**UNIVERSIDADE METODISTA DE PIRACICABA**

**FACULDADE DE ENGENHARIA, ARQUITETURA E URBANISMO**

**PROGRAMA DE PÓS GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO**

# DESENVOLVIMENTO ENXUTO DE *SOFTWARE* EMBARCADO: PROPOSTA DE AÇÕES PARA UMA EQUIPE DA INDÚSTRIA DE DISPOSITIVO MÓVEL.

ELAINE CRISTINE GOUVÊA MARTINS

ORIENTADOR: PROF. DR. FERNANDO CELSO DE CAMPOS

SANTA BÁRBARA D'OESTE

2008

**UNIVERSIDADE METODISTA DE PIRACICABA**

**FACULDADE DE ENGENHARIA, ARQUITETURA E URBANISMO**

**PROGRAMA DE PÓS GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO**

# DESENVOLVIMENTO ENXUTO DE *SOFTWARE* EMBARCADO: PROPOSTA DE AÇÕES PARA UMA EQUIPE DA INDÚSTRIA DE DISPOSITIVO MÓVEL.

ELAINE CRISTINE GOUVÊA MARTINS

ORIENTADOR: PROF. DR. FERNANDO CELSO DE CAMPOS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção, da Faculdade de Engenharia, Arquitetura e Urbanismo, da Universidade Metodista de Piracicaba – UNIMEP, como requisito para obtenção do Título de Mestre em Engenharia de Produção.

SANTA BÁRBARA D'OESTE

2008

Ao meu esposo Cícero  e

aos nossos filhos Ester, Davi e Paula.

# AGRADECIMENTOS

A Deus, pela Vida!

À minha família, pelo incentivo e encorajamento em todos os momentos.

Ao meu orientador Professor Dr. Fernando Celso de Campos pela confiança, suporte, direcionamento e colaboração demonstrados durante todo o curso.

Aos meus pais Orlando (em memória) e Jandira Gouvêa que não pouparam esforços para despertar em seus filhos o anseio pelo saber e a integridade no agir.

À equipe de desenvolvimento de *software* da empresa “Alvo”.

**Martins, Elaine Cristine Gouvêa. Desenvolvimento Enxuto de *Software* Embarcado: Proposta de Ações para uma Equipe da Indústria de Dispositivo Móvel. 2008.** Dissertação (Mestrado em Engenharia de Produção) – Faculdade de Engenharia, Arquitetura e Urbanismo, Universidade Metodista de Piracicaba, Santa Bárbara d'Oeste.

## RESUMO

*Os princípios enxutos têm se mostrado universais e bem sucedidos em melhorar resultados, que é o objetivo de toda organização empresarial. Esse trabalho apresenta uma revisão bibliográfica e propostas de ações relacionadas à transição de processo de desenvolvimento de software embarcado em cascata para desenvolvimento enxuto, no contexto da empresa global de mercado de massa "Alvo". A metodologia utilizada foi pesquisa-ação, suportado por um projeto Digital Six Sigma DMAIC. Um projeto piloto foi realizado, no qual os alinhavamentos organizacionais foram estabelecidos ao processo de desenvolvimento de software. Os resultados encontrados nesse projeto piloto mostraram que mesmo usando uma abordagem híbrida, práticas ágeis podem ajudar organizações maduras a se tornarem mais flexíveis e serem mais produtivas.*

**PALAVRAS-CHAVE:** Desenvolvimento Enxuto de *Software*; Pensamento Enxuto; Processo Ágil.

**Martins**, **Elaine Cristine Gouvêa. Lean Embedded Software Development: Actions Proposals to a Team at Mobile Devices Industry. 2008.** Dissertation (*Master Degree in Production Engineering*) – Faculdade de Engenharia, Arquitetura e Urbanismo, Universidade Metodista de Piracicaba, Santa Bárbara d'Oeste.

## ABSTRACT

*Lean principles have proven to be universal and successful at improving results. This dissertation presents a bibliographic review and some actions proposals to transitioning from waterfall development into lean embedded software development, at a mass market company called "Alvo". The methodology was research-action, supported by a Digital Six Sigma DMAIC project. A pilot project was carried out where organizational alignments have been stablished along the software development process. The outcomes from this pilot project showed that even in an hybrid approach, agile practices can help mature organizations to become more flexible and be more productive.*

**KEYWORDS:** Lean Software development; Lean Thinking; Agile Process.

# SUMÁRIO

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| DSS | *Digital Six Sigma* |
| DMAIC | *Define, Measure, Analyze, Improve, Control* |
| DMADV | *Define, Measure, Analyze, Design, Verify* |
| DFSS | Desenho para *Six Sigma* |
| PC | *Personal Computer* (computador de uso pessoal) |
| XP | *Extreme Programming* |
| ASD | *Adaptive Software Development* |
| DSDM | *Dynamic Systems Development Model* |
| FDD | *Feature Driven Development* |
| LD | *Lean Development* |

# LISTA DE FIGURAS

# LISTA DE QUADROS

## 1. Introdução

Desde a introdução de uma economia baseada na indústria, todos os setores se esforçam para melhorar a produtividade. As organizações enfrentam o desafio de se manterem competitivas num ambiente de estímulos e ameaças.

Portanto, as empresas precisam repensar seu processo produtivo, de inovação, posicionamento no mercado global, e são levadas a um processo de mudança.

O que se demanda das empresas é que essas tenham habilidade de serem ágeis, respondendo rapidamente às mudanças das necessidades do cliente e forças do mercado, mantendo o valor ao cliente, custo, e qualidade dos produtos e serviços.

A forma, intensidade e velocidade com a qual esse processo é conduzido determinarão o sucesso ou o fracasso de tal decisão.

Será visto como uma empresa de desenvolvimento de *software* embarcado está conduzindo o processo de transição para uma abordagem enxuta, baseada na teoria e nos princípios do Sistema Toyota de Produção e seus desdobramentos.

### 1.1. Justificativa

Dispositivos móveis têm se transformado de meros aparelhos de comunicação de voz em computadores de mão capazes de uma variedade de tarefas, como suporte a *emails*, navegação pela *Internet* e transmissão de vídeo. Consequentemente, os fabricantes têm focado na melhoria das capacidades de *software* de seus produtos, que significa adicionar mais suporte para o desenvolvimento de *software*.

Os aparelhos celulares ganharam múltiplos *gigabytes* de armazenamento, mais telas funcionais, mais capacidades de potência de áudio e funções de vídeo, como visualizar *video clips*, e assim os dispositivos se tornaram mais próximos de um computador de uso pessoal (*PC*) (MAIER, 2006).

*Software* é o que sustenta essas capacidades. Portanto, os fabricantes desses dispositivos precisam encontrar maneiras de disponibilizá-lo nos seus sistemas. E ainda fazer com que aplicações hoje basicamente disponíveis nos computadores de uso pessoal se tornem acessíveis.

Claramente isso conduz a um foco em custo, qualidade e desempenho, levando à necessidade de integrar o conhecimento a partir de várias fontes, como mercado, clientes, usuários, competidores, órgãos reguladores. A comunidade de negócios está

requisitando processos de desenvolvimento de *software* mais ágeis, que levem em consideração o *business case* de novos produtos.

Para atender a essa necessidade, essa dissertação apresenta uma reflexão e proposta de ações sobre o processo de transformação de uma organização para "enxuta" na unidade que desenvolve *software* embarcado num dispositivo móvel.

## 1.2. Objetivos

**Objetivo Geral:** sistematizar um conjunto de propostas consistentes com a realidade do desenvolvimento de *software*, correlacionados à teoria de Pensamento Enxuto e à sua aplicação.

**Objetivos específicos são: a**presentar proposta de quatro frentes para a aplicação de princípios Enxutos:

- Adequar conceitos aos processos globais da empresa
- Pesquisar junto aos desenvolvedores de *software* para se levantarem alternativas de "ganhos imediatos"
- Estabelecer base de referência para avaliação dos resultados
- Propor piloto para processos ágeis

## 1.3. Metodologia da Pesquisa

Para que os objetivos deste trabalho sejam atingidos, além do levantamento de dados empíricos e da elaboração de uma revisão bibliográfica, uma fundamentação teórica é requerida e está baseada em estudos sobre Pesquisa Científica e Metodologias.

Salomon (1991) define o trabalho científico como uma atividade que, por meio de uma metodologia rigorosa, se presta à pesquisa e à análise por escrito de questões ou problemas levantados.

Nesta seção serão demonstrados a Abordagem e o Método de Pesquisa Científica utilizados neste trabalho e suas respectivas justificativas.

**A Abordagem Utilizada**

De acordo com Creswell (1994), existem duas abordagens amplas mais utilizadas na pesquisa científica: a Abordagem Quantitativa e a Qualitativa.

Godoy (1995) explica que “pesquisa qualitativa não procura enumerar ou medir os eventos estudados, parte de focos ou questões de interesse amplo que vão se definindo à medida que o estudo se desenvolve”.

Segundo a mesma autora, muitos dos aspectos envolvidos somente são percebidos no transcorrer da execução da pesquisa empírica, ao contrário de uma pesquisa quantitativa na qual “o pesquisador conduz seu trabalho a partir de um plano estabelecido *a priori*, com hipóteses claramente especificadas e variáveis operacionalmente definidas”. A autora completa: “Quando o estudo é de caráter descritivo e o que se procura é o entendimento do fenômeno como um todo, na sua complexidade, é possível que uma análise qualitativa seja a mais indicada”.

Segundo Martins (1999), a “pesquisa quantitativa requer que o pesquisador possa manipular o objeto de estudo de forma a selecionar variáveis independentes de variáveis dependentes e isolar certas inferências no experimento, tornando-o mais confiável e previsível”.

A abordagem utilizada neste trabalho é a qualitativa, sendo esta justificável uma vez que, objetivando a ampliação dos conhecimentos a respeito do pensamento enxuto, pretende-se observar sua viabilidade e efetiva implantação, buscando identificar novos aspectos envolvidos e novas relações entre os aspectos levantados na literatura.

Além disso, as questões relacionadas à escolha, implementação e utilização do Pensamento Enxuto no desenvolvimento de *software* embarcado é algo complexo, de amplitude diferente de quando se fala na manufatura e dos tradicionais processos e metodologias aplicados ao desenvolvimento de um produto de *software*, uma vez que suas características de integração e abrangência funcional trazem impactos em diversas áreas da organização de maneira simultânea.

**O Método de Pesquisa Utilizado**

Segundo Bryman (1989), os principais métodos utilizados em pesquisas organizacionais são: a pesquisa experimental, a pesquisa de avaliação (*survey*), o estudo de caso e a pesquisa-ação (*action research*).

De acordo com Kuri Chu (2002), a Pesquisa Experimental tem como principal característica o controle do pesquisador sobre as variáveis envolvidas no estudo, podendo manipular as variáveis independentes de maneira controlada, obtendo diferentes resultados para as diversas situações. Neste tipo de pesquisa, segundo a autora, o pesquisador deve definir com cuidado quais são as variáveis dependentes e independentes do problema, quais são os materiais e a instrumentação que será utilizada, qual o procedimento que será adotado e quais as medidas que são relevantes para a pesquisa.

Sobre a Pesquisa de Avaliação (*Survey*), Kuri Chu (2002) explica que este método de pesquisa envolve a coleta sistemática de dados, geralmente utilizando entrevistas e questionários aplicados em um determinado momento pelo próprio pesquisador ou por meio de auto-aplicação, que contenham as informações a serem classificadas adequadamente. Outro ponto destacado pela autora é que as amostras podem tornar-se muito grandes. Assim, ao se adotar o *survey* como método, deve-se ter em mente as dificuldades de acesso.

O método de Estudo de Caso, segundo Yin (1989), é uma “pesquisa empírica que investiga um fenômeno contemporâneo dentro de um contexto real de vida, no qual as fronteiras entre fenônemo e contexto não são claramente evidentes e no qual múltiplas fontes de evidências são utilizadas”. Segundo o autor, o método de estudos de caso é o mais adequado quando se procura responder questões do tipo como? e por que? e também quando o objeto de estudo é historicamente novo com poucas possibilidades de busca e controle das ocorrências.

Na Pesquisa-Ação (*Action Research*), segundo Bryman (1989), deve haver colaboração entre pesquisador e agentes por um interesse mútuo no diagnóstico e na solução do problema do estudo de caso em que estão envolvidos.

De acordo com Martins (1999), “para realizar esse tipo de pesquisa, o pesquisador precisa envolver-se diretamente com a organização estudada, passando a ser virtualmente um membro dela”.
Thiollent (1997) *apud* Kuri Chu (2002) aborda alguns pontos de atenção com relação ao desenvolvimento de uma pesquisa-ação:

- ética e negociação;
- compromisso participativo com melhorias e mudanças;
- orientação interrogativa e crítica por parte do pesquisador e agentes;
- instrumentalidade sem exclusão do espírito crítico;
- cientificidade e objetividade em termos principalmente da imparcialidade, do consenso entre os envolvidos e do rigor científico.

Partindo da análise dos métodos disponíveis, a pesquisa realizada neste trabalho utiliza o método de Pesquisa-Ação (*Action Research*), justificável perante as impossibilidades da realização de um *Survey* no qual o pesquisador deve se manter distante dos envolvidos e de uma pesquisa experimental onde o mapeamento de variáveis dependentes e independentes não está sob o controle do pesquisador.

O método de Pesquisa-Ação é bastante adequado para se atingir o principal objetivo deste trabalho que é a proposição de um conjunto de ações que possa auxiliar uma empresa nos processos de desenvolvimento de produto embarcado de *software* desde a sua especificação até a sua entrega ao cliente para validação, atendendo às especificações e condições de contorno técnicas e funcionais do produto final.

Neste caso, a atuação do pesquisador não se restringe somente à execução da pesquisa, mas sim do exercício de um papel como membro da equipe de projeto dentro da organização, tendo acesso a todas as ocorrências, desde discussões estratégicas, políticas e diretrizes, necessidades do negócio e restrições até discussões técnicas e financeiras, pode-se esperar que o resultado final do trabalho seja bastante enriquecido.

Desta forma, é possível dizer que o estudo sobre os aspectos envolvidos nas análises e propostas de ações de implantação, condizente com as necessidades e realidade das organizações, por meio da aplicação do método de pesquisa-ação, poderá contribuir com a sistematização de conhecimentos práticos obtidos durante o projeto a partir de uma proposta de roteiro teórico para aplicação em processos deste tipo de produto.

A pesquisa tem dois pontos principais no desenvolvimento da argumentação à conclusão do trabalho: uma revisão bibliográfica, e a proposta de conjunto de ações para o desenvolvimento enxuto de produto de *software*. A revisão bibliográfica teve o enfoque que possibilita visualizar a relevância do Pensamento Enxuto expandido para o desenvolvimento de *software*, ver figura 1.

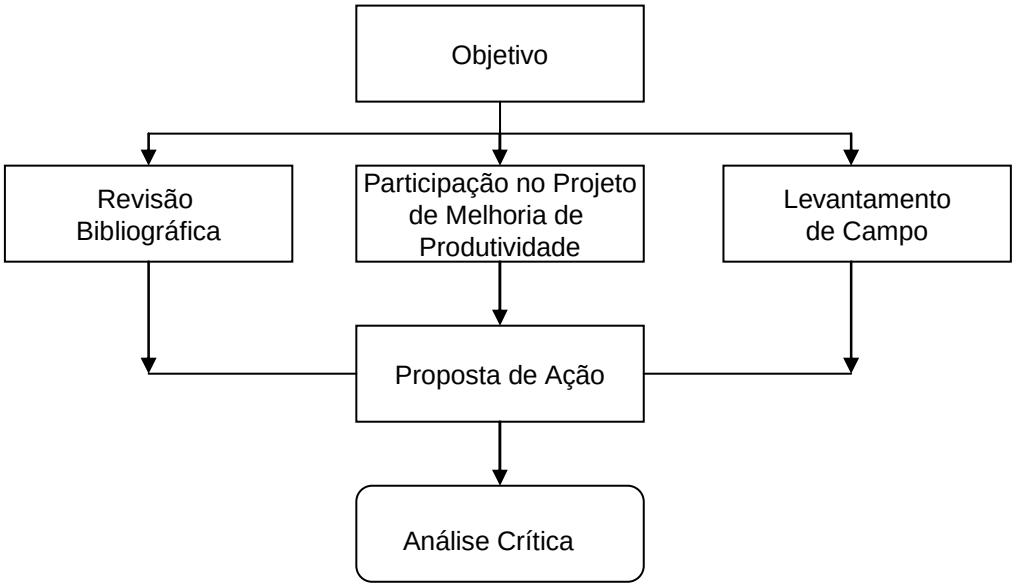


**Figura 1 -** *Outline* da pesquisa

A proposta de conjunto de ações para uma empresa de desenvolvimento de *software* embarcado em dispositivo móvel é baseada na participação do pesquisador em projeto de melhoria de produtividade implementado na organização, como membro da equipe formada com gerentes, desenvolvedores e membros do time de qualidade, e por meio de apresentações, relatórios e análise dos dados do atual processo de desenvolvimento de produto *software*, assim como das restrições e processos da empresa.

## 1.4. Estrutura de Dissertação

Este trabalho foi estruturado em 4 capítulos.

Capítulo 1 – Introdução, justificativa e relevância, objetivo geral e objetivos específicos, metodologia da pesquisa e estrutura da dissertação. Enfim, este capítulo fornece uma visão geral e contexto necessário para compreensão das pretensões do trabalho.

Capítulo 2 – Apresenta uma revisão bibliográfica do Pensamento Enxuto na manufatura, e a extrapolação de seu conceito para o desenvolvimento de produto, e mais especificamente no desenvolvimento de *software*, e pincelando algumas implicações essenciais no gerenciamento de projeto. O que se pretende é dar o embasamento teórico para se fazerem os levantamentos de quais ações são aplicáveis e podem ser propostas para uma unidade de desenvolvimento de *software* embarcado na Indústria de dispositivo móvel, para se obter vantagem competitiva.

Capitulo 3 – Tem como objetivo apresentar algumas características do desenvolvimento de *software* da indústria de dispositivos móveis. E, além disso, expõe-se o contexto da empresa e propõem-se alternativas para se implementar os princípios de pensamento enxuto.

Capitulo 4 – Realizam-se as discussões finais e apresentam-se as conclusões da dissertação e as propostas de trabalhos futuros.

## 2. Pensamento Enxuto

Nos primórdios da produção manufatureira, o produtor artesanal era trabalhador altamente qualificado, que fazia uso de ferramentas simples e flexíveis, para produzir individualmente e com exclusividade. Porém, apresentava duas grandes desvantagens econômicas: resultava em grandes *lead times* e era cara demais para a maioria das pessoas tornando-se, com o passar do tempo, inviável comercialmente (WOMACK *et al.*, 1992).

Após a Primeira Guerra Mundial, Alfred Sloan, da General Motors e Henry Ford, da Ford Motors, conduziram a mudança de séculos de produção artesanal de bens – cuja liderança era européia – para a chamada Era da Produção em Massa.

Este sistema de produção, que foi utilizado primeiramente nas indústrias automobilísticas americanas, foi posteriormente difundido nas indústrias da Europa.

O produtor em massa, por sua vez, utilizava-se de profissionais excessivamente especializados para projetar produtos que eram manufaturados por trabalhadores sem qualificação ou semi-qualificados, em máquinas dispendiosas e especializadas em uma única tarefa. Por ser dispendiosa a mudança de um produto, este era mantido como padrão o maior tempo possível e com métodos de trabalho muitas vezes monótonos e obsoletos. Com isso, o consumidor obtinha preços mais baixos, em detrimento de variedade e qualidade.

A produção em massa deixava muito a desejar em termos de competitividade e atendimento aos anseios consumistas emergentes (WOMACK *et al.*, 1992). O aspecto chave para este tipo de produção residia na completa e consistente intercambialidade das peças, em sua simplicidade e na facilidade de ajustá-las entre si. Foram estas inovações que tornaram a linha de montagem possível, reduzindo-se drasticamente os custos de fabricação e aumentando a qualidade do produto, superando os problemas da produção artesanal.

O surgimento do conjunto de filosofias e técnicas da Manufatura Enxuta na indústria japonesa – um pioneirismo de Eiji Toyoda e Taiichi Ohno, da Toyota, ocorreu após a Segunda Grande Guerra.

*Lean Thinking* (ou "Pensamento Enxuto") é um termo cunhado por James Womack e Daniel Jones para denominar uma filosofia de negócios baseada no Sistema Toyota de Produção que olha com detalhe para as atividades básicas envolvidas no negócio e

identifica o que é o desperdício e o que é o valor a partir da ótica dos clientes e usuários.
O produtor enxuto, em contraposição aos dois anteriores, combina as vantagens da produção artesanal e em massa, evitando a rigidez desta e os altos custos da primeira. Assim, a manufatura enxuta emprega equipes de trabalhadores multiqualificados em todos os níveis da organização, além de perseguir custos sempre declinantes, nível zero de estoque, e de desenvolver ou adquirir máquinas altamente flexíveis, para produzir uma maior e sempre crescente variedade de produtos, tendo sempre em mente a máxima satisfação do cliente – a qualidade aplicada (WOMACK *et al.*, 1992).
Os "produtores enxutos" almejam a perfeição: fazer a coisa certa, no lugar certo e na hora certa na primeira vez. É claro que essa perfeição é algo praticamente inatingível, ou então seu custo é altíssimo pelos padrões atuais, mas sua busca incessante continua gerando efeitos surpreendentes.
Outra diferença recai sobre o modo como as pessoas trabalham. Enquanto a maioria delas achará seu trabalho mais estimulante – inclusive os operários de chão de fábrica – à medida que a produção enxuta vai se disseminando e sua produção aumentando, poderá ocorrer que suas tarefas se tornem mais estressantes. Isso porque um dos objetivos-chave desse sistema de produção é trazer a responsabilidade para a base da pirâmide organizacional, responsabilidade essa que significa liberdade para controlar o próprio trabalho – uma vantagem – mas que aumenta o temor de cometer erros que acarretem prejuízo, certamente uma desvantagem em nossa mentalidade de insegurança no emprego e de parcos estímulos aos processos de tomada-de-decisão nesse nível.
A palavra japonesa *muda* é o centro da revolução *Lean*.
*Muda* significa desperdício: especificamente qualquer atividade que absorva recursos mas com resultados sem valor. Esse desperdício pode ocorrer de diferentes formas e há diferentes métodos para eliminá-los (OHNO, 1988). Alguns foram identificados por Taiichi Ohno, executivo da Toyota: erros que são subsequentemente retificados, produção e inventário sem demanda, processos que não são realmente necessários, movimentação desnecessária de pessoas e “coisas”, espera, e produtos e serviços que não atendem às necessidades do cliente.
Portanto, os conceitos básicos de Pensamento Enxuto (Ohno, 1988) e aplicados a desenvolvimento de produtos podem ser definidos como a seguir (WALTON, 1999):

**1. Valor e Cadeia de Valor**

Identificar o valor de um produto e o processo mais efetivo (em termos de custo e

cronograma) para atender o valor é um dos maiores objetivos de “enxuto”. Valor é a capacidade provida ao cliente na hora certa, no preço apropriado, como definido em cada caso pelo cliente. O mais efetivo processo é alcançado pela execução do menor número de passos de valor-adicionado e sem passos que não adicionam valor. Isso é conseguido por meio do mapeamento da cadeia de valor. A cadeia de valor são as atividades específicas requeridas para desenhar, ordenar, e prover um produto, desde o conceito à entrega,  do materia bruto às mãos do cliente.

