimento de *software* sob encomenda, doravante denominada apenas Empresa X, conforme os critérios de definição do sujeito apresentados anteriormente na seção 1.4.5.

Ressalta-se que todos os nomes apresentados são fictícios, conforme solicitado pela empresa, por questões de sigilo.

5.1 Roteiro de execução do estudo

O estudo foi estruturado e conduzido segundo o seguinte roteiro:

- Entrevista estruturada, conduzida através de *e-mail*, com um dos sócios-proprietários da empresa, o qual também exerce a função de Diretor de Tecnologia. Nesta etapa, informações iniciais sobre a empresa foram coletadas, para caracterização desta e para definição das etapas seguintes. As perguntas realizadas estão elencadas no Apêndice A deste trabalho.
- Entrevista não estruturada com o Gerente da Unidade C, para conhecer a unidade e entender sua estrutura e seu o papel nos projetos.
- Entrevista semiestruturada com o Gerente da Unidade C, buscando compreender a penetração da filosofia enxuta na unidade. Os tópicos discutidos durante a entrevista estão elencados no Apêndice B deste trabalho.
- Observação direta, natural e não estruturada, realizada durante três visitas, em horários diversificados e não pré-definidos pelo pesquisador, com duração média de uma hora por visita, para compreender o andamento real das atividades.
- Aplicação de um questionário, cujas perguntas estão elencadas no Apêndice C deste trabalho, a três desenvolvedores da Unidade C, cujos critérios de escolha serão detalhados na seção 5.3.2, com o objetivo de auxiliar o entendimento de quanto os conceitos enxutos são compreendidos e aplicados por parte dos colaboradores diretamente relacionados com a produção do *software*.

5.2 Caracterização da empresa

A Empresa X caracteriza-se como uma empresa brasileira de desenvolvimento de *software* sob encomenda, existente há quase 25 anos no mercado.

Em território nacional, a empresa atua através de três unidades quem realizam desenvolvimento de *software*, situadas em cidades distintas, sendo que a Unidade C, alvo deste estudo, existe há cerca de cinco anos, sendo a mais nova. Tal unidade localiza-se em uma cidade pequena do interior do estado de São Paulo, com menos de 100 mil habitantes. Além disso, a empresa também tem presença em outras localidades do país através de escritórios comerciais.

Embora com atuação primariamente nacional, a empresa atua há cerca de 10 anos de forma intensa no mercado internacional, possuindo uma unidade subsidiária nos Estados Unidos da América, atualmente com projetos para clientes dos EUA, Japão, Portugal e Itália.

O quadro funcional é composto por aproximadamente 100 funcionários, somando-se todas as unidades e escritórios das diversas localidades. A Unidade C conta com aproximadamente 15 pessoas, variando levemente em picos de demanda. Visto que dados sobre o faturamento anual da empresa não foram disponibilizados, os quais poderiam ser utilizados para classificação precisa do porte desta segundo a legislação vigente, este trabalho considerou o Critério de Classificação de Empresas quanto ao Número de Empregados, sugerido pelo SEBRAE/SC, o qual dispõe considerar-se a empresa como de médio porte quando seu quadro funcional estiver entre 100 e 499 funcionários, conforme replicado no Quadro 5.1.

Quadro 5.1 - Critérios de classificação de empresas sugerido pelo SEBRAE/SC.

Tamanho	**Intervalo de classificação**
Micro	Até 19 empregados
Pequena	Entre 20 e 99 empregados
Média	Entre 100 e 499 empregados
Grande	Mais de 500 empregados

Fonte: adaptado de SEBRAE/SC (2008).

Conforme explicado pelo Diretor de Tecnologia, a empresa possui uma carteira com pouco mais de 200 clientes, sendo que cerca de 30 destes são mais ativos, possuindo vários projetos em execução durante um longo período de tempo, o que ajuda a causar alguns picos de demanda que precisam ser compensados pelas unidades com algumas variações de mão-de-obra, seja por deslocamento de desenvolvedores entre as unidades por um curto período de tempo, seja com a busca de novos profissionais no mercado.

Sobre a organização funcional, esclareceu-se que a Empresa X segue a sugestão de nomeação proposta pelo Scrum, onde não existe diferenciação por função entre os colaboradores técnicos, da maneira como ocorre em ambientes tradicionais. Ou seja, não há separação de funções como Arquiteto de Softwares, Analista de Sistemas, Testador etc. sendo

todo colaborador técnico denominado Desenvolvedor, independentemente do tempo de casa, experiências ou habilidades, evidenciando-se assim a importância dada pela empresa a colaboradores com perfil multifuncional e multidisciplinar.

A Unidade C possui seu quadro funcional todo composto por Desenvolvedores, porém um deles assume concomitantemente às atividades de desenvolvimento também atividades de gerência, respondendo pelas questões administrativas da unidade. Este profissional é nomeado neste trabalho simplesmente como Gerente, para facilitar o entendimento, embora oficialmente seja tratado dentro da empresa como Desenvolvedor, assim como todos os outros funcionários da Unidade C, o que lhe confere uma posição de igualdade perante os outros.

Ao se considerar a Empresa X como um todo, o quadro funcional é composto, aproximadamente, em 75% por desenvolvedores e em 25% por funcionários exclusivamente administrativos, como diretoria, gerência, financeiro, recursos humanos, etc.

A Empresa X é especializada no desenvolvimento de *software* utilizando tecnologias da empresa norte-americana Microsoft, como soluções em nuvem com Windows Azure e criação de soluções personalizadas com a plataforma .NET, incluindo-se aí a linguagem de programação C#, as bibliotecas ASP.NET, WPF e Silverlight e o banco de dados SQL Server, embora alguns projetos específicos tenham utilizado banco de dados Oracle.

Este direcionamento para tecnologias Microsoft é estratégico para a empresa, sendo inclusive certificada como Microsoft Gold Certified Partner, fazendo parte do seleto grupo de apenas 70 empresas brasileiras que possuem tal certificação (MICROSOFT, 2013).

Sobre as metodologias adotadas no ciclo de vida de desenvolvimento de *software*, a Empresa X passou a maior parte de sua existência seguindo preceitos prescritivos, começando a procurar alternativas em práticas emergentes há cerca de 12 anos, sendo que a adoção oficial de Scrum em toda a empresa iniciou há pouco mais de 4 anos.

A adoção do Scrum como o modelo de processos oficial da Empresa X foi uma decisão estratégica, largamente incentivada pelo alto escalão da organização, principalmente pelo fato do citado Diretor de Tecnologia ser grande entusiasta da metodologia, atuando inclusive como instrutor e palestrante, possuindo o título de Certified Scrum Trainer.

Este processo de adoção do Scrum foi aplicado em todas as unidades de desenvolvimento de *software* da empresa, sendo que, inicialmente, somente em alguns projetos piloto e, posteriormente, na grande maioria dos demais. Segundo o Diretor de Tecnologia, mais de 95% dos projetos atualmente na Empresa X seguem os processos do Scrum. O investimento foi tão enfático por parte da organização que mais de 75% dos

desenvolvedores participaram de treinamentos oficiais para o Scrum e obtiveram o título de Certified Scrum Developer.

Além do Scrum, segundo o Diretor de Tecnologia, a Empresa X aplica também alguns outros conceitos emergentes, como Test-Driven Development, mas não em todos os projetos. Além disso, alguns processos tradicionais continuam sendo aplicados, como Gestão de Riscos e Gestão de Requisitos, segundo ele por agregarem vantagens gerenciais sem impactar negativamente no processo de desenvolvimento de *software*.

5.2.1 Antes do pensamento enxuto

O processo de adoção do Scrum começou pouco tempo após a abertura da Unidade C e, desta forma, mesmo os desenvolvedores mais antigos da unidade (que atuam nesta desde o início), praticamente não presenciaram um cenário exclusivamente baseado na engenharia de *software* tradicional.

O desenvolvedor que responde como Gerente da unidade, por sua vez, atua pela Empresa X há quase 10 anos e descreveu como ocorria o processo antes da adoção de metodologias enxutas. As informações mais relevantes, para comparação do cenário anterior com o cenário atual, estão agrupadas nesta seção juntamente com alguns comentários sobre o assunto feitos pelo Diretor de Tecnologia.

O modelo de processo empregado pela empresa era o RUP, de maneira muito próxima ao descrito na seção 3.2.2, o que é esperado quando se segue um modelo de processo fortemente prescritivo. Durante este período, a empresa enquadrava-se como de pequeno porte, tendo duas unidades de desenvolvimento com um total de funcionários entre 30 a 60 dependendo da época.

Sobre o processo, resumidamente, os projetos passavam por um período de definições e planejamento inicial, passando por um longo período de levantamento de requisitos, modelagem de dados e fechamento do escopo. O Gerente da unidade explicou que, com toda a documentação e burocracia processual existente na época, o tempo que restava para o desenvolvimento do *software* propriamente dito era de cerca de 30% do tamanho do projeto, sendo que em praticamente todos os projetos acabava não sendo o suficiente, chegando a causar desvios de cumprimento de prazos de 40% a 80% em relação ao estimado.

O Diretor de Tecnologia enfatizou o quanto o modelo de processos prescritivo adotado no passado causava uma ilusória sensação de controle sobre o projeto. Com a pretensão equivocada por parte dos gestores de que um escopo fortemente definido pudesse ser seguido como um plano infalível e pouco flexível, qualquer necessidade de mudança durante o

projeto, algo comum por parte do cliente, era motivo para atritos muitas vezes corrosivos para a imagem da organização perante este e até para a continuidade do projeto. “O processo tradicional ‘escondia’ a real necessidade do cliente embaixo de requisitos especificados inicialmente, fazendo com que a gestão de requisitos acirrasse com o cliente constantemente, ao invés de atendê-lo em suas reais necessidades – as quais, conforme sabemos hoje, são inevitavelmente mutáveis”, afirmou ele.

As equipes eram altamente especializadas dentre as disciplinas preconizadas por este modelo, ou seja, existiam divisões na empresa, como: equipe de analistas de sistemas, equipe de desenvolvedores de interfaces, equipe de desenvolvedores de componentes, equipe de testadores de *software*. Um exemplo deste cenário: uma unidade da empresa não possuía todas as equipes necessárias em todas as disciplinas durante as fases do projeto, portanto era comum que as atividades de planejamento, análise e modelagem do *software* fossem responsabilidade de uma equipe na Unidade A e que as atividades de implementação fossem responsabilidade de uma equipe da Unidade B, enquanto que os testes e a garantia da qualidade fossem responsabilidade de outra equipe, novamente na Unidade A.

Este cenário causava diversos gargalos entre as fases, por exemplo: um analista de sistemas montava uma especificação de componente a ser desenvolvimento, mas aguardava outras especificações dependentes ficarem prontas antes de passar aos desenvolvedores. Quando a especificação estava com o desenvolvedor, caso este enfrentasse problemas para entender ou mesmo encontrasse algum erro na especificação, precisava parar e aguardar a disponibilidade do analista (por telefone ou por e-mail), o qual muito provavelmente já estava especificando outro componente e não estaria, portanto, disponível para dedicar muito tempo ao problema.

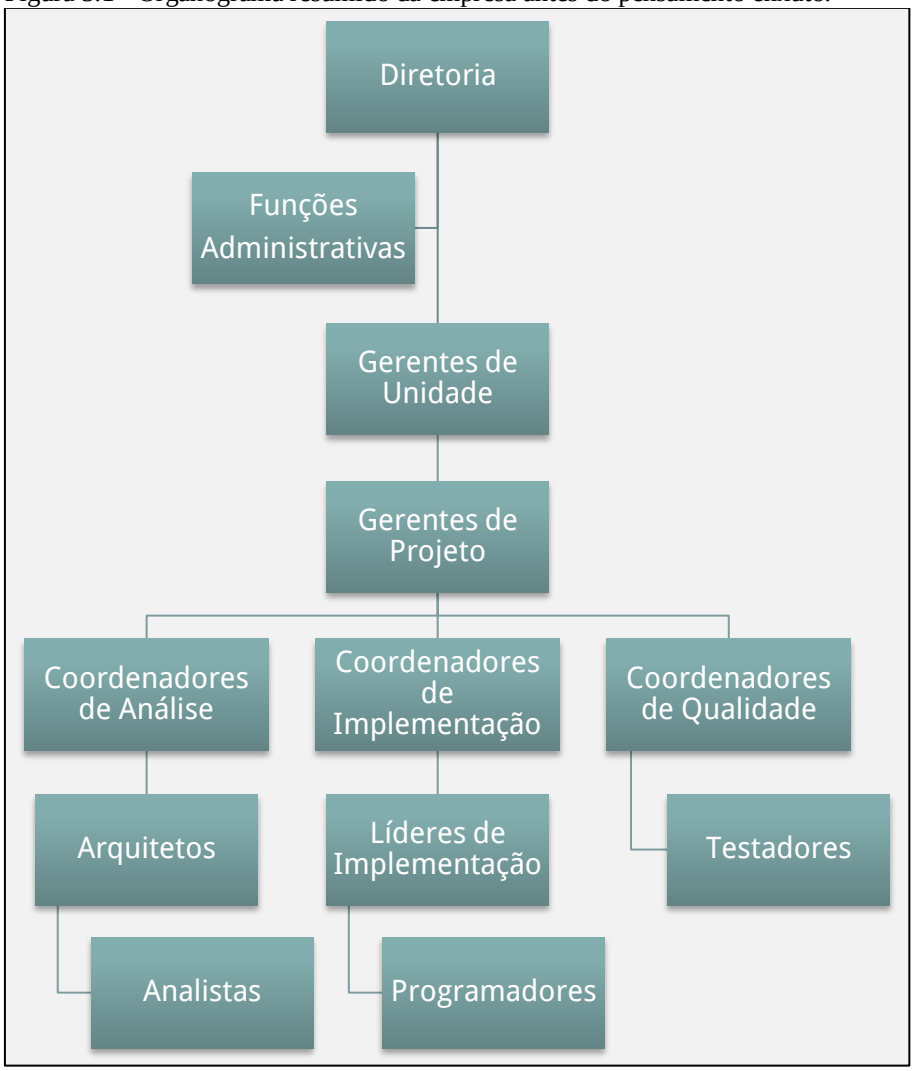
Esta situação era recorrente e se agravava ainda mais pelo fato de cada equipe ter suas metas e medições de desempenho independentes. Por exemplo: um desenvolvedor precisava cumprir determinado prazo para entregar o componente e muitas vezes liberava algo com defeitos, que acaba causando muito trabalho ao testador para detectar e documentar tudo; com isso, o desenvolvedor “ganhava” tempo até que o componente fosse recusado e voltasse para a realização de correções.

A comunicação era um ponto crítico, as equipes de diferentes disciplinas não se entendiam e não se respeitavam, além do costume recorrente de “jogar a culpa” em outra equipe quando qualquer problema acontecia.

Cada equipe possuía toda uma hierarquia própria, como coordenadores de célula, gerentes de projetos, gerente de unidade, diretores de área. Confrontos entre responsáveis de equipes distintas também era recorrente.

