

Princípios do Desenvolvimento de Software *Lean*



Curso de Verão 2008 - IME/USP

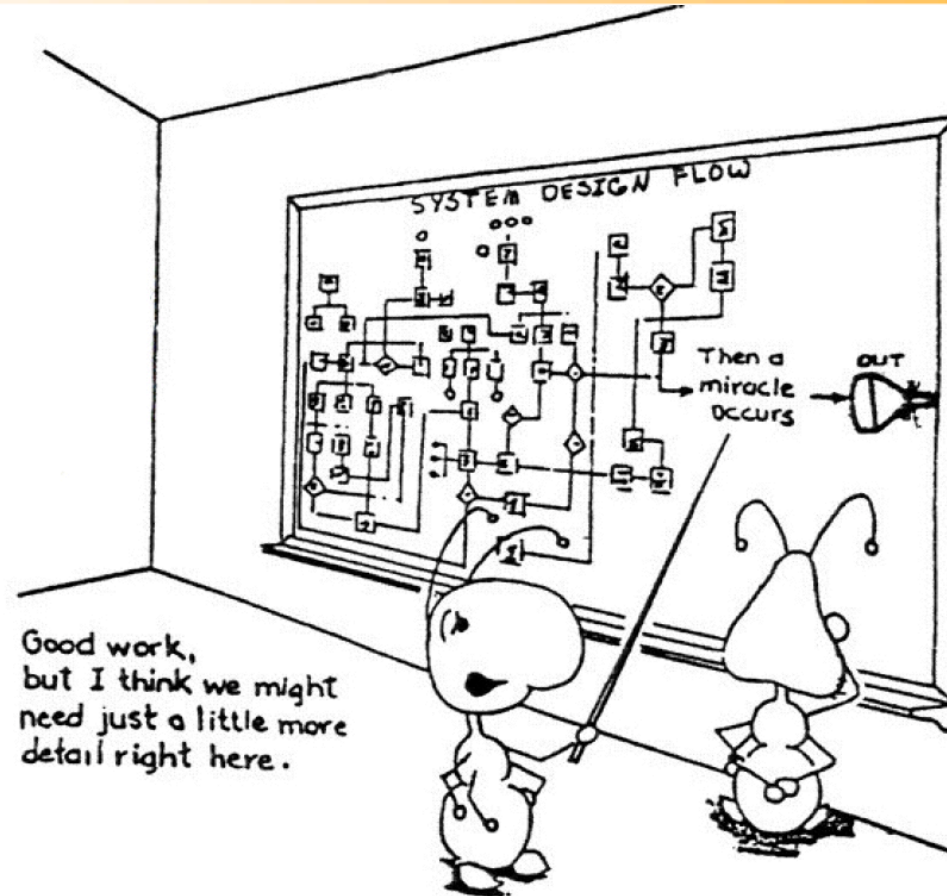
www.agilcoop.org.br

Danilo Sato

Dairton Bassi

24/Jan/2008

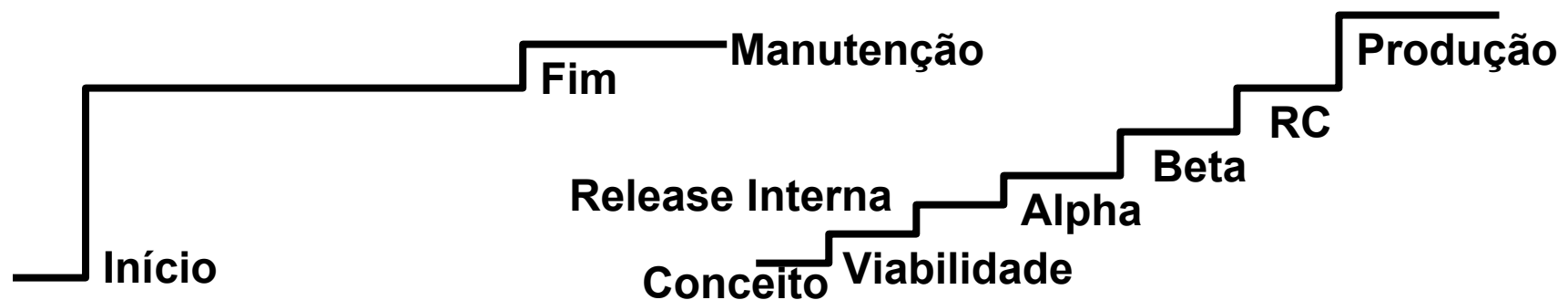
Introdução



- Qual o papel do software?
- Por que desenvolvemos software?