Uma dificuldade que se apresenta é quando se tenta definir valor somente no contexto do desenvolvimento do produto. Ao final do processo de desenvolvimento de produto, valor é somente parcialmente percebido. O desenho pode eventualmente satisfazer o usuário final, mas tem que passar pela produção, operações, suporte e possivelmente *upgrades* antes que o ciclo de vida possa ser avaliado. Aspectos de “valor” podem incluir um desenho fácil de produzir a baixo custo; um desenho que satisfaça os requisitos do cliente com um nível aceitável de risco; ou um fornecedor de infra-estrutura que suporta a produção. Tudo isso contribui para o sucesso do produto.

Há também valor que flui para futuros desenvolvimentos como preservação de capital humano e experiência, sinergia com outros produtos.

Uma definição plausível de valor no desenvolvimento de produto é produtos com a informação correta entregues na hora certa, para processos/clientes *downstream*, onde é quantificado pela forma, ajuste e adequação, função e oportunidade de produtos de informação (WALTON, 1999).

## 2. Desperdício

Desperdício é sempre referido no contexto de *enxuto* como *muda*. Há dois tipos: Tipo I é encontrado em atividades que adicionam valor ao cliente, são necessárias na estrutura de desenvolvimento para entregar o produto. Tipo II é encontrado em atividades que não criam valor e podem ser eliminadas imediatamente.

## 3. Fluxo

Fluxo é descrito como a realização progressiva de tarefas ao longo da cadeia de valor de tal forma que um produto prossegue do desenho ao lançamento, entrega, da matéria bruta às mãos do cliente sem interrupção, refugos ou retrocessos.

**4. Puxado**

Puxado é definido como um sistema de produção cascateada e entrega em que nada é produzido pelo fornecedor acima na cadeia até que o cliente abaixo sinalize a necessidade. As três seguintes características são condições necessárias para o puxado:

- Sincronização (*Timing*);
  Sincronização se refere a alinhar o *takt time* de processos interconectados de tal forma que o apropriado "*timing*" está adequado, assim possibilitando fluxo e permitindo que puxado seja bem-sucedido;
- Alinhamento (*Position*);
  Alinhamento descreve posicionamento apropriado que é necessário para o puxado ocorrer. Num sentido de manufatura, isso poderia significar posição física. Num ponto de vista de desenvolvimento, isso poderia significar formato e localização de aquivo adequados.

**5. Perfeição**

Esse é um princípio baseado no conceito de que não há fim no processo de reducão de esforço, tempo, espaço, custo e erros. Basicamente, esse é o conceito de Kaizen (Imai, 1986) ou melhoria contínua. Sempre tentar ser melhor, mais barato e mais rápido deve ser o objetivo constante de todos envolvidos nos fluxos de valor. A busca do aperfeiçoamento contínuo em direção a um estado ideal deve nortear todos os esforços da empresa, em processos transparentes onde todos os membros da cadeia (montadores, fabricantes de diversos níveis, distribuidores e revendedores) tenham conhecimento profundo do processo como um todo, podendo dialogar e buscar continuamente melhores formas de criar valor.

Para Ohno (1988), a identificação e eliminação completa de todos os desperdícios constituem a base do Sistema *Toyota* de Produção.

Shingo (1996) classifica sete tipos de desperdícios identificados no Sistema *Toyota* de Produção:

- **Super produção**, produzir excessivamente ou cedo demais, gerando quebra no fluxo de peças e informações e excesso de inventário;
- **Espera**, longos períodos de ociosidade de pessoas, peças e informação, gerando quebra no fluxo e *lead times* longos;

- **Transporte excessivo**, movimentação excessiva de pessoas, peças e informação, gerando desperdício de capital, tempo e energia;
- **Processos inadequados**: utilização errada de procedimentos, ferramentas, sistemas e procedimentos, debilidade de ferramentas e de senho de produto;
- I**nventário desnecessário**, armazenamento excessivo e falta de informação;
- **Movimentação desnecessária**: desorganização do ambiente de trabalho, gerando baixo desempenho dos aspectos ergonômicos;
- **Produtos defeituosos** problemas no processo, na qualidade dos produtos e baixo desempenho nas entregas.

O *Toyota Production System* é representado por uma estrutura em que os objetivos principais, que são melhor qualidade, o menor custo, o menor *Lead Time* e alta moral estão sustentados por duas colunas:

- a primeira é composta pelo *Just-in-Time*, ou seja, a parte certa, quantidade certa e tempo certo, tendo como princípio a busca do Fluxo Contínuo, o *Takt Time* e Sistema Puxado, Mudança Rápida e Logística Integrada.
- A segunda pelas Pessoas Altamente Motivadas
- A terceira pelo *Jidoka* (autonomação), com Paradas Automatizadas, Separação pessoa-máquinas, prova de erro, controle de Qualidade na Estação, e os 5 Por quês.
- Na base na qual se busca a estabilidade temos o *Heijunka*, o Trabalho Padronizado, Gerenciamento Visual, Manutenção Produtiva Total (TPM) e o *Kaizen*.

Smaley (2006), afirma que no modelo *Toyota* cinco etapas devem ser cumpridas para se chegar a uma manufatura enxuta. Estas etapas fazem parte do *Total Flow Management* do Sistema *Toyota* de Produção, porém, para cada empresa dependerá de seu estado atual. As etapas descritas são:

1. Estabilidade básica dos equipamentos;
2. Aperfeiçoar o fluxo da produção;
3. Nivelamento da produção;
4. Desenvolver o sistema puxado
5. Desenho do fluxo de valor

Sobre os princípios e ferramentas que sustentam a manufatura enxuta, segue breve revisão em que se busca o entendimento destas relações na composição do Sistema *Toyota* de Produção:

**1. *Just-in-time* (JIT).**

Conforme Tubino (1997), o *Just-in-time* está baseado em três princípios básicos: um sistema de gerenciamento da produção, um sistema de garantia da qualidade e um sistema de manutenção preventiva total. Tem como objetivo principal minimizar ou eliminar desperdício e atividades que não agregam valor ao produto, sendo que a produção deve seguir rigorosamente as quantidades, nas datas certas e com qualidade.

Crawford, *apud* Pires (1995), descreve oito principais razões para implementação do *Just-in-time*, que são: reduzir inventários, reduzir custos de manufatura, reduzir ciclos de produção, aumentar a qualidade dos produtos, reduzir necessidade de espaço físico, atingir uma posição competitiva melhor, aumentar a margem de lucro e aumentar a eficiência da mão de obra.

**2. Tempo *Takt***

***Takt Time*** é o ritmo de produção necessário para atender a um determinado nível considerado de demanda, dadas as restrições de capacidade da linha ou célula. O tempo *takt* desenvolve papel importante na manufatura enxuta , pois é ele que determina o ritmo de funcionamento da fábrica , se constituindo na base para o sistema puxado (*Kanban)* e pode-se entender como o tempo que rege o fluxo de materiais em uma linha ou célula.

**3. Jidoka - Autonomação**

Conforme Kosaka (2006), o *Jidoka*, teve origem com a automação da máquina de tear fabricada por Sakichi Toyoda (1867-1930), fundador da *Toyota Automatic Loom Works*, e consiste na transferência da inteligência humana para equipamentos, automatizando-os de modo a permitir que as máquinas detectem a produção de uma única peça defeituosa e suspendam imediatamente seu funcionamento enquanto se solicita ajuda. O *Jidoka* tem como objetivo principal garantir a qualidade dentro do processo, e tanto pode ser aplicado à máquina quanto em atividades que envolvam o homem.

**4. *Heijunka* - Nivelamento**

O *Heijunka*, conforme Niimi (2004) significa produção nivelada. Um dos objetivos do *Heijunka é a* determinação de um lote fixo de produção e é a chave para se atingir a estabilidade nos processos produtivos, constituindo-se numa parte crítica para a implementação da metodologia.

**5. Estabilidade Básica**

A estabilidade básica para Smalley (2006) implica na previsibilidade geral e disponibilidade constante em relação à mão de obra, máquinas, materiais e métodos (4Ms). Na manufatura enxuta, a estabilidade está apoiada em metodologias como o 5S, o *TPM*, *Kaizen*, Trabalho padronizado.

Segundo Kishida *et al. (2006),* o Trabalho Padronizado é uma ferramenta enxuta básica centrada no movimento e trabalho do operador e aplicada em situações de processos repetitivos, visando a eliminação de desperdícios. Trata de estabelecer procedimentos precisos para o trabalho de cada um dos operadores em um processo de produção, baseado em três elementos:

- **Tempo *takt***, o ritmo em que os produtos devem ser produzidos para atender a demanda do cliente.
- **Seqüência de trabalho** em que um operador realiza suas tarefas dentro do tempo *takt*.
- **Estoque padrão de processo**, incluindo os itens nas máquinas exigidos para manter o processo operando suave e continuamente.

O trabalho padronizado permite a prática do *Just-in-Time* e do *Jidoka* (Autonomação) além de assegurar uma estabilidade básica nos processos para garantir que eventuais melhorias sejam mantidas de forma contínua.

## 2.1. Desenvolvimento Enxuto de Produtos

Segundo Walton (1999), a essência de “enxuto” é muito simples, mas não se pode dizer o mesmo sob a perspectiva de implementação.

Progresso maior tem sido visto em implementação e atenção às práticas do chão de fábrica. Entretanto, o nível de implementação e educação em outras áreas, como desenvolvimento de produto é muito menor (WALTON, 1999).

O desenvolvimento de produto é um conjunto de atividades que se inicia com a

percepção de uma oportunidade de mercado e finaliza na produção, venda e entrega de um produto (ULRICH *et al.*, 2004).

Processo de desenvolvimento ágil de produto é aquele em que rapidamente se introduz uma sucessão constante de melhorias incrementais de produto – que pode ser chamado de “novos” produtos – que são realmente planejados como “variações sobre um tema”, baseado em partes comuns e arquitetura de produto modular, tornando o tempo de resposta ao mercado mais rápido (ANDERSON, 2003).

A área de desenvolvimento de produto é rica em oportunidades para melhoria. O tamanho do tempo que se leva para desenvolver um novo produto; o grau ao qual o produto satisfaz os requisitos do cliente; e a facilidade com que novos produtos podem ser produzidos são todas áreas em que a maioria das companhias pode fazer dramáticas melhorias.

## 2.2. Desenvolvimento Enxuto de *Software*

O sonho de toda empresa é ter *software* desenvolvido com menor custo, mais rapidamente e de melhor qualidade. Pesquisas e investimentos têm sido feitos em tecnologias, metodologias, linguagens, ferramentas porém ainda continua um desafio.

Os princípios da manufatura enxuta hoje são aplicados não somente à manufatura mas extrapolaram para o domínio de gestão de toda a empresa, no desenvolvimento do produto.

O ciclo de vida de um produto de *software* cobre desde a captura e decisão de quais novas funcionalidades ou características devem ser incorporadas, prazos, aceitação do cliente até a entrega do produto.

Wirth (1995) lamenta que quase todo *software* tenha “crescido” gordo. O autor atribui isso ao crescente desempenho do *hardware* e à complexidade desnecessária construída no *software*. Obviamente, *software* gordo necessita desenvolvimento de *software* gordo.

As causas para complexidade são que os produtos de *software* abraçam qualquer funcionalidade que os usuários possam querer, assim como um desenho monolítico para todas as funcionalidades concebíveis e recomenda boa engenharia por refinamentos graduais do produto que pode ser entendido como melhoria contínua ou *kaizen* (IMAI, 1986).

*Softwares* sofisticados são complexos e boas soluções seriam desenvolvidas via processos de mentalidade iterativa e consumo de tempo. Em vista de restrições de

tempo, as inadequações do *software* são corrigidas via adições concebidas rapidamente aumentando complexidade e tamanho do *software*.

As práticas da produção enxuta – guias específicos sobre o que fazer – não podem ser transplantados diretamente de uma planta da manufatura para o desenvolvimento de *software*. Muitas tentativas de aplicar as práticas da produção enxuta ao desenvolvimento de *software* têm sido mal-sucedidas porque gerar bom *software* não é um processo de produção, mas sim processo de desenvolvimento.

Segundo Poppendieck e Poppendieck (2003), desenvolvimento é bastante diferente de produção. Pensar no desenvolvimento é como criar uma receita e produção como seguir a receita. Essas são atividades bem diferentes, e elas devem seguir diferentes abordagens. Desenvolver uma receita é um processo de aprendizado envolvendo tentativa e erro. De fato, a idéia total de desenvolver uma receita é tentar muitas maneiras num tema e descobrir o melhor prato. Isso é um processo iterativo em que se desenvolve, degusta, realimenta a receita, o que gera valor com o ciclo de *feedback.* Uma vez que o *chef* desenvolveu uma receita, preparar o prato significa seguir a receita. Isso é equivalente à manufatura, na qual o objetivo é reproduzir fiel e repetidamente uma receita com o mínimo de variação.

Segundo Bohem e Turn (2005), qualidade no desenvolvimento de *software* resulta num sistema com integridade percebida e integridade conceitual. Integridade percebida significa que a totalidade do produto atinge o equilíbrio de funcionalidade, usabilidade, confiabilidade, e economia que satisfaz o cliente. Integridade conceitual significa que os conceitos centrais do sistema trabalham juntos de uma forma suave e coesa. Então, qualidade do desenho significa realização de objetivo ou adequação de uso mais do que conformidade a requisitos. Desenvolvimento pretende produzir soluções apropriadas para problemas únicos de clientes.

### 2.2.1. Os sete princípios do Desenvolvimento Enxuto de *Software*

Os conceitos básicos do Pensamento Enxuto são: Valor, Cadeia de Valor, Desperdício, Fluxo, Puxado e Perfeição, conforme descrito nesse capítulo 2.

Conforme Raman (1998), esses princípios podem ser aplicados ao desenvolvimento de *software*. O fundamental é entender claramente o que o cliente quer e o que ele considera “valor” e encontrar a maneira mais direta de se implementar isso, mantendo o fluxo, sem parada. Em desenvolvimento de *software,* essa abordagem poderia ser obtida

de uma integração coordenada, diária e contínua de pedaços do projeto, para validação pelo próprio cliente. Puxado é a base para o conceito de inventário *Just-in-time* (JIT), mas em *software*, pode significar que somente o que é requerido pelo cliente deve ser contruído. Em relação à perfeição, há dois aspectos. O primeiro é eliminar defeitos enquanto se desenvolve o produto. O segundo, é remover o desperdício.

O foco do desenvolvimento de *software* deve ser ter uma resposta rápida a uma necessidade identificada. Mecanismos devem ser "criados" para encurtar o tempo. E o desenvolvimento rápido deve ser considerado importante desde o início, para que haja mudança de pardigmas e se mudem as práticas do paradigma de produção em massa para o Pensamento Enxuto.

Poppendieck e Poppendieck (2006), baseados nesses conceitos básicos do Pensamento Enxuto, propuseram sete princípios que formam o coração do desenvolvimento enxuto de *software*, que são:

1. Eliminar desperdício
2. Construir com qualidade
3. Criar conhecimento
4. Adiar compromissos
5. Entregar rápido
6. Respeitar as pessoas
7. Otimizar o todo

Segue sumário desses sete princípios.

#### 2.2.1.1. Eliminar o Desperdício

Analogamente ao que Ohno (1988) falou sobre a produção, em desenvolvimento de *software* se começa o relógio da linha do tempo quando se recebe uma ordem para endereçar uma necessidade do cliente e para quando o *software* está implantado. Desenvolvimento de s*oftware* enxuto foca em reduzir a linha do tempo removendo tudo que não adiciona valor. Para eliminá-lo, é preciso reconhecê-lo. Não há substituto para desenvolver um profundo entendimento do que o cliente realmente valorizará uma vez que comece a usar o *software*. E muitas vezes o usuário muda de opinião quando vê o *software* em ação. Portanto, segundo Poppendieck (2007) qualquer coisa que se faça que não adiciona valor, ou que retarde a possibilidade do cliente usar o *software* quando ele quiser, é desperdício.

Em manufatura, inventário é desperdício. Em desenvolvimento de *software*, inventário é um trabalho parcialmente feito, pois não pode ser utilizado pelo cliente, ao mesmo tempo que a sua manutenção se complica. Outra forma, é *churn* (ou mudanças frequentes de requisitos), mas que segundo Poppendieck e Poppendieck (2006), está associado a grandes inventários de trabalhos parcialmente feitos, com longo tempo de desenvolvimento. A maior fonte de desperdício são as funcionalidades extras.. Há um custo enorme em se desenvolver funcionalidades extras num sistema de *software*. Elas adicionam complexidade ao código base que elevará seu custo numa taxa alarmante, e tornando cada vez mais caro para testar, manter, e dramaticamente reduzindo sua vida útil. É necessário um processo que permita desenvolver o que realmente agrega valor ao cliente, assim como ele possa validar ao longo do desenvolvimento.

Não se deve estabelecer o escopo com uma lista de tudo que um sistema possa ter. Uma possibilidade, conforme Poppendieck e Poppendieck (2006) é a abordagem flexível mudando incrementalmente, evoluindo funcionalidades objetivando atividades e valores específicos de clientes.

A Cadeia de Valor é uma série de ações para desenvolver um produto de valor para o cliente, do pedido à entrega. É importante que toda a cadeia de valor seja melhorada. Em desenvolvimento de *software*, não basta entender os requisitos do cliente, mas é igualmente importante que o desenho seja desenvolvido meticulosamente para que refinamentos e evoluções graduais sejam possíveis (RAMAN, 1998).

Segundo Poppendieck (2007), analogamente à definição feita para a manufatura (Shingo, 1996), desperdício em *software* é identificado como:

- Extra Produção: Funcionalidades extras; funcionalidades desnecessárias => deve-se desenvolver de acordo com o definido nos requisitos; desenvolver de acordo com os requisitos imediatos dos clientes; desenvolver somente o que é crítico e necessário para o cliente no curto intervalo de tempo;
- Inventário: Requisitos de sistema esperando para serem desenvolvidos; excesso de documentação => deve-se desenvolver código, não documentação; ter frequentes entregas, não acumular código; detalhamento de requisitos somente para a iteração corrente;
- Passos extras de processamento: Codificar diretamente a partir de definições; esclarecer diretamente com o cliente, significando que cliente é parte integrante do time de desenvolvimento; documentação somente para o necessário;

- Movimentação: Remover linhas extras de comunicação; ter desenvolvedores junto aos clientes; encontrar informação; ter todos na mesma sala;
- Defeitos: Testar cedo e frequentemente; desenvolvimento dirigido por teste
- Espera: Não deixar clientes esperando; entregar com frequência, ciclos de iteração rápida; reduzir tempo de tomada de decisão; comunicação face-a-face para entendimento imediato, incluindo clientes => entregar em incrementos curtos;
- Transporte: Entregar trabalho diretamente para o cliente; evitar transferências entre participantes (ie, analista para programador para testador para implementador para cliente).

Um outro princípio que é o coração do desenvolvimento de *software* é construir com qualidade e será detalhada a seguir.

#### 2.2.1.2. Construir com Qualidade

Segundo Poppendieck e Poppendieck (2006), o objetivo é construir qualidade no código desde o início. É evitar criar defeitos. Requer uma organização altamente disciplinada. Esse princípio claramente se mapeia no conceito básico de “perfeição” do Pensamento Enxuto.

Segundo Shingo (1988), devem-se controlar as condições para não permitir o defeito entrar. Se necessárias as inspeções, então que sejam feitas logo depois do passo que ocorreu. Quando o defeito é encontrado, deve-se parar a linha, encontrar a causa, e solucioná-lo imediatamente. Sistemas de acompanhamento de defeitos são filas de trabalhos parcialmente feitos, ou de retrabalho.

Na indústria de *software* esse slogan “faça certo na primeira vez” tem sido usado para defender grandes quantidades de documentação e passos e pessoas intermediários no processo, pensamento de produção em massa. Seria melhor traduzido em *software* como “Teste Primeiro”. Portanto, esse domínio seria acoplar ciclos curtos de *build* com testes automatizados (POPPENDIECK, 2007).

Outro princípio a ser tratado é o de criar conhecimento, definido a seguir.

#### 2.2.1.3. Criar Conhecimento

Desenvolvimento de *software* é um processo de criação de conhecimento. É um

processo análogo à criação de uma receita nova para determinados ingredientes determinados pelo cliente. Esse princípio se baseia no conceito básico da melhoria contínua.

Um processo de desenvolvimento de *software* focado em criar conhecimento deve considerar que o desenho evolua durante o código, e não gastará tempo em fechá-lo prematuramente. Assim como o processo deve estar em constante reavaliação e melhoria (HIGHSMITH, 2002).

MacCormack (2003) identificou quatro práticas que conduziram a desenvolvimentos de *software* bem sucedidos:

- Versões de *software* disponíveis para o cliente avaliar e dar retorno de um conjunto mínimo de funcionalidades, o mais cedo possível.
- Integração contínua, se possíveis diárias.
- Um time ou um líder com a experiência e instintos para tomar boas decisões
- Arquitetura modular que suporte facilmente adicionar novas funcionalidades.

O próximo princípio para o desenvolvimento de *software* é adiar o quanto possível, o "assumir" compromissos.

#### 2.2.1.4. Adiar compromissos

Segundo Poppendieck e Poppendieck (2006), o desafio para o respondedor é decidir quanto tempo ele pode esperar antes de ter que tomar decisão crítica. Decisões irreversíveis de cronograma devem ser feitas na última chance de fazê-las, antes que seja muito tarde. Primeiramente, devem ser tomadas as decisões reversíveis, que podem ser facilmente alteradas. Ohno (1988) disse que planos mudam muito facilmente. Acordos mundiais nem sempre vão de acordo com o plano e pedidos mudam rapidamente em resposta a alterações nas circunstâncias.

Esse conceito está relacionado com o "puxado", pois somente quando um cliente mais abaixo da corrente pede é que o fornecedor corrente acima deve produzi-lo. Essa é a base do conceito de inventário *Just-in-time* num ambiente de manufatura. A pergunta a ser feita é: qual é "o último momento" que se pode decidir pela aquisição, desenvolvimento, manufatura de um determinado item, de tal modo que os riscos para o negócio sejam mitigados, inventário minimizado e ao mesmo tempo não prejudique o fluxo?

De acordo com Boehm e Turner (2005) as abordagens de desenvolvimento de *software*

são baseadas na expectativa de que um plano é um compromisso. E por isso pode ser medido o resultado contra o plano original. Mas no ambiente empresarial, competitivo e de constante mudanças, fechar requisitos e planejamentos baseados nos primeiros acordos iniciais, são prematuros. Pois, no início, o próprio cliente pode não saber exatamente o que ele precisa.

A comunidade de *software* tem uma tendência de incluir funcionalidade que se 'pensa' ser bom para o cliente ter. O resultado é um *software* gordo e desnecessariamente complexo (WIRTH, 1995).

Portanto, o processo de desenvolvimento de *software* deve executar as atividades da sua cadeia de valores de tal forma que, baseadas nas prioridades do cliente, as decisões possam ser tomadas, "no momento adequado ou quão mais tarde possível", para não dificultar ou bloquear mudanças futuras, ou que as decisões não sejam tomadas muito cedo no ciclo de desenvolvimento, e que aquele requisito se torne um inventário de *software*, sem uso.

Outro princípio do desenvolvimento de *software* que deve ser abordado é a entrega rápida, conforme detalhado à frente.

#### 2.2.1.5. Entregar rápido

A idéia é entregar *softwar*e tão rápido que o cliente não tenha tempo de mudar de idéia (POPPENDIECK e POPPENDIECK, 2006). Isso requer uma qualidade muito boa e um entendimento profundo do cliente. Será necessária uma equipe com pessoas engajadas e que pensam e são confiáveis para tomar decisões e ajudar um ao outro.