Baseado nos relatos sobre a organização e a estrutura hierárquica da Empresa X no período anterior ao pensamento enxuto, o organograma apresentado na Figura 5.1 foi elaborado para facilitar o entendimento do cenário anterior.

Figura 5.1 - Organograma resumido da empresa antes do pensamento enxuto.



Fonte: elaborada pelo próprio autor.

Conforme pode ser observado, existia uma hierarquia considerável desde os funcionários técnicos até o alto escalão decisório, com concentração de decisões em diversos níveis, do operacional ao estratégico. Desta maneira, a liberdade decisória era praticamente nula nos níveis inferiores. Embora o organograma não evidencie, segundo o Gerente também ocorria uma espécie de subordinação dos programadores para com os analistas, pois as decisões para agregação de valor ao negócio só eram permitidas se provenientes das equipes

de análise, enquanto cabia aos programadores apenas transformar em código o que fora especificado, com o mínimo de questionamentos possíveis.

Para o Gerente da Unidade C, este excesso de especialização das equipes também causava dois outros problemas: dependência excessiva de determinados indivíduos de destaque dentro da equipe, que acabavam sendo os únicos que resolviam boa parte dos problemas mais complexos; e a real capacidade de cada indivíduo era subutilizada, visto que não existia liberdade para atuação ampla dentro do ciclo de vida do *software*, apenas atuando-se sempre nas mesmas atividades.

O Diretor de Desenvolvimento também citou que, por um bom período de tempo durante a década de 2000, a empresa acreditou que os fracassos em projetos importantes e os problemas recorrentes, de maneira geral, seriam justamente por não estarem totalmente preparados e maduros para o cumprimento integral de todas as ações e documentação necessárias segundo os modelos prescritivos.

Contou ainda o Diretor de Desenvolvimento que se cogitou, em determinado momento naquela época, que a empresa só alcançaria um patamar realmente adequado de produtividade e qualidade se investisse pesado em um processo cada vez mais rígido e prescritivo. Contudo, principalmente por inviabilidade financeira para o investimento na adoção de modelos de maturidade de processos, como CMMI e MPS.BR, mas também por desejo dos sócios-proprietários em buscar alternativas inovadoras e diferentes, optou-se pelo investimento da adoção de uma filosofia enxuta para a organização.

Pouco antes do início de 2009, a Empresa X começou o processo de adoção oficial do Scrum como metodologia enxuta para condução dos projetos. Inicialmente em alguns projetos pilotos, começou a apresentar resultados interessantes após o primeiro ano de adoção.

5.3 O caso estudado

Neste tópico, será apresentada a descrição detalhada das informações coletadas através das entrevistas, da observação direta e do questionário aplicado a alguns desenvolvedores. Durante sua condução, serão citados princípios conceituais e práticas das abordagens enxutas discutidas na seção anterior, expondo-se então como a empresa está aplicando-as.

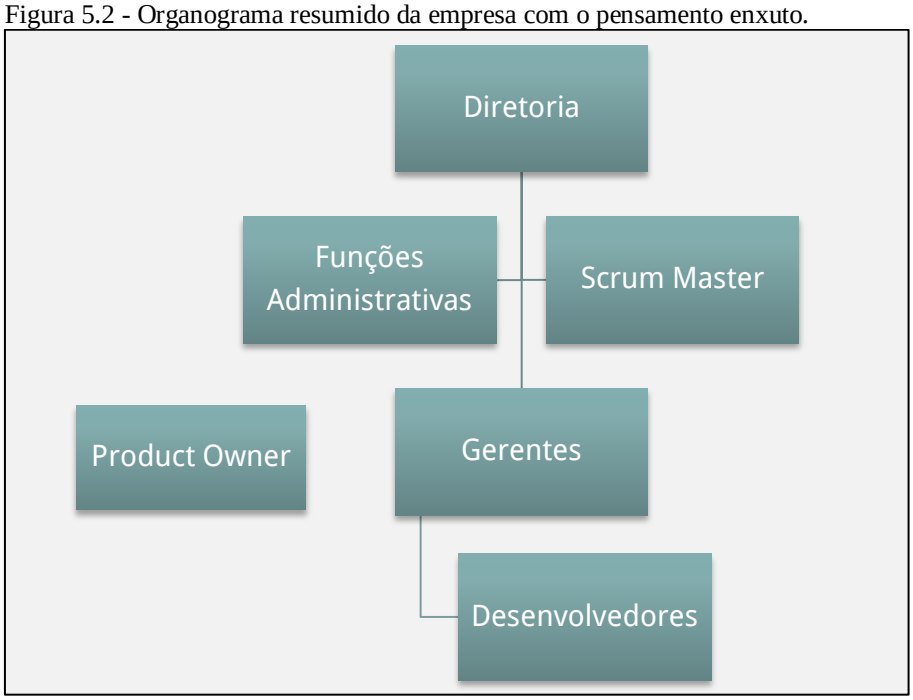
Conforme já descrito, a metodologia ágil que a empresa oficialmente declara utilizar é a metodologia Scrum, entretanto outras práticas foram detectadas durante o estudo, conforme será apresentado durante o decorrer da presente seção deste trabalho.

Não apenas meras escolhas metodológicas para o processo, a filosofia da empresa, a cultura organizacional e o comportamento de todos os envolvidos, especialmente dos

desenvolvedores, demonstrou-se fortemente relacionada ao pensamento enxuto. Questionado sobre isso, o Diretor de Tecnologia disse conhecer o STP e enfatizou que Scrum, assim como todos os outros conceitos emergentes que a empresa vem adequando em seu processo, são realmente evoluções do pensamento enxuto para a engenharia de *software* nos quais os aspectos comportamentais são levados ao extremo. Segundo ele, "o maior desafio é justamente habilitar em todas as pessoas o autogerenciamento e a pró-atividade efetiva, certamente um questão de cunho filosófico e cultural, acima de tudo".

O organograma revisado da Empresa X após a adoção da filosofia enxuta, conforme apresentado abaixo na Figura 5.2, demonstra como a hierarquia e o excesso de controle gerencial realmente foram enxugados como parte da adoção de maior agilidade no processo.

Figura 5.2 - Organograma resumido da empresa com o pensamento enxuto.



Fonte: elaborada pelo próprio autor.

É interessante também observar que os Desenvolvedores só respondem ao Gerente da unidade sobre questões administrativas, financeiras, alocação de recursos, etc. Decisões técnicas e de projeto não devem necessitar da atuação de um Gerente. Também vale destacar que o Scrum Master não é subordinado a nenhuma gerência e é encarado como uma função de apoio, auxiliando nas decisões e na correta utilização do Scrum, como um "conselheiro" disponível para todos os Desenvolvedores de qualquer projeto.

Outro ponto de destaque é a forma como o papel do Product Owner é encarado pela empresa. Ao mesmo tempo em que não faz parte da organização, pois sempre se trata de uma

pessoa externa, representante do cliente para o projeto, é visto como parte vital do projeto e, por isso, mesmo sem ligações no organograma foi representado graficamente.

Curiosamente, quando o Gerente da Unidade C foi questionado sobre conhecer o STP e o pensamento enxuto no primeiro contato realizado durante o estudo, este respondeu que não fazia ideia e que para ele, até então, existia apenas a utilização de Scrum na empresa. Após algum tempo do estudo sendo conduzido, ele argumentou que estudou melhor o assunto, repensou sua resposta e que passou a explicar sua visão desta forma: "Acho que existem muitos nomes diferentes para as mesmas coisas! Scrum parece ser uma ótima forma de atuar com uma filosofia enxuta na empresa... Provavelmente aplicamos muito mais coisas do que realmente temos ciência que aplicamos".

5.3.1 Aplicação dos princípios enxutos

Buscando compreender de que maneira as práticas aplicadas pela empresa são condizentes e relacionadas aos princípios enxutos, optou-se por conduzir este tópico do estudo baseando nos princípios do Lean Software Development conforme descrito por Poppendieck e Poppendieck (2003), por já representarem um bom mapeamento para a engenharia de *software* dos princípios originais do STP.

As descrições contidas nos tópicos a seguir foram organizadas com base em entrevista não-estruturada realizada com o Gerente que responde pela Unidade C, em conjunto com algumas situações presenciadas pelo autor deste estudo durante as observações diretas.

5.3.1.1 Elimine o desperdício

A empresa entende que o desperdício é efetivamente a maior barreira para a geração de valor e também para a obtenção de resultado financeiro.

A própria adoção do Scrum foi fortemente abraçada por toda a alta cúpula da empresa com relativa facilidade justamente por representar uma grande oportunidade de eliminação de desperdícios durante o ciclo de vida de desenvolvimento de *software*, principalmente em relação às grandes divergências que existiam corriqueiramente entre o que era delimitado no escopo dos projetos e o que efetivamente ocorria em seu andamento.

Enfatizou-se que a adoção do Scrum definitivamente contribuiu para eliminar a antecipação de funcionalidades e incentivou a busca constante por soluções simples. Inclusive, o Gerente enfatizou fortemente que desenvolver a solução mais simples possível para determinado problema é o desafio constante ao qual os desenvolvedores são submetidos

e, quando novos indivíduos entram na empresa, conhecendo ou não Scrum previamente, são prontamente respaldados por desenvolvedores mais experientes para entender a importância de atuar desta maneira.

Em busca de uma efetividade ainda maior na busca por soluções simples, a empresa vem utilizando TDD em vários projetos, embora ainda sem tornar esta abordagem um padrão. Como TDD não é uma prática endossada oficialmente pelo Scrum, é acordado com o cliente a utilização desta em cada projeto. Observa-se que muitas decisões sobre os procedimentos internos das equipes só são tomadas em comum acordo com o cliente (assim como esta, outras serão apresentadas posteriormente).

Questionado sobre qual tipo de desperdício ainda perdura no processo mesmo com as melhorias trazidas pelo Scrum e pelas demais abordagens enxutas que a empresa vem empregando, o Gerente respondeu que perdura o desperdício de tempo produtivo, explicando que a implantação do Scrum fortaleceu os desenvolvedores e diminuiu drasticamente o retrabalho mas, por outro lado, gerou brechas de tempo produtivo desperdiçado quando o desenvolvedor encerra a tarefa com a qual se comprometeu em determinado dia e, após este período, pode não se dedicar a nenhuma outra atividade relevante à empresa. Para o Gerente, se o desenvolvedor não possui real comprometimento para buscar novas atividades naquele dia, ao invés de simplesmente esperar a próxima reunião diária, bastante tempo produtivo pode ser desperdiçado. Isto enfatiza a dependência no fator humano inerente à atividade de desenvolvimento de *software* e que é um problema também em metodologias enxutas.

Entretanto, a empresa não estaria interessada em eliminar completamente esta situação, uma vez que a liberdade em escolher o que fazer com seu tempo livre também serve como incentivo aos desenvolvedores, que se dedicam ainda mais para encerrar suas atividades, e até como motivador para que se sintam mais felizes no trabalho.

Outro exemplo, citado pelo Gerente, de que este suposto desperdício de tempo produtivo acaba sendo benéfico, foi em relação às paradas para descanso (os chamados intervalos para *coffee break*), pois são momentos em que oficialmente o desenvolvedor estaria em horário produtivo mas, apesar de não estar diretamente dedicado à solução do problema, funcionam como momentos de intensificação de criatividade. “Muitas vezes, pensar em um código que solucione um problema exige muita criatividade. Parar, tomar um café, rir um pouco e ‘esfriar a cabeça’ permite que a pessoa volte mais concentrada e com novas ideias para resolver o problema”, argumentou ele.

Como conclusão da reflexão sobre este princípio enxuto, o Gerente concorda que a eliminação de desperdícios nunca termina e que, embora no momento pareça não existirem

mais desperdícios críticos no processo, um olhar mais aprofundado precisará ser aplicado sobre este princípio enxuto.

5.3.1.2 Amplifique o aprendizado

A principal medida destacada pelo Gerente trata-se de colocar pessoas menos experientes atuando em projetos com pessoas mais experientes. Estas pessoas "menos experientes" referem-se tanto à pouca experiência profissional (pessoas recém formadas ou ainda em formação acadêmica na área), quanto pessoas já experientes no mercado mas recém contratadas pela empresa. Ou seja, a empresa incentiva o aprendizado do processo usando Scrum e da assimilação da cultura corporativa existente por meio do contato direto com pessoas mais experientes na organização, privilegiando assim o aprendizado por meio do conhecimento tácito, o qual, segundo Nonaka e Takeuchi (1997), corresponde ao conhecimento mais subjetivo relacionado à experiência vivenciada de maneira prática, trocado entre indivíduos geralmente através da comunicação física informal.

Em todos os projetos este contato entre desenvolvedores menos e mais experientes ocorre durante as Reuniões de Planejamento do Sprint, durante as Reuniões Diárias de 15 Minutos e durante algumas paradas para descanso durante o dia. Em alguns projetos, novamente quando acordado com o cliente, a empresa vem empregando a prática de Programação em Pares para que os menos experientes tenham seu aprendizado impulsionado, principalmente nas primeiras iterações. Segundo o Gerente, raramente projetos mais longos (com mais de quatro Sprints), permanecem utilizando-se de Programação em Pares. Ou seja, a empresa enxerga esta prática como um impulsionador do aprendizado para o início dos projetos, mas não como uma prática vantajosa para a produtividade no longo-prazo.

O Gerente citou também que, em projetos que precisem passar por mudança de desenvolvedores – algo fortemente evitado pela direção, mas que em certos momentos é inevitável por saída de colaboradores da empresa ou por mudanças em prioridades entre diversos projetos – a utilização de Programação em Pares apresenta-se como um mecanismo eficiente de transição rápida de conhecimento.

Em um caso específico, contou ele, "procedeu-se com o que poderia ser chamado de 'Programação em Cinco' durante algum tempo", onde todos os cinco desenvolvedores escalados para o projeto permaneceram codificando a mesma História, através de telas compartilhadas em tempo real. Compartilhar as telas e codificar em conjunto não é algo que tenha ocorrido por tanto tempo com exceção deste caso citado, mas é algo que acontece esporadicamente por um período curto (de alguns minutos) segundo ele.

Outra medida existente refere-se à valorização de certificações técnicas, como o programa Microsoft Certified Professional (MICROSOFT LEARNING, 2013), inclusive com apoio financeiro por parte da empresa para desenvolvedores, ajudando com os custos referentes ao valor dos exames. Entretanto, não existe uma estruturação formal de incentivo ao estudo para certificações.

Por fim, é importante ressaltar que estudos sobre o Scrum também foram incentivados pela empresa, principalmente no início de sua aplicação. Diversos colaboradores receberam treinamentos oficiais e alguns até certificaram-se na metodologia.

Como conclusão da reflexão sobre este princípio enxuto, o Gerente acredita que a empresa poderia impulsionar mais a amplificação de aprendizado, principalmente com medidas que incentivem os desenvolvedores a utilizar seus tempos livres (citados no tópico anterior) como momentos ideais para o estudo e a experimentação, ou seja, a aplicação deste princípio enxuto ainda pode ser melhorada na empresa.