Introdução

- Como é o seu processo de desenvolvimento de software?
 - Quem é seu cliente?
 - Produção/manufatura ou novo produto?
 - Gerenciado como projeto ou produto?



Introdução

- Como clientes:
 - “Resolvam completamente meus problemas”
 - “Não desperdicem meu tempo”
 - “Ofereçam exatamente o que eu preciso”
 - “Entreguem valor exatamente onde e quando eu preciso”
 - “Reduzam o número de decisões que eu preciso tomar para resolver meus problemas”

Introdução

- Empresas que aplicam conceitos *Lean*:

The Dell logo, consisting of the word "DELL" in a bold, blue, sans-serif font. The letter "E" is stylized with a diagonal slash.

Produção / Manufatura

The Toyota logo, featuring the word "TOYOTA" in red, uppercase, sans-serif font, preceded by the Toyota symbol (three overlapping ellipses).

Produção / Manufatura +
Desenvolvimento de Produtos

The ZARA logo, consisting of the word "ZARA" in a blue, serif font.

Cadeia de Suprimento

The Google logo, consisting of the word "Google" in its multi-colored, sans-serif font.

Desenvolvimento de Software

Origens do Pensamento *Lean*

- O Sistema de Produção da Toyota:
 - Taiichi Ohno:
 - Fluxo “Just-In-Time”
 - Complexidade vs. Economia de Escala
 - *Automation*
 - “*Stop-the-Line*”
 - Shigeo Shingo:
 - Produção sem estoque
 - Trabalho organizado em tarefas pequenas
 - Zero Inspeções
 - “*Mistake-proof*”

Origens do Pensamento *Lean*

- Os valores foram expandidos para outras áreas:
 - Produção *Lean*
 - Manufatura / Operações *Lean*
 - Cadeia de Suprimentos *Lean*
 - Desenvolvimento de Produtos *Lean*
- Desenvolver software é criar um novo produto!
 - Sempre aparece algo novo

Problemas com Software

- Alguns dos motivos:
 - Requisitos que mudam rápido e constantemente
 - Tomada de decisões centralizada
 - Gerenciamento rígido do escopo
 - Práticas “tradicionais” de desenvolvimento (linear)
 - Pouco foco na qualidade do software produzido

Desenvolvimento de Software

Lean

- Princípios *Lean* aplicados ao software:
 1. Elimine Desperdícios
 2. Inclua a Qualidade no Processo
 3. Crie Conhecimento
 4. Adie Comprometimentos
 5. Entregue Rápido
 6. Respeite as Pessoas
 7. Otimize o Todo

1. Elimine Desperdícios

“Desperdício é tudo aquilo que não agrega valor ao cliente”

-- Taiichi Ohno

- Este é o principal princípio *Lean*
- Software funcionando é o que vai trazer valor ao cliente
- É preciso aprender a identificar desperdícios

1. Elimine Desperdícios

- Os sete desperdícios de software:
 - a) Trabalho incompleto (“em-progresso”)
 - b) Processos a mais
 - c) Funcionalidades a mais
 - d) Troca de tarefas
 - e) *Handoffs*
 - f) Atrasos
 - g) Defeitos

1.a) Trabalho Incompleto

- Artefatos inacabados consomem recursos sem trazer retorno
- Reclamações (“*churn*”):
 - Requisitos > Especificando muito cedo
 - Teste > Testando muito tarde
- Documentação não-codificada
- Código não-sincronizado
- Código não-testado
- Código não-implantado