Organizações enxutas, segundo Kishida e Silva e Guerra (2006) trabalham para padronizar, mas esses padrões existem porque eles incorporam o melhor do conhecimento corrente sobre como fazer o trabalho. Eles formam uma linha base contra a qual espera-se que trabalhadores experimentem encontrar melhores caminhos para fazer seu trabalho. Existem para serem desafiados e melhorados.

A idéia de fluxo é fundamental para a produção enxuta. Se você só faz o que adiciona valor, então você deve fazê-lo quão rápido um fluxo permite.

"Puxado" significa que nada é feito a menos e até que o processo seguinte requeira isso. O efeito do "puxado" é que a produção não é baseada em previsão: compromisso é adiado até a demanda ser apresentada para indicar o que o cliente realmente quer (WOMACK e JONES, 1996).

A idéia de enxuto é ser capaz de fazer o produto tão rápido quanto possível para atender à ordem.

Em desenvolvimento enxuto de *software*, a idéia é maximizar o fluxo de informação e entregar valor. Documentos que não são úteis ao cliente são substituídos por testes automatizados. Esses testes asseguram que o valor do cliente é entregue ambos inicialmente e no futuro quando as trocas inevitáveis são necessárias (UDO e KOPPENSTEINER, 2003).

Em desenvolvimento de *software*, a chave para a entrega rápida é dividir o problema em lotes menores (incrementos) puxados por uma demanda e teste de cliente. O mecanismo mais simples e efetivo de implementar desenvolvimento enxuto é entregar incrementos de valor de negócio real em períodos de tempo curto (POPPENDIECK e POPPENDIECK, 2006).

A ênfase é parear um time de desenvolvimento capacitado com um time de cliente capacitado e dar-lhes a responsabilidade e autoridade para desenvolver o sistema em pequenos e rápidos incrementos, conduzidos pela prioridade e *feedback* do cliente.

É claro que no desenvolvimento enxuto de *software*, não poderia faltar o respeito às pessoas.

#### 2.2.1.6. Respeitar as pessoas

Respeito às pessoas significa em desenvolvimento de *software* olhar sob o ponto de vista das pessoas fazendo o trabalho. Uma empresa que respeita seus colaboradores desenvolve bons líderes, que fomentam engajamento, pessoas pensantes e foca seus esforços em criar produtos bons. Força de trabalho de especialistas deve ser nutrida. Planejamento e controle baseado em responsabilidade, onde ao invés de se falar às pessoas o quê e como fazer desenvolve-se uma organização onde pessoas decidem por elas mesmas (WOMACK *et al*., 1992).

Para uma organização ser capaz de dizer que centraliza naqueles que adicionam valor, e todas são importantes, então as pessoas que fazem o trabalho são o centro de recursos, informação, autoridade de Desenho de Processo, autoridade de Tomada de decisão, centro da Energia Organizacional.

Segundo Poppendieck (2007), centralizar em pessoas que adicionam valor significa melhorar a capacitação dos desenvolvedores por meio de treinamento. Significa formar times que desenham seus próprios processos e endereçam os problemas completamente.

Isso significa que os grupos de trabalho e gerentes existem para suportar os desenvolvedores, não para lhes dizerem o que fazer.

O pensamento enxuto no desenvolvimento de *software* deve "afetar" várias organizações, e cujo "caminho" deve ser otimizado. Esse é outro princípio a ser visto.

#### 2.2.1.7. Otimizar o todo

Há uma quantia considerável de ganhos para serem obtidos com contratos de estrutura, acordos de *outsourcing* e interações *cross*-funcionais com incentivos corretos que garantem que todos estão focados na otimização do todo.

Segundo Morien (2005), com certa frequência, a maior barreira em adotar práticas enxutas é organizacional. Como produtos movem de um departamento para outro, uma grande lacuna frequentemente desenvolve, especialmente se cada departamento tem seu próprio conjunto de medidas de desempenho que não estão relacionadas às medidas de desempenho dos departamentos vizinhos.

Isso pode criar o que é conhecido como medida sub-otimizada, porque cria um comportamento que cria uma otimização local em detrimento de otimização global.

Uma das maiores medidas de sub-otimização em desenvolvimento de *software* ocorre quando os gerentes de projeto são medidos pelo *Earned Value*, que é o custo inicialmente estimado para as tarefas que devem ser completadas. A idéia é que não deve ser gasto nada a mais do que foi estimado. O problema é que, isso requer que o gerente de projeto construa um inventário de descrições de tarefas e estimativas. O inventário de tarefas requeridas para cálculo de *Earned Value* entra no caminho de entrega de valor de negócio real e degrada hora extra. Quando se tem que decidir entre entregar valor ou *earned value*, *earned value* usualmente vence (MORIEN, 2005).

Para impedir esses problemas, organizações enxutas estão usualmente estruturadas ao redor de times que mantém responsabilidade pelo valor de negócio como um todo, acima de medidas intermediárias (POPPENDIECK, 2007).

Os princípios básicos de eliminar desperdício (Poppendieck e Poppendieck, 2006), de dar poder aos trabalhadores da linha de frente, responder imediatamente aos pedidos do cliente e otimização da cadeia de valor são fundamentais ao pensamento enxuto. Quando aplicado ao desenvolvimento de *software*, esses conceitos provêm uma poderosa infra-estrutura para melhoria.

O desenvolvimento de *software* por intermédio de processos ágeis “surge” como uma alternativa de se sustentarem os princípios do desenvolvimento enxuto descritos, conforme detalhado no próximo item.

### 2.2.2. Processos Ágeis

Os processos ágeis são um conjunto de práticas que focam no desenvolvimento rápido de *software*. De acordo com Highsmith (2002), agilidade baseia-se nos aspectos de que adotar mudança é melhor do que tentar resistir a ela e que o foco deve estar no talento e capacitação dos indivíduos e times. No desenvolvimento de *software*, o que precisa de melhoria é responder rápida e efetivamente à mudança, o que requer minimizar o custo de transferência de conhecimento, custo de captura de conhecimento (documentos!) e o tempo entre tomar a decisão e explorar seus resultados para aprender em consequência de implementá-la. E outro aspecto é estar preparado para mudanças.

Conforme Lindvall *et al.* (2004) nos anos recentes, o uso, interesse e controvérsias sobre métodos ágeis tem crescido dramaticamante. Ainda há necessidade de se esclarecer em quais ambientes e sob quais condições métodos ágeis fucionam melhor. Embora a maioria das organizações tenha necessidades similares, ainda há necessidade de se terem evidências antes de adotar novos métodos em grandes organizações por causa de suas complexidades e da necessidade de integrar novas tecnologias e processos aos existentes.

No desenvolvimento ágil, Highsmith (2002), o período de desenvolvimento é dividido em unidades chamadas iterações. Uma iteração é um período no qual processos são executados para desenvolver o escopo de uma função.

A primeira iteração começa no início do desenvolvimento. Nesse ponto, os requisitos do cliente são claramente indicados ao time de desenvolvimento, que os implementa de acordo com a prioridade. Também, a data final é decidida quando a iteração começa. O tempo típico é de uma semana a um mês. Mesmo se os requisitos não forem todos implementados, a data final é mantida. Todos os requisitos que não foram implementados (que foram os de mais baixa prioridade) são adiados. Se todos os requisitos foram implementados antes da data final, novos requisitos do cliente podem ser trabalhados.

É dada mais prioridade para a data de entrega que para o escopo (MORIEN, 2005).

No gerenciamento de projeto tradicional, o conjunto de atividades do projeto é quase

inteiramente definido na primeira etapa de captura de requisitos onde se requer que os requisitos sejam completamente apurados e estabelecidos, e travados, antes que atividades subsequentes sejam executadas (MORIEN, 2005).

A segunda importante característica do processo de desenvolvimento tradicional é a natureza linear das fases. O modelo em cascata "puro" acontece cada fase por vez, quando os resultados pré-determinados são alcançados e aceitos. Ao longo do tempo, algumas mudanças foram feitas a esse modelo para permitir realimentação limitada, ver figura 2.

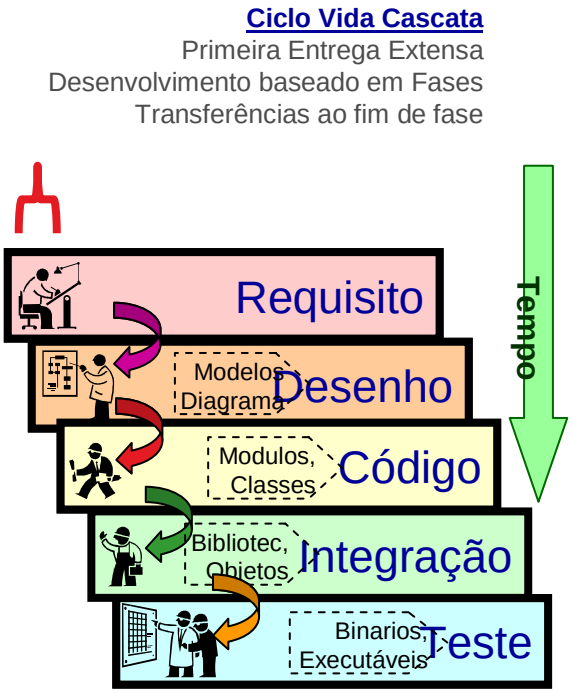


**Figura 2** – Tradicional Ciclo de Vida em Cascata
Fonte: Adaptado de Morien (2005)

Já no processo ágil (Morien, 2005) as mudanças são aceitas como um aspecto inevitável do projeto, e o gerente de projeto deve acomodar isso. Assim, tempo e orçamento podem ser definidos antes do escopo.

O gerenciamento ágil de projeto persegue a produção de requisitos de alta-prioridade, que são entregues aos clientes regularmente, de forma incremental, trabalhando num período fixo de tempo para completude, ou num orçamento fixo para despesas. Dessa maneira, as funcionalidades mais necessárias do sistema são entregues. Decisões são constantemente feitas para entregar as funcionalidades (UDO e KOPPENSTEINER, 2003).

A indústria do *software* se deparou com usuários que somente sabem o que eles querem depois que eles vêem uma versão inicial do *software*. Isso significa que os requisitos não são totalmente entendidos no início do projeto e mudam durante a fase de desenvolvimento de *software* (MORIEN, 2005).

Planejamento é grandemente simplificado porque as datas são sempre conhecidas antecipadamente, e os times, com o dono do produto conduzindo, são responsáveis por determinar prioridades. Acompanhar é simples também, por causa de reuniões diárias e demonstrações frequentes ilustram o progresso (MORIEN, 2005) e (UDO e KOPPENSTEINER, 2003).

Provavelmente, a maior diferença esteja na batalha data *versus* escopo, e nesse caso, data sempre vence. Isso é, o tamanho da iteração determina o escopo ao invés de o escopo determina o tamanho do ciclo de desenvolvimento. Em métodos dirigidos por plano, escopo determina tempo, e duas variáveis (escopo e tempo) variavam ao longo do ciclo. Como os métodos ágeis fixam o tempo e deixa este definir o escopo, somente uma variável permance (o escopo do que pode ser construído), como mostrado na Figura 3.

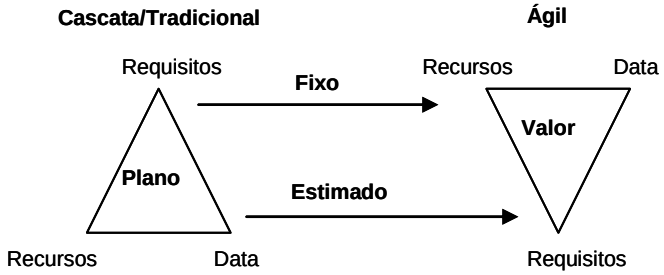


**Figura 3**: Método dirigido pelo plano (tradicional) versus dirigido por valor (ágil)

Fonte: Adaptado de Morien (2005)

Conforme Boehm e Turner (2005) quando a iteração é terminada, mesmo que todos os requisitos do cliente não tenham sido implementados, as funcionalidades de mais alta prioridade estarão implementadas e o *software* operável. Assim que a primeira iteração é terminada, a segunda começa. Frequentemente, as iterações têm o mesmo tamanho em termos de dias, sendo essa uma efetiva maneira para o time aprender o ritmo de desenvolvimento dentro do período de tempo. Quando uma iteração é executada, o *software* é desenvolvido e o escopo implementado é estendido. Também, o *software* pode ser entregue ao cliente toda vez que uma iteração é completada.

Para a priorização de requisitos e a definição de requisitos para cada iteração, o ideal é

que o próprio cliente defina. Porém, o papel do cliente pode ser feito pelo “dono do produto” dentro do time de desenvolvimento, que representa os interesses do cliente, assim como deve entender e assumir a posição do cliente. (BOEHM e TURNER, 2005).

O programa de teste é criado ao mesmo tempo em que o código é desenvolvido. O programa desenvolvido é testado por esse programa de teste, e as tarefas são completadas.

Ao final de uma iteração, uma reunião de “retrospectiva” é realizada onde todos os detalhes relevantes à iteração são revisados, por exemplo, o código, teste, e reuniões, e mesmo ar condicionado, ventilação, iluminação. São revisadas as atividades que continuarão na próxima iteração, discutem-se problemas que ocorreram, soluções encontradas, e confirma-se sua implementação na próxima iteração. E são acordados os novos esforços que devem ser feitos na próxima iteração (BOEHM e TURNER, 2005).

Portanto, a essência do Ágil, Highsmith (2002), é:

- Ciclo de Vida iterativo
  - Feedback rápido e Aprendizado e Correção em ciclos curtos
  - Mais (e mais rápidos) ciclos de desenvolvimento
  - Confrontar riscos e caminhos errados o mais cedo possível
- Times colaborativos
  - Times produzem melhores resultados que Indivíduos
  - Envolvimento do stakeholder possibilita decisões melhores
- Desenvolvimento com Qualidade validada continuamente
  - Testes de regressão automatizadas
  - Integração contínua
  - Desenvolvimento conduzido por testes
- Escopo e Custo eficientes
  - Custo de mudança decrementado
  - Menos tolerância para atividades e escopo de baixo valor

A iteratividade e colaboração melhoram a qualidade e o tempo do ciclo.

Mais ciclos (e mais rápidos) de desenvolvimento garante ciclos de *feedback* em cada nível da escala, confronta riscos e caminhos errados mais cedo, assim como *feedbacks* frequentes permitem correção de planos e processos no decorrer do ciclo.

Segundo Poppendieck e Poppendieck (2003), desenvolvimento ágil procura alavancar cooperação sobre a negociação.

O fundamento dos métodos ágeis é o ciclo de vida iterativo. Métodos ágeis requerem uma mudança do ciclo de vida em cascata para o iterativo, e oferecem um suporte para o ciclo de vida iterativo, conforme ilustrado na Figura 4.

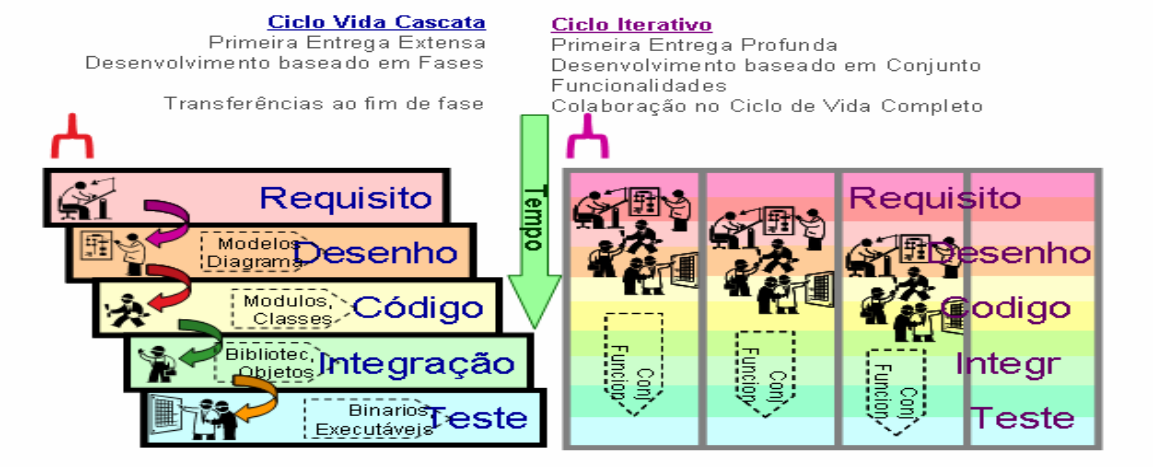


**Figura 4**- Quadro comparativo entre métodos tradicionais e iterativos

Fonte: Adaptado de Morien (2005)

Sob a perspectiva de distribuição de atividades em execução e pessoas alocadas, a figura 5 mostra a diferença entre cascata e ágil.

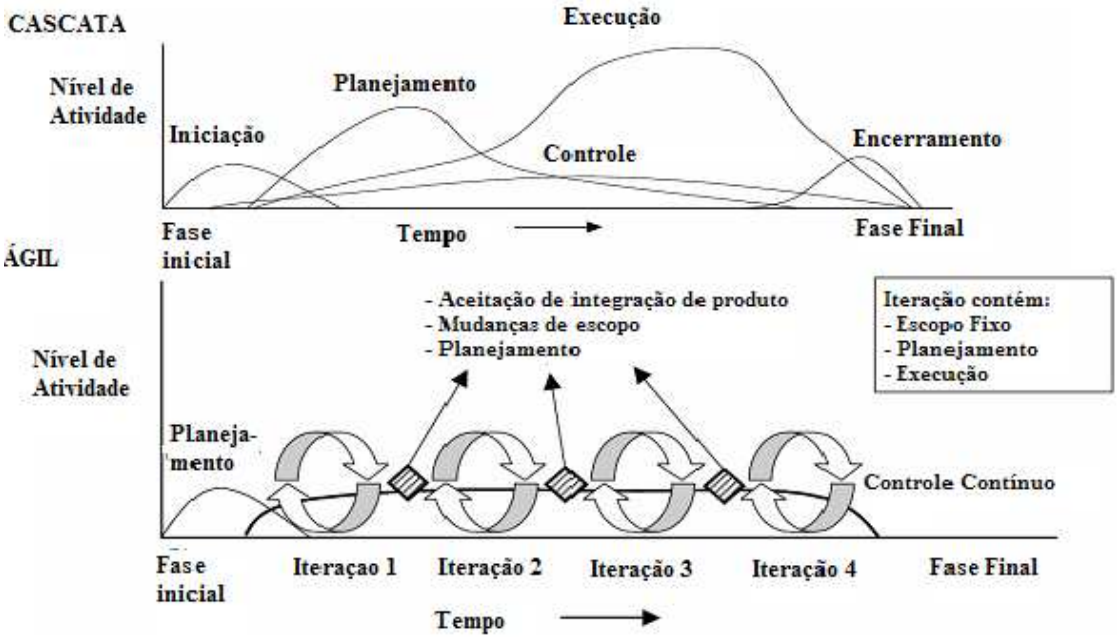


**Figura 5 -** Fases de planejamento e controle – Cascata e Ágil

Fonte: Adaptado de Udo e Koppenstein (2003)

O retorno em Investimentos esperado nos Métodos Ágeis (HIGHSMITH, 2004):

- Investimento em Ágil
  - Criar um ambiente de teste de regressão automatizado
  - Mais planejamento e supervisão dos líderes de projeto
- Retorno de Ágil
  - Reduzir taxa de introdução de falha, melhor qualidade
  - Mais rápida resolução de problemas
- Período de recuperação de Investimento
  - Times geralmente recuperam seus investimentos no primeiro projeto
  - O segundo projeto oferece oportunidade para ganhar produtividade

Relatórios de indústrias sobre a adoção de Ágil:

- "*Agile* Adoption Poll" Report, 2005, baseado nas respostas de 232 participantes:
  - 42% não estão usando métodos Ágeis: 26% não conhecem + 16% conhecem, mas não usam;
  - 14% estão investigando + 4% com projetos piloto + 3% rejeitaram após análise;
  - 37% adotaram os métodos Ágeis na companhia: 17% adoção de algumas práticas + 12% implantação em alguns projetos + 8% implantado em todos os novos projetos.
- August 2006 "Survey Says: *Agile* Works in Practice", baseado na resposta de 4232 profissionais de Tecnologia da Informação:
  - 65% trabalham em organizações que adotaram um ou mais técnicas do desenvolvimento ágil
  - 41% trabalham em organizações que adotaram uma ou mais metodologias ágeis
  - 60% relatam incremento de produtividade
  - 66% relatam incremento de qualidade
  - 58% reportam melhoria na satisfação do *stakeholder*
- *Agile* 2006 Market Survey, baseado na reposta de 136 pessoas / 128 organizações em Outrubro/2005, pela Digital Focus:
  - 46% de companhias de tamanho médio já adotaram Ágil amplamente na companhia;
  - Somente 12% de companhias grandes estão usando práticas ágeis

amplamente na companhia, mas 44% estão usando em pelo menos um projeto;

- 51% dizem que falta de conhecimento/capacidade é uma das 3 barreiras para adoção de Ágil;
- 56% dos executivos vêem falta de experiência/capacidade como uma barreira para adoção de Ágil;
- 51% dos profissionais de não-TI estão olhando para Ágil para melhorar velocidade e previsibilidade no desenvolvimento de *software*
- 47% dos profissionais de TI procuram por Ágil para ajudá-los a gerenciar o escopo de projetos sendo mais responsivo a mudanças.

Fatores motivadores para mudança para Ágil, VersionOne 2006 (Figura 6). Em organizações ao redor do mundo, desenvolvimento Ágil representa um salto significativo em direção a entregas com valor.

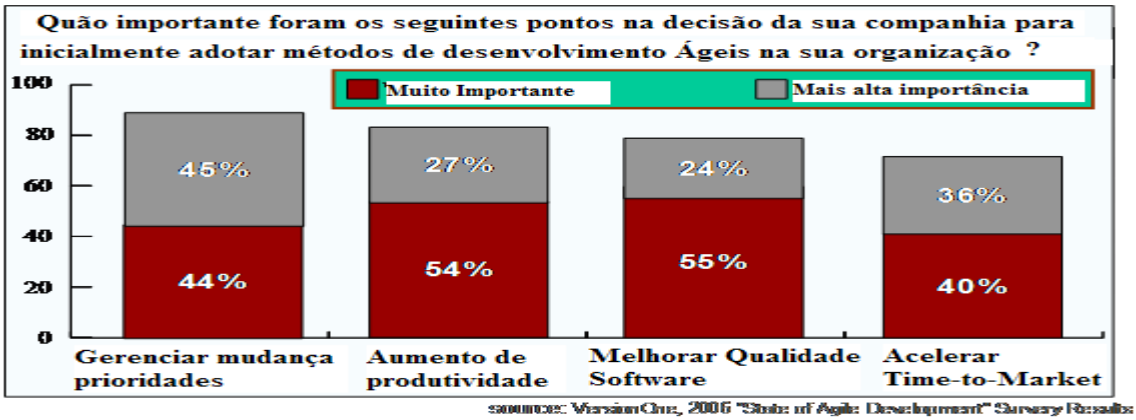


**Figura 6 -** Fatores motivadores para mudança para Ágil

Fonte: VersionOne 2006.

No mundo do desenvolvimento ágil, há várias abordagens de desenvolvimento. Baseado em Highsmith (2002), as principais metodologias são:

1. *Adaptive Software Development* (ASD)
2. *Crystal Methods*
3. *Dynamic Systems Development Model* (DSDM)
4. *Extreme Programming (XP)*
5. *Feature Driven Development* (FDD)
6. *Lean Development (LD)*
7. *Scrum*

Segue breve resumo das principais metodologias.