5.3.1.3 Adie comprometimentos

Para o Gerente, é complexo dizer que adiar comprometimentos é realmente possível, visto que um cliente, ao investir em um *software* sob encomenda, parte do conceito de que está adquirindo não o produto, mas uma expectativa sobre algo que ainda não existe e que deverá ser criado da forma como se espera. Neste sentido, o maior comprometimento é o projeto como um todo e, este comprometimento não pode ser adiado, inclusive sendo regido contratualmente. Entretanto, os pequenos comprometimentos sobre cada funcionalidade desejada são adiados graças ao processo iterativo, aplicado através da divisão em Sprints, assim como também é aplicado de forma parecida em outras metodologias ágeis.

Além da condução em Sprints auxiliar no princípio de adiar comprometimentos, destaca-se que o modelo de vendas da empresa também foi adaptado para respaldar o Scrum. Salvo casos específicos, grande parte dos projetos são negociados e vendidos como horas de esforço produtivo. Ou seja, a empresa passou a negociar as horas de seus desenvolvedores durante o projeto, ao invés de negociar em escopo fechado pelo produto a ser criado. O Gerente explicou que existe um trabalho muito forte das equipes de vendas em demonstrar ao possível cliente que este modelo é mais vantajoso para ambas as partes, estabelecendo-se a definição do Product Backlog pelo Product Owner em detrimento de um escopo fechado e, ao invés de pagar pelo cumprimento deste escopo, o cliente “aluga” os serviços dos desenvolvedores durante as iterações que durarem para o desenvolvimento destas funcionalidades, com total liberdade de manipular o Product Backlog, acrescentando ou

modificando itens, visto que isto não implicará em quebras de contrato, apenas estenderá o tempo em que o time de desenvolvedores permanecerá atuando pelo projeto – e a empresa sempre recebendo corretamente por isso.

Questionado sobre o perfil do Product Owner, foi descrito como sendo um indivíduo (e, em raras exceções, mais de um indivíduo) que representa o cliente com um perfil estritamente de negócio, ou seja, com conhecimento profundo das necessidades do negócio. Em todos os projetos conduzidos pela empresa, o Product Owner participa ativamente das Reuniões de Planejamento do Sprint, discutindo sobre os rumos que o projeto tem tomado e as próximas necessidades desejadas, ou seja, quais itens do Product Backlog são mais importantes para a próxima iteração.

Em alguns casos, se desejado pelo cliente, o Product Owner participa também das Reuniões Diárias em momentos críticos do projeto, para manter-se ciente do andamento e entender melhor possíveis problemas que estejam ocorrendo, contudo, sem grandes interferências, visto que as decisões técnicas são exclusivas ao time de desenvolvedores.

Como conclusão da reflexão sobre este princípio enxuto, o Gerente entende que o processo é adequado e, seguindo-se o Scrum corretamente, não existem comprometimentos antecipados e que este princípio enxuto está efetivamente em execução na empresa.

5.3.1.4 Entregue rápido

As entregas rápidas são reforçadas pelo próprio processo utilizado, orientado em iterações na forma de Sprints, conforme sugere a abordagem Scrum.

A empresa utiliza Sprints com duração de duas semanas, um tamanho relativamente curto em comparação à sugestão de trinta dias encontrada na bibliografia sobre Scrum, conforme apresentado anteriormente na seção 4.6.

Conforme explicado pelo Gerente, as funcionalidades existentes no Product Backlog são quebradas em Histórias pequenas que caibam dentro do intervalo de duas semanas e que possam proporcionar um incremento de *software* funcional após este intervalo. Questionado sobre a veracidade dessa expectativa, ele informou que efetivamente as Histórias são escritas e priorizadas de modo a proporcionar algo funcional ao final do Sprint e que isto ocorre desde a primeira iteração de um projeto mas, realmente, a Definição de "Pronto" é bem trabalhada com o cliente antes do início do projeto para que as expectativas nas primeiras iterações não sejam demasiadamente equivocadas, pois as rápidas entregas inicias são importantes para demonstrar o andamento do projeto e proporcionar *feedback* rápido, mas raramente representam um *software* realmente passível de uso pelo cliente desde o início.

Como conclusão da reflexão sobre este princípio enxuto, o Gerente entende que o processo é adequado e, seguindo-se o Scrum corretamente, as entregas incrementais proporcionam *feedback* rápido e mantêm o cliente satisfeito com a evolução do projeto, ao mesmo tempo em que não comprometem a equipe com tarefas em excesso, ou seja, este princípio enxuto está efetivamente em execução na empresa.

5.3.1.5 Valorize a equipe

A avaliação da aplicação deste princípio estendeu-se por diferentes entendimentos do sentido deste conceito.

Considerando a valorização da equipe no sentido de proporcionar autonomia, a empresa valoriza e compreende que os desenvolvedores são os especialistas técnicos e que compete totalmente a eles todas as decisões neste sentido durante o projeto.

O Scrum Master restringe-se exclusivamente ao papel de garantir que o processo está sendo seguido de maneira adequada, não influenciando em decisões técnicas. Inclusive, o Scrum Master é um indivíduo único para toda a empresa, e "flutua" entre os vários projetos conforme a necessidade. Ficou evidente que este papel não é crítico na visão da empresa e que sua atividade não é constante no dia-a-dia dos projetos.

Dentro de cada equipe, desenvolvedores mais experientes atuam como líderes, porém de forma natural. Ou seja, pessoas mais experientes vão assumindo decisões mais críticas e guiando os menos experientes, sem que exista, porém, uma hierarquia formal ou algum controle sobre isso.

Entretanto, se por um lado a equipe como um todo tem total autonomia sobre os projetos, por outro lado não existem mecanismos formais de reconhecimento e valorização de indivíduos. Ou seja, não existem mecanismos de detecção de desenvolvedores melhores. À medida em que um desenvolvedor torna-se mais experiente e assume mais responsabilidades em vários projetos, vai ganhando destaque na empresa e sendo escalado para projetos mais importantes, mas isto ocorre de forma não estruturada.

Como conclusão da reflexão sobre este princípio enxuto, o Gerente reconheceu que, ao mesmo tempo em que as equipes são valorizadas, cada indivíduo pode passar despercebido e que, possivelmente, mecanismos mais específicos poderiam ser criados para que a empresa tenha melhor visão sobre o desempenho e a evolução de cada pessoa.

5.3.1.6 Adicione segurança

Para facilitar a análise deste princípio enxuto, foi sugerido ao Gerente responder separadamente sobre as medidas que adicionam segurança ao processo e as que adicionam segurança sobre a qualidade do produto.

Sobre as medidas que adicionam segurança ao processo, ele foi enfático em dizer que seguir o Scrum adequadamente é certamente a medida mais importante, visto que todo o fluxo sugerido pela metodologia é focado em adicionar segurança. Como práticas mais importantes dentro desse fluxo, foram citadas as estimativas realizadas em grupo através do Jogo de Planejamento, a priorização de histórias de usuário conduzida sempre pelo Product Owner, a condução do trabalho em Sprints curtos, as Reuniões Diárias de acompanhamento e a utilização de Folga (de 1 a 3 dias, dependendo do projeto) no planejamento de cada Sprint, para proporcionar um espaço de tempo para ajuste e acomodação de desvios entre o estimado e o executado para as histórias do Sprint.

Questionado se o Scrum realmente é seguido à risca e em todos os projetos, foi respondido que a empresa faz todo o possível para que seja dessa forma e que o Scrum Master é a garantia de que será compreendido e executado adequadamente. Mas, em situações raras, não é seguido em algum projeto por decisão proveniente do cliente, acordada antes mesmo do início do projeto. Ele ainda destacou que, se há alguns anos no início da adoção do Scrum era mais desafiador explicar ao cliente as vantagens deste modelo, hoje é difícil encontrar algum cliente que não queira conduzir o projeto desta maneira.

Quantos a medidas que adicionam segurança sobre a qualidade do produto, destacou-se como principais práticas a Revisão de Código em Pares, Repositório Único de Código, Integração Contínua, elaboração de Testes Unitários e Entregas Incrementais.

Observa-se que todas estas práticas são abordadas em detalhes pela XP, contudo a empresa não declara utilizar Programação Extrema oficialmente ou mesmo uma forma híbrida, como ScrumXP. Questionado sobre isso, o Gerente declarou que não conhece a fundo a XP para garantir que realmente utilizam esta abordagem.

Vale destacar também, conforme citado anteriormente, que o Test-Driven Development é utilizado em alguns projetos, quando em comum acordo com o cliente, como uma prática adicional de segurança que favorece tanto o processo quanto a qualidade do produto, visto que guia o fluxo de desenvolvimento com base nos resultados esperados (testes unitários criados antes da codificação da história) ao mesmo tempo em que garante que os

resultados permanecerão de acordo após implementações futuras (teste de regressão automatizado).

Como conclusão da reflexão sobre este princípio enxuto, o Gerente acredita que a constante utilização do Scrum, em conjunto com todas as demais práticas citadas, provenientes de outras abordagens ágeis, proporciona a adição de segurança ideal tanto para o processo quanto para a garantia da qualidade do produto e que este princípio enxuto está efetivamente em execução na empresa.

5.3.1.7 Otimize o todo

No contexto de visão do todo sobre o que ocorre no projeto, todos os integrantes da equipe aproveitam da facilidade oferecida pelo Scrum através das Reuniões Diárias de Planejamento, onde rapidamente todos ficam a par do que foi realizado no dia anterior por cada, quais os problemas encontrados e o que se espera realizar hoje. Este modelo de autogerenciamento oferece uma ótima oportunidade para que todos estejam cientes do que ocorre e também possam sugerir melhorias – ao processo e ao produto sendo criado – de forma simples e produtiva.

Mesmo em projetos compostos por pessoas em locais físicos diferentes (em outras unidades da empresa ou mesmo alocados em clientes em outros países), a participação nestas reuniões ocorre da mesma forma, através da infraestrutura de conferência existente na sala de reuniões: câmera, microfones e projetor, para que as pessoas possam se ver e conversar em tempo real.

A padronização do fluxo de trabalho através do Scrum entre todas as unidades da empresa oferece também a garantia de que todos conheçam o funcionamento do processo e, portanto, tenham facilidade para detectar pontos de melhoria e sugerir mudanças. Entretanto, não se realizou um mapeamento da cadeia de valor para oferecer uma visão em detalhes sobre o processo, com os tempos demandados para cada etapa e os desperdícios entre as atividades e, portanto, este tipo de informação detalhada para otimização do todo, conforme se sugere para este princípio enxuto, não foi empregada pela empresa.

Existem ainda listas de discussão internas por e-mail, para proporcionar uma visão geral do todo em relação às escolhas técnicas de vários projetos. Ou seja, embora as equipes possuam total liberdade sobre decisões técnicas tomadas em seus projetos, estas listas de discussão proporcionam uma maneira de conhecer que tipos de decisões estão sendo tomadas em outros projetos da empresa que se utilizam das mesmas tecnologias e plataformas. Estas listas por e-mail também representam uma forma simples, mas eficaz, para que

desenvolvedores que estejam enfrentando um problema técnico busquem apoio de outros desenvolvedores, de outros projetos e unidades, que possam já ter passado por problemas similares.

Se possibilidade de conhecimento e sugestão de otimização de processos de toda a empresa existem e são naturalmente favorecidos pelo Scrum, não é possível dizer o mesmo da visão do todo sobre o produto. Não existe um mecanismo para proporcionar visão do todo sobre o produto que está sendo concebido, ou seja, o que já foi discutido neste trabalho e estudado por Petersen (2010) parece realmente um problema inerente às metodologias ágeis: os desenvolvedores tem uma visão especifica do que está sendo criado naquele Sprint mas pouca visão do que realmente espera-se que o produto se torne por parte do cliente.

Mesmo que o Product Backlog possa oferecer uma visão antecipada de funcionalidades futuras, o Gerente avaliou que realmente um desenvolvedor não tem visão real de como o projeto irá prosseguir no futuro, principalmente para projetos grandes e, desta forma, pouco pode fazer para tomar decisões técnicas imediatas que ofereçam melhores resultados para situações que encontrará no futuro. Ele ainda refletiu sobre este princípio enxuto, avaliando que o equilíbrio para melhorar sua adequação é muito tênue, uma vez que tentativas de antecipar funcionalidades com base em uma análise antecipada do projeto como um todo seria um retrocesso aos processos tradicionais da engenharia de *software* e representaria uma antítese ao principal princípio enxuto, a eliminação de desperdícios.

5.3.2 Percepção dos desenvolvedores

As entrevistas conduzidas com o Diretor de Tecnologia e com o Gerente da unidade proporcionaram uma visão de alto nível sobre a aplicação dos conceitos enxutos e sua importância para a empresa, entretanto percebeu-se necessário também compreender a percepção dos desenvolvedores sobre o processo no dia-a-dia de trabalho.

As descrições contidas nos próximos tópicos foram organizadas com base em questionário aplicado aos desenvolvedores, cujo roteiro de perguntas realizadas encontra-se disponível no Apêndice C deste trabalho.

Para a condução do questionário, solicitou-se ao Gerente da Unidade C que apontasse cinco desenvolvedores da unidade com disponibilidade para respondê-lo e que possuíssem diferentes tempos de atuação na empresa e de experiência profissional. A partir daí, o autor deste trabalho se comunicou diretamente com estes desenvolvedores através de e-mail, solicitando as respostas das questões elencadas no Apêndice C, a fim de consolidar o perfil

pessoal destes. Em seguida, três desenvolvedores foram escolhidos para continuidade no questionário, cujos dados consolidados estão apresentados no Quadro 5.2 a seguir.

Quadro 5.2 - Perfil pessoal dos desenvolvedores respondentes do questionário.

Questão	**Desenvolvedor A**	**Desenvolvedor B**	**Desenvolvedor C**
C.2.1	27 anos	20 anos	24 anos
C.2.2	Processamento de Dados	Análise e Desenvolvimento de Sistemas	Análise e Desenvolvimento de Sistemas
C.2.3	2008	2014 (Previsão)	2012
C.2.4	8 anos	7 meses	4 anos
C.2.5	5 anos	7 meses	2 anos
C.2.6	Desenvolvedor	Desenvolvedor	Desenvolvedor
C.2.7	4 anos	7 meses	4 anos
C.2.8	Sim e possui certificação sobre isso	Somente treinamentos ao entrar na empresa	Sim e possui certificação sobre isso

Fonte: elaborado pelo próprio autor.

Conforme pode ser observado através dos dados apresentados, o perfil dos desenvolvedores escolhidos para responder o questionário foi propositalmente variado dentre as opções disponibilizadas, uma vez que se considerou mais relevante para os resultados do estudo analisar percepções com vieses distintos de forma a obter revelações mais imparciais.