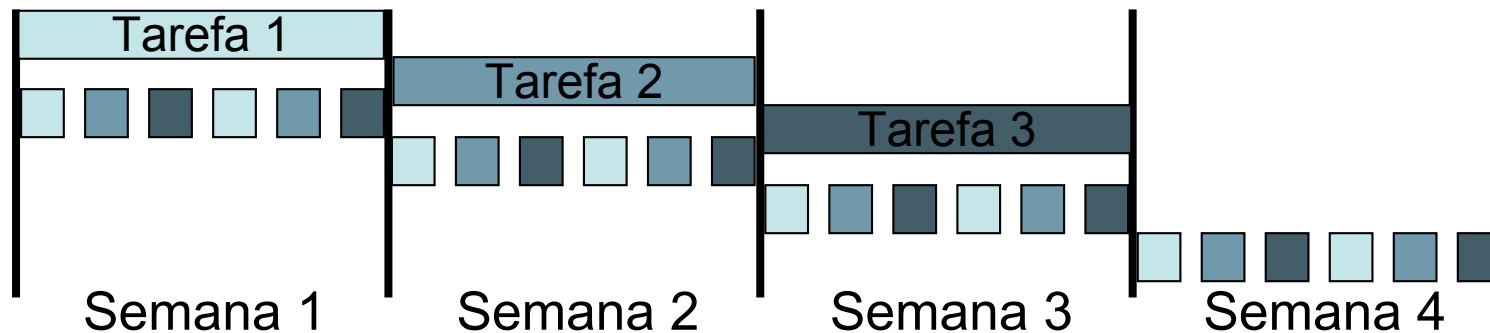
1.b) Processos a mais

- Burocracia desnecessária
- Documentação desnecessária
- Atividades de gerenciamento
- Bom teste para avaliação:
 - Existe algo ou alguém esperando pelo que está sendo produzido?

1.c) Funcionalidades a mais

- Jim Johnson:
 - 45% das funcionalidades implementadas não são utilizadas nunca
 - 19% das funcionalidades implementadas são raramente utilizadas
- Código não-utilizado introduz complexidade
- Complexidade: inimigo da manutenção
- Mito: Especificar cedo reduz o risco

1.d) Troca de Tarefas



- Como sincronizar desenvolvimento novo e manutenção?
 - Rotação de pessoas
 - Alocar uma parte da manhã
 - Triagem agressiva + atendimento imediato a manutenções urgentes
 - Implantação semanal + tratar manutenção como funcionalidades do release

1.e) *Handoffs*

- Metáfora: Aprender a andar de bicicleta
- Conhecimento tácito é difícil de transmitir
- Quanto mais *handoffs*, maior é a perda de conhecimento
- Dicas:
 - Reduza *handoffs*
 - Use meios de comunicação eficazes
 - Libere partes do trabalho para apreciação e *feedback*

1.f) Atrasos

- Programadores precisam tomar decisões a cada 15 minutos
- É impossível assumir que toda informação necessária estará documentada
- Opções na hora da dúvida:
 - Tentar descobrir a resposta
 - Trocar de tarefa
 - Adivinhar e prosseguir

1.f) Atrasos

- Exemplos:
 - Esperar pelo entendimento completo dos requisitos
 - Esperar meses pela aprovação do projeto
 - Esperar pela alocação das pessoas
 - Esperar pela disponibilidade das pessoas alocadas
 - Processo de controle de alterações
 - Esperar pelo sistema inteiro ficar completo para ter as funcionalidades-chave
 - Esperar o código passar pelos testes
 - Esperar para comunicar defeitos (QA no final)

1.g) Defeitos

- O custo dos defeitos aumenta com o tempo
- Equipes ágeis se esforçam ao máximo para evitar defeitos
- Em caso de defeito, fazem o máximo para curar a raíz do problema
- Testes automatizados são investimentos!
- Use os testes como *design* do sistema (TDD)

2. Inclua a Qualidade no Processo

*“Inspeccionar para prevenir defeitos é bom;
Inspeccionar para encontrar defeitos é
desperdício”*

-- Shigeo Shingo

- Não deixe os testes para o final
- Ciclos de teste muito longos geralmente gastam mais tempo corrigindo defeitos
- Ao invés de se esforçar para gerenciar defeitos, evite-os

2. Inclua a Qualidade no Processo

- Prevenindo defeitos com vários tipos de teste:

		Perspectiva do Negócio			
Suporte à Programação	Testes de Histórias Interesse de Negócio (Design do Produto)	Testes de Usabilidade	Testes Exploratórios	Crítica ao Produto	
	Testes de Unidade Interesse do programador (Design do Código)	Testes de Propriedade Segurança, Carga, Combinatório			
		Perspectiva da Tecnologia			

3. Crie Conhecimento

“Não existe bala de prata”