**1. *Adaptive Software Development* (ASD)**

Desenvolvido por Highsmith (2000), evoluiu a partir de *Rapid Application Development* (RAD) em 1992. ASD é desenhado para adotar mudança em ambientes complexos e incertos. ASD reconhece que falha é "*ok*", e os times devem continuamente adaptar ao projeto específico e à situação. O ciclo de vida envolve um processo iterativo de Especular-Colaborar-Aprender. Planejamento ocorre na fase de Especular. O termo "especular" é um termo mais impreciso que "planejar", e reconhece incerteza. Desenvolvimento ocorre na fase de Colaboração. A fase de Aprendizado reflete no projeto que o time pode adaptar o projeto e seu processo. O papel do gerente de projeto é facilitar a colaboração do time.

**2. *Crystal Methods***

Desenvolvido por Alistair Cockburn, tem uma estrutura que matricia as necessidades de comunicação (tamanho de time), prioridades de projeto, e complexidade de sistema. Embora a estrutura do *Crystal* atue como um guia de processo, processos são basicamente talhados ao time. Comunicação e pessoas vêm primeiro e processos em segundo. Controle é limitado. *Crystal* segue somente duas regras absolutas: uso de ciclos de desenvolvimento incremental menos que quatro meses de duração, e usa *workshops* de reflexão para adaptar a metodologia (UDO e KOPPENSTEINER, 2003).

**3. *Dynamic Systems Development Model* (DSDM)**

Originada na Inglaterra no meio dos anos 90, também surgiu a partir de *Rapid Application Development* (RAD). É controlado por um consórcio, cujo ingresso é quase inexpressivo, e tem documentação extensiva, serviço e suporte disponível. É a única metodologia Ágil que se diz de acordo com a ISO 9000. O modelo tem três processos: Modelagem funcional, desenho & desenvolvimento, e implementação. Sub-processos existem para cada área principal, e todos processos são iterativos. Iterações são curtas e "tempo-definido" (UDO e KOPPENSTEINER, 2003).

**4. *Extreme Programming* (XP)**

É provavelmente a metodologia Ágil mais popular e que gera maior interesse. Segundo Highsmith (2002), foca em interações de cliente e desenvolvedor. Também identifica o ambiente apropriado para uso: times no mesmo espaço físico, de 10 pessoas ou menos

desenvolvedores com um cliente no local dedicado integralmente ao projeto. Como outra metodologia Ágil, desenvolvimento ocorre em iterações curtas (três semanas ou menos).

**5. *Feature Driven Development* (FDD)**

Diferentemente das outras metodologias Ágeis, FDD alega ser baseada em processos "repetitíveis". Dá mais ênfase na modelagem. Projetos começam com um modelo de esqueleto que é continuamente atualizado à medida que o desenho amadurece. Funcionalidades são definidas como funcionalidade de negócios que podem ser entregues em incrementos de duas semanas ou menos. Funcionalidades dirigem o ciclo de vida de desenvolvimento em iterações curtas de duas semanas (similar a todas metodologias Ágeis). FDD também dá grande ênfase em gerenciamento de projeto. Requisitos são gerenciados por todo o ciclo de vida. Variações no desenvolvimento de funcionalidades de 10% ou mais vai para a gerência para decisões de escopo (reduzir funcionalidades ou mover o cronograma) (UDO e KOPPENSTEINER., 2003).

**6. *Lean Development* (LD)**

**E**mergiu a partir do movimento de produção enxuta na manufatura nos anos 80. É estrategicamente focada em outros modelos ágeis. O objetivo é simultaneamente reduzir o tempo, custo, e defeitos de desenvolvimento de *software* em 30%. A fim de atingir esses objetivos, mudanças incrementais não são suficientes. Maiores melhorias requerem pensamento radical e suporte da gerência senior. Requer adoção no mais alto nível de gerência a fim de assegurar sucesso (POPPENDIECK e POPPENDIECK, 2006).

**7. *Scrum***

Desenvolvido por Ken Schwaber e Jeff Sutherland, por volta de 1995. Tem mais ênfase em gerenciamento de projeto que qualquer outra metodologia Ágil. *Scrum* emprega um processo iterativo e incremental que sustenta um conjunto de práticas e regras que incorporam a transparência, inspeção, e adaptação.

Projetos com caos controlado requerem diferentes técnicas de projeto, e *Scrum* usa processos explícitos de monitoramente e mecanismos constantes de *feedback*. Desenvolvimento ocorre em iterações de 30-dias chamadas "*sprints*". No início de cada iteração, o time revê o que será feito, e é determinado o que pode se tornar uma

funcionalidade que poderia ser “entregue” ao cliente.

O “coração” da produtividade do *Scrum* ocorre durante as iterações. O time decide a melhor maneira de construir a funcionalidade. A lista de requisitos é chamada de *backlog*.

Reuniões diárias de “*scrum*” (30 minutos ou menos) acontecem com todo o time para identificar bloqueios e prover *feedback* de estado. As perguntas que devem ser respondidas são: O que você fez desde a última reunião? O que você planeja fazer até a próxima? E quais impedimentos estão no caminho para você atingir seus compromissos? Requisitos são colocados “em espera” constante durante o *sprint* para prover alguma estabilidade num ambiente de mudança rápida (HIGHSMITH, 2002).

Os principais papéis são: o dono do produto, o time e o *Scrum Master*. O dono do produto representa os interesses dos que “suportam” o projeto e o resultado, e é o responsável por priorizar o *backlog* do produto para cada iteração. O time é responsável pelo desenvolvimento da funcionalidade. O *scrum master* é reponsável pelo processo, por orientar os desenvolvedores e adequar à cultura da organização.

Um artefato utilizado é o gráfico *burndown,* que mostra a quantidade de horas que faltam. É uma maneira fácil de visualizar correlação entre a quantia de trabalho que resta num ponto no tempo e o progresso do time do projeto.

Segue cenário comparativo entre os vários métodos ágeis, apresentado pela Figura 7.

| Nome do Método | Pontos Chaves | Características Especiais | Fraquezas Identificadas |
|---|---|---|---|
| ASD (*Adaptative Software Development*) | Cultura adaptativa, colaboração, componente dirigido por missão baseado em desenvolvimento iterativo | Organizações são vistas como sistemas adaptativos, criando uma ordem emergente de indivíduos interconectados fora da *web* | ASD é mais conceito e cultura do que prática de *software* |
| Crystal Methods | Família de métodos. Cada um tem valores primordiais e princípios. Técnicas, papéis, ferramentas e padrões variam. | Princípios de desenho em métodos. Habilidade em selecionar o método mais adequado baseado no tamanho e criticidade do projeto. | Muito cedo prá estimar: somente dois de quatro métodos sugeridos existem. |
| DSDM (*Dynamic Systems Development Model*) | Aplicação de controles para RAD, uso de "tempo estabelecido" (*timeboxing*), times com poder, | Primeiro método de desenvolvimento ágil de *software*, use de prototipagem, vários papéis de usuários: "embaixador", "visionário", e "consultor". | Enquanto o método está disponível, somente membros do consórcio têm acesso a *white papers.* |
| XP (*Extreme Programming*) | Desenvolvimento dirigido pelo cliente, times pequenos, *builds* diárias. | Refatoração - re-desenho constante do sistema para melhorar o desempenho e resposta a mudança | Enquanto práticas individuais são cabíveis para muitas situações, visão geral e práticas gerenciais recebem menos atenção |
| FDD (*Feature Driven Development*) | Processo de 5-passos, desenvolvimento baseado em componente orientado a objeto (i.e. funcionalidade). Iterações bem curtas: De horas a duas semanas | Simplicidade do método, desenha e implementa sistema por funcionalidade, modelo de objeto. | FDD foca somente no desenho e implmentação. Precisa de outras abordagens de suporte. |
| LD (*Lean Development)* | Reduzir tempo, custo, e defeitos de desenvolvimento de *software.* | Requer melhoria incremental, otimização global | Maiores melhorias requerem suporte da gerência sênior a fim de assegurar sucesso |
| Scrum | Time de desenvolvimento independente, pequeno, auto-organizado, ciclos de versão a cada 30 dias | Força uma mudança de paradigma de "definido e repetitível" para "nova visão de desenvolvimento de produto de Scrum" | Enquanto Scrum detalha em específico como gerenciar o ciclo de 30 dias, testes de integração e aceitação não são detalhados |

**Figura 7 –** Tabela Comparativa dos Métodos Ágeis

Fonte: Adaptado de Abrahamsson *et al*. (2006)

**Novas Medidas de Sucesso**

Uma grande questão a ser discutida é como medir o sucesso de um projeto, com a perspectiva ágil, já que no gerenciamento tradicional se valoriza o escopo e custo (esforço e prazo).

Segundo Morien (2005), times e organizações evoluem em "conformidade com o plano" para a "habilidade de responder a mudança". Essa transição envolve mover da tradicional estrutura de quebra por trababalho para o "foco de entrega de valor" implementando requisitos baseados na prioridade. Os *stage gates* são substituídos com medidas de sucesso baseados no código funcionando, testado e demonstrado. O plano é fluido e flexível; o "real" é o melhor que pode ser alcançado diante dos fatos ocorridos. Mais importante, o "real" é potencialmente despachado.

O desenvolvimento de *software* tem algumas especializações ou particularidades em relação à aplicação fim. A seguir, será descrito o desafio do desenvolvimento para dispositivo móvel.

### 2.2.3. *Software* para Dispositivo Móvel

Desenvolver *software* para um dispositivo móvel, telefones celulares e *PDA*s (*Personal Digital Assistants*) representa um desafio. Há uma forte tendência em se trocar a potência de processamento, capacidade de armazenamento e capacidade gráfica de um computador pessoal por um dispositivo móvel, onde tamanho, resposta instantânea e carga útil de bateria são limitantes. O objetivo é manter o desempenho e o manuseio (SALMRE, 2005).

Os dispositivos móveis têm telas menores, diferentes cores e resoluções, capacidade limitada de entrada de dados do usuário e capacidade de processamento.

Implementar aplicações para esses equipamentos não é somente portar de um computador pessoal, mas é desenvolver uma arquitetura diferente, para suprir limitações de desempenho e usabilidade.

As empresas de base tecnológica enfrentam o permanente desafio da necessidade de desenvolver novos produtos e tecnologias. A convergência das tecnologias e empresas da *Internet* e da telefonia móvel formam um mercado, ainda muito recente, mas que se encontra hoje em franca expansão em todo o mundo. Esse mercado, ainda não

consolidado, é caracterizado por constantes lançamentos de novos produtos em escala global e introduções de novas tecnologias.

Segundo Vainio  e Tuunanen e Abrahamsson (2005), existe uma lacuna nos atuais métodos de desenvolvimento dos sistemas de informação e *software* de engenharia para incorporar informação dos *stakeholders* que contribuem para um produto de sucesso no mercado.

No mercado móvel, a porção de ciclo de vida de produto durante os quais lucros podem ser ganhos tem se tornado progressivamente menor devido aos rápidos avanços tecnológicos. Assim, produtos de *software* móvel são mais e mais desenvolvidos como uma família, que ajuda uma empresa a reduzir o custo de desenvolver produtos individuais graças ao reuso de uma plataforma comum de produto. Isso significa que a coleção de elementos comuns, particularmente os elementos de tecnologia fundamentais é implementada por toda uma faixa de produtos. O que permite que funcionalidades específicas possam ser configuradas para produtos específicos (MACGRATH, 2001).

O benefício da abordagem de desenho de produto baseado em plataforma depende da habilidade da empresa em converter o esforço investido no desenvolvimento da plataforma na redução de custo de desenvolver variações individuais. Em adição aos investimentos, criar uma plataforma requer recursos para definir uma arquitetura apropriada para o produto, sobre a qual um conjunto de produtos pode ser desenvolvido e com a qual um mercado grande suficiente possa ser atingido.

Usuários são uma fonte crítica de inovação para fabricantes de produtos, que comprometem recursos consideráveis em relacionamentos de longo prazo (MACGRATH, 2001).

Produzir produtos de *software* bem-sucedidos para mercados de massa em celulares requer incorporar elementos de mercado no processo de desenvolvimento.

Os princípios estabelecidos de engenharia de *software* são ainda válidos para dispositivos móveis.  Os computadores de uso pessoal oferecem um ambiente rico para desenvolvimento de aplicação onde questões de desempenho e armazenamento não são tão críticos, já que recursos estão disponíveis ao desenvolvedor.

Entretanto, nos dispositivos móveis essas questões são cruciais, onde capacidade de processamento, tamanho da tela, capacidade de entrada de dados, podem produzir uma interface de usuário que frustra, dadas as limitações. O desenho do *software* de celulares requer capacitações especiais  (SALMRE, 2005).

Segundo Vainio  e Tuunanen e Abrahamsson (2005), a mais crítica diferença de aplicações em dispositivos móveis e nos *PC*s é como as pessoas os usam. No *PC*, pessoas gastam tempo "surfando" pela *Internet*, trabalhando em documentos, ou se comunicando (por *email*, mensagem instantânea, etc). Tendem a ser atividades longas, que têm um carater exploratório.  Com dispositivos móveis, tipicamente o usuário faz atividades rápidas, respondendo ou fazendo um pedido imediato de alguma outra pessoa ou processo. Quando o usuário faz uma chamada telefônica ou envia uma mensagem de texto, está interrompendo alguém.

Para ser útil, aplicações em celulares devem oferecer uma visão diferente do mesmo dado e processos que uma aplicação num *PC,* de tal forma que ofereça um acesso instantâneo aos elementos chaves que o usuário possa necessitar (SALMRE, 2005)

Ao desenvolver aplicações móveis, a necessidade de demonstrar claro retorno em investimento é tão grande quanto com qualquer outro tipo de desenvolvimento de *software*. Apesar disso, é maior o desafio para construir aplicações móveis e oferecer potencial para retorno em investimento de diversas maneiras (HAYES *et al.*, 2003).

Conforme Vainio  e Tuunanen e Abrahamsson (2005), o negócio de *software* para celulares está se tornando incrementalmente complexo, de evolução rápida,  entrelaçado devido a vários fóruns de indústrias, coalisões tecnológicas e parcerias. Isso tem um efeito no desenvolvimento de *software*, devido à quantidade e variedade de informação de *stakeholders* que contribuem para o aumento de sucesso de produto.

Nos mercados de dispositivos móveis e sem fio, é um desafio coletar, analisar, e processar a informação certa e integrá-la ao desenvolvimento de produto.

É necessário efetivamente integrar o conhecimento de várias fontes, tais como mercado, clientes, usuários, competidores e partes regulatórias, para produzir mais barato, e produtos com melhor funcionamento e usabilidade. Para atender a essa necessidade, a comunidade de negócio requer processos de desenvolvimento de *software* mais leves e ágeis que também levem em consideração o *business case* dos novos produtos.

Segundo Bolwijn *et al.* (1999), nas indústrias como a de "sem-fio" e "móvel", onde muitas companhias são capazes de produzir uma corrente contínua com uma larga variedade de produtos de alta qualidade com eficiência de custo,  elas precisam ser inovativas para serem bem-sucedidas. Eles caracterizam da seguinte maneira: "Na empresa inovativa, redução de custo, melhoria de qualidade e aumento de flexibilidade está tudo embutido na contínua busca por quebras de paradigmas em todas as áreas envolvidas: com o objetivo derradeiro de entregar produtos extraordinários em termos

de preço, qualidade e desempenho". Adicionalmente, adaptação a ambientes de mudanças se tornou uma competência estratégica para muitas oraganizações. Assim como uma organização precisa cuidadosamente alinhavar a infra-estrutura à sua própria realidade, usando as características mais apropriadas para as prioridades estratégicas e industriais.

É claro que o contexto em que o desenvolvimento de produtos e *software* para a indústria aqui apresentada se dá num contexto de trabalho em equipe muito forte.

#### 2.2.3.1. O trabalho em equipe

O ambiente de desenvolvimento de uma grande empresa de dispositivos móveis de uma empresa de telefonia, normalmente é grande, por ser um mercado que demanda grande variedade de produtos, novidades tecnológicas, além de ser um equipamento de utilidade, para prover comunicação entre as pessoas, baseada em (VAINIO e TUUNANEN e ABRAHAMSSON, 2005).

Os laboratórios de desenvolvimento estão distribuídos globalmente, onde é comum que os membros do time de desenvolvimento, clientes, representantes dos clientes, gerência de projetos e produtos, times de integração de produtos possam estar fisicamente distribuídos, em fusos horários possivelmente até com pouca sobreposição do horário comercial.

É importante enfatizar que os produtos têm seus *business cases* desenvolvidos para atingir diferentes faixas econômicas de usuários, em diferentes regiões e culturas do mundo (VAINIO e TUUNANEN e ABRAHAMSSON, 2005).

Portanto, o trabalho em equipe e a definição de papéis de cada time são fundamentais, e a coordenação sincronizada desse trabalho é essencial para que o resultado seja compatível com a demanda, com os requisitos estabelecidos pelas regiões, e que o mercado absorva e atenda aos anseios do usuário final.

A comunicação entre os times, tanto da coordenação para os executores assim como dos desenvolvedores com seus pares é básica para que haja convergência dos resultados.

A sincronização para que todos os times tenham e mantenham a mesma visão do produto a ser desenvolvido ocorre por meio de discussão de planos, acompanhamento e controle de cronograma e objetivos parciais e finais, reuniões periódicas ou sob demanda, avisos, alertas, relatórios e comunicados gerais enviados e distruídos para cada grupo que necessita daquela informação, assim como deve haver alguns níveis de

documentação elaboradas e mantidas consistentemente para garantir o acordado e o entendimento de todos os times envolvidos.

E cada time localmente deve também manter a consistência de prazos, escopo, esforço acordados com as diretivas globais.

### 2.2.4. Dispositivos Móveis no mundo e desenvolvimento de *software* no Brasil

Segundo revelou a consultoria iSuppli em novembro de 2006 (Nystedt, 2006), o número de assinantes de telefones móveis em todo mundo pulará de 2,6 bilhões de pessoas neste ano para 4 bilhões em 2010, graças ao desenvolvimento de aparelhos de baixo custo. Novos clientes em nações em desenvolvimento como Índia e China, África e Oriente Médio estão por trás do significativo aumento entre usuários de telefones celulares, disse o estudo, enquanto a indústria móvel afirmou que a responsável é a crescente penetração de aparelhos mais baratos.

Esse aumento tem relação com a crescente noção de que telefones celulares se tornaram direitos básicos para a população mundial.

A iniciativa de telefones de baixo custo começou em 2005 como uma forma de conectar nações usando redes já existentes. A Associação GSM (GSMA) formulou a idéia após perceber por meio de estudos que mais de um bilhão de usuários no mundo poderiam usar telefones celulares se pudessem pagar pelos aparelhos.

Conforme Vainio  e Tuunanen e Abrahamsson (2005), dispositivos celulares têm se transformado de meros telefones em computadores de mão capazes de uma variedade de tarefas, como suporte a emails, navegar pela Internet e transmitir vídeo. Consequentemente, os fabricantes têm focado na melhoria das capacidades de software de seus produtos, que significa adicionar mais suporte para o desenvolvimento de software e convencimento de fornecedores em desenvolver aplicações para eles.

Devido ao fato de preços de dispositivos móveis estarem rapidamente baixando 46% desde 1999, faz com que estes se tornem produtos de commodities ao invés de produtos exclusivamente de alta tecnologia. Claramente isso leva a um foco em custo, qualidade e desempenho, levando à necessidade de integrar o conhecimento a partir de várias fontes, como mercado, clientes, usuários, competidores, órgãos reguladores. A comunidade de negócios está requisitando processos de desenvolvimento de software

mais ágeis, que levem em consideração o business case de novos produtos (VAINIO e TUUNANEN e ABRAHAMSSON, 2005).

Segundo a Negócio Oportuno (2007), é meio natural portanto que a área de dispositivo móvel diminua o foco em hardware para centrar-se em software.

Essa mudança ocorre porque os fabricantes acreditam que está cada vez mais difícil diferenciar seus produtos em função do hardware. Assim, melhorias têm surgido ao se tirar vantagem com as novas capacidades de dados, que repaginaram o mercado de telefones celulares.

Os aparelhos celulares ganharam múltiplos gigabytes de armazenamento, mais telas funcionais, mais capacidades de potência de áudio e funções de vídeo, como visualizar video clips, e assim os dispositivos se tornaram mais próximos de um computador de uso pessoal (VAINIO e TUUNANEN e ABRAHAMSSON, 2005).

Software é o que sustenta essas capacidades. Então os fabricantes desses dispositivos precisam encontrar maneiras de disponibilizá-lo nos seus sistemas. E ainda tornar possível que terceiros possam desenvolver aplicações e integrá-las ao telefone, fazendo com que aplicações hoje basicamente disponíveis nos computadores de uso pessoal se tornem acessíveis.

#### 2.2.4.1. Panorama do desenvolvimento de *software* no Brasil

O Brasil possui um setor de software bem desenvolvido, com áreas de grande expertise, segundo dados da SOFTEX e do Massachussets Institute of Technology (MIT), o mercado de software era da ordem de US$ 7,7 bilhões em 2001. São características de nossa indústria de software, a dispersão pelo território nacional e um grande número de empresas, várias das quais micro e pequenas.

Segundo relatório do Ministério de Desenvolvimento, Indústria e Comércio Exterior (2005), dados apresentados pela revista Computerworld nos informam que já são mais de 20 os centros de desenvolvimento de software espalhados por todo o Brasil, numa intensa atividade da qual fazem parte software houses nacionais, instituições de ensino e pesquisa, incubadoras de empresas e grandes companhias multinacionais da área de tecnologia da informação. A produção, por enquanto, ainda é meio aleatória, incluindo desde software de banco de dados textual, sistemas de automação industrial, software de gestão empresarial, até jogos para aparelhos celulares.

Os investimentos feitos no Brasil pelas grandes multinacionais da indústria de informática e de telecomunicações vêm sendo feitos como contrapartida da indústria de tecnologia da informação aos incentivos fiscais que recebe por conta do Processo Produtivo Básico (PPB), previsto na Lei de Informática 11.077/2004, em vigor desde 1991 e renovada em 2004 até 2019, cujo propósito é estimular a produção local. Os softwares são incorporados aos produtos vendidos no Brasil e no exterior.

Segundo reportagem da Negócio Oportuno (2007), no Brasil surgiu um movimento na base industrial, em que multinacionais instalaram centros de competência em suas filiais.

Institutos de Pesquisas foram criados com foco em pesquisa e desenvolvimento (P&D) na área de Tecnologia da Informação e Comunicação no Brasil e na capacitação de recursos humanos para este mercado, dentro dos parâmetros da Lei de Informática.

A lei beneficia empresas que investem 4% do seu faturamento em P&D, com a redução da alíquota do IPI (Imposto sobre Produtos Industrializados), tornando seus produtos mais competitivos no mercado.

Mão-de-obra qualificada, flexibilidade para trabalhar em equipe e fuso horário compatível com o mercado norte-americano, o principal do mundo, são algumas das vantagens competitivas do Brasil.

Segundo Motta (2005), a Índia lidera e vai continuar a liderar o mercado de offshoring por muitos anos mas existe um movimento, ainda tímido, que conta com a participação do governo, de entidades representativas dos exportadores e produtores de software e de empresas para que o Brasil atinja um outro patamar no setor de tecnologia da informação. A Lei de Informática é citada como um dos atrativos para as grandes empresas criarem os centros de desenvolvimento de software no Brasil. Pela lei, a empresa tem benefícios fiscais com a produção local, mas deve investir cerca de 2,5% de sua receita bruta em pesquisa e desenvolvimento no país. Ele ressalta, no entanto, que sem mão-de-obra qualificada o empreendimento não se mantém. O Brasil possui algumas vantagens competitivas, que qualifica o país para a disputa por esse mercado. Em relação à qualidade dos softwares, a maioria está apta, ou até mesmo já recebeu, selos de excelência dos mais altos níveis, como ISO, CMM Certification (SEI CMM) e QSM methodology.