Sobre o motivo para se prosseguir com apenas três desenvolvedores, e não com os cinco iniciais, optou-se por resguardar suas identidades e garantir que o Gerente não soubesse quem foram os escolhidos para que, desta forma, não pudesse interferir ou influenciar as respostas de alguma maneira. Além disso, o número inicial de cinco desenvolvedores apontados para a seleção de três foi exigência do Gerente da Unidade C, dada a inviabilidade de disponibilizar mais desenvolvedores por conta de muitas atividades nos projetos em andamento no momento da condução do estudo.

Dentre os perfis analisados, o Desenvolvedor A é o mais experiente da unidade, já tendo inclusive atuado em outras duas empresas antes desta, porém só tendo iniciado a atuação com metodologias enxutas na Empresa X. Vale relembrar o já citado anteriormente na seção 5.2: o Gerente da Unidade C é oficialmente também desenvolvedor, e seria, portanto, o mais experiente da unidade e há mais tempo na empresa. Entretanto, por assumir também estas atribuições administrativas de maneira concomitante e por ter participado deste estudo através de entrevista, optou-se por não considera-lo para o questionário, tornando assim o Desenvolver A o mais experiente neste contexto.

Já o Desenvolvedor B é o extremo oposto, estando em início de carreira, ainda cursando sua graduação e há pouco tempo na empresa. Este, durante os três primeiros meses

em período de experiência, conseguiu se adequar bem ao processo de trabalho, gerando resultados satisfatórios, tendo sido efetivado em seguida. Foi acrescido à equipe durante um projeto em andamento, em uma seleção mais rápida que o normal, por se tratar de reposição da vaga de um ex-funcionário que havia deixado a empresa. Embora inicialmente a escolha de um desenvolvedor com pouca experiência para participar do estudo possa parece contraditória, as percepções deste desenvolvedor foram muito relevantes, principalmente por não estar "viciado" no processo empregado na empresa, tendo inclusive contribuído várias vezes com críticas enquanto os outros só enxergaram pontos positivos.

Por fim, o Desenvolvedor C possui um tempo de experiência intermediário, formou-se recentemente e já atuou em outra empresa antes enquanto cursava sua graduação, inclusive tendo estudado e atuado com metodologias enxutas também naquele período, antes de atuar na Empresa X, estando, portanto, já acostumado a um modelo de processos ágil.

Estas respostas também deixam evidente que a empresa realmente não diferencia funções entre os funcionários técnicos, sendo todos denominados Desenvolvedores, independentemente da experiência ou do tempo de atuação pela empresa. As atribuições dentro do ciclo de vida de desenvolvimento de *software* também são genéricas e abrangentes, não ocorrendo práticas como deixar funcionários menos experientes exclusivamente conduzindo testes de *software*, para adquirirem mais experiência, antes de começarem a programar suas próprias rotinas.

Conhecido o perfil dos desenvolvedores respondentes do questionário, a seguir serão apresentadas as respostas fornecidas por cada um deles para as perguntas realizadas, conforme constantes no Apêndice C.3.

5.3.2.1 Percepções sobre a questão C.3.1

Você sente que está evoluindo seus conhecimentos enquanto desenvolve suas atividades? Se sim, que meios são oferecidos para incentivar seu aprendizado?

Esta questão foi elaborada no intuito de detectar a percepção dos desenvolvedores respondentes sobre o aprendizado, inclusive quanto aos mecanismos da empresa para seu incentivo, estando diretamente relacionada ao princípio enxuto "Amplifique o aprendizado".

O Desenvolvedor A respondeu de forma muito positiva sobre esta questão, confirmando o que já havia sido respondido pelo Gerente (conforme exposto no tópico 5.3.1.2), principalmente sobre a importância dada pela empresa na busca por certificações. Este desenvolvedor argumentou que o fato da empresa ter buscado treinamentos oficiais em Scrum durante o processo de adoção desta metodologia foi determinante não só para

impulsionar o aprendizado sobre isso aos desenvolvedores, como também impulsionar a mudança cultural necessária, uma vez que a grande maioria das pessoas já estava acostumada a trabalhar seguindo modelos de processos prescritivos. Este desenvolvedor ainda enfatizou que a empresa valoriza muito a evolução profissional de cada indivíduo e apoia suas buscas individuais por novos desafios, contando que recentemente passou pouco mais de dois meses no escritório da empresa nos EUA, melhorando sua fluência em inglês e visitando um cliente americano que fazia questão de contar com uma presença mais constante e próxima de alguém da equipe de desenvolvimento durante as primeiras iterações de um projeto crítico, para melhorar a comunicação e garantir que os primeiros requisitos – e mais críticos – seriam corretamente levantados, compreendidos e desenvolvidos.

O Desenvolvedor B disse "estar empolgado", em suas próprias palavras, visto que a empresa informou que arcaria com os custos de exames de certificação. Contudo, o desenvolvedor fez a ressalva de que, sendo novo na empresa (e na carreira), está ciente de que ainda tem muito que aprender, mas não necessariamente tem visão efetiva de qual seria o melhor caminho a seguir. Na prática, a escolha da certificação para a qual está estudando no momento não foi dele, mas uma sugestão de um amigo, pois seria uma certificação valorizada no mercado e que poderia melhorar seu rendimento financeiro. Em nenhum momento o Desenvolvedor B citou em sua resposta sobre escolher estudar determinado assunto pela importância disso na equipe ou em algum projeto específico, parecendo realmente mais preocupado com um possível retorno financeiro deste investimento em aprendizado.

O Desenvolvedor C elogiou fortemente o programa de treinamentos da empresa, principalmente no que se refere aos treinamentos sobre Scrum, inclusive pelo fato do Diretor de Tecnologia da empresa ser instrutor oficial da metodologia. Disse também já ter recebido treinamentos rápidos de outros assuntos, quando participou de projetos com pessoas mais experientes que ele. Contudo, enfatizou que várias vezes, por rotatividade de mão-de-obra, pessoas inexperientes precisam ser acrescidas a um projeto em andamento (principalmente dada a grande quantidade de projetos médios e longos existentes na empresa), então nem sempre existe muito tempo hábil para capacitar adequadamente as novas pessoas. Apesar da crítica citada, o desenvolvedor assegurou que em nenhum momento essa situação impactou realmente de forma negativa, a ponto de causar o fracasso o projeto, tendo, no máximo, gerado alguns pequenos problemas e retrabalhos que foram rapidamente absorvidos pelo processo iterativo, pela prática de folga ao planejar os Sprints e pelas efetivas reuniões de retrospectiva ao final de cada iteração.

As respostas dos desenvolvedores corroboram com a percepção já citada do Gerente. Também enfatizam que, embora existam incentivos da empresa para o aprendizado, a falta de uma formalização – e talvez até de um roteiro dos conhecimentos técnicos que se espera que os desenvolvedores evoluam – não permite o aproveitamento máximo deste conceito.

Por fim, os três desenvolvedores respondentes também elogiaram a leveza e a liberdade que o ambiente lhes transmite para o aprendizado, pois alegaram que todos os desenvolvedores da Unidade C são muito próximos e se ajudam mutuamente, ensinando coisas aos outros, evoluindo em grupo, mesmo que atuando em projetos separados.

5.3.2.2 Percepções sobre a questão C.3.2

Quem pode tomar decisões técnicas durante o projeto (como escolher novos padrões de código, arquitetura, escolher plug-ins e bibliotecas, reescrever algum código utilizado por vários componentes, etc.)?

Esta questão foi elaborada no intuito de validar a percepção dos usuários sobre a auto-organização da equipe em relação ao que se espera no pensamento enxuto e que inclusive já havia sido detectado nas conversas com o Gerente e durante a observação direta na unidade.

Sobre esta questão, o Desenvolvedor A esclareceu que parte das definições técnicas refere-se a acordos previamente estabelecidos durante a venda do projeto para o cliente. Por exemplo, por se tratar de uma empresa especializada em tecnologias Microsoft, raramente plataformas de outras empresas, como o Java da Oracle, são vendidas como solução ao cliente. Desta forma, diversos conceitos relacionados à plataforma tecnológica preferencial da empresa já são pré-estabelecidos e não compete a ninguém da equipe (ou do cliente) decidir por algo diferente em nenhum momento. Para outras decisões mais especificas, como a escolha do modelo arquitetural da aplicação, a utilização de *plug-ins* e bibliotecas de código para apoiar e/ou automatizar determinados recursos, bem como os padrões e estilos de codificação que serão adotados para o projeto, normalmente são decisões tomadas exclusivamente pela equipe de desenvolvimento. Este desenvolvedor ainda assume que os mais experientes na Unidade, como ele, certamente são mais determinantes nas escolhas da equipe, até por uma questão de respeito e admiração por parte dos outros integrantes.

O Desenvolvedor B apresentou uma visão que corrobora completamente o exposto acima, somente acrescentando que no projeto específico que vem trabalhando desde sua chegada à empresa, grande parte das decisões de desenvolvimento foram comunicadas ao Product Owner, que possui um perfil de Gestor de TI dentro do cliente, conhecendo não apenas as necessidades do negócio, mas também necessidades específicas tecnológicas sobre

o *software* que estava sendo concebido. Este desenvolvedor, contudo, garantiu que nunca aconteceu do PO "obrigar" a equipe de desenvolvimento a adotar uma escolha técnica, sempre tendo sido o cenário inverso, com a equipe propondo alguma novidade técnica e o PO aprovando (ou recusando) conforme sua compreensão da relevância – e da adequação – da referida decisão para o produto em questão.

O Desenvolvedor C destoou um pouco das respostas dos outros dois, dizendo que nem sempre se sentiu livre pra propor e aplicar todas as escolhas tecnológicas que queria. Argumentou que a escolha sobre a utilização de alguns padrões de arquitetura, interface, componentes e bibliotecas é relativamente unificada para toda a empresa, e que as listas de discussão e trocas de e-mails entre desenvolvedores de todas as unidades oferece um mecanismo de padronização que, por outro lado, algumas vezes se parece mais com um entrave. Ele exemplificou uma situação que, ao sugerir a utilização de uma biblioteca de código de interface gráfica específica para um projeto, não bastou apenas convencer a equipe de que seria uma boa escolha, visto que após várias trocas de mensagens com outros desenvolvedores de outras unidades (e no caso citado, inclusive o envolvimento do Diretor de Tecnologia na decisão), chegou-se à conclusão que seria mais prudente codificar naquele projeto da mesma maneira como já havia sendo realizado em outros anteriormente.

Com base nestas respostas, e em tudo que foi observado pessoalmente durante a condução deste estudo, que os desenvolvedores sentem-se livres para ter ideias e sugerir inovações em cada novo trabalho, mas que existe um controle administrativo por parte da Direção de Tecnologia e uma tendência de padronização da maneira de se codificar entre as equipes, que não permite total liberdade de decisão.

5.3.2.3 Percepções sobre a questão C.3.3

Como é o controle de atividades na equipe? Como cada um sabe o que o outro está fazendo, se está com problemas, se depende de alguma coisa de outra pessoa da equipe ou de pessoas externas (como definições por parte do cliente)?

Esta questão foi elaborada no intuito de possibilitar a compreensão sobre os mecanismos utilizados para que toda a equipe saiba como está ocorrendo a evolução do Sprint e quais os problemas que podem lhe comprometer.

Os três desenvolvedores respondentes do questionário citaram como principal mecanismo as Reuniões Diárias, que ocorrem em pé, com duração máxima de 15 minutos. Todos os passos mais importantes que ocorreram no dia anterior são expostos nesta reunião e todos os grandes problemas que podem representar uma barreira na conclusão do Sprint são

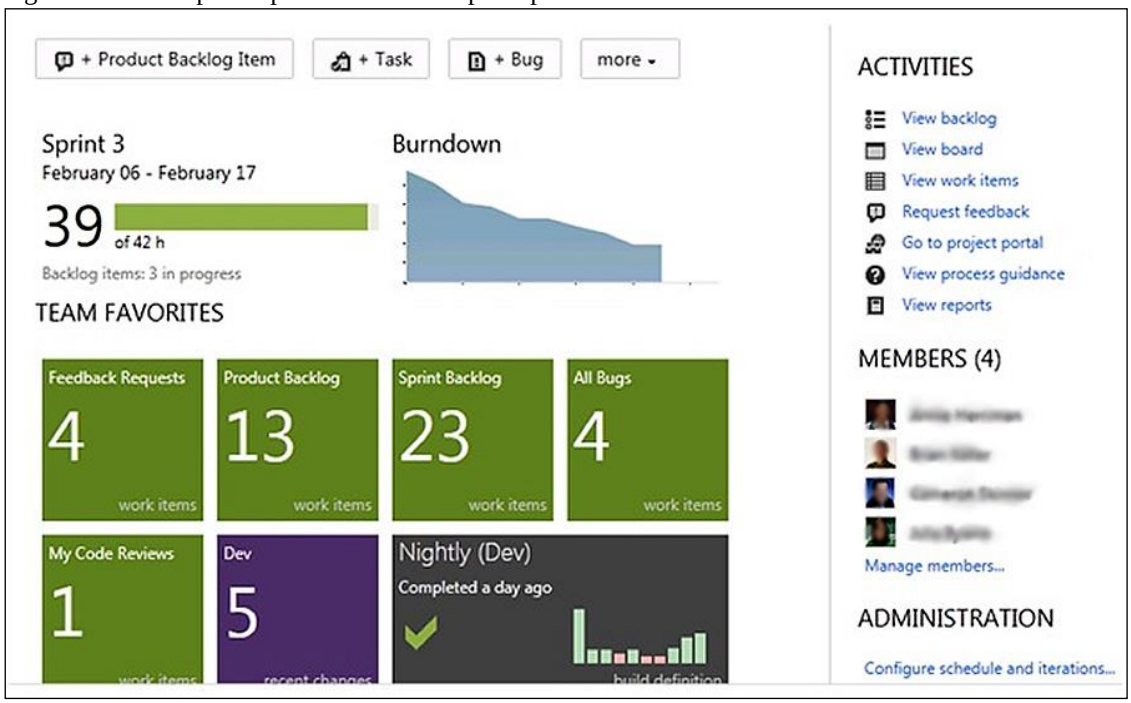
explanados a todos os participantes, que rapidamente podem sugerir alguma contingência ou disponibilizar ajuda. Como se pressupõe que tais reuniões sejam realmente rápidas e diretas, nas situações onde um problema é complexo demais para ser resolvido naquele momento, alguns desenvolvedores continuam discutindo e procurando soluções sobre ele com mais calma após a reunião.

Os desenvolvedores também acrescentaram um comentário importante sobre esta questão de todos acompanharem o andamento do projeto, o que inclusive foi corroborado durante a observação direta: a utilização das ferramentas visuais da plataforma Team Foundation Service da Microsoft.

Através desta plataforma, centralizam-se recursos importantes para a condução ágil do desenvolvimento de *software*, como a elaboração do Product Backlog, a distribuição das tarefas entre os membros da equipe em cada Sprint, a geração de gráficos em tempo real sobre o andamento do projeto, como Burndown Chart e quadro Kanban digital para acompanhamento do andamento das tarefas.