Somando-se às vantagens acima mencionadas, o Brasil possue outras só como posição geográfica, língua falada nos negócios, custos de produção e prestação de serviços.

Finalmente, a mão de obra é barata se comparada à européia e à americana, principalmente em decorrência da diferença no câmbio entre o real e o dólar/euro. Mas a flutuação do valor da moeda traz consigo uma sazonalidade e questionamentos sobre o custo final.

Fica claro que apesar de todas estas vantagens e, tirando o fato de que o governo deveria incentivar mais o setor, tanto negocial, fiscal e juridicamente, para transformá-lo em algo parecido com o que vemos atualmente na Irlanda e Índia, a questão é única e exclusivamente, que nosso mercado interno ainda é fragmentado (NEGÓCIO OPORTUNO, 2007).

A maioria de nossas empresas de tecnologia não possui o volume adequado para competir mundialmente, em igualdade de condições com empresas estrangeiras. Em grande escala, elas não têm como competir em logística, promoção, distribuição, produção e capitação de recursos baratos para financiamento de projetos.

Uma das consequências desta falta de concentração é que muitas vezes não se tem uma quantidade suficiente de pessoal em uma mesma empresa para pesquisar, desenvolver, promover, vender e prestar serviços aos softwares e produtos produzidos.

### 2.2.5. Digital Six Sigma

O conceito Six Sigma diz respeito a variação e capacidade de processo. Em alto nível significa eliminar variação removendo causas especiais e comuns, proporcionando melhoria na capacidade do processo (SMITH e FINGAR, 2003).

Na tentativa de melhorar o processo, devem-se identificar aquelas variáveis de processo com o maior impacto no desempenho do processo e trabalhar na sua otimização.

As implicações de *Six Sigma* na indústria são profundas. Por exemplo, em 1999, a empresa *General Electric Company* gastou mais de um bilhão de dólares em iniciativas *Six Sigma* e recebeu mais de dois bilhões em benefícios para o ano fiscal (PANDE et al., 2000). Enquanto *Six Sigma* tem obtido grande impacto na indústria, a comunidade acadêmica ainda está atrás da formalização do entendimento (LINDERMAN et al., 2003).

O nome *Six Sigma* é um conceito que foi originado pela Motorola Inc. nos Estados Unidos da América por volta de 1985. Naquele tempo, eles estavam com o desafio da competição japonesa na indústria de eletrônicos e precisava fazer drásticas melhorias no seu nível de qualidade (HARRY e SCHROEDER, 2000). O nome sugere um objetivo

(3.4 defeitos por milhão de oportunidades (DPMO)). Foi uma maneira da Motorola expressar seus objetivos de qualidade de 3.4 DPMO onde uma oportunidade de defeito é uma falha de processo que é crítico para o cliente.

De acordo com Linderman et al. (2003), nem todos processos devem operar no nível *Six Sigma*. O nível apropriado dependerá da importâmcia estratégica do processo e o custo da melhoria relativa ao benefício. Então, na tentativa de desenvolver os conceitos e principios, *Six Sigma* é definido como um método organizado e sistemático para melhoria de processo estratégico e desenvolvimento de novos produtos e serviços sustentados por métodos estatísticos e o método científico para fazer drásticas reduções nas taxas de defeito definidas pelo cliente. Essa definição destaca a importância de melhorias baseadas na definição do cliente sobre um defeito, não em considerações internas.

*Six Sigma* usa um método estruturado para endereçar os mais importantes problemas do negócio. As metodologias estruturadas são: *Define-Measure-Analyze-Improve-Control* (DMAIC) para resolução de problemas; *Define-Measure-Analyze-Design-Verify* (DMADV) para desenvolvimento de processo ou re-engenharia; Desenho para *Six Sigma* (DFSS) para desenvolvimento de novo produto. Seja qual for o método escolhido, é importante que o método seja cuidadosamente seguido e uma solução não seja oferecida até o problema ser claramente identificado.

A metodologia utilizada nessa pesquisa-ação segue uma sequência específica com o acrônimo DMAIC (*define, measure, analyze, improve and control*) e utiliza ferramentas existentes para executar cada passo da sequência.

Um importante conceito no controle do processo estatístico é de que todo fenômeno mensurável tem uma distribuição estatística. Isso significa que a saída do processo variará ao redor dos seus valores típicos.

O desvio padrão do parâmetro de interesse, sigma, é uma medida de variação.

Esse parâmetro mede o grau ao qual o processo, com seu nível corrente de variação, é capaz de satisfazer os requisitos do cliente. É definido como a menor diferença entre o limite de tolerância do cliente máximo e mínimo (upper or lower customer-tolerable limit) e a média do processo, dividido pelo desvio padrão do processo.

Após serem vistos os conceitos de manufatura enxuta, desenvolvimento enxuto de produto e de software, assim como o desafio de se desenvolver software para dispositivo móvel, será apresentada a experiência da empresa “Alvo” num projeto de transição para desenvolvimento enxuto.

# 3. Pesquisa-Ação: Proposta de Conjunto de Ações na Empresa “Alvo”

A empresa aqui denominada “Alvo” é uma empresa global líder em comunicação. Como muitas empresas hoje, a empresa “Alvo” tem o desafio de diminuir seus custos operacionais enquanto mantém a competividade em tecnologias de ponta para estar à frente em inovação. A empresa investe fortemente em pesquisa e desenvolvimento de produtos para sustentar seu posicionamento diante dos desafios do mercado.

No domínio dos dispositivos móveis, o desenvolvimento de produtos de *software* requer a incorporação de elementos de mercado ao processo de desenvolvimento a fim de ganhar uma vasta base de clientes para o produto.

Algumas características da indústria de desenvolvimento desses dispositivos mundialmente, assim como no Brasil podem ser vistas no Anexo 1, e a abordagem necessária para o desenvolvimento desses dispositivos, no capítulo anterior.

A seguir a empresa “Alvo” será caracterizada quanto ao processo de desenvolvimento de software, aplicado ao caso a ser analisado, assim como a análise dos dados encontrados.

A experiência a seguir detalhada diz respeito a uma das unidades da empresa, no Brasil.

## 3.1. Empresa “Alvo”

A seguir serão descritos os aspectos relevantes ao desenvolvimento de software na empresa “Alvo” e a transição para processos ágeis.

### 3.1.1. Ciclo de Vida do produto

Na empresa Alvo, o processo de desenvolvimento de novos produtos segue uma adaptação do modelo proposto por Cooper (2000), com cinco estágios representando os maiores eventos.

O time de desenvolvimento de *software* no Brasil é responsável pelo estágio 3 – ver Figura 8.

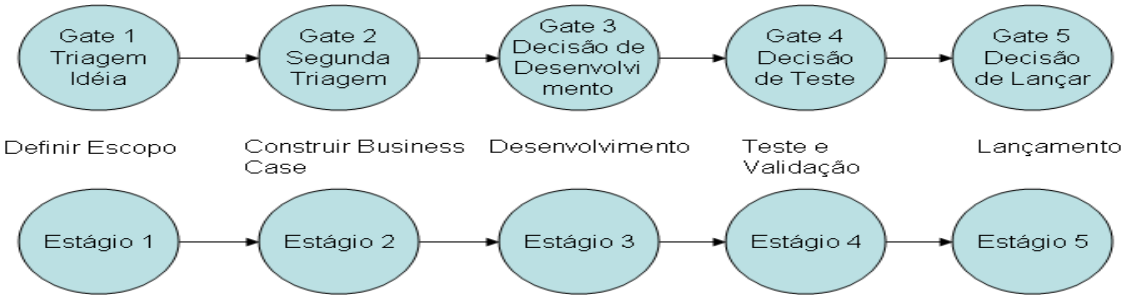


**Figura 8**- Modelo *Stage Gate*

Fonte: Modificado de Cooper (2000)

Os estágios são *cross*-funcionais. Cada estágio consiste de um conjunto de atividades paralelas empreendidas por pessoas de diferentes áreas funcionais na empresa. Os participantes do time do projeto executam tarefas chaves para ajuntar informação necessária para avançar o projeto para a novo *gate* ou ponto de decisão. Os *gates* entre os estágios servem de pontos de verificação de qualidade e controle. Para gerenciar riscos, via o método *Stage Gate*, as atividades num dado estágio devem ser desenhadas para agrupar informação – técnica, financeira, e relacionada a operações – a fim de mitigar os riscos técnicos e de negócios. Os *gates* também têm um formato comum que inclui entregas, critérios e saídas.

É necessário detalhar o ambiente global de desenvolvimento, realizado a seguir.

### 3.1.2. Dispositivos móveis

Um grande segmento do portifólio da empresa “Alvo” é o telefone celular, que é um equipamento de telefonia e comunicação de voz ao qual foram agregadas diversas aplicações de *software* ou *hardware* que interagem entre si, e que o tornam cada vez mais um computador de mão, tanto em termos de capacidade de processamento e armazenamento quanto em funcionalidades.

**Estrutura funcional - componente**

O telefone celular (Figura 8) é tratado como um produto composto por vários sub-produtos embarcados. Esses sub-produtos são na verdade aplicações completas, mapeadas em componentes. Portanto, o telefone celular é um sistema composto por

componentes, *hardware* ou *software*.


**Figura 9**- Exemplo de Produto Celular e cada componente como produto

Fonte: adaptada de

http://papotech.com.br/blog/wp-content/uploads/2007/01/indexhero20070109.jpg

Os componentes são desenvolvidos de forma distribuída globalmente nos diversos centros de excelência em todos os continentes.

A um time no Brasil foi atribuída a responsabilidade de prospectar, desenvolver e evoluir um desses componentes e suas dependências sistêmicas.

As demandas do mercado considerando as diversas regiões do mundo são mapeadas em funcionalidades para os vários sistemas.

Após serem definidos e discutidos os requisitos de todas as funcionalidades a serem desenvolvidas e integradas, o grupo de desenvolvimento do componente é organizado em estrutura de projetos, seguindo o ciclo de vida de produto, desde a definição de requisitos até a aceitação pelos clientes.

Há uma estrutura de especialistas, gerentes de desenvolvimento e de projeto que implementa essas novas funcionalidades.

Os produtos são desenvolvidos sobre “plataformas” que simplificando quer dizer estruturas básicas de *hardware* e *software* que sustentam “famílias” de produtos, com diferentes configurações e aplicações.

Para isso, o reuso de *software* deve ser um dos objetivos finais. O componente deve ter uma arquitetura flexível que propicie portabilidade de funcionalidades entre os diversos telefones de diferentes tipos, modelos, plataformas e entre as várias tecnologias de transmissão celular (CDMA, GSM ou 3G).

O sistema deve possuir um poderoso processo de gerenciamento de requisitos e de configuração/customização de *software* que permita que a interface do usuário possa ser única para cada produto, de cada tecnologia, para cada operadora de telefonia móvel: língua nativa da região para comandos, alertas, respostas a comandos, sons e ícones.

A interoperabilidade deve ser garantida. Para cada operadora alvo, há validação da respectiva configuração, assim como dos protocolos de comunicação definidos pelos órgãos normatizadores.

Tanto a verificação funcional do componente de *software* quanto testes de robustez devem ser executadas e validadas.

Após a integração das várias funcionalidades, o componente passa por uma seqüência de fases de teste de validação junto a cada região/operadora-alvo, focando nos principais aparelhos da plataforma. Esta fase é denominada fase de testes de interoperabilidade, que segue até a aceitação pelo cliente.

É claro que com um mercado competitivo, ávido por "novidades" tecnológicas, diferenciadas funcionalidades e estéticas de produto, os executivos têm estabelecido objetivos agressivos de velocidade ao mercado demandando uma significativa diminuição de tempo de desenvolvimento de produtos e projetos, além de manter e aprimorar a qualidade e inovação.

## 3.2. Motivadores da Mudança

Acelerar o passo de entrega de *software* se tornou foco em resposta ao ambiente competitivo. Projetos se tornaram maiores em tamanho num esforço de atender às funcionalidades priorizadas pelo mercado.

O ambiente competitivo não se restringe somente a uma região ou a concorrentes de outras empresas, mas também na comparação com outros *sites* da própria empresa, onde possa haver qualificação técnica e custos menores associados ao processo de desenvolvimento de *software*.

O time do Brasil que desenvolve um componente integrado em plataformas de produtos de telefones celulares vem desenvolvendo, aplicando e colhendo frutos desde 2003 de

várias iniciativas de melhoria de qualidade, em aprimoramento de processo de desenvolvimento, gerenciamento de projeto e contenção de defeitos.

A partir do segundo semestre de 2007 a gerência executiva estabeleceu a prioridade de melhoria na produtividade, mantendo os resultados obtidos nas iniciativas anteriores, em termos de qualidade e custo.

O relato a seguir tem como objetivo apresentar propostas, identificar possíveis dificuldades para a execução de um piloto, tomando-se como base os conceitos do pensamento enxuto.

Assim como numa empresa de manufatura onde as mudanças não poderiam acontecer somente no chão de fábrica, no caso da empresa "Alvo", as mudanças não poderiam ocorrer somente no desenvolvimento de *software*. Assim, outros departamentos envolvidos como *marketing* técnico, que define os requisitos a serem desenvolvidos, vendas, engenharia de sistemas, gerenciamento de projetos também deveriam sofrer alterações.

O escopo desse trabalho portanto está restrito a mudanças sugeridas na área de desenvolvimento de *software* (estágio 3 – Stage Gate) de um componente de *software* desenvolvido no Brasil, que passa a ser denominado componente "alvo".

## 3.3. Processo de mudança para Enxuto

Os princípios enxutos provaram seu valor além do ambiente de manufatura, e também é reconhecido como um método que produz melhoria em velocidade e resultados de qualidade enquanto abaixa os custos. Entretanto, somente quando Poppendieck e Poppendieck (2003) desenvolveram os conceitos de Desenvolvimento Enxuto de *Software* que a conexão entre "Enxuto" e "Ágil" pôde ser conceituada e realizada.

A grande questão a ser respondida para a empresa "Alvo" é: "Como podemos atender a mais pedidos dos clientes, com a mesma quantidade de recursos e melhorando a qualidade de nosso produto?"

Para endereçar a questão de uma aceleração na entrega, ou melhoria da produtividade, um projeto foi iniciado, utilizando método de processo de melhoria *Digital Six Sigma (DSS) DMAIC* (*Define, Measure, Analyze, Improve and Control*) (SMITH e FINGAR, 2003). Desse modo, a condução e resultados do projeto poderiam ser acompanhados detalhadamente pela gerência e responsáveis.

## 3.4. O projeto de melhoria de produtividade

Segue descrição dos principais pontos do projeto de melhoria de produtividade, são discutidos.

### 1. Metodologia utilizada para o projeto de melhoria da produtividade

A empresa “Alvo” se utliza da metodologia DSS DMAIC, para alavancar processos de melhoria, possuindo treinamentos e estrutura mundial de suporte, acompanhamento e controle dos projetos.

O projeto é proposto para uma organizaçào mundial de qualidade, que avalia o escopo, prazo, custos e qual o retorno de investimento previsto. Dessa forma, quando aprovado, a estrutura mundial passa a acompanhar as fases do projeto e seu retorno financeiro para a organização.

O projeto foi aprovado em agosto/2007 e o seu final está previsto para junho/2008.

### 2. Equipe montada para desenvolver projeto de melhoria da produtividade

O projeto foi organizado com os seguintes participantes: um *Black Belt*, um coordenador (candidato a *Green Belt*), três gerentes de desenvolvimento de *software*, um gerente de projeto e três desenvolvedores de *software*.

O pesquisador faz parte dessa equipe como um dos gerentes de desenvolvimento de *software*.

### 3. Objetivo do projeto de melhoria

Implantar transição de processos e ações com o fim de aumentar em 20% a produtividade média, em número de linhas desenvolvidas por hora para todas as funcionalidades desenvolvidas pelo componente “alvo”.

Sob a perspectiva de Enxuto, num processo ágil, parece estranho se pensar que produtividade seja medida considerando a quantidade de linhas de código implementada durante o desenvolvimento, pois o foco é entregar valor para o cliente. Não necessariamente esses parâmetros são diretamente proporcionais. Mas, no projeto, por necessidade de avaliação quantitativa, será considerada produtividade o aumento de 20% na quantidade de linhas de código geradas.

### 4. Escopo das ações a serem propostas

O escopo deveria ficar restrito ao desenvolvimento de novas funcionalidades, do

momento em que os requisitos são definidos e acordados com a plataforma do produto, até o momento em que o *software* daquela funcionalidade tiver sido desenvolvido, testado funcionalmente e integrado à plataforma do produto do dispositivo móvel.

## 5. Cronograma

| **Fase** | **Início** | **Fim** |
|---|---|---|
| *Define* | Ago/2007 | Set/2007 |
| *Measure* | Out/2007 | Nov/2007 |
| *Analyze* | Dez/2007 | Dez//2007 |
| *Improve* | Jan/2008 | Jun/2008 |
| *Control* | Jul/2008 | Ago/2008 |

## 6. Base de dados histórica dos desenvolvimentos já realizados

A equipe do DSS DMAIC tinha como entrada no seu processo de investigação uma base de dados histórica bastante completa, para todas funcionalidades de *software* desenvolvidas ao longo dos últimos 3 anos no componente "alvo", com as seguintes informações:

- identificação da funcionalidade, desenvolvedores de *software*, líder técnico, gerente de projeto, engenheiros de sistema, tamanho em quantidade de linhas de código (estimado e real), prazo (estimado e real), esforço (estimado e real), percentual de tempo e esforço gastos em cada fase do ciclo de vida do desenvolvimento, além de documentos que descrevem o escopo, dependências de outros componentes, problemas encontrados, riscos gerenciados. Os dados relacionados a tempo são realizados em horas;
- outro dado importante mantido é a referência da versão do processo de desenvolvimento de software utlizada.

O processo de atualização dessa base é assim definido:

- A partir do momento que os requisitos são estabelecidos, estimativas, planejamento e alocação de recursos são realizadas, essa é a primeira atualização realizada nessa base de dados para aquela funcionalidade;
- Semanalmente, essa base é atualizada com os dados relacionados ao "realizado". Dados relacionados ao andamento do projeto são atulizados pelos gerentes de projeto, e cada desenvolvedor atualiza a quantidade de horas

trabalhadas nas atividades/tarefas a ele alocadas no cronograma.

- Correlacionada a essa base de dados, para cada funcionalidade são armazenados todos os casos de teste executados, assim como todas as falhas encontradas e em que fase do teste (do funcional até o teste do campo), assim como a causa raiz do problema.

Foi selecionado um conjunto de 61 funcionalidades por terem sido funcionalidades desenvolvidas seguindo o mesmo processo de desenvolvimento.

## 7. Hipóteses iniciais do projeto *DSS DMAIC*

Foi realizado um levantamento na base de dados históricos de um conjunto de 61 funcionalidades desenvolvidas no último ano, que haviam seguido o mesmo processo de desenvolvimento, no estágio 3 do *Stage-Gate* (ou seja, do desenvolvimento dos requisitos à entrega do *software* para testes do componente integrados ao produto) .

A primeira observação foi de que quanto maior a funcionalidade em termos de quantidade de linhas de código, a tendência era se ter uma maior quantidade de linhas desenvolvidas por hora. Mas uma análise detalhada precisava ser feita (Figura 9).

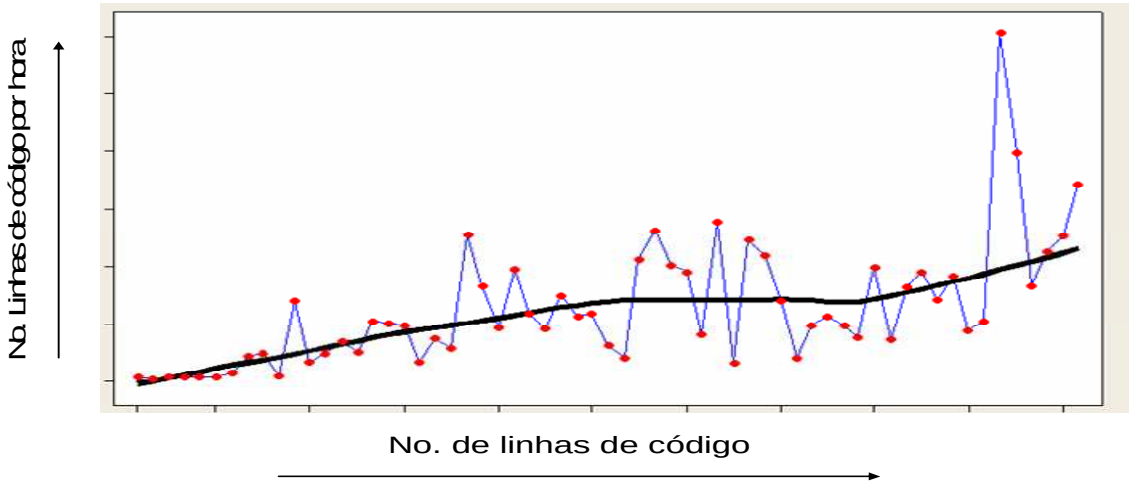


**Figura 10**- Dados históricos de produtividade de linhas de código por hora

As seguintes perguntas fariam parte da investigação do projeto:

- qual o peso do processo no desenvolvimento das funcionalidades?
- no processo, que parece “pesado”, o que gera valor, o que é necessário porque a organização de produto pede, o que é feito que poderia ser descartado?
- nas atividades que geram valor, existe uma maneira melhor e mais eficiente para

executá-las?

- a produtividade está relacionada diretamente ao conhecimento que o desenvolvedor tem do assunto?

**8. Execução do projeto de melhoria**

Todas as informações obtidas, discutidas pelo time do projeto, assim como acompanhamento e apresentação aos reponsáveis, são armazenados na Intranet da empresa "Alvo".

Todo o planejamento e execução das tarefas, e decisões quanto às ações a serem tomadas estão a cargo da equipe, inclusive de envolver outros, segundo necessidade.

Reuniões semanais de acompanhamento foram executadas.

### 3.4.1. Fase *Define* – DSS DMAIC – Agosto/2007 a Setembro/2007

A equipe do projeto decidiu que se teriam quatro frentes para a aplicação de princípios Enxutos, que serão descritas a seguir:

- Adequações realizadas para se transicionar para processos ágeis
- Pesquisa junto aos desenvolvedores de *software* para se levantarem "ganhos imediatos"
- Estabelecer base de referência para avaliação
- Proposta de piloto para processos ágeis - transição de um método de desenvolvimento em cascata para um composto de Enxuto e Ágil.

Detalha-se a seguir cada etapa proposta para o processo de transição.

#### 3.4.1.1. Adequações realizadas para transicionar para processos Ágeis

Era necessário investigar quais teorias dariam sustentação às mudanças esperadas, e também pontuar alguns aspectos importantes quando se trata de introduzir processos ágeis no desenvolvimento de um produto de *software*, que é embarcado em outro produto.

**1. Situação global de processos ágeis na empresa "Alvo"**

Foi necessário primeiro se estabelecer o que seria possível e o que não seria possível ser alterado no processo global de desenvolvimento de *software* da empresa.

Na organização de dispositivos móveis, a iniciativa de se usarem processos ágeis num

desenvolvimento de *software* tinha o aspecto de pioneirismo.

Porém em organizações de desenvolvimento de *software* para infra-estrutura de rede já havia sido experimentado, cada um com diferentes abordagens.

- O time de DSS DMAIC consultou as organizações globais, através de reuniões, audio-conferências, *emails* para alinhamento de qual era o posicionamento global da empresa em relação a processos ágeis.
  - trabalhos na área de processos ágeis haviam sido escritos sobre experiências pilotos (Lindvall *et al*., 2004), principalmente utilizando XP (Extreme Programming);
  - O aspecto mais importante "aprendido" e o maior desafio na experiência dessas outras organizações foi o de que adotar práticas ágeis envolve integrar cada piloto com os processos existentes no ambiente do projeto;
  - Portanto, adequações teriam que ser feitas, seja qual fosse o processo utilizado.
- A organização que define processos e que provê a garantia de qualidade foi consultada do que seria aceito ou não.

Era preciso capacitar alguns membros do projeto mais adeaquadamente sobre como seria um processo de desenvolvimento ágil:

- participação de membro da equipe em *workshop* mundial da empresa "Alvo" sobre a transição para processos ágeis;
- Leitura, reuniões de discussão e participação de treinamentos em universidade do Brasil sobre o assunto.

A recomendação da empresa foi a de se usar *Scrum*, e a equipe do projeto DSS DMAIC considerou também que esse método seria o mais apropriado dado que as adequações seriam factíveis e era o processo que mais se "aproxima" de gerência de projeto.

**2. Adequações realizadas**

Os principais aspectos do processo de desenvolvimento de *software* da empresa "Alvo" que precisam ser "costurados" para o uso de processos ágeis estão apresentados na figura 10:

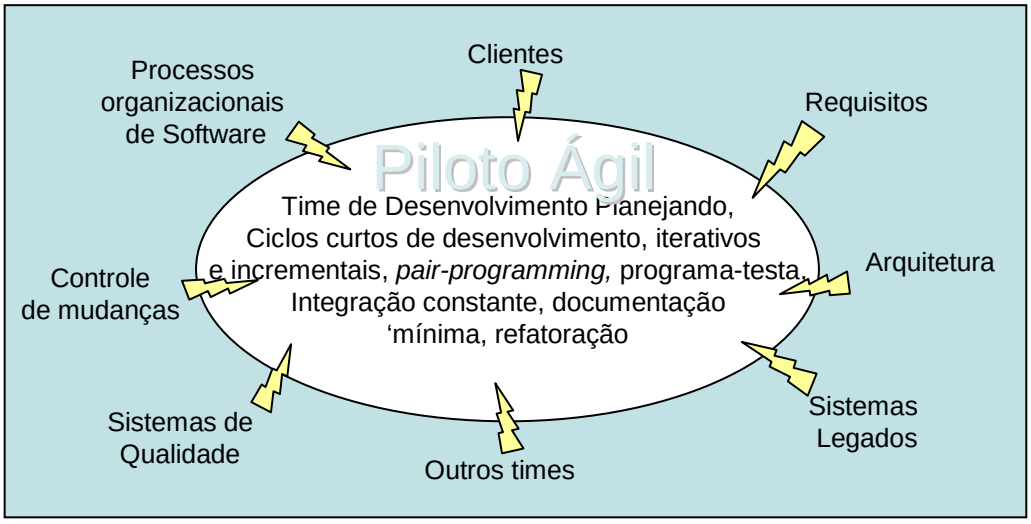


**Figura 11:** Adequando *Scrum* à organização de dispositivos móveis
Legenda: os "raios" mostram as incompatibilidades entre o piloto *Scrum* e o ambiente
Fonte: Modificado de Lindvall *et al*. (2004)

A seguir são detalhadas as adequações feitas para cada área chave que houve incompatibilidade entre o Scrum e o ambiente de desenvolvimento da empresa "Alvo":

1. Clientes e Requisitos
2. Arquitetura
3. Sistemas legados
4. Outros times
5. Sistemas de qualidade
6. Controle de mudanças
7. Processos organizacionais de *software*

### 2.1. Clientes e Requisitos

Na metodologia ágil, a negociação de requisitos com o cliente, mesmo ao longo do ciclo de desenvolvimento do produto é natural. Ou seja, mantém-se a data de entrega, mas aceita-se alterar o escopo. Na empresa "Alvo", esse é um valor que não é admissível.
Esse valor de colaboração do cliente sobre negociação de contrato originou-se na consultoria de Tecnologia de Informação (TI) onde predominam aplicações de negócios baseados em *web.*
Como se trata de um produto de mercado de massa, os requisitos são definidos por

grupos técnicos de *marketing*, que são a ponte de ligação entre as demandas dos clientes mundiais com o grupo de desenvolvimento. Os clientes estão pulverizados nas organizações mundiais. Com isso, a prioridade de entrega de requisitos durante o desenvolvimento, se torna praticamente impossível.

Desenvolvimento de sistema de larga escala possuem muitas camadas entre os desenvolvedores e os clientes, que são: cliente ⇔ times de campos ⇔ pessoal de *marketing* ⇔ times de negócios ⇔ pessoal de engenharia (desenho) de sistemas ⇔ arquitetos ⇔ desenvolvedores.

A presença de um "cliente" para definir prioridades, esclarecer conflitos ou até mesmo de exluir requisitos, não existirá no dia-a-dia, como previsto como um benefício dos processos ágeis.

Portanto, para a organização do componente "alvo", os requisitos quando acordados no início do desenvolvimento se tornam todos com a mesma prioridade e obrigatórios de serem entregues.

## 2.2. Arquitetura de *software* embarcado

*Software* embarcado é a parte de um produto que é esperado mudar. Se não precisa mudar, então poderia ser *hardware*. *Software* trata os trabalhos de interação com o usuário. Por isso, o *software* tem que ser "arquitetado" para ser fácil mudá-lo (BOEHM e TURNER, 2005).

À medida que o tempo passa, modificar *software* de produção, ou seja, acrescentar funcionalidades a um produto já lançado no mercado, tende a se tornar mais difícil e caro.

*Software* embarcado tem aspectos bastante intrinsecos quanto à sua integração ao produto: com outros componentes, sistema operacional, arquitetura do *software* apoiado na infra-estrutura de camadas de comunicação com a rede, interface gráfica de cada produto.

Para entrega de desenvolvimento do componente entende-se integração desse "pacote" à linha de desenvolvimento do *software* do telefone celular.

O componente não tem "vida própria". Somente quando embarcado e integrado ao produto é que ele atingiu o objetivo.

O desenvolvimento das funcionalidades no componente "alvo" portanto teria que respeitar as interfaces com outros componentes já existentes ou negociar explicitamente

qualquer alteração que os afetasse.

## 2.3. Sistemas Legados

De uma forma regular, para cada produto (dispositivo móvel) novo, raramente os produtos embarcados começam do zero. Sempre partem de uma base já desenvolvida. Portanto, em qualquer desenvolvimento, deve-se considerar o custo de manter o legado íntegro, onde todos os testes contínuos e de regressão, propostos pelos processos ágeis, devem incorporar e validar os casos de uso da parte do legado que estão sendo alterados.

O processo regular de novos produtos embarcados se trata de agregar novas funcionalidades ao produto anterior. Ou seja, trata-se de uma evolução incremental.

Portanto, ao se desenvolverem novas funcionalidades no componente "alvo", também deve-se garantir que o legado continua funcionando conforme especificações. Isso faz com que a avaliação futura de qualquer iniciativa enxuta, em termos de qualidade, vai contar também, a qualidade embutida no legado.

## 2.4. Outros times

Os requisitos são agrupados em funcionalidades, que podem ter que ser implementados *cross*-componentes, portanto *cross*-times. Isso significa que algumas estratégias e acordos, e definições de interface devem ser resolvidas no início do desenvolvimento.

Ou seja, a proposta dos processos ágeis de se ter a decisão do que será trabalhado somente em alguma iteração, e tem relação com outros componentes, não é viável, já que outros times disparam suas atividades no paralelo e com dependências mútuas.

Todas as interfaces têm que ser resolvidas no tempo definido pelo ciclo de desenvolvimento do produto como um todo (no caso, na fase de definição, cascata).

## 2.5. Sistemas de Qualidade

O Sistema de Qualidade define metas e métricas que o novo desenvolvimento deve fornecer os dados pedidos, ou os adequá-los semanticamente ou deve pedir *"abono"*.

No caso, o grupo do DSS DMAIC propôs incialmente que inspeções de código fossem abonadas para o processo ágil, já que a proposta do processo é fazer e testar, e testar

com regressão. Mas a organização de qualidade não aceitou e então houve adequação nas tarefas para que as inspeções fossem mantidas.
Todos os dados e métricas pedidas pela plataforma foram proporcionados.

## 2.6. Controle de Mudanças

O time de desenvolvimento em posse dos requisitos estima tamanho e esforço de implementação e se compromete com uma data chave de "entrega". Definido e acordado o escopo, e esforço estimado, inicia-se todo o planejamento, que se reflete num cronograma detalhado.

- Recursos são alocados baseados na estimativa realizada no momento que se "trava" o escopo.
- Todo o processo de desenvolvimento é tratado como um projeto, que segue as recomendações do PMBOK-2004.
- O acompanhamento das atividades é feito por meio de indicadores requeridos pela plataforma de desenvolvimento, assim como internamente ao componente é feito o acompanhamento segundo indicadores que fazem a relação do planejado com o real, *cost and size performance indicators* (PMBOK, 2004).

Portanto, a plataforma de produto gerencia desde a definição dos requisitos específicos de cada componente e mudanças só ocorrem caso a gerência de produto assim o aceitar.
No caso, o acordado foi que todos os requisitos acordados no início do desenvolvimento seriam entregues, conforme data definida e acordada com a plataforma de produto.

## 2.7. Processos globais de desenvolvimento de *software*

Um aspecto que é bastante evidente na empresa "Alvo" é que por se tratar de uma empresa global, e com processos fortemente definidos e com fino controle de ritmo de produtos, dados e métricas armazenados, qualquer proposta que extrapole o domínio do componente ou de qualquer fator decisório externo será muito difícil de ser implementada ou aprovada. Ou seja, a equipe definiu que só seria proposto o que fosse aplicável para que as interfaces para integração com o produto fossem mantidas.
O processo de desenvolvimento atual do componente e da organização dos produtos segue a metodologia em cascata, devido a abordagem de fases lineares.
Se o desenvolvimento do componente respeita o escopo, datas de entrega e recursos, e

fornece os dados para acompanhamento do desenvolvimento, a organização global não interfere no modelo de processo sendo utilizado.

**3. Decisão pelo *Scrum***

Foi necessário determinar quais práticas ágeis poderiam oferecer a maior ajuda para os projetos, e verificar se a organização estava madura o suficiente para adotar um ciclo de vida iterativo. Nesse ponto, é preciso que o processo atual esteja estável.

As práticas ágeis mais publicadas, discutidas e utilizadas são o *Extreme Programming* (*XP*) e *Scrum.*

Dado que mundialmente, em outras organizações da empresa "Alvo", o *Scrum* era a metodologia ágil que tem aspectos fortes de gerenciamento de projeto e era a recomendada, e com mais suporte, ela foi selecionada, porém teve que ter manter os alinhavamentos citados anteriormente, para que pudesse ser utilizada:

- Liderança Técnica Forte – planejamento de iterações requer boa decomposição técnica;
- Alguma variação nas capacitações do time – flexibilidade dos recursos;
- Metodologia orientada a objetos;
- Ciclo de vida iterativo, com rápido *feedback* e aprendizado em ciclos curtos (~30 dias);
- Ciclo de desenvolvimento incremental, com "sub-funcionalidades" prontas a cada iteração;
- Qualidade é validada continuamente durante o desenvolvimento, por meio de testes de regressão;
- Integração de código contínua;
- Reuniões diárias de acompanhamento das atividades;
- Reuniões semanais para acompanhamento do projeto.

### 3.4.2. Fase de *Measure* – Outubro e Novembro de 2007

Nessa fase do projeto, foram executadas as ações planejadas para dar sustentação às mudanças.

#### 3.4.2.1. Pesquisa junto aos desenvolvedores de software para se levantarem “ganhos imediatos”

Foi elaborada uma pesquisa para que todos os desenvolvedores de software e gerentes de projeto da organização do componente “alvo”, de forma voluntária, pudessem opinar e sugerir melhorias para aspectos já observados ou já experimentados pelos próprios desenvolvedores.

Essa pesquisa tem aspecto relevante da filosofia Enxuta, que atribui grande valor ao conhecimento proveniente do colaborador direto do processo.

A pesquisa foi divulgada via email para todo o time (universo de 100 desenvolvedores, incluindo os times de garantia de qualidade e de gerenciamento de integração), e disponibilizada na Intranet, durante 10 dias.

A instrução era de que o desenvolvedor deveria dar sugestão (ões) de ganho(s) imediato(s) para o desenvolvimento de *software* do componente “alvo”. E explicitava o que “ganho imediato” queria dizer:

- Fácil de implementar
- Rápida e barata de implementar (pouco esforço)
- Implementação está sob o controle do nosso time
- Trará benefícios para nossa produtividade

Foram apresentadas: 96 sugestões, 30 respondentes.

As sugestões foram analisadas pelo time de DSS DMAIC, uma a uma, e identificadas a qual fase do processo de desenvolvimento impactava, conforme figura 11.

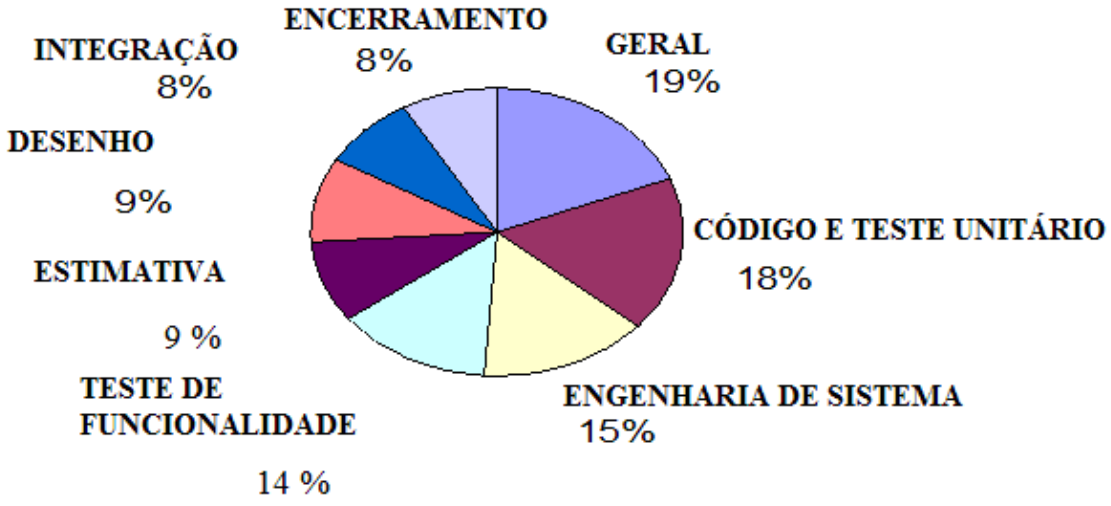


**Figura 12 –** Proporção de sugestões de “ganhos imediatos” por fase do ciclo de desenvolvimento

As sugestões foram classificadas em 4 classes: não aplicável; já implantada – verificar se implantação foi adequada; é sugestão de melhoria, mas não é “ganho imediato”; ou é “ganho imediato”.

Os itens entendidos como já implantados que só precisaria verificar a implantação, foram encaminhados para os responsáveis de cada área. Assim como sugestões pertinentes, mas que não se tratavam de “ganho imediato”.

Os “ganhos imediatos” foram assim encaminhados:

- 5 itens para gerência de projeto
- 2 para fase de testes de funcionalidades
- 10 para a fase de arquitetura e engenharia de sistemas
- 7 para a fase de estimativa de tamanho e esforço do projeto
- 5 para *cross*-fases

Desses “ganhos imediatos”, os que se aproximaram de um processo “ágil” foram:

- Evitar ter somente um desenvolvedor alocado para o desenvolvimento de uma funcionalidade;
- Usar “página wiki” (wikipedia) para dados da funcionalidade em desenvolvimento – melhorar a comunicação;
- melhorar definição de papéis entre gerente de projeto e liderança técnica;
- simplificar a fase de desenho, indo para o código “o quanto antes”, integrando as fases;
- desenvolvedores instalados proximamente quando trabalhando no desenvolvimento da mesma funcionalidade (*war room).*

#### 3.4.2.2. Estabelecer base de referência para avaliação

Para todo programa em implantação, é necessário se estabelecerem dados e indicadores que possam ser usados como referência para avaliação.

Com a base histórica bastante completa dos desenvolvimentos de *software*, aspectos relevantes ao desenvolvimento capturados na retrospectiva do projeto foram avaliados. Alguns indicadores foram definidos para serem medidos previamente à implantação e seriam capturados e calculados após a experiência piloto.

**1. Tamanho das funcionalidades desenvolvidas**

Pela base histórica, o time definiu quais as dimensões dos projetos para que esses

pudessem ser agrupados e analisados num conjunto de amostras mais significativas para que não houvesse distorção na análise dos resultados. Funcionalidades muito pequenas foram desprezadas. Portanto, foram consideradas 54 funcionalidades desenvolvidas:

- 19 com < 150 linhas de código, tamanho pequeno;
- 15, com 150 >= linhas de código < 1000, tamanho médio;
- 20, com >= 1000 linhas de código, tamanho grande.

## 2. Base de referência de Produtividade

Foi feita a linha de base de produtividade por tamanho do *software* desenvolvido

- Calculou-se a quantidade de linhas de código desenvolvidas por hora, desde o momento em que se acorda o desenvolvimento, com a definição dos requisitos, até a entrega da funcionalidade, pronta para ser integrada ao produto, para testes do produto, incluindo todas as etapas do processo.
- Foram consideradas as 54 funcionalidades citadas anteriormente.
- Feito o cálculo da média de quantidades de linhas de código produzidas por hora, com intervalo de confiança de 95%, cujo resultado está exposto na figura 13, apresentadas por tamanho total da funcionalidade. Pôde-se ver que o tamanho da funcionalidade influencia a produtividade: funcionalidades grandes possuem uma maior produtividade.

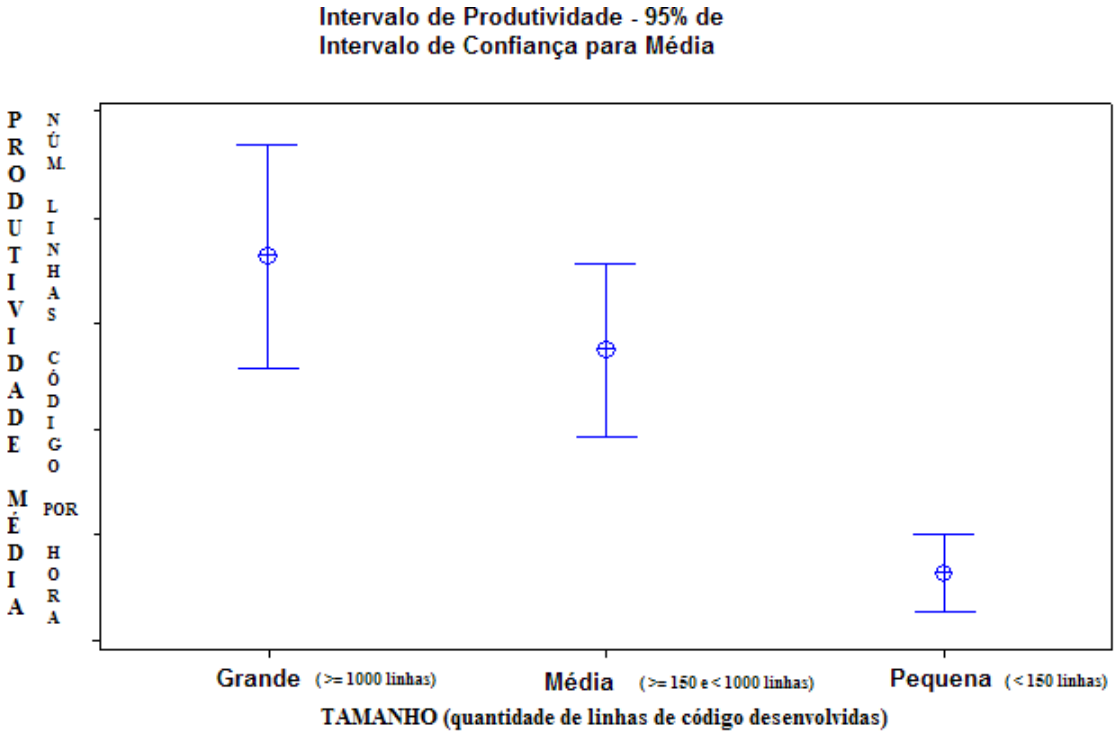


Figura 13 - Comparação da produtividade por tamanho das funcionalidades

3. **Comparação da produtividade do componente “alvo”com a indústria**

O componente “Alvo” trata-se de um sistema software embarcado, na área de telecomunicações. A figura 14 apresenta um *benchmark* de produtividade de desenvolvimento de *software* (quantidade de linhas de código produzidas por hora) entre a empresa “Alvo” e outras empresas: da área de Telecom, empresas que produzem *software* embarcado, adapatado de McConnell (2006).

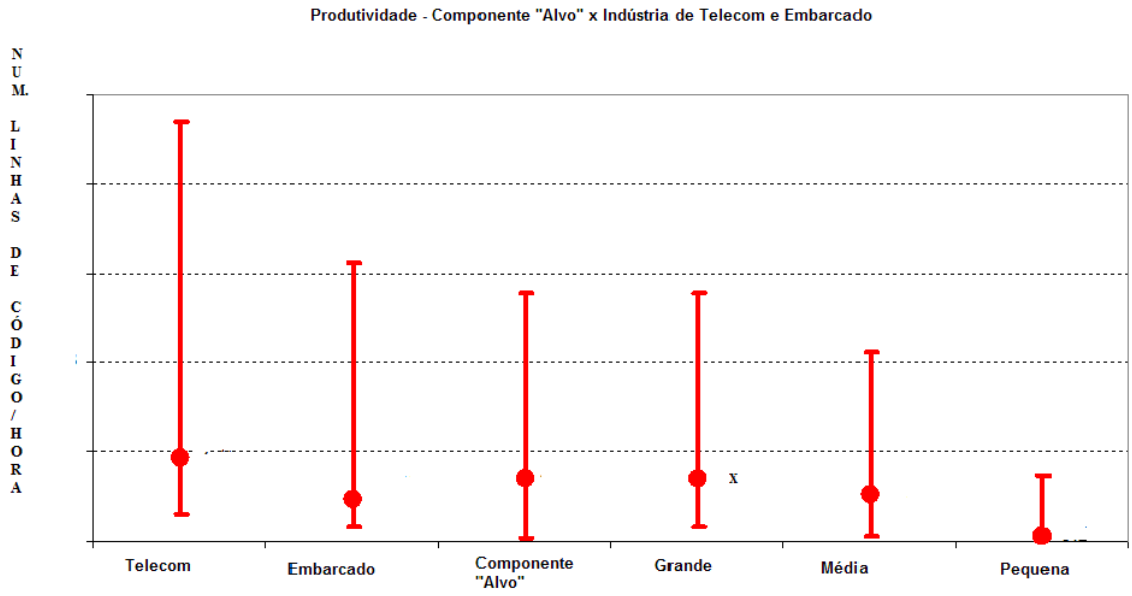


**Figura 14 -** Adaptação da tabela 5-2 Productivity Rates for Common Project Types
Fonte: Adaptado de McConnell (2006)

### 3.4.2.3. Fase de Analysis – Dezembro de 2007

Nessa fase, os dados capturados são analisados e define-se a estratégia de implantação das ações.