A seguir, na Figura 5.3, demonstra-se um exemplo de visualização do painel de controle principal proporcionado pelo TFS, demonstrando um resumo do andamento deste Sprint do projeto, bem como diversos números de destaque para facilitar o acompanhamento visual do andamento do projeto por parte de todos os integrantes.

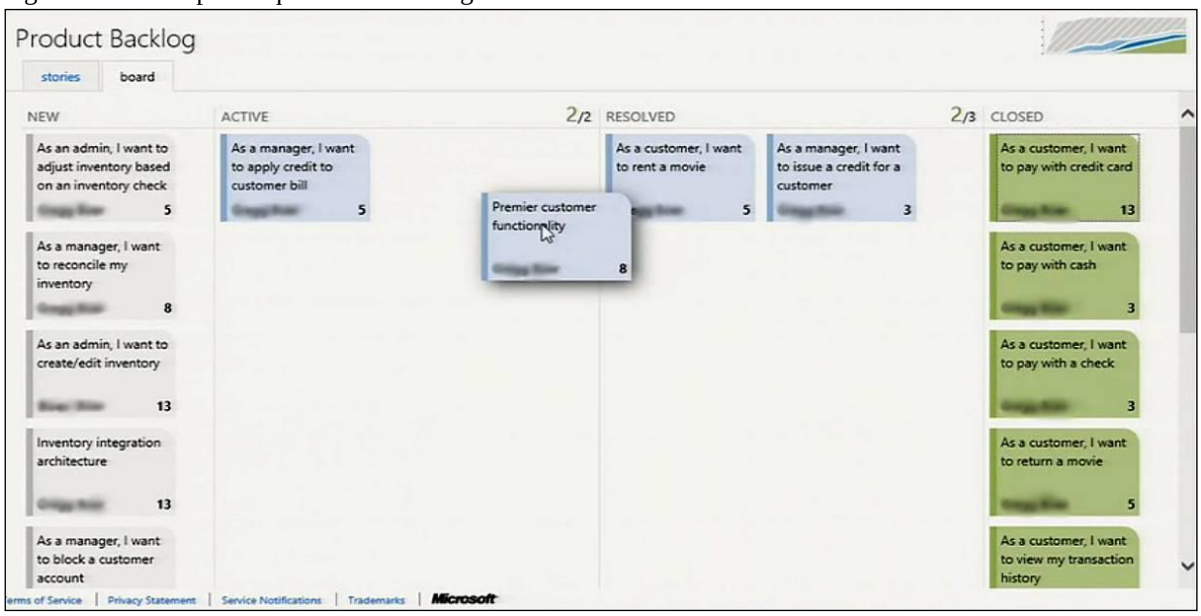
Figura 5.3 - Exemplo do painel de controle principal do Team Foundation Service.



Fonte: captura de tela cedida pela empresa.

Esta plataforma oferece mecanismos de apoio estruturados com base nos termos utilizados no Scrum, mas também disponibiliza recursos para abordar outras práticas enxutas, como o já citado quadro Kanban, conforme este exemplo abaixo na Figura 5.4.

Figura 5.4 - Exemplo de quadro Kanban digital do Team Foundation Service.



Fonte: captura de tela cedida pela empresa.

Com base nesta constatação de que não se utiliza a ferramenta Kanban de forma física, sem que esteja, portanto, constantemente visível a todos, questionou-se ao Gerente da unidade se este tipo de Kanban digital efetivamente surte o efeito de manter todos informados e, principalmente, porque não utilizar um quadro físico constante visível a todos. A resposta foi direta e coerente: embora em muitos projetos os desenvolvedores estejam atuando fisicamente no mesmo local, dentro da Unidade C, não são raras as situações onde desenvolvedores situados em localidades diferentes participam do mesmo projeto. Neste cenário, um quadro Kanban estampado em uma parede de nada serve se os integrantes da equipe em outros locais físicos não podem acompanha-lo.

Sobre esta mesma indagação, o Desenvolvedor C ainda acrescentou o seguinte pensamento objetivo: “as tarefas são conduzidas o tempo todo olhando para a tela do computador. O melhor lugar para acompanhar o andamento dos outros é ali mesmo, onde já estamos olhando sempre”.

Portanto, com base nas respostas desta questão e na observação direta realizada, percebe-se que as ferramentas utilizadas apoiam adequadamente o acompanhamento do projeto e, ao mesmo tempo, suportam diretamente diversos conceitos enxutos, auxiliando na correta condução do fluxo do processo.

5.3.2.4 Percepções sobre a questão C.3.4

Você tem conhecimento apenas das tarefas da Sprint atual (e das anteriores que participou) ou sabe sobre o andamento de todo o projeto, o que o cliente espera do *software* e tem ideia do que esperar das iterações futuras?

Esta questão foi elaborada com o intuito de validar a percepção do todo por parte de cada indivíduo da equipe, a fim de detectar se a percepção obtida durante a entrevista com o Gerente seria confirmada.

Todos os desenvolvedores responderam com o entendimento de que possuem conhecimento sobre todo o cenário no projeto, porém dentro do escopo necessário, ou seja, não é pretensão de ninguém (nem mesmo do Product Owner) saber tudo que o produto poderá se tornar após anos de projeto, mas sim saber tudo que o produto deverá se tornar dentro de um intervalo limitado. Para isso é importante não só uma concepção adequada do Product Backlog, mas também a visibilidade e o conhecimento de todos, mesmo que muitas tarefas ainda demorem a ser adicionadas em algum Sprint.

O Desenvolvedor B, embora citando o mesmo tipo de situação, destacou que diversas vezes já se sentiu "perdido", não sabendo ao certo por que a História que estava desenvolvendo era relevante ao *software*. Porém, os outros dois desenvolvedores não fizeram qualquer comentário sobre isso, podendo-se supor que a falta de experiência do Desenvolvedor B foi a causa desta sensação.

5.3.2.5 Percepções sobre a questão C.3.5

Se alguém lhe pede ajuda ou informa que precisa de alguma coisa sua pra prosseguir com a atividade dele, mas isso não está atualmente contemplado na previsão inicial sobre o que você está trabalhando, qual sua ação?

Esta questão foi elaborada com o intuito de compreender o posicionamento dos desenvolvedores sobre a colaboração entre suas atividades.

Sobre pedidos de ajuda, todos os três desenvolvedores responderam com uma posição muito positiva, argumentando que o ambiente é muito colaborativo e de ajuda mútua. Os desenvolvedores rapidamente se acostumam com isso e, desta forma, não ficam presos e isolados ao enfrentarem problemas, sempre tirando dúvidas com os colegas que estão em volta na unidade.

Em casos mais específicos, onde sejam dúvidas sobre a utilização de uma nova tecnologia ou sobre alguma regra de negócio que não se tenha grande conhecimento, e que

ninguém próximo na unidade consiga responder, as listas de discussão internas que interligam todos os desenvolvedores costumam surtir resultados rápidos na maioria das vezes, conforme explicou de forma mais aprofundada o Desenvolvedor C.

Já sobre interdependência entre Histórias que impeçam em algum momento a evolução, o Desenvolvedor A disse ser um caso muito raro, uma vez que durante as Reuniões de Planejamento do Sprint existe um esforço muito grande e coletivo para não alocar Histórias que não possam ser concluídas por completo por dependerem de algo que ainda não será desenvolvido.

No caso de codificação de Histórias dependentes que precise inevitavelmente ocorrer de forma concomitante, o Desenvolvedor A ainda argumentou que as Reuniões Diárias auxiliam essa organização, uma vez que nelas todos os integrantes da equipe expõe o que já realizaram no dia anterior e o que pretendem realizar neste novo dia de atividades. No momento em que alguma atividade prevista se choque com a necessidade de outra ainda não realizada, a equipe pode reposicionar as tarefas do Sprint para adequar melhor o uso do tempo, uma vez que o importante é terminar o Sprint com as Histórias previstas concluídas, independentemente da ordem em que foram executadas.

O Desenvolvedor B citou um exemplo de uma situação onde pegou para desenvolver uma atividade que se mostrou muito mais complexa do que havia sido estimado durante o planejamento inicial da iteração e que não poderia ser concluída a tempo, dependendo de muitas outras tarefas não previstas. Naquele caso, não foi possível terminar a História, mas como foi possível detectar isto durante o decorrer do Sprint, foi viável antecipadamente deixar o Product Owner ciente da situação, evitando maiores crises com o cliente e, inclusive, permitindo que o próprio PO escolhesse qual outra História do Product Backlog poderia ser "puxada" para dentro do Sprint e conduzida no lugar desta, resultado em algumas mudanças inesperadas na condução do Sprint, mas gerando um impacto muito pequeno na condução do projeto como um todo.

O Desenvolvedor A também citou que, num momento de grande dúvida, quando a equipe não consegue decidir qual a melhor ação a tomar dentro de um imprevisto que inviabilize o cumprimento de parte ou de todo o Sprint, o papel do Scrum Master é efetivamente relevante para auxiliá-las na condução das escolhas mais adequadas dentro do que a metodologia Scrum prega.

5.3.2.6 Percepções sobre a questão C.3.6

Se você encerra uma tarefa antes do planejado, o que costuma fazer com o tempo que sobra? O ambiente favorece de alguma forma a utilização desse tempo?

Esta questão foi elaborada no intuito de avaliar muitas facetas dos desenvolvedores, desde sua propensão a realizar estimativas precisas, seu interesse em estudar nas horas vagas ou em ajudar os demais, até seu entendimento sobre responsabilidade e auto-cobrança.

O Desenvolvedor A começou sua resposta garantindo que o cenário ideal para qualquer equipe ágil é evoluir sua capacidade de estimativa a cada iteração, chegando num ponto onde sobrar tempo torna-se algo incomum. Por outro lado, o próprio desenvolvedor esclarece que as estimativas já são realizadas considerando todo o tempo necessário do início ao fim do processo, ou seja, desde sua concepção até o final de todos os testes que garantam seu estado de pronto e que, neste intervalo, o mecanismo de Folga é adequado para preparar a equipe para imprevistos e que, quando estes não ocorrerem, é possível mesmo sobrar algum tempo, que normalmente acaba sendo utilizado para ajudar outras pessoas, seja presencialmente na unidade, seja nas listas de discussão internas da empresa, ou até em fóruns e *sites* de ajuda em programação públicos na Internet.

Já o Desenvolvedor B concentrou sua resposta em justificar que sua produtividade ainda não é similar à dos desenvolvedores mais experientes, que está ciente disso e estudando bastante para melhorar. Desta forma, raramente encontra-se em situações onde sobra tempo e que, não fosse a colaboração de outros desenvolvedores, provavelmente atrasaria algumas atividades que considera muito difíceis dentro do processo de desenvolvimento de *software*, como *layout* de interfaces ou criação de comandos SQL para execução no banco de dados.

Por fim, o Desenvolvedor C respondeu de forma próxima ao Desenvolvedor A, mas contrastante ao garantir que é normal sobrar algumas folgas durante o dia e que adorava estes momentos para adiantar o estudo pra alguma certificação, ou mesmo pra relaxar vendo algum vídeo ou visitando alguma rede social. Ele garantiu que estas brechas são muito importantes pra reestabelecer a criatividade e encontrar novas soluções para os problemas, de forma parecida com o que já havia sido respondido pelo Gerente no item 5.3.1.1 deste trabalho.

5.3.2.7 Percepções sobre a questão C.3.7

O processo de desenvolvimento de *softwares* adotado pela empresa é adequado? Os envolvidos nos projetos (você, os outros desenvolvedores, os clientes) estão satisfeitos com este processo?

Esta questão foi elaborada buscando compreender a satisfação do desenvolvedor quanto ao processo adotado pela empresa, bem como sua visão sobre a satisfação de todos os envolvidos, além de proporcionar espaço para possíveis reclamações sobre as abordagens utilizadas que por ventura não tivessem sido detectadas até então durante a pesquisa.

Os três desenvolvedores começaram a resposta a esta questão elogiando o modelo de processos utilizado pela empresa, enfaticamente elogiando o Scrum. Também elogiaram similarmente o empenho na empresa em seguir corretamente esta metodologia ágil e também a cultura de busca constante por melhoria no processo.

Após este ponto, as percepções dos desenvolvedores A e B permaneceram satisfatórias, declarando acreditar que todos os participantes dos projetos parecem felizes com os procedimentos adotados pela empresa e que raramente enfrentam atritos ou recusas por parte dos clientes.

O Desenvolvedor C ainda acrescentou que acredita que este sucesso na utilização do Scrum se deve também, em boa parte, à correta aplicação do conceito de Product Owner, bem como a grande dedicação deste. Este desenvolvedor citou que, na empresa em que trabalhou anteriormente, era o Gerente dos desenvolvedores quem exercia o papel de PO, blindando a equipe do contato direto com o cliente, o que escondia os problemas e manipulava as decisões sempre a favor de um relaxamento da equipe quanto ao cumprimento dos prazos, ou seja, o processo ágil era apenas um pretexto para o não cumprimento das responsabilidades dos desenvolvedores, além de manter inúmeros problemas inerentes às abordagens prescritivas, como fragilidade a mudanças, estimativas imprecisas, falta de transparência e insatisfação por parte do cliente, de maneira geral.

Já o Desenvolvedor B apresentou uma opinião contraditória após os elogios iniciais, alegando que, a partir de seu ponto de vista, a grande liberdade documental e o pouco tempo dedicado às fases de concepção e análise acabam causando uma dependência excessivamente grande na Refatoração, o que faz com que ele se sinta "patinando" ao criar algo, com a sensação ruim de que vai precisar gastar muito esforço futuramente para refazer e readequar boa parte do seu código. Os outros desenvolvedores não levantaram este tipo de reclamação em nenhum momento, sendo possível que seja um efeito colateral da pouca experiência deste desenvolvedor, ou seja, uma capacidade ainda pequena do "fazer certo da primeira vez", o que corroboraria as críticas de autores como Abrahamsson *et al.* (2010), quando ressalta que um processo ágil requer uma equipe mais preparada e experiência, senão o risco de retrabalho excessivo por falta de definições de projeto inviabiliza os ganhos da agilidade.

Dados os resultados positivos e as vantagens apontadas por todos que participaram deste estudo, é possível também supor que equilibrar alguns desenvolvedores menos experientes em meio a uma equipe predominantemente mais experiente, tanto em metodologias ágeis quanto em desenvolvimento de *software* de maneira geral, representa um bom caminho para contornar este problema de retrabalho causado pela diminuição no tempo dedicado à concepção e análise do projeto.

5.4 Resultados

Durante a descrição do caso estudado no decorrer desta seção, diversos pontos positivos sobre a utilização de princípios enxutos já foram citados. Contudo, a seguir os resultados detectados estão agrupados e resumidos para facilitar o entendimento.