1. **Possíveis causas que possam ter afetado a produtividade**

A equipe tentou identificar possíveis causas que pudessem ter afetado a produtividade do desenvolvimento: complexidade do projeto, experiência dos desenvolvedores, indefinição dos requisitos ou mudança de requisitos ao longo do ciclo, desenvolvedores envolvidos em mais de um projeto paralelamente, dependência de outros componentes do produto, identificar desperdícios.

- Para complexidade da funcionalidade, não há dados suficientes na base de dados histórica para se ter uma referência para comparação. *Software* embarcado tem

características que o tornam inerentemente mais complexos, dependentes do ambiente em que estão inseridos e da quantidade de interfaces externas.

- A Figura 15
- O que se pôde notar é que desenvolvedores com baixa experiência possuem baixa produtividade. O que surpreendeu nos dados históricos é que desenvolvedores com maior experiência possuem menor produtividade (ver figura 15). Porém, aos desenvolvedores de mais experiência, usualmente são atribuídas funções “além código”, como liderança de time, inspeções dos códigos gerados, negociar interfaces. O que afeta a quantidade de linhas de código geradas.

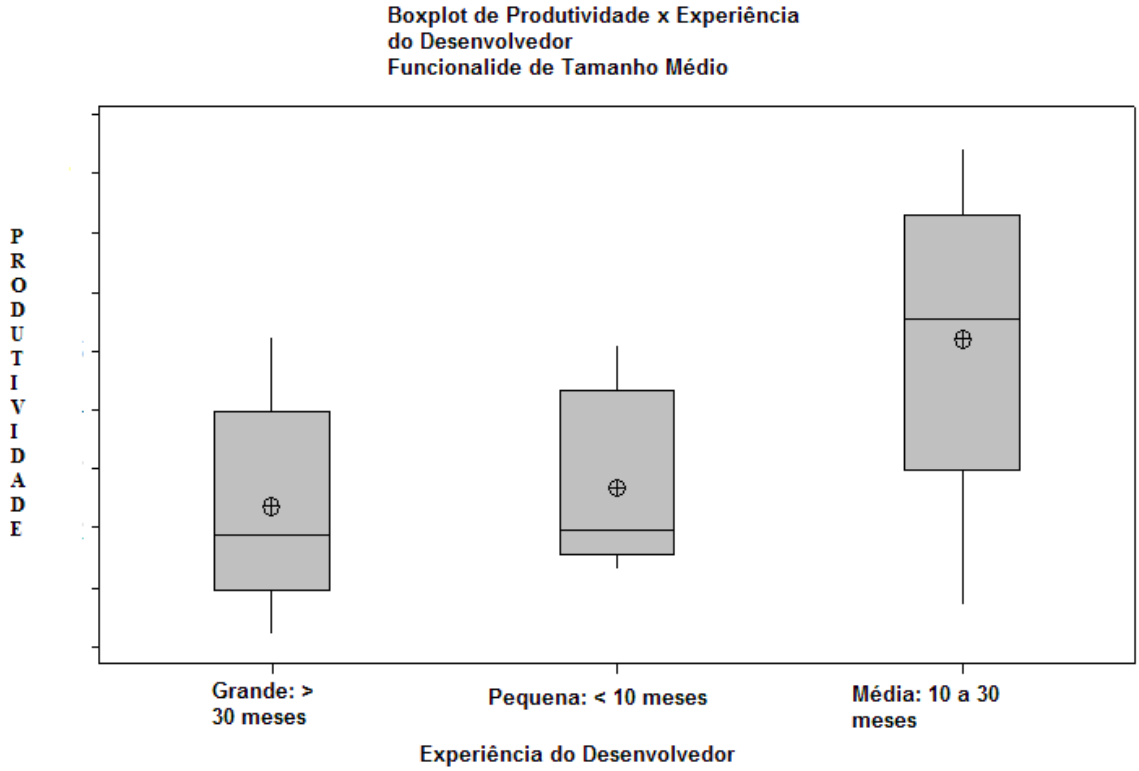


**Figura 15 –** Produtividade por experiência do desenvolvedor

**2. Estabelecer base de referência da densidade de defeito das funcionalidades**

Densidade de defeito, na empresa “Alvo”, é calcular a quantidade de defeitos encontrados depois que a funcionalidade foi integrada no produto (estágio 4 – *Stage Gate*) até o momento de lançamento (estágio 5 – *Stage Gate*) sobre a quantidade de linhas de código desenvolvidas.

Devem ser escolhidas algumas funcionalidades já desenvolvidas, projetos já finalizados e com a base histórica calcular a densidade de defeito.

A classificação do erro também é importante: ou seja, os erros que devem ser considerados são os erros introduzidos pelo código gerado por aquela funcionalidade.
Esse valor deve ser utilizado na avaliação da qualidade do produto gerado com as novas práticas em relação ao anterior.
Porém, para o escopo desse projeto, decidiu-se não acompanhar, pois a equipe teria que esperar o todo o ciclo do produto terminar para se ter os dados necessários. Cabe aqui proposta para um outro projeto DSS DMAIC para essa avaliação.

### 3. Cálculo da Eficiência do Ciclo do Processo (PCE)

Somente trabalho que adiciona funcionalidades pelas quais os clientes externos estão dispostos a pagar é valor adicionado. Uma segunda categoria de trabalho necessário para satisfazer um cliente interno ou requisitos regulatórios é conhecido como valor adicionado organizacional. Todos outros (retrabalho, múltiplas aprovações, movimentações desnecessárias, etc) não adicionam valor. A medida do grau de valor adicionado é PCE, segundo GEORGE (2002), Parnell-Klabo (2006):

$$PCE = \frac{\text{tempo de valor adicionado}}{\text{total de } \textit{lead time}}$$

Esse é um ponto fundamental dentro do Pensamento Enxuto, já que, se uma atividade não adiciona valor, ou não é obrigatória por questões do negócio / regulamentação / aspectos legais, então é desperdício.
No caso da empresa “Alvo”, em que o produto desenvolvido está embarcado em um produto de mercado de massa, sem dúvida, é necessário ter a visão do usuário (cliente final), mas o cliente direto do desenvolvimento do componente é toda a estrutura organizacional que é responsável pelo produto como um todo (o telefone celular).
Com isto em mente, é importante que a cadeia de valores reflita não somente o que adiciona valor ao produto mas qual a demanda que a organização de produto faz também, como representante do usuário final.
O time do projeto DSS DMAIC, a partir da base de dados histórica, começou a fazer um mapeamento das atividades registradas para identificar o que era desperdício.
Mas as atividades registradas na base histórica estavam num nível muito alto, ou seja, muito mais relacionadas à fase do processo do que um detalhamento de cada atividade envolvida. Portanto, também não se tinha o tempo gasto em cada tarefa, para se

calcular a eficiência do processo.

A equipe do projeto DSS DMAIC elaborou então um detalhamento das atividades relacionadas a cada fase do processo do desenvolvimento. Foi mapeada uma centena de tarefas de alto nível, com o necessário detalhamento das sub-tarefas e quais requerem navegação entre organizações departamentais múltiplas.

Para o projeto piloto então, o time DSS DMAIC definiu que o time de desenvolvimento do piloto iria registrar o tempo gasto em cada uma das tarefas detalhadas para que após o final do projeto piloto usando processos ágeis, se começasse a haver dados para que fossem analisados os gargalos e o PCE calculado.

Outro aspecto que vale ser examinado num futuro processo de melhoria de processos ágeis, é que no princípio de “Eliminar Desperdício” classificados por Poppendieck e Poppendieck (2003), dos sete tipos classificados como extra produção, inventário, passos extras de processamento, movimentação, defeitos (retrabalho), os mais prevalentes são extra produção com funcionalidades desenvolvidas e que não são consumidas pelo usuário final, gerando inventário de requisitos esperando para serem desenvolvidos.

Em relação à produção de funcionalidades que acabam não sendo consumidas pelo usuário final é necessário ter uma abordagem bastante específica, já que a visão do usuário final assim como os requisitos estão, em termos organizacionais, no escopo da estrutura de produto final. Porém o time local, como o detentor maior do conhecimento do assunto, deve ter uma ação mais questionadora e pragmática sobre os requisitos que são apresentados.

Deve haver uma atividade para se identificarem os fatores chaves para o desperdício. Seguem alguns aspectos da organização que podem afetar:

- Muitas camadas de interface e tomada de decisão sobre as funcionalidades a serem disponibilizadas ao usuário final;
- A decisão sobre quais funcionalidades foram acordadas para o produto final acontece tarde no ciclo. Isso significa que a priorização dos requisitos é bastante prejudicada;
- Execução manual de casos de teste sem o benefício da automação, limitando a regressão frequente de garantia de qualidade;
- Funcionalidades que no final do ciclo do produto ficarão ”desligadas” provocam um grande inventário de outros requisitos que não foram implementados por

falta de recursos, além de dificultar a manutenção do *baseline*.

### 3.4.3. Fase de *Improve* – Janeiro a Junho de 2008

Essa fase, ainda em andamento no momento de fechamento dessa dissertação, basicamente compreende a identificação de qual seria a funcionalidade a ser desenvolvida, a preparação física do ambiente onde os desenvolvedores seriam colocados (*war room*), assim como quais os dados que seriam capturados durante o desenvolvimento, e começou em outubro/2007. Seguem dados sobre o projeto piloto.

#### 3.4.3.1. Projeto Piloto

Após o grupo do DSS DMAIC ter estudado e definido quais seriam as restrições e condições para aplicação de processo ágil, a equipe especificou o uso de *Scrum* como o processo ágil no processo de desenvolvimento de *software*.

Em discussão com *stakeholders*, houve entendimento de que para direcionar o momento de mudança, o projeto piloto precisaria refletir uma entrega típica de projeto na plataforma, em termos de tamanho e complexidade.

Esse era um importante atributo na seleção de piloto porque quando se tenta um novo método, podem-se ter diferentes abordagens dependendo do nível de risco e demanda por resultados quantitativos. Limitar a projetos pequenos e de baixo risco pareceu ao time de DSS DMAIC que falhariam em produzir resultados convincentes e tangíveis, necessários para superar as restrições organizacionais e conduzir a mudanças no cenário.

**1. O desenvolvimento**

Após alguma análise de funcionalidades que seriam desenvolvidas, foi selecionada uma funcionalidade com:

- estimativa de aproximadamente 37.000 linhas de código;
- duração: de outubro de 2007 a junho de 2008;
- seguindo *Scrum*, mas adaptado:
    - *backlog* (conjunto de requisitos acordados e comprometidos) total logo no início do projeto;

- Procurador do cliente ou dono do produto – representante da voz do cliente não existiu durante o desenvolvimento, somente houve discussão de entendimento dos requisitos e não de quais poderiam ser entregues prioritariamente;
- Coube ao time de desenvolvimento, entender as necessidades do cliente e estabelecer um cronograma de desenvolvimento mais propício para se chegar aos objetivos finais;
- funcionalidade tinha interfaces e dependências com times externos: especificações e implementações feitas de acordo com o prazo e processo global define;
- o time foi colocado proximamente, para discussões e suporte (*war room*)
- *Sprints* de aproximadamente 1 mês, onde casos de uso da funcionalidade foram desenhados, codificados, e testados unitariamente;
- As inspeções foram feitas mais tarde, em outros *sprints*. Isso porque ainda houve tentativa de se minimizarem a quantidade de inspeções de código, o que a organização de qualidade global não aceitou;
- O último *sprint* ficou planejado somente para teste da funcionalidade como um todo, e correção de eventuais falhas encontradas;
- Documentação automatizada: desenho, diagramas de classes, relatórios de inspeção, de ferramentas de garantia de qualidade;
- Integração frequente com o “trem” do produto;
- casos de uso consistentes e prontos para uso;
- Time de desenvolvimento foi envolvido na estimativa de cada *sprint*, no *backlog* daquele *sprint* (conjunto de requisitos que seriam implementados);
- reuniões diárias em-pé de aproximadamente 15 minutos, acompanhando o que foi feito no dia, se atingiu o planejado, quais as dificuldades que estavam encontrando, quanto falta fazer, quais os problemas que já se enxergava para os dias seguintes;
- Ao final de cada *Sprint*, reuniões de lições aprendidas foram feitas;
- Apesar de se ter um planejamento de teste de funcionalidade ao final, durante os *sprints*, também foram feitos;
- Cartazes foram usados como uma ferramenta visual para mostrar problemas e atrasos (conforme Figura 16 – *Burndown chart*). Esse gráfico mostra a

quantidade horas de trabalho planejada por semana *versus* o real, mostrando que as curvas foram muito similares, porém o esforço empreendido foi bem maior que o planejado.

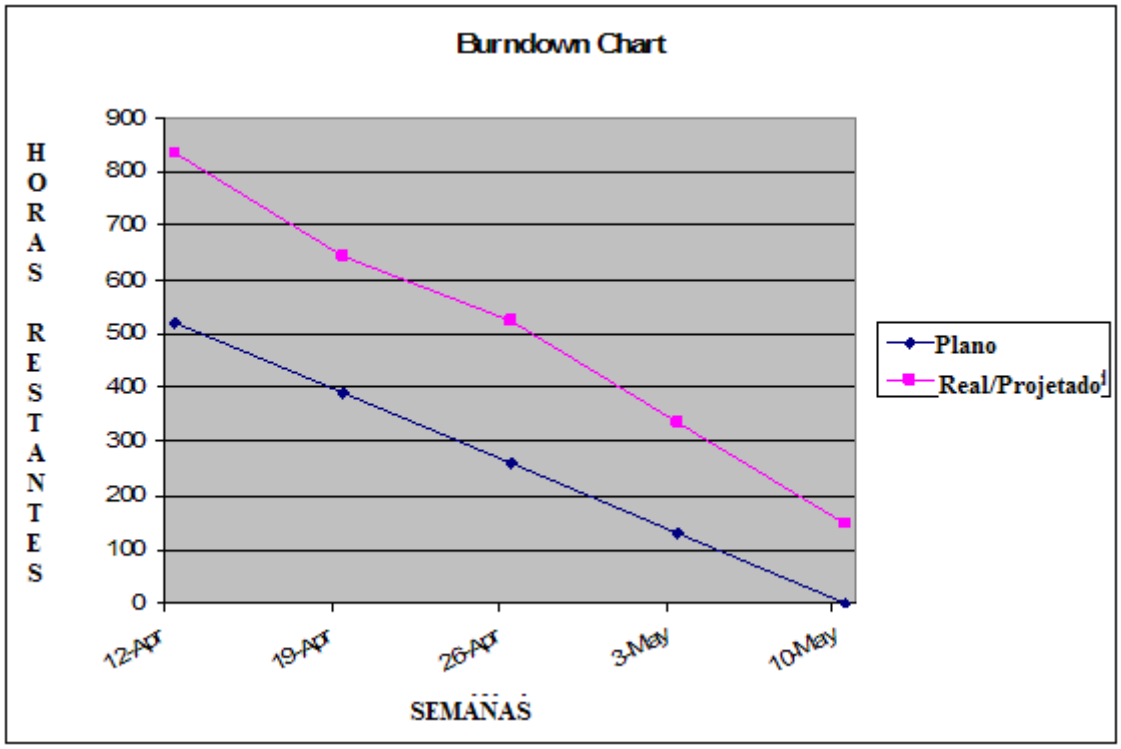


**Figura 16 –** Gráfico de *Burndown* do Projeto Piloto

**2. Os resultados**

Apesar de o desenvolvimento ainda estar em andamento, alguns resultados já podem ser apresentados:

- Membros do projeto DSS DMAIC que atuaram no piloto: Coordenador atuou como *Scrum Master*, 2 desenvolvedores, 1 dos gerentes gerenciou o desenvolvimento da funcionalidade;
- A produtividade média para funcionalidades grandes será chamada "**X**";
- Montando uma base de dados para funcionalidades desenvolvidas com processos ágeis: diariamente os desenvolvedores armazenam as atividades detalhadamente realizadas durante o dia assim como a respectiva quantidade de horas;
- A cada *Sprint*, a produtividade melhorou, mostrando, que com a adaptação ao processo, os resultados acompanham o amadurecimento do time (ver

Quadro 1). Nesse quadro, para todos os *sprints*, a produtividade exposta, ainda não contém o tempo gasto para completar 100% de inspeção de código, assim como 100% de testes unitários. Mas mesmo assim, vê-se que a produtividade melhorou.

| Sprint | Produtividade |
|---|---|
| Sprint 2 | X + 67% |
| Sprint 3 | X + 152% |
| Sprint 4 | X + 162% |

Quadro 1 – *Sprint* x Produtividade do Projeto Piloto

- duração: 7 meses (até final de abril com planejamento de 9 meses);
- número de *Sprints*: 7 de desenvolvimento planejados (6 completados), 1 de teste de funcionalidade planejado;
- Esforço: 4334 horas (até *Sprint* 6), 5550 horas (total estimado);
- Linhas de código: 33680 (até *Sprint* 6), 37100 (total estimado);
- Produtividade: base de comparação é de X linhas de código/hora (cascata)
- Produtividade: X + 122% linhas de código/hora (até *Sprint* 6), X + 93% linhas de código/hora (final: total estimado).

**3. O que ainda está planejado para a fase de *Improve***

Na fase de *Improve*, apesar de ainda não se ter terminado o projeto piloto, como já se tem resultados que mostram resultados esperados para funcionalidades grandes, a equipe DSS DMAIC decidiu:

- Propor à gerência senior, a implantação de processos ágeis para desenvolvimento de funcionalidades grandes e médias;
- O processo de implantação e acompanhamento deve ser realizado durante a fase de *Control;*
- Para funcionalidades pequenas, será necessário ainda verificar se os ganhos e práticas realizadas por esse piloto são aplicáveis;

Com a fase de *Improve* já equacionada, apesar de ainda se terem importantes atividades a serem tratadas, a equipe deve começar a planejar o que será a próxima e última fase do projeto.

### 3.4.4. Fase de *Control* – Julho a Agosto de 2008

Nessa fase, será realizada o processo de formalização dos resultados encontrados durante o projeto DSS DMAIC, assim como o acompanhamento da implantação dos processos ágeis para as funcionalidades médias e grandes a serem desenvolvidas pelo time de desenvolvimento do componente "alvo".

### 3.4.5. Pontos de Reflexão sobre a proposta de ação

Ainda que não completo o projeto DSS, algumas considerações podem ser feitas:

**1. Resultado obtido pelo projeto piloto**

O objetivo do projeto DSS de melhorar em 20% a produtividade média foi atingido para funcionalidades médias e grandes vai ser trabalhada ainda para as pequenas:

- Para funcionalidades médias e grandes
  - foi atingido.
  - Deve-se levar em consideração ainda que outros desenvolvimentos devem continuar sendo medidos
- No caso do piloto, por se tratar de um projeto grande, a produtividade histórica (processo cascata) era de X linhas de código por hora. No projeto piloto (processo ágil scrum) foi de X + 93% linhas de código por hora (projetada), apresentado no Quadro 2.

| Processo usado | Produtividade (linhas de código por hora) |
|---|---|
| Cascata | X |
| *Scrum* | X + 93% |

**Quadro 2 –** Comparação entre a produtividade histórica e com processo ágil (*Scrum*)

- O objetivo do projeto era de se ter um aumento de 20%.
- Para funcionalidades pequenas:
  - deve ainda dentro do escopo do projeto DSS ser investigada.
- O projeto piloto teve interface de desenvolvimento com outros componentes, e a interface foi garantida, conforme processo global definiu.

**2. Avaliação feita pela equipe do projeto piloto**

A avaliação foi feita em reunião de levantamento das lições aprendidas:

- As reuniões diárias melhoram a disciplina de trabalho e possibilitam trabalhar na retirada de “pedras” do caminho, melhorando a visibilidade e pro-atividade frente aos problemas;
- Comunicação entre desenvolvedores e gerência do projeto melhorou, pois há visibilidade do andamento, assim como a previsibilidade aumentou;
    - A *wiki* (enciclopédia *on-line*) funcionou como um quadro de comunicação eficiente, contendo todas as informações necessárias para o time;
- Time de desenvolvimento é mais motivado e colaborativo: a proximidade (*war room*) permite discussão de problemas e indicação de possibilidades técnicas de resolução;
- Muitas adaptações tiveram que ser feitas pelo próprio time de desenvolvimento, para que a integração constante de código fosse facilitada, assim como identificar quais módulos tinham sido inspecionados e quais ainda precisariam ser, ou quais classes de base foram integradas e quais as que deveriam estar prontas para que a aplicação pudesse utilizá-la;
- Problemas que seriam identificados somente no fim do ciclo de desenvolvimento, quando no ciclo de vida de cascata, agora são identificados durante um ciclo curto, e como os testes se repetem ao longo de todos os *Sprints*, os casos de uso são mais intensivamente testados;
- Devesenvolvedores estão mais expostos, diariamente, ao ter que apresentar os resultados do dia, assim como o planejamento para o dia seguinte. Essa transparência incomoda algumas pessoas.

Foram também levantados alguns aspectos a serem considerados para os próximos desenvolvimentos de funcionalidades, usando processos ágeis.

**3. Algums aspectos a serem considerados para os próximos projetos ágeis:**

- Com a mudança de paradigma de planejamento – ou seja, o planejamento está ocorrendo todos os dias, mas principalmente, a cada novo *Sprint* – e a base de dados histórica está sendo re-montada, o processo ainda precisa ser revisitado e novos cálculos de esforço por fase precisam ser alterados;
- Os desenvolvedores devem ter disciplina de catalogar as atividades diárias com detalhes, para que a base histórica seja criada. A gerência deve estar atenta. Caso contrário informações serão perdidas para os próximos desenvolvimentos;

- Testes de regressão automatizados, que é uma das questões abordadas por processos ágeis (codifica, testa, ou até mesmo, codificação dirigida por testes), é muito difícil de ser feito dado que o desenvolvimento das funcionalidades é feito sobre componente “alvo” já existente. Ou seja, o produto cresce incrementalmente. Os testes do legado deveriam ser desenvolvidos, o que seria um custo muito grande.

**4. Quanto ao processo ágil**

- *Scrum* foi selecionado por ser indicado pela empresa global, onde outras organizações haviam utilizado;
- O time foi adotando algumas práticas ágeis e o processo foi sendo adaptado e “aprendido” ao longo do desenvolvimento;
- Valores, princípios do processo organizacional e interfaces preservadas;
- O que foi garantido: desenvolvimento com iterações curtas e incrementais
- A cada *Sprint*, foi melhorando a produtividade do time;
- Numa organização grande e global, como a empresa “Alvo”, um projeto não pode ser verdadeiramente independente, mas precisa interagir e seguir as regras da organização como um todo.

**5. Desenvolvimento *Cross*-times**

- A diferença entre uma grande organização e uma pequena, talvez seja essa: mesmo projetos individuais dependem de outros ambientes de diversas maneiras. O time deve ser capaz de comunicar e coordenar ações com outros times na organização, e o *software* desenvolvido deve ser integrado suavemente ao sistema maior;
- Os times frequentemente estão distribuídos pelas várias localidades físicas e fusos horários, aumentando a necessidade de formalização de comunicação, e diminuindo a agilidade para tomada de decisão e de notificações;
- Times usando diferentes abordagens de processo de desenvolvimento para definir uma determinada interface: para o time utilizando processo ágil, apenas uma parte das decisões precisariam ser tomadas num determinado instante do projeto, mas para o outro time, que utiliza processo tradicional necessitará da decisão total num outro momento.

**6. Integração contínua do desenvolvimento ao "produto"**

- Integração contínua significou aos desenvolvedores integrarem seus códigos da funcionalidade constantemente no repositório de código do componente "alvo". A cada modificação, já era integrado. Isso contribuiu na agilidade em detetar e remover rapidamente defeitos relacionados à integração do código dos outros membros, assim como contribuiu em dividir o esforço de integração em porções menores e mais fáceis;
- A integração constante do código ao "produto" permaneceu no escopo do desenvolvimento da funcionalidade e não sob a gerência de configuração do produto;
- Na empresa "Alvo", a gerência de configuração e integração do produto somente é envolvida quando o desenvolvimento da funcionalidade estiver completo para ser integrado à versão oficial do produto. É uma decisão que não cabe ao time de desenvolvimento da funcionalidade e sim à gerência do produto.

**7. *Pair-programming***

- O conceito é aceito pela empresa "Alvo", como possível de trazer melhoras na produtividade e qualidade, mas dada a complexidade e tamanho do sistema todo, a empresa continua valorizando a inspeção como um processo mais apropriado para se ter um grupo discutindo os casos de uso e revendo a implementação e casos de teste para garantir a sua abrangência.

**8. Princípios do Desenvolvimento Enxuto**

Os pontos de reflexão abordados acima demonstram que os princípios de desenvolvimento enxuto detalhados na revisão bibliográfica dessa dissertação foram tratados na aplicação do processo ágil *Scrum*: construir com qualidade (através de testes por todo o ciclo de vida de desenvolvimento), criar conhecimento, adiar compromissos e entregar rápido (com integração contínua e incremental, entrega de incrementos funcionais em ciclos curtos), respeitar as pessoas (considerando o time contribuindo, opinando, planejando e endereçando o desenvolvimento).

Um aspecto foi parcialmente atingido: otimizar o todo. Parcialmente porque no que diz respeito à estrutura funcional do componente "alvo", sim houve otimização com foco

total de todos os participantes ou envolvidos no desenvolvimento do projeto piloto: tanto desenvolvedores, testadores, gerente de projeto, de configuração, de integração. Mas ao tratar da empresa “Alvo” como um todo, o projeto DSS DMAIC teve que ficar restrito a proposta de mudanças somente no escopo do componente “alvo”.

Ainda há um aspecto essencial para o conceito de “enxuto”, que é a eliminação de desperdício que não pôde ser trabalhado completamente.

Os desperdícios comumente encontrados no ciclo de desenvolvimento de *software,* que podem ser mapeados mais facilmente no ciclo de desenvolvimento usado no componente “alvo” (*stage* gate 3 – escopo dessa pesquisa-ação) foram tratados: passos extras de processamento, movimentação, defeitos, espera, transporte. Mas ainda não foi escopo desse projeto avaliar o impacto de cada atividade contida no ciclo de desenvolvimento de *software* quanto a valor adicionado, avaliação de gargalos ou propostas de melhorias, ou formalização da cadeia de valor.

Aspectos de extra-produção e inventário de requisitos não foram abordados. Isso porque a estrutura de decisão sobre o que deve ser desenvolvido num determinado produto, para determinados mercados, ainda há muitas organizações envolvidas onde precisariam ser trabalhados conjuntamente os conceitos de “enxuto”. Há no componente “alvo” muitos requisitos ou até mesmo funcionalidades desenvolvidas porém desligadas no produto final, ou porque o cliente não tem interesse, ou porque o cliente ainda não tem infra-estrutura necessária para habilitar aquela funcionalidade.

Os aspectos não trabalhados nesses dois princípios (otimizar o todo global e desperdício) se “encontram” na causa raiz e reforçam a característica necessária para “enxuto” de ter que se trabalhar num nível mais alto de decisão e sustentação gerencial/executiva em relação a decisões relacionadas a produtos, e as organizações relacionadas.

A seguir as conclusões dessa dissertação são sumarizadas.

## 4. Conclusão

O objetivo desse trabalho era investigar quais os princípios de desenvolvimento enxuto de *software* seriam factíveis para uma unidade da empresa global “Alvo” que desenvolve *software* embarcado para dispositivos móveis, do mercado em massa, e propor um conjunto de ações para a melhoria de sua produtividade.

No desenvolvimento de *software* a dificuldade é composta pela complexidade da estrutura, dada a sua natureza com alto nível de variabilidade. A “repetiblidade” parece só ser possível até o nível de padrão de trabalho. Para se obter os mesmos efeitos como o do sistema puxado, uma complexa rede de iteratividade, gerenciamento de projeto, teste, integração, e cooperação do time deve ser construída.

Foi realizada uma pesquisa-ação, na qual foi relatado um conjunto de avaliações e iniciativas que foram propostas até a execução de um projeto piloto para se transicionar o processo de desenvolvimento de *software* de tradicional em cascata para processos ágeis *scrum*. O pressuposto para o projeto de mudança era de que o escopo das mudanças não poderia alterar a interface com outras organizações de desenvolvimento de outros componentes nem impedir que a gerência do produto obtivesse os dados que fossem requeridos.

O grande desafio foi integrar eficientemente as novas práticas com o ambiente global da empresa “Alvo”, que possui processos de desenvolvimento fortemente especificados, alinhavados e controlados, assim como integrar novos dados levantados durante o desenvolvimento com sistemas de qualidade e métricas bem definidas e acompanhadas.

Com a proposta de mudança de processo, a base de dados histórica do componente “alvo”, que já continha dados referentes há três anos, deverá ser reconstruída. Portanto, atenção especial deve ser dada à coleta e integridade dos dados armazenados, já que esses serão o alicerce para próximos planejamentos, e replicação de conhecimento e experiência para outros projetos e organizações. Assim como o aspecto de testes automatizados e com regressões devem ser ainda concebidos, já que a base sobre a qual as funcionalidades são construídas, e portanto alteradas, devem também serem retestadas e validadas. Deve ser mantida a qualidade do legado.

A interação e dependência do componente com outros componentes do produto pode ser um aspecto delicado a ser tratado, pois as implementações e necessidades de negociação de interfaces podem ocorrer em momentos diferentes.

A experiência piloto no desenvolvimento de *software* de uma funcionalidade no componente “alvo” implementou um processo iterativo, com ciclos de aproximadamente 4 semanas, as especificações e codificações eram feitas concomitantemente, testes exaustivos foram realizados, código funcional disponibilizado a cada ciclo e o plano de iteração refletiu exatamente o que o time estava fazendo. Também, o desenvolvimento ágil contou com um time altamente motivado, cooperativo, cujo esforço coletivo do time era produzir código funcional, e a integração foi realizada continuamente.

Há aspectos que precisam de maior investigação, que extrapolaram o escopo desse projeto mas que certamente, a partir deste, serão necessários: cálculo da densidade de defeito, análise da eficiência dos testes integrados durante o desenvolvimento, identificação das atividades que formam a cadeia de valores, qual o *takt time* do novo ciclo, eliminação de desperdícios, e trabalhar com a gerência de produto para que o conceito de enxuto possa ser absorvido por organizações que estabelecem e priorizam os requisitos junto aos clientes. Somente quando outras organizações forem envolvidas é que medidas mais abrangentes poderão evitar que a organização dispenda esforço de gerência e desenvolvimento em inventário de requisitos ou funcionalidades não demandas pelos clientes. Esses itens ampliam o conceito de “desenvolvimento ágil” e sustentam mais adequadamente alguns princípios de “desenvolvimento enxuto” de *software*, abrindo possibilidades de se trabalhar em melhoria contínua e cadeia de valor. São oportunidades para que princípios enxutos sejam perseguidos.

A medida de sucesso para o projeto de transição foi realizada considerando linhas de código produzidas por hora. O objetivo do projeto era de se ter um aumento de 20% na produtividade. Atingiu-se 93%. Porém, para um projeto piloto, único, a gerência do componente “alvo” está considerando que houve uma melhoria significativa, atingindo os objetivos, mas que a base histórica ainda deve ser realimentada para que a produtividade média seja restabelecida.

O que se pôde concluir no caso na empresa “Alvo”, que usou uma abordagem híbrida, é que práticas ágeis podem ajudar organizações maduras a se tornarem mais flexíveis e serem mais produtivas.

## 4.1. Futuras investigações

Como em qualquer investigação, responder a uma questão sempre conduz a outras. Portanto, há espaço para estudos futuros relacionados ao tema dessa dissertação, como:

- Avaliação da experiência piloto de desenvolvimento de *software* embarcado aplicada no processo global da empresa "Alvo":
  - Qual pode ser o efeito multiplicador do processo ágil em outros componentes?
  - A partir da re-estrutura da base de dados histórica do ciclo de desenvolvimento de *software*, quais são os gargalos e quais os desperdícios que podem ser atacados para que a produtividade seja ainda mais melhorada?
  - Para se ter uma estrutura Enxuta e Ágil, para todo o ciclo do produto, que ações devem existir nas organizações de *marketing* técnico, que são os "porta-vozes" dos clientes, para as organizações de desenvolvimento?
  - Um projeto para se avaliar a densidade de defeito, sob o efeito da implantação de processos ágeis.
- A Teoria das Restrições aplicada à Gerência de Projetos de Desenvolvimento de *Software*
  - Quais os efeitos no planejamento e acompanhamento do projeto numa estrutura de desenvolvimento enxuto?
  - Quais aspectos relevantes a serem considerados na criação da corrente crítica?
- Produção nivelada em desenvolvimento de *software*
  - *Store Management* é uma proposta viável? Existem outras?
  - Consegue-se ter uma produção nivelada somente com os conceitos de processos ágeis?
  - Quais parâmetros devem ser considerados na análise de nivelamento de produção de *software*?
  - Ainda não está claro no processo de desenvolvimento enxuto de *software* como tratar o nivelamento de produção, e portanto como serão encontrados os gargalos. A literatura ainda é escassa sobre o assunto, principalmente por causa da viariabilidade intrínsica ao *software*.

- Desenvolvimento de *Software* e Subcontratados
    - As tarefas para os subcontratados devem ser bem definidas para o caso onde há uma licitação, e portanto precisam estipular o que é requerido e o que será oferecido, baseado num plano, *milestones* e entregas, com detalhes suficientes para que o custo seja estimado.
    - Como adequar essas condições com processos ágeis?

## Referências Bibliográficas

Abrahamsson, P.; Dooms, K.; Vodde, B.; Still, J.. (2006) *Agile software development of embedded systems: Practical results.* 32nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). August 29-September 1, 2006

Ambler, Scott (2006). *Survey Says: Agile Works in Practice.* August 2006. Acessado em nov/2007.
http://www.ddj.com/dept/architect/191800169
http://www.ambysoft.com/surveys/

Anderson, D. J. (2003). *Agile Management for Software Engeineering: Applying the Theory of Constraints for Business Results.* Prentice-Hall.

Barcaui, A. B.; Quelhas, O. (2004). *Corrente Crítica: Uma alternativa à gerência de projetos tradicionais*. Revista Pesquisa e Desenvolvimento Engenharia de Produção n.2., p. 1 – 21, julho 2004

Barnett, L. (2006). *Agile Survey Results: Solid Experience and Real Results.* Sept 2006. Acessado em Nov/2007.
http://www.agilejournal.com/content/view/93/33

Boehm, B.; Turner, R. (2005). *Management Challenges to Implementing Agile Processes in Traditional Development Organizations.* IEEE Software. Published by IEEE Computer Society. Sep/Oct 2005, p. 30 – 39.

Bolwijn, P. T.; Kumpe (1990). *Manufacturing in the 1990s – Productivity, Flexibility and Innovation*. Long Range Planning, vol. 23.

Bryman, A. (1989). *Research Methods and Organization Studies.* Londres: Uniwin Hyman.

Casarotto Filho, N.; Fávero, J. S.; Castro, J. E. E. (1999). *Gerência de Projetos / Engenharia Simultânea*. São Paulo: Editora Atlas S.A.

Cooper, R. (2000). *Winning with New Products Doing it Right*. IVEY Business Journal, vol. July-August 2000, p. 54-60.

Creswell, J.W. (1994). *Research Design: Qualitative & Quantitative Approaches.* Londres: Sage.

Cusumano, M.; Nobeoka, K. (1998). *Thinking Beyond Lean* . New York, NY, The Free Press.

Digital Focus *Agile 2006 Market Survey.* Acessado em Nov/2007.
http://biz.yahoo.com/prnews/060724/nym110.html?.v=43
http://www.digitalfocus.com/research/research.php

Gane, C; Sarson, T. (1979). *Structured System Analysis: Tools and techniques*. 1st Edition, Prentice-Hall

George, M. (2002). *Lean Six Sigma*, McGraw-Hill, New York

Godoy, A.S. (1995). *Introdução à pesquisa qualitativa e suas possibilidades.* Revista de Administração de Empresas. São Paulo: EAESP-FGV. Mar./Abr., v. 35, n. 2, p. 57-63.

Gupta, A. K.; Wilemon, D. (1996). *Changing Patterns in Industrial R&D Management*. Journal of Product Innovation, vol. 13.

Harry, M..J.; Schroeder, R.. (2000). Six *Sigma: The* Breakthrough Management *Strategy Revolutionizing the World's Top Corporations.* Doubleday, NY.

Hayes, K.; Kuchinskas, S. (2003). *Going Mobile: Building Real-Time Enterprise with Mobile Applications that Work (no pagination)*. CMP Books.

Highsmith, J. (2002). *Agile Software Development Ecosystems*. Boston, MA: Addison Wesley.

Highsmith, J. (2000). *Adaptative Software Development: A Collaborative Approach to Managing Complex Systems.* Addison Wesley.

Highsmith, J. (2004). *Agile Project Management: Creating Innovative Products.* Boston, MA: Addison Wesley.

Highsmith, J., (2002). What is agile software development? Crosstalk. The Journal of Defense Software Engineering, p. 4-9.

Imai, M. (1986). *Kaizen, The Key to Japan's Competitive Success*. McGraw-Hill Publising Co, New York.

Kuri Chu, M.G.P. (2002). *Diagnóstico da Estratégia Competitiva e de Produção em uma Unidade de Negócios.* Dissertação (Mestrado em Engenharia de Produção). Universidade Federal de São Carlos. São Carlos.

Kishida, M.; Silva, A. H.; Guerra, Ezequiel. (2006). *Benefícios da Implementação do Trabalho Padronizado na ThyssenKrupp*. *Lean* Institute do Brasil, publicado em 16/10/2006

Korzeniowski, P. (2006). *Software Shift in the Mobile Device Market*. TechNewsWorld, Oct, 4th/2006. Acessado em Nov/2007.
http://www.technewsworld.com/story/53403.html

Larman, C. (2004). *Agile & Iterative Development*. Addison-Wellesy, Boston, MA.

Lidvall, M.; Muthig, D.; Dagnino, A.; Wallin, C.; Stupperich, M.; Kiefer, D.; May, J.; Kähkönen, T. (2004). *Agile Software Development in Large Organizations.* IEEE Software. Published by IEEE Computer Society. Dec 2004 (Vol. 37, No. 12) p. 26-34

Linderman, K.; Schoroeder, R.G.; Zaheer S.; Choo, A.S. (2003). *Six Sigma: a goal-*

*theoretic perspective.* Journal of Operations Management 21 (2003) 193-203.

Martins, R.A. (1999). *Sistemas de Medição de Desempenho: um modelo para estruturação do uso.* Tese (Doutorado em Engenharia de Produção). Escola Politécnica, Universidade de São Paulo. São Paulo.

McConnell, S. (2006). *Software Estimation: Desmistifying the Black Art.* Microsoft Press: Redmond , Washington.

McGrath, M. E. (2001). *Product Strategy for High-Technology Companies: Accelerating Your Business to Web Speed*. New York: McGraw-Hill Companies.

McManus, H. (1999). *Lean Engineering*. Cambridge, MA, Massachusetts Institute of Technology.

MethodsAndTools.com. *Agile Adoption Poll*. May 2005. Acessado em Nov/2007. http://www.methodsandtools.com/facts/facts.php?may05 http://www.agile-lu.org/wikka.php?wakka=AdoptionOf*Agile*Methods

Ministério do Desenvolvimento, Indústria e Comércio Exterior – Secretaria de Tecnologia Industrial, Coordenação de Comércio Eletrônico. (2005). *Polos de Desenvolvimento de Software*. Versão 1, dezembro/2005.

Mirshawka, V. (1994). *TPM a moda brasileira***.** Makron Books, São Paulo.

Morien, R. I. (2005). *Agile management and the Toyota way for software project*. 3rd IEEE International Conference on Industrial Informatics (INDIN 2005), Perth, WA, Aug 10 2005. IEEE, Perth, WA. – p. 516-522

Motta, C. (2005). *Retorno dos Investimentos em Empresas de Tecnologia: Fusões, Aquisições e Mercado de Capitais.* Centro Brasileiro de Estudos Jurídicos da Tecnologia da Informação. Tendências Tecnológicas: http://***www.cbeji.com.br*****.** Acessado em Nov/2007

Negócio oportuno - *Desenvolver softwares e aplicativos destinados ao mercado mundial***.** Portal da Administração. Acessado em Nov/2007 *http://www.portaldaadministracao.org/2007/07/negcio-oportuno-desenvolver-softwares-e.html*

Niimi, A. (2004). *Sobre o Nivelamento (Heijunka)***.** Adaptado do "Manufacturing Week", Chicago, Fev 2004.

Nystedt, D. (2006). *Mercado global terá 2,6 bilhões de usuários de celulares em 2006*. IDG Nes Service, Taipei, publicada em 10 de novembro de 2006. Acessado em Nov/2007 *http://idgnow.uol.com.br/telecom/2006/11/10/idgnoticia.2006-11-10.7698735687*

Ohno, T. (1988). *Toyota Production System: Beyond Large Scale Production. New York:* Productivity Press.

Pande, P.S.; Neuman, R.P.; Cavangh, R.R. (2000). *The Six Sigma Way: How GE, Motorola, and Other Top Companies Are Honing Their Performance*. McGraw Hill, NY.

Parnell-Klabo, E. (2006). *Introducing Lean Principles with Agile Practices at a Fortune 500 Company.* Proceedings of AGILE 2006 Conference (AGILE'06), IEEE Computer Society, p. 232 - 242

Pires, S. (1995). *Gestão Estratégica da Produção.* São Paulo: Ed.UNIMEP.

PMBOK – (2004). A *Guide To The Project Management Body Of Knowledge* (PMBOK®) (3rd ed.). (2004).Newtown Square, PA: Project Management Institute.

Poppendieck, M.; Poppendieck, T. (2006). *Implementing Software Development: From Concept to Cash*. Editora: Addison-Wesley Professional.

Poppendieck, M.; Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit for Software Development Managers*. Boston, MA: Addison Wesley

Poppendieck, M.. (2007). *Lean Software Development*. 29th International Conference on Software Engineering (ICSE'07 Companion). IEEE Computer Society . p. 165 - 176

Porter, M. E. (1991). *Estratégia competitiva: técnicas para análise de indústrias e da concorrência*. 5ª ed. Rio: Ed. Campus.

PMDOI. *Project Management Declaration of Interdependence.* ***http://www.pmdoi.org/***

Raman, S. (1998). *Lean Software Development – Is it feasible?*. IEEE. Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE. Bellevue, WA, USA. 31 Oct-7 Nov 1998, Vol 1, pp C13/1-C13/8

Ribeiro, H. (1994). *5S A Base para a Qualidade Total*. Salvador,BA: Casa da Qualidade.

Richards, K. (2006). *Early mainstream: Agile develops in the enterprise.* July 2006. Acessado em Nov/2007
http://www.versionone.net/pdf/ADT_Article.pdf

Roetzheim, W. (2000). *Estimating Software Costs*. Software Development Magazine On-Line, Oct 15th / 2000. http://www.sdmagazine.com/documents/s=821/sdm0010d/

Rosson, R. (1994). *Self-Directed Work Teams at Texas Instruments Defense Systems & Electronics*. Group Management. Cambridge, MA, MIT.

Rother, M.; Shook, J. (1998). *Aprendendo a enxergar.Mapeando o fluxo de valor para agregar valor e eliminar desperdício.* São Paulo, *Lean* Institute Brasil.

Salmre, I. (2005). *Writing Mobile Code: Essential Software Engineering for Building Mobile Applications (no pagination).*Addison Wesley Professional.

Salomon, D.V. (1991). *Como fazer uma Monografia.* 2. ed. São Paulo: Martins Fontes.

Shingo, S. (1996). *O Sistema Toyota de Produção do Ponto de Vista da Engenharia de Produção*. 2ª ed. Porto Alegre: Artes médicas, 1996.

Smaley, A. (2006). *Estabilidade é a Base para o Sucesso da Produção Lean*. Lean Institute Brasil, São Paulo.

Smith, H.; Fingar, P. (2003). *Digital Six Sigma: Integrating continuous improvement with continuous change and continuous learning.* Acessado em Nov/2007. *http://www.businessprocesstrends.com*

Smith R. (1997). *The Historical Roots of Concurrent Engineering Fundamentals*. IEEE Transactions on Engineering Management 44(No 1).

Stalk Jr, G.; Hout, T. M. (1992). *Competindo contra o tempo*. Rio de Janeiro: Campus.

Techtarget Network. (2005). Acessado em nov/2007. http://searcheio.techtarget.com/sDefinition/0..sid19_gci810510.00.html

Thiollent, M. (1997). *Pesquisa-Ação nas Organizações.* São Paulo: Atlas.

Turk, D.; France, R.; Rumpe, B. (2002). *Limitations of Agile Software Processes*. Third International Conference on eXtreme Programming and *Agile* Processes in Software Engineering (XP 2002), p. 43-46

Udo, N.; Koppensteiner, S.(2003). *Will agile development change the way we manage software projects? AGILE from a PMBOK guide perspective*. in *Proc. PMI Global Congr., Project Manage. Inst.,* Baltimore, MD, Sep. 22–23, 2003, p 1-7.

Ulrich, K.; Eppinger, S. D. (2004). *Product Design and Development* . New York, NY, McGraw-Hill, Inc.

Vainio, A. M.; Tuunanen, T.; Abrahamsson, P. (2005). *Developing Software Products for Mobile Markets: Need for Rethinking Development Models and Practices*.

Proceedings of the 38th Hawaii International Conference on System Sciences. (HICSS'05) - Track 7 - Volume 07, p. 189 - 198

Vargas, R. V. (2002). *Análise de valor agregado em projetos*. Rio de Janeiro: Editora Brasport.

VersionOne (2006). *State of Agile Development Survey* **.** Acessado em Nov/2007 http://www.versionone.net/surveyresults.asp

Walton, M. (1999). *Strategies for Lean Product Development: A Compilation of Lean Aerospace Initiative Research*. Research Paper WP99-01-91. Cambridge, MA: Massachusetts Institute of Technology.

Wirth, N. (1995). *A Plea for Lean Software*. IEEE Computer, Feb 2005, Vol 28, No 2, p. 64-68.

Womack, J. (2006). *Mura, Muri, Muda?*. *Lean* Enterprise Institute do Brasil, Ago 2006, p. 1 - 3

Warmkessel, J. (1998*). Introduction to the Product Value Stream*. Cambridge, MA.

Womack J.; Jones D. T. (1996). *Lean Thinking*. New York, NY, Simon & Schuster.,

Womack, J. P.; Jones, D. T.; Ross, D. (1992). *The Machine that Changed the World*. New York, NY: Rawson Associates.

Yeo, K. T. (2005). *Critical failure factors in information system projects.* International Journal of Management and Enterprise Development. Vol 2, number 2/2005, p 219 - 239

Yin, Robert K. (1989). *Case study research. Design and methods.* London: Sage Publications.