Enfatiza-se ainda que, pelo caráter deste estudo exploratório e qualitativo, não se pretende quantificar os resultados detectados, inclusive a empresa também não os quantificou precisamente durante a implantação de novos métodos em seus processos. Portanto, os tópicos resumem as percepções análogas entre os participantes do estudo sob o ponto de vista dos fatores produtividade, qualidade, redução de custos e satisfação dos clientes.

5.4.1 Produtividade

Conforme apresentado anteriormente na seção 5.2.1 do presente trabalho, os problemas de cumprimento de prazos chegavam a 80% em relação ao estimado, segundo o Gerente da Unidade C, e eram praticamente constantes. Sobre esta questão, ele afirmou: “De todos os projetos que participei naquele período (e certamente foram mais de dez projetos), somente um não atrasou e era um pequeno algoritmo de migração de dados bem simples, que foi superestimada pela equipe de vendas, então terminamos bem antes do prazo final. Nunca presenciei eficiência em cumprimento de prazos em nenhum projeto naquela época”.

Em contrapartida, o Gerente informou que nunca houve um único problema de cumprimento de prazos grave após a adoção do modelo enxuto. Esta afirmação contundente representaria uma melhoria próxima 100% no cumprimento de prazos em relação ao cenário anterior, um resultado absolutamente expressivo, não fosse uma questão relevante a ser enfatizada: a definição de “Pronto” do Scrum pode ser bem genérica e permitir que o cliente considere sucesso algo que na verdade está apenas parcialmente concluído.

Para o Diretor de Tecnologia, este modelo em incrementos adotado por todas as abordagens enxutas, e muito enfatizado no Scrum, é justamente um dos maiores pontos

positivos para a produtividade na visão do cliente, pois elimina expectativas precipitadas e, graças a adiar comprometimentos, permite entregar pequenos lotes de incrementos mais concretos e relevantes ao negócio. Ainda segundo ele, é possível que as equipes ainda estejam longe a produtividade ideal quanto ao desenvolvimento do *software* como atividade técnica, mas o resultado oferecido ao cliente passa a percepção de grande ganho de produtividade.

O Gerente também citou que as ferramentas utilizadas possuem relatórios para estimativa de produtividade por desenvolvedor e que, neste sentido, o número de soluções que um desenvolvedor consegue produzir por dia continua próximo do cenário anterior, com variações pequenas na casa de 5% a 10%, dependendo do tempo do projeto, da experiência do profissional e das tecnologias utilizadas para o desenvolvimento. Para ele, o ganho em produtividade com metodologias enxutas/ágeis não está no indivíduo, mas sim na equipe, justamente pela eliminação de desperdícios discutida na seção 5.3.1.1.

Do ponto de vista dos Desenvolvedores, não se falou diretamente sobre ganho de produtividade, mas todos citaram que gostam e se sentem satisfeitos com o processo de desenvolvimento enxuto, se sentindo mais capazes de entregar *software* funcional ao final de cada iteração. Este enfoque no cliente, eliminando-se desperdícios de projeto que não entreguem efetivamente o que o cliente espera, apresentou-se claramente na Unidade C.

Resumindo-se a avaliação sobre os resultados positivos quanto à produtividade, este estudo presenciou pessoas altamente satisfeitas com os ganhos obtidos neste sentido e, mesmo que a produtividade de cada indivíduo não seja drasticamente maior pelo uso de abordagens enxutas, a equipe como um todo produz *software* de forma muito mais eficiente.

5.4.2 Qualidade

Quanto aos resultados sobre o fator qualidade, ainda relacionando-se com a afirmação citada sobre entregar *software* funcional ao final de cada iteração, demonstrou-se realmente de difícil mensuração, pois no contexto de entregar o que o cliente espera, realmente visualizou-se que o próprio modelo de processo do Scrum, bem como os mecanismos de adição de segurança discutidos anteriormente na seção 5.3.1.6, contribuem diretamente para que o produto criado seja exatamente – e apenas – o que o cliente necessita.

Todos os respondentes foram enfáticos em afirmar que os *softwares* criados segundo o Scrum não sofrem de problemas como recusas por não-aceitação por parte do cliente. Na pior das hipóteses, uma funcionalidade que não agrada o cliente é transformada em um novo item do Product Backlog e entra em um Sprint conforme a viabilidade de capacidade produtiva versus demanda atual.

Quanto a características técnicas para mensuração da qualidade de *software*, o estudo confirmou uma percepção já levantada em Pressman (2011), de que métodos enxutos não se preocupam claramente com questões como qualidade de arquitetura, de código, de padrões de projeto e, como já discutido, possuem preocupação quase nula com documentação. Neste sentido, seria possível considerar estes *softwares* de baixíssima qualidade sob o ponto de vista de abordagens prescritivas como o RUP.

Contudo, durante as observações diretas, foi possível presenciar a percepção discutida em Silva (2011), de que métodos enxutos modificam as preocupações sobre padronização e documentação de algo estático para algo dinâmico. Um bom exemplo são os casos de testes unitários escritos dentro do próprio código do sistema, principalmente nos projetos que se utilizam de TDD. Conforme argumenta o autor citado, e foi possível confirmar com este estudo, este tipo de mecanismo funciona como uma documentação executável que garante o bom funcionamento do *software* desenvolvimento e, de quebra, oferece uma possibilidade de entendimento facilitado sobre cada rotina do código, auxiliando em manutenções futuras.

Visto que nenhum dos Desenvolvedores participantes falou sobre dificuldade de manutenção e, inclusive, uma vez que a Unidade C possui projetos usando Scrum com mais de três anos de duração sem problemas nesse sentido, é possível sugerir-se que as práticas existentes são realmente adequadas para garantir a qualidade de codificação e documentação necessárias tanto para o cliente quanto para a equipe de desenvolvimento.

Resumindo-se a avaliação sobre os resultados positivos quanto à qualidade, confirmou-se o enfoque total no atendimento às necessidades do cliente, mas, ainda assim, visualizou-se que as equipes não sofrem para realizar manutenções em projetos longos e, portanto, o nível de qualidade técnica dos produtos criados parece ser adequado.

5.4.3 Redução de custos

Conforme já citado anteriormente, a empresa não teve interesse em revelar dados financeiros, tanto para a classificação de seu rendimento anual quanto para a análise de resultados sobre custos e lucros da utilização de métodos enxutos.

Embora sem números concretos para validar as percepções apresentadas, o Diretor de Tecnologia foi enfático ao afirmar que a redução do TCO (sigla em inglês para *total cost of ownership*, traduzido como custo total de propriedade) do *software* desenvolvido com filosofia enxuta é evidente, tanto para a empresa de desenvolvimento quanto para o cliente.

Conforme explicou ele, para a empresa reduz-se o custo tanto para produzir quanto para manter o *software*, pois não se perde capacidade produtiva com a criação de funções não

necessárias para o cliente. Além disso, a estruturação da empresa enxuta com o Scrum requer um organograma bem reduzido, conforme apresentado na seção 5.3 deste trabalho, ocasionando uma considerável redução de custos com diferentes níveis de gestão.

Ainda segundo o Diretor de Tecnologia, para o cliente a redução do custo total de propriedade ocorre no sentido de que, conforme discuto anteriormente neste estudo, a empresa não vende o produto como um todo, mas sim as horas de trabalho da equipe. Neste modelo de venda adotado pela Empresa X, paga-se pelo produto apenas enquanto existir interesse nisso – no desenvolvimento de novas funções, em sua evolução, em sua manutenção – e, com o enfoque nas necessidades do negócio, conforme também já apresentado neste estudo, se oferece uma garantia forte de que não se investirá em algo que não receberá.

Além de um TCO mais vantajoso, tanto para fornecedor quanto para cliente, o Diretor de Tecnologia também argumentou que, desta forma, o ROI (do inglês *return on investment*, traduzido como retorno sobre o investimento) é favorecido pelo pensamento enxuto, novamente por causa da eliminação de desperdícios e também pelas entregas rápidas, uma vez que o cliente recebe *software* funcional o mais rápido possível, além de receber exatamente o que precisa, não mais (o que representaria custo adicional desnecessário) ou menos (o que representaria investimento inadequado).

Sobre essa questão de custos, o Gerente da Unidade C só acrescentou que o organograma enxuto da empresa realmente representa redução de custos para manter o processo em funcionamento. Não realizou nenhum comentário adicional sobre isso, inclusive sobre o fato de responder como gerente da unidade mas ser oficialmente desenvolvedor, que poderia representar prejuízo financeiro para ele enquanto vantajoso em custos para a empresa.

Resumindo-se a avaliação sobre os resultados positivos quanto à redução de custos, confirmou-se que o processo enxuto proporciona um modelo menos custoso para a empresa de desenvolvimento de *software* e, possivelmente, também para o cliente. Entretanto, por ter sido apenas uma opinião e por não terem ocorrido conversas com clientes durante o estudo de caso, não foi possível confirmar esta percepção.

5.4.4 Satisfação dos clientes

Sobre a satisfação dos clientes em relação à empresa, tanto o Diretor de Tecnologia quanto o Gerente da Unidade C informaram que antes dos métodos enxutos, os atritos e disputas entre os interesses do cliente versus os interesses da equipe (em busca de evitar mudanças de escopo, para não impactar os prazos e os custos) eram constantes e corroíam

lentamente a relação entre as partes. Muitos clientes participavam de algum projeto e desistiam de estabelecer novos contratos posteriormente.

Com o estabelecimento do cenário enxuto, os clientes passaram a fazer parte ativa do projeto (principalmente graças ao papel do Product Owner) e as disputas por visões distintas deixaram de existir, uma vez que não existe mais excesso de comprometimentos antecipados que deixam a equipe engessada contra mudanças.

Conforme informou o Gerente, mesmo em situações onde o Product Owner participa ativamente do projeto, assistindo às reuniões diárias de planejamento, a equipe não se sente acuada e não atua tentando esconder os problemas existentes. A cultura de parceria com o cliente já está muito arraigada nas equipes e é fortemente trabalhada desde as vendas até os primeiros contatos com desenvolvedores quando da entrada de novos clientes.

O Diretor de Tecnologia comentou também que o número de clientes estava crescendo de forma acelerada, além de ter aumentado a fidelidade de clientes antigos em conduzir novos projetos. Informou também que uma parcela dos clientes inclusive estava procurando a Empresa X justamente por ouvir falar de sua utilização bem sucedida de Scrum. Esta situação demonstra uma tendência de mercado, segundo ele, uma vez que vários competidores diretos também começaram a se preocupar com metodologias ágeis e estão tentando suas próprias implementações em seu processo, com os clientes buscando cada vez mais este tipo de ambiente para a condução de seus projetos.

Embora com várias argumentações positivas, o Diretor de Tecnologia também citou que projetos ágeis costumam apresentar resultados apenas no médio prazo, o que assusta inicialmente clientes desacostumados com este modelo. Mas, segundo ele, como os resultados são muito positivos quando começam a aparecer, nunca ocorreu de um cliente realmente desistir do projeto antes disso.

Resumindo-se a avaliação sobre os resultados positivos quanto à satisfação dos clientes, embora satisfação seja algo subjetivo, os envolvidos no estudo demonstraram que a relação com os clientes é mais transparente e ativa do que no cenário anterior. Além disso, dado o aumento no número de cliente e a redução no número de clientes antigos perdidos, é possível sugerir que os métodos enxutos efetivamente focam-se nas necessidades dos clientes e que isso melhora a relação entre ambas as partes.

6 Considerações finais

Este trabalho buscou analisar a aplicação da produção enxuta no desenvolvimento de *software*, objetivando explorar quais conceitos oriundos de abordagens denominadas enxutas e/ou ágeis podem ser aplicados concomitantemente com sucesso em um caso real.

Para tanto, descreveu-se o STP e o pensamento enxuto, demonstrando-se o histórico de sua evolução e seus principais conceitos práticos e filosóficos.

Foram também apontadas características do processo de desenvolvimento de *software* segundo modelos prescritivos, comumente denominados engenharia de *software* tradicional, bem como os problemas recorrentes em projetos baseados nestes, citados por diversos autores, seguindo-se com a apresentação dos princípios enxutos para desenvolvimento de *software* e das principais abordagens enxutas da atualidade, as quais se dispõem como possíveis soluções para estes problemas dos modelos prescritivos.

Demonstrou-se, por fim, através do estudo de caso, um exemplo de aplicação de tais conceitos e ferramentas, explorando-se qualitativamente as percepções de diferentes níveis funcionais da empresa sobre as melhorias proporcionadas pela adoção destas abordagens, bem como desafios quanto à sua adoção e possíveis problemas ainda persistentes.

6.1 Conclusões e contribuições

Conforme exposto durante a revisão bibliográfica acerca das metodologias enxutas para desenvolvimento de *software*, e corroborado durante o estudo de caso, a produção enxuta pode contribuir positivamente para o processo de desenvolvimento de *software*.

Através da revisão bibliográfica, também se realizou um confronto de opinião sobre a relação entre a denominação “metodologias ágeis” e a denominação “metodologias enxutas” durante a seção 4.1. As visões de diversos autores foram confrontadas e se mostraram diversas e inconclusivas. De maneira geral, boa parte das visões apresentadas tenderam a dizer que são conceitos equivalentes ou complementares.

Este trabalho adotou o entendimento de “enxuto” e “ágil” como conceitos equivalentes quando aplicado ao desenvolvimento de *software*, sendo esta postura validada durante o estudo de caso, observando-se que o Scrum - e as outras práticas de outras metodologias comumente denominadas ágeis aplicadas pela empresa – relacionam-se diretamente à adequação aos princípios enxutos do Lean Software Development, bem como algumas das práticas do Scrum e do XP se mostraram relacionadas diretamente inclusive com conceitos do STP, como Kanban, Andon e Kaizen.

Esta questão validada pelo estudo representa contribuição relevante para pesquisas futuras tanto na engenharia de produção quanto na engenharia de *software*, sendo possível afirmar com maior segurança, conforme levantado durante o estudo de caso, que "metodologias enxutas" e "metodologias ágeis" são apenas variações de nomenclatura e representam conceitos equivalentes no desenvolvimento de *software*.

O estudo de caso também contribuiu demonstrando que estas diferentes abordagens podem coexistir se complementando. Este cenário se demonstrou viável e também vantajoso à organização de desenvolvimento de *software* sob encomenda estudada, sendo possível a partir deste estudo propor a hipótese de que tais vantagens possam existir em empresas similares.

Além disso, demonstrou-se também a possível viabilidade de aplicação destes conceitos ágeis/enxutos em empresas menores, sem grande disponibilidade de recursos financeiros, por não terem demonstrado necessitar de grandes investimentos para implementação e, pelo contrário, demonstraram redução de custos em relação ao cenário anterior da Empresa X com metodologias prescritivas.

Por fim, embora sem resultados quantitativos, demonstrou-se como consenso entre todos que participaram da pesquisa os resultados positivos, inclusive tendo a empresa podido crescer bastante – saindo do pequeno porte e tornando-se uma empresa de médio porte – sem necessidade de investimentos financeiros estratosféricos.

6.2 Limitações da pesquisa

Embora a pesquisa tenha proporcionado relevantes contribuições citadas e tenha cumprido os objetivos propostos com sucesso, é relevante enfatizar algumas limitações desta que podem influenciar sua possibilidade de utilização como referência para generalização.

Não foi possível atuar em todas as unidades de desenvolvimento de *software* da empresa, principalmente por questões de distância e tempo disponível para condução da pesquisa, tendo-se limitado à menor e mais recente unidade da empresa. Embora as respostas do Diretor de Tecnologia tenham sido genéricas e referentes a toda a organização, as demais informações representam um escopo limitado em relação à empresa como um todo.

Conforme apontado na seção 5.3.2, dos quinze desenvolvedores da unidade estudada, apenas três foram selecionados para responder ao questionário. O principal motivo foi devido a impedimentos por parte da empresa em ceder mais desenvolvedores para esta etapa do processo, bem como para resguardar a identidade dos respondentes para que pudessem expor suas opiniões com mais transparência. De qualquer maneira, as opiniões de apenas três

desenvolvedores não condizem com uma amostra de tamanho relevante para a concepção de afirmações generalistas sobre o assunto.

Por fim, enfatiza-se também que os resultados apresentados quanto às melhorias obtidas pela empresa, bem como os vários pontos positivos destacados por todos os indivíduos que participaram, não se baseiam em números ou cálculos matemáticos. A empresa não dispõe de medições numéricas e comparativas para as variáveis em questão, tendo os resultados sido, portanto, apenas percepções descritivas e puramente qualitativas, embora corroboradas similarmente entre todos os indivíduos participantes. Ou seja, embora sejam relevantes para o caráter qualitativo exploratório do estudo, não servem como base para conclusões quantitativas.

6.3 Trabalhos futuros

O presente estudo teve um enfoque exploratório qualitativo, possibilitando entender melhor um caso real de desenvolvimento de *software* em um ambiente enxuto. Contudo, a fim de possibilitar a concepção de generalizações mais embasadas, pretende-se futuramente a expansão deste estudo na forma de uma pesquisa quantitativa, elaborando-se e aplicando-se um questionário mais específico a desenvolvedores e gestores de outras empresas que utilizem uma ou mais abordagens enxutas/ágeis.

Em outra vertente, visualiza-se também a possibilidade de buscar um mapeamento mais preciso entre os conceitos das diferentes abordagens estudadas, comparando-se de forma mais aprofundada os princípios e as práticas entre elas.

Por fim, visualiza-se ainda a possibilidade de estudar mais a fundo as ramificações e possíveis soluções para os diferentes tipos de desperdício no processo de desenvolvimento de *software*, em especial ao desperdício de tempo produtivo detectado na empresa durante a condução do estudo, com vistas a encontrar maneiras de melhorar o processo neste sentido.

Referências

ABES. **Mercado brasileiro de software**. Associação Brasileira de Engenharia de Software, 2010. Disponível em: <http://www.abes.org.br/templ3.aspx?id=306&sub=596>. Acesso em: 14 nov 2011.

ABRAHAMSSON, P.; BABAR, M. A.; KRUCHTEN, P. **Agility and architecture: can they coexist?** IEEE Software, Los Alamitos, EUA, Março/Abril 2010. p. 16-22.

ABRAHAMSSON, P.; WARSTA, J.; SIPONEN, M.; RONKAINEN, J. **New directions on agile methods:** a comparative analysis. In: 25TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. Washington, EUA: IEEE Computer Society, 2003. p. 244-254.

ABRAN, A.; MOORE, J. W.; BOURQUE, P.; DUPUIS, R.; TRIPP, L. L. **Guide to the software engineering body of knowledge (SWEBOK)**. Los Alamitos, EUA: IEEE Computer Society, 2004.

AGILCOOP. **Métodos ágeis no Brasil: estado da prática em times e organizações**. Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2012. Disponível em: <http://ccsl.ime.usp.br/agilcoop/files/metodos_ageis_brasil_estado_da_pratica_em_times_e_organizacoes.pdf>. Acesso em: 16 set 2012.

ANDERS, D. **Agile management for software engineering:** applying the theory of constraints for business results. Upper Saddle River, EUA: Prentice Hall, 2004.

ANDERSON, D. J. **Lean Software Development**. Technical Articles for Visual Studio Application Lifecycle Management, 2012. Disponível em: <http://msdn.microsoft.com/en-us/library/vstudio/hh533841.aspx>. Acesso em: 12 mai 2013.

ANDERSON, L.; ALLEMAN, G.; BECK, K.; BLOTNER, J.; CUNNINGHAM, W.; POPPENDIECK, M.; WIRFS-BROCK, R. **Agile management, an oxymoron:** who needs managers anyway? In: 18TH ANNUAL CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES AND APPLICATIONS. Nova York, EUA: Association for Computing Machinery, 2003. p. 275-277.

BASSI FILHO, D. L. **Experiências com desenvolvimento ágil**. 170 p. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2008. Disponível em: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-06072008-203515>. Acesso em: 25 fev 2013.

BECK, K. Embracing change with Extreme Programming. **IEEE Computer**, Los Alamitos, EUA, v. 32, n. 10, p. 70-77, Outubro 1999.

BECK, K. **Extreme Programming explained:** embrace change. Redwood, EUA: Addison-Wesley Professional, 2000. 190 p.

BECK, K. **Test-driven development:** by example. Boston, EUA: Addison-Wesley Professional, 2003.

BECK, K. **Extreme Programming explained:** embrace change. 2 ed. Boston, EUA: Addison-Wesley Professional, 2004. 224 p.

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; 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.; THOMAS, D. **Manifesto for Agile Software Development**. The Agile Manifesto, 2001. Disponível em: <http://agilemanifesto.org>. Acesso em: 19 nov 2013.

BEEDLE, M.; DEVOS, M.; SHARON, Y.; SCHWABER, K.; SUTHERLAND, J. Scrum: an extension pattern language for hyperproductive software development. **Pattern Languages of Program Design**, v. 4, p. 637-651, 1999.

BRYMAN, A. **Research methods and organization studies**. Londres: Routledge, 1989.

CERVO, A. L.; BERVIAN, P. A. **Metodologia científica**. 5ª ed. São Paulo: Pearson Pretince Hall, 2002.

CESCHI, M.; SILLITTI, A.; SUCCI, G.; PANFILIS, S. D. **Project management in plan-based and agile companies**. IEEE Software, Los Alamitos, EUA, Maio/Junho 2005. p. 21-27.

CHARETTE, R. **Why software fails**. IEEE Spectrum, Los Alamitos, EUA, 2005. p. 42-49.

CHARETTE, R. N. **Risk, lean development & profit:** getting back to basics. In: LEAN SOFTWARE & SYSTEMS CONFERENCE. Atlanta, EUA: Lean Systems Society, 2010. p. 2.

CLARK, M. **Bubble, bubble, build's in trouble**. Pragmatic Automation in Java, 2007. Disponível em: <http://www.pragmaticautomation.com/cgi-bin/pragauto.cgi/Monitor/Devices/BubbleBubbleBuildsInTrouble.rdoc>. Acesso em: 7 jun 2013.

COAD, P.; LUCA, J. D.; LEFEBVRE, E. **Java modeling in color with UML**. Englewood Cliffs, EUA: Prentice-Hall, 1999.

COCKBURN, A. **Agile software development**. Boston, EUA: Addison-Wesley, 2002.

CONBOY, K.; COYLE, S.; WANG, X.; PIKKARAINEN, M. **People over process: key challenges in agile development**. IEEE Software, Los Alamitos, EUA, Julho/Agosto 2011. p. 48-57.

COSTA, V. B. **Status do uso de metodologias ágeis na indústria de software brasileira**. 57 p. Departamento de Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2011.

CUKIER, D. **Programação Extrema: Extreme Programming ou simplesmente XP**. Blog da Locaweb, 20 jun 2008. Disponível em: <http://blog.locaweb.com.br/metodologias-ageis/programacao-extrema-extreme-programming-ou-simplesmente-xp>. Acesso em: 30 abr 2013.

CUKIER, D. **Padrões para introduzir novas ideias na indústria de software**. 150 p. Dissertação (Mestrado em Ciências) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2010.

DEGIRMENCI, T. **Standardization and certification in lean manufacturing**. 98 p. Tese (Mestrado em Ciências Aplicadas em Engenharia Mecânica) - Departamento de Engenharia Mecânica, Universidade de Waterloo, Waterloo, Canadá, 2008.

DEMING, W. E. **Out of the crisis**. Cambridge, EUA: Massachusetts Institute of Technology, 1986.

DYBÅ, T.; DINGSøYR, T. **What do we know about agile software development?** IEEE Software, Maryland, EUA, Setembro/Outubro 2009. p. 6-9.

FOWLER, M. **Agile versus lean**. Martin Fowler's Blog, 26 jun 2008. Disponível em: <http://martinfowler.com/bliki/AgileVersusLean.html>. Acesso em: 2013 mai 3.

GERIC, F. **Demystifying Kanban**. Agility Business Solutions, 2011. Disponível em: <http://oldblog.agilitybiz.net/Lists/Posts/Post.aspx?ID=46>. Acesso em: 7 jun 2013.

GHINATO, P. **Sistema Toyota de Produção: mais do que simplesmente Just-in-Time**. EDUCS, 1996. p. 169-189.

GHINATO, P. Sistema Toyota de Produção. In: ALMEIDA, A. T. D.; SOUZA, F. M. C. **Produção & Competitividade:** aplicações e inovações. Recife: UFPE, 2000.

GIL, A. C. **Como elaborar projetos de pesquisa**. São Paulo: Atlas, 1991.

GIL, A. C. **Métodos e técnicas da pesquisa social**. 5ª ed. São Paulo: Atlas, 1999.

HIBBS, C.; JEWETT, S.; SULLIVAN, M. **The art of lean software development**. Sebastopol, Ucrânia: O'Reilly Media, 2009.

HIGHSMITH, J. Foreword. In: POPPENDIECK, M.; POPPENDIECK, T. **Lean software development:** an agile toolkit. Redwood, EUA: Addison-Wesley Professional, 2003.

IBM. **RUP: best practices for design, implementation and effective project management**. IBM Rational Unified Process (RUP), 2013. Disponível em: <http://www.ibm.com/software/rational/rup/>. Acesso em: 23 nov 2013.

IEEE STANDARDS ASSOCIATION. **IEEE Standard Glossary of Software Engineering Terminology:** Standard 610.12-1990. Nova York, EUA: IEEE, 1993.

IKONEN, M. **Lean thinking in software development: impacts of Kanban on projects**. 113 p. Tese (Doutorado em Ciência da Computação) - Faculdade de Ciências, Universidade de Helsinque, Helsinque, Finlândia, 2011.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The unified software development process**. 1 ed. Upper Saddle River, EUA: Pearson Education, 1999.

JANZEN, D.; SAIEDIAN, H. **Test-driven development concepts, taxonomy, and future direction**. IEEEE Computer Society, Los Alamitos, EUA, Setembro 2005. p. 43-50.

JOHNSON, J. **ROI, it's your job**. In: THIRD INTERNATIONAL CONFERENCE ON EXTREME PROGRAMMING AND AGILE PROCESSES IN SOFTWARE ENGINEERING. Alghero, Italy: Università degli Studi di Genova, 2002. Disponível em: <http://ciclamino.dibe.unige.it/xp2002/talksinfo/johnson.pdf>. Acesso em: 14 ago 2012.

JONSSON, H. **Lean software development:** a systematic review. In: RESEARCH METHODOLOGY COURSE MINI CONFERENCE. Västerås, Sweden: Etteplan, 2012.

KATAYAMA, E. T. **A contribuição da indústria da manufatura no desenvolvimento de software**. 114. Dissertação (Mestrado em Ciências) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2010.

KISTE, R. P.; MIYAKE, D. I. **Abordagem Lean no desenvolvimento de software ágil:** um estudo de caso em uma editora. In: IX CONGRESSO NACIONAL EM EXCELÊNCIA DE GESTÃO. Rio de Janeiro: Universidade Federal Fluminense, 2013. p. 27.

LANE, M. T.; FITZGERALD, B.; ÅGERFALK, P. J. **Identifying lean software development values**. In: 20TH EUROPEAN CONFERENCE ON INFORMATION SYSTEMS. Barcelona, Espanha: ESADE, 2012. p. 15.

LIKER, J. K.; HOSEUS, M. **Toyota culture:** the heart and soul of The Toyota Way. Nova York, EUA: McGraw-Hill, 2008.

LINDSTROM, L.; JEFFRIES, R. Extreme programming and agile software development methodologies. **Information Systems Management**, v. 21, n. 3, p. 41-61, 2004.

LOBO, E. J. R. **Guia prático de engenharia de software**. São Paulo: Digerati Books, 2009. 128 p.

MADISON, J. **Agile-Architecture interactions**. IEEE Software, Los Alamitos, EUA, Março/Abril 2010. p. 41-48.

MARCONI, M. A.; LAKATOS, E. M. **Metodologia científica:** ciência e conhecimento científico, métodos científicos, teoria, hipóteses e variáveis, metodologia jurídica. 3ª ed. São Paulo: Atlas, 2000.

MCT. **Resultado da pesquisa qualidade no setor de software brasileiro**. Ministério da Ciência e Tecnologia, 2009. Disponível em: <http://www.mct.gov.br/index.php/content/view/320673.html>. Acesso em: 11 jun 2011.

MENDONÇA, C. D. Enacting Scrum and Agile with Visual Studio 2012. **ALM Magazine**, v. 1, n. 2, p. 2-15, Fevereiro 2013.

MESO, P.; JAIN, R. Agile software development: adaptive systems principles and best practices. **Information Systems Management**, v. 23, n. 3, p. 19-30, Summer 2006.

MICROSOFT. **Microsoft Pinpoint**. Pesquisa de parceiros por competência, 2013. Disponível em: <http://pinpoint.microsoft.com/pt-BR/companies/search?fcrc=BRA&fs=253&sort=name>. Acesso em: 3 set 2013.

MICROSOFT LEARNING. **Programa Microsoft Certified Professional**. Microsoft.com, 2013. Disponível em: <http://www.microsoft.com/learning/pt/br/start/start-overview.aspx>. Acesso em: 27 jun 2013.

MIDDLETON, P. Maintenance management: from product to process. **Journal of Software Maintenance**, v. 7, n. 1, p. 63-73, 1995.

MIDDLETON, P. Lean software development: two case studies. **Software Quality Journal**, v. 9, n. 4, p. 241-252, 2001.

MIDDLETON, P.; JOYCE, D. Lean software management: BBC Worldwide case study. **IEEE Transactions on Engineering Management**, v. 59, n. 1, p. 20-32, fev 2012.

MORGAN, J. M.; LIKER, J. K. **Sistema Toyota de desenvolvimento de produto:** Integrando pessoas, processo e tecnologia. Porto Alegre: Editora Bookman, 2008. 391 p.

NAUR, P.; RANDELL, B. **Software engineering:** report of a conference sponsored by the NATO Science Committee. In: NATO SOFTWARE ENGINEERING CONFERENCE, 7–11 October 1968. Garmisch, Alemanha: NATO Scientific Affairs Division, 1969.

NOGUEIRA, J.; JONES, C.; LUQI. **Surfing the edge of chaos:** applications to software engineering. In: COMMAND AND CONTROL RESEARCH AND TECHNOLOGY SYMPOSIUM. Monterey, EUA: Naval Post Engineering, 2000. p. 1-13. Disponível em: <http://www.dodccrp.org/events/2000_CCRTS/html/pdf_papers/Track_4/075.pdf>.

NONAKA, I.; TAKEUCHI, H. **The new new product development game**. Harvard Business Review, 64, n. 1, 1986. p. 137-146.

NONAKA, I.; TAKEUCHI, H. **Criação de conhecimento na empresa:** como as empresas japonesas geram a dinâmica da inovação. 20 ed. Rio de Janeiro: Elsevier Brasil, 1997.

NONAKA, I.; TAKEUCHI, H. **Gestão do conhecimento**. Porto Alegre: Bookman, 2004.

NOYES, B. **Rugby, Anyone?** Fawcette Technical Publications, Junho 2002. Disponível em: <http://controlchaos.squarespace.com/storage/scrum-articles/Brian%20Noyes%20on%20Scrum%20June%202002.pdf>. Acesso em: 27 abr 2013.

OHNO, T. **Sistema Toyota de Produção:** além da produção em larga escala. Porto Alegre: Bookman, 1997.

PALMER, S.; FELSING, J. M. **A practical guide to Feature-Driven Development**. Englewood Cliffs, EUA: Prentice-Hall, 2002.

PARDAL, L. C. M.; PERONDI, L. F. P.; VALERI, S. G. **A filosofia enxuta no desenvolvimento de produto e suas origens**. In: 2º WORKSHOP EM ENGENHARIA E TECNOLOGIA ESPACIAIS. São José dos Campos, São Paulo: INPE - Instituto Nacional de Pesquisas Espaciais, 2011. p. 1-8.

PETERSEN, K. **Implementing lean and agile software development in industry**. 309 p. Tese (Doutorado em Engenharia de Software) - Escola de Computação, Instituto de Tecnologia de Blekinge, Karlskrona, Suíca, 2010.

PETERSON, K.; WOHLIN, C. **Measuring the flow in lean software development**. Software Practice and Experience, abr 2010. p. 975-996.

POPPENDIECK, M.; POPPENDIECK, T. **Lean software development:** an agile toolkit. Redwood, EUA: Addison-Wesley Professional, 2003.

POPPENDIECK, M.; POPPENDIECK, T. **Implementing lean software development:** from concept to cash. Upper Saddle River, EUA: Addison-Wesley Professional, 2006.

POPPENDIECK, M.; POPPENDIECK, T. **Leading lean software development:** results are not the point. Redwood, EUA: Pearson Education, 2009.

PORTER, M. E.; MILLAR, V. E. **How information gives you competitive**. Harvard Business Review, Boston, EUA, 1995. p. 1-12.

PRESSMAN, R. S. **Engenharia de software:** uma abordagem profissional. 7 ed. Porto Alegre: Bookman, 2011.

PROJECT MANAGEMENT INSTITUTE. **PMBOK:** a guide to the Project Management Body of Knowledge. 5 ed. Newtown Square, EUA: PMI, 2013. 589 p.

RANDELL, B. **The 1968/69 NATO Software Engineering Reports**. Department of Computing Science, Newcastle University, 2001. Disponível em: <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/>. Acesso em: 22 fev 2013.

RETAMAL, A. M. **Feature driven development**. Wikimedia Commons, 2007. Disponível em: <http://pt.wikipedia.org/wiki/Ficheiro:Fdd.png>. Acesso em: 17 mai 2013.

ROACH, S. **White collar productivity:** a glimmer of hope? Nova York, EUA: Morgan Stanley, 1988.

ROESCH, S. M. A. **Projetos de estágio e pesquisa em administração**. São Paulo: Atlas, 1999.

ROTHER, M.; SHOOK, J. **Learning to see:** Value Stream Mapping to create value and eliminate Muda. 3 ed. Cambridge, EUA: Lean Enterprise Institute, 2008.

SALGADO, A.; MELCOP, T.; ACCHAR, J.; REGO, P. A.; FERREIRA, A. I. F.; KATSURAYAMA, A. E.; MONTONI, M.; ZANETTI, D. **Aplicação de um processo ágil para implantação de processos de software baseado em Scrum na Chemtech**. In: IX SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE. Belém: Universidade Federal do Pará, 2010. p. 351-358.

SATO, D. T. **Uso eficaz de métricas em métodos ágeis de desenvolvimento de software**. 155 p. Dissertação (Mestrado em Ciências) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2007.

SCHACH, S. R. **Engenharia de software:** os paradigmas clássico e orientado a objetos. 7 ed. Porto Alegre: AMGH, 2010.

SCHIEL, J. **Enterprise-scale agile software development**. Florence, EUA: CRC Press, 2009.

SCHWABER, K.; BEEDLE, M. **Agile software development with Scrum**. Upper Saddle River, EUA: Prentice Hall, 2001.

SCHWABER, K.; SUTHERLAND, J. **Guia do Scrum:** um guia definitivo para o Scrum. Tradução de Fábio Cruz. 2 ed. [S.l.]: Scrum.org, 2011. 18 p. Disponível em: <http://www.scrum.org/Scrum-Guides/>. Acesso em: 24 abril 2013.

SEBRAE/SC. **Legislação: critérios de classificação de empresas**. Serviço Brasileiro de Apoio às Micro e Pequenas Empresas, 2008. Disponível em: <http://www.sebrae-sc.com.br/leis/default.asp?vcdtexto=4154>. Acesso em: 11 set 2013.

SELBY, R. **Software engineering:** Barry W. Boehm's lifetime contributions to software development, management, and research. Hoboken, EUA: Wiley-IEEE Press, 2007.

SHINGO, S. **O Sistema Toyota de Produção do ponto de vista da Engenharia de Produção**. Tradução de Eduardo Schaan. 2 ed. Porto Alegre: Artmed, 1996. 296 p.

SHORE, J.; WARDEN, S. **The art of agile development**. Sebastopol, EUA: O'Reilly Media, 2007.

SILLITTI, A.; SUCCI, G. Foundations of agile methods. In: LUCIA, A. D. *et al.* **Emerging methods, technologies, and process management in software engineering**. Nova Jersey, EUA: John Wiley & Sons, 2007. Cap. 11, p. 249-270.

SILVA, F. L. C. **Mapeamento e aplicação da produção enxuta para o processo de desenvolvimento de software**. 255 p. Dissertação (Mestrado em Engenharia de Produção) - Universidade Estadual do Norte Fluminense, Campos dos Goytacazes, 2011.

SOMMERVILLE, L. **Engenharia de software**. 8 ed. São Paulo: Addison Wesley, 2007.

STANDISH GROUP. **The chaos report**. The Standish Group International, Boston, EUA, 1995.

STANDISH GROUP. **Chaos summary for 2010**. The Standish Group International, Boston, EUA, 2010.

SUTHERLAND, J.; SCHWABER, K. **Business object design and implementation**. In: OOPSLA '95 WORKSHOP. Michigan, EUA: Universidade de Michigan, 1995. p. 118.

TELES, V. M. **Extreme Programming:** aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec Editora, 2004.

TELES, V. M. **Um estudo de caso da adoção das práticas e valores do Extreme Programming**. 181. Dissertação (Mestrado em Informática) - Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

TEMPRADO, E. M.; BENDITO, E. R. **Lean software development and agile methodologies for a small software development organization**. 74 p. Dissertação (Mestrado em Ciências com Ênfase em Tecnologia da Qualidade) - Escola de Engenharia, Universidade de Borås, Borås, Suécia, 2010.

TERA PRUDENT SOLUTIONS. **Waste reduction by Muda, Mura and Muri**. Reference Materials, 2012. Disponível em: <http://www.tera-tps.com.au/material.htm>. Acesso em: 22 jul 2013.

THIMBLEBY, H. Delaying commitment. **IEEE Software**, v. 5, n. 3, p. 78-86, 2002.

VENNERS, B. **Test-driven development: a conversation with Martin Fowler (Part V)**. Artima Developer, 2002. Disponível em: <http://www.biology.emory.edu/research/Prinz/Cengiz/cs540-485-FA12/resources/testDrivenDev.pdf>. Acesso em: 10 mai 2013.

VRIENS, C. **Certifying for CMM level 2 and ISO 9001 with XP@Scrum**. In: AGILE DEVELOPMENT CONFERENCE. Nova York, EUA: IEEE Computer Society, 2003. p. 120-124.

WOMACK, J. Mura, Muri, Muda? **Jim Womack e-Letters**, Lean Enterprise Academy, 7 jul 2006.

WOMACK, J. P.; JONES, D. T. **A mentalidade enxuta nas empresas:** elimine o desperdício e crie riqueza. Rio de Janeiro: Editora Campos, 1998. 427 p.

WOMACK, J. P.; JONES, D. T.; ROOS, D. **A máquina que mudou o mundo**. 8ª ed. Rio de Janeiro: Editora Campus, 2004.

WYSOCKI, R. K. **Effective project management:** tradicional, adaptive, extreme. 5 ed. Indianápolis, EUA: John Wiley & Sons, 2011.

YIN, R. K. **Estudo de caso:** planejamento e métodos. 3ª ed. Porto Alegre: Bookman, 2005.

Apêndice A: Entrevista aplicada ao Diretor de Tecnologia

O presente apêndice apresenta a estrutura da entrevista conduzida por e-mail com o Diretor de Tecnologia da empresa estudada, para obtenção das informações iniciais sobre a empresa e sobre a unidade visitada, conforme citado na seção 5.1 deste trabalho.

As questões estão elencadas a seguir:

- Idade da empresa?
- Idade da unidade visitada?
- Número total de funcionários da empresa?
- Número total de funcionários da unidade?
- Proporção de funcionários por função?
- Quantidade aproximada de clientes?
- Principais tecnologias utilizadas para o desenvolvimento?
- Quanto tempo da utilização de metodologias ágeis?
- Quais metodologias ágeis são aplicadas oficialmente?
- Percentual de projetos utilizando metodologias ágeis?
- Quantidade de profissionais treinados nestas metodologias?
- Resumo das melhorias obtidas em relação ao processo prescritivo.
- Resumo dos problemas/desafios pela adoção das metodologias ágeis.
- Expectativa sobre tais metodologias no futuro?
- Há valorização por parte dos clientes sobre elas?
- Há atenção da concorrência na adoção delas?
- Outras informações que se deseje citar e não foram questionadas.

Apêndice B: Entrevista aplicada ao Gerente

O presente apêndice apresenta as perguntas realizadas durante a entrevista semiestruturada, conduzida presencialmente com o Gerente da unidade estudada, conforme citado na seção 5.1 deste trabalho.

As questões estão elencadas a seguir:

- Como a empresa encara o desperdício durante o processo?
- Existem ações específicas para se evitar desperdícios?
- De que maneira garantem-se entregas rápidas?
- O tamanho das iterações é adequado?
- Entre cumprir o prazo ou eliminar defeitos, o que é mais importante?
- Existem ações para amplificar o aprendizado dos colaboradores?
- De que maneira um profissional é reconhecido?
- E a equipe como um todo, como é garantida sua valorização?
- Como se dá a negociação para adiar comprometimentos antecipados?
- O processo favorece a segurança, proporcionando mecanismos que efetivamente diminuem as chances do projeto fracassar?
- Quais as medidas de segurança para garantia da qualidade dos produtos?
- Os desenvolvedores possuem uma “visão do todo”, entendendo o escopo geral no qual estão inseridas suas atividades específicas?
- Quais as ações organizacionais para melhorar o processo como um todo em detrimento de apenas melhorias pontuais em determinadas etapas?

Apêndice C: Questionário aplicado aos Desenvolvedores

O presente apêndice apresenta a estrutura do questionário elaborado para obtenção das informações sobre a percepção dos desenvolvedores da unidade estudada.

C.1 Introdução

Caro desenvolvedor, o presente questionário representa parte do estudo que está sendo conduzido em sua unidade para entendimento da filosofia enxuta aplicada ao desenvolvimento de *software*, sendo importante conhecer suas opiniões e entender suas ações diante de vários cenários fictícios, mas que podem ser comuns no seu dia-a-dia de trabalho.

Enfatiza-se que respostas diretas, objetivas e transparentes são determinantes para a validade do estudo. Vale citar que o estudo é conduzido sob total sigilo e que suas considerações serão utilizadas sempre de forma anônima.

C.2 Perfil pessoal

Esta seção do questionário visa obter informações gerais sobre seu perfil, suas atribuições e experiência, ajudando a definir seu nível de conhecimento sobre a empresa e sobre os assuntos abordados pela pesquisa.

C.2.1 Qual sua idade?
C.2.2 Qual sua formação acadêmica?
C.2.3 Quando concluiu sua graduação ou qual a previsão de conclusão?
C.2.4 Qual seu tempo de experiência profissional na área?
C.2.5 Há quanto tempo está na empresa?
C.2.6 Qual sua função na empresa atualmente?
C.2.7 Há quanto tempo estuda/utiliza metodologias enxutas/ágeis?
C.2.8 Já participou de cursos sobre metodologias enxutas/ágeis?

C.3 Questionário

Esta seção representa o corpo do questionário e envolve diversas perguntas que buscam explorar se e como princípios enxutos são abordados nos projetos.

C.3.1 Você sente que está evoluindo seus conhecimentos enquanto desenvolve suas atividades? Se sim, que meios são oferecidos para incentivar seu aprendizado?

C.3.2 Quem pode tomar decisões técnicas durante o projeto (como escolher novos padrões de código, arquitetura, escolher *plug-ins* e bibliotecas, reescrever algum código utilizado por vários componentes, etc.)?

C.3.3 Como é o controle de atividades na equipe? Como cada um sabe o que o outro está fazendo, se está com problemas, se depende de alguma coisa de outra pessoa da equipe ou de pessoas externas (como definições por parte do cliente)?

C.3.4 Você tem conhecimento apenas das tarefas da Sprint atual (e das anteriores que participou) ou sabe sobre o andamento de todo o projeto, o que o cliente espera do *software* e tem ideia do que esperar das iterações futuras?

C.3.5 Se alguém lhe pede ajuda ou informa que precisa de alguma coisa sua pra prosseguir com a atividade dele, mas isso não está atualmente contemplado na previsão inicial sobre o que você está trabalhando, qual sua ação?

C.3.6 Se você encerra uma tarefa antes do planejado, o que costuma fazer com o tempo que sobra? O ambiente favorece de alguma forma a utilização desse tempo?

C.3.7 O processo de desenvolvimento de *softwares* adotado pela empresa é adequado? Os envolvidos nos projetos (você, os outros desenvolvedores, os clientes) estão satisfeitos com este processo?