-- Fred Brooks

- Incentive o compartilhamento de conhecimento tácito
- Buscar um processo “padrão” engessa
- O processo deve ser continuamente melhorado

3. Crie Conhecimento

- Método científico (***Plan-Do-Check-Act***):
 - Enquadre o problema
 - Procure pela raiz do problema
 - Proponha uma solução
 - Implemente a solução
 - Verifique os resultados
 - Analise e adapte seus padrões
- Mito: Predições criam previsibilidade

4. Adie Comprometimentos

- Decisões irreversíveis devem ser tomadas o mais tarde possível (*last responsible moment*)
- É preciso definir o momento da decisão
 - Quando houver mais informação
- Flexibilidade arbitrária também é ruim
- Um bom líder saberá alocar flexibilidade
- Mito: Um plano é um comprometimento

4. Adie Comprometimentos

- *Design* baseado em conjunto (*set-based*)
 - Na incerteza, experimente diversas soluções
 - Agende o momento da decisão
 - Sempre haverá uma solução que funciona
 - Paradoxo: Isso não é desperdício!
- Exemplo: Toyota Prius
 - 15 meses do conceito ao lançamento
 - 10 opções de motores híbridos desenvolvidos durante os 4 primeiros meses
 - Motores híbridos viraram item opcional

5. Entregue Rápido

“A moral da história é que devemos encontrar uma maneira de entregar software tão rápido, que nossos clientes não tenham tempo de mudar de idéia”

-- Mary Poppendieck

- Competir com base na velocidade traz grande vantagem competitiva
- Mito: Rápido e sujo vs. Lento e limpo

5. Entregue Rápido

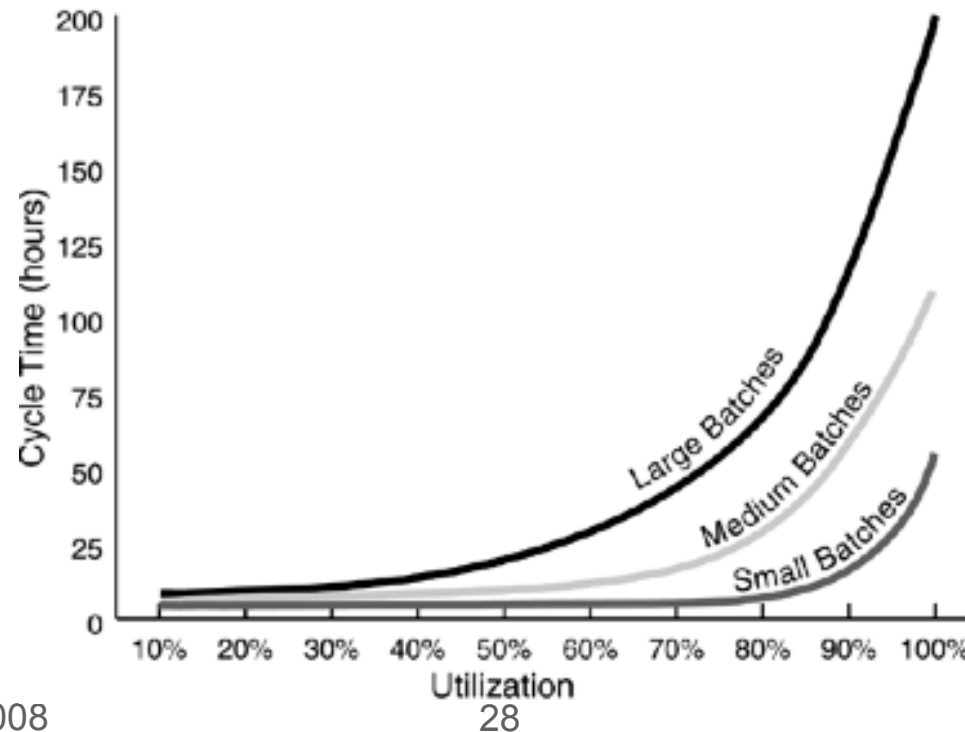
- Sistemas *Pull* em software (reduzem inventário)
- Radiadores de Informação



5. Entregue Rápido

- Teoria das filas:

- Tempo do ciclo =
$$\frac{\# \text{ Coisas em processo}}{\text{Taxa Média para completar}}$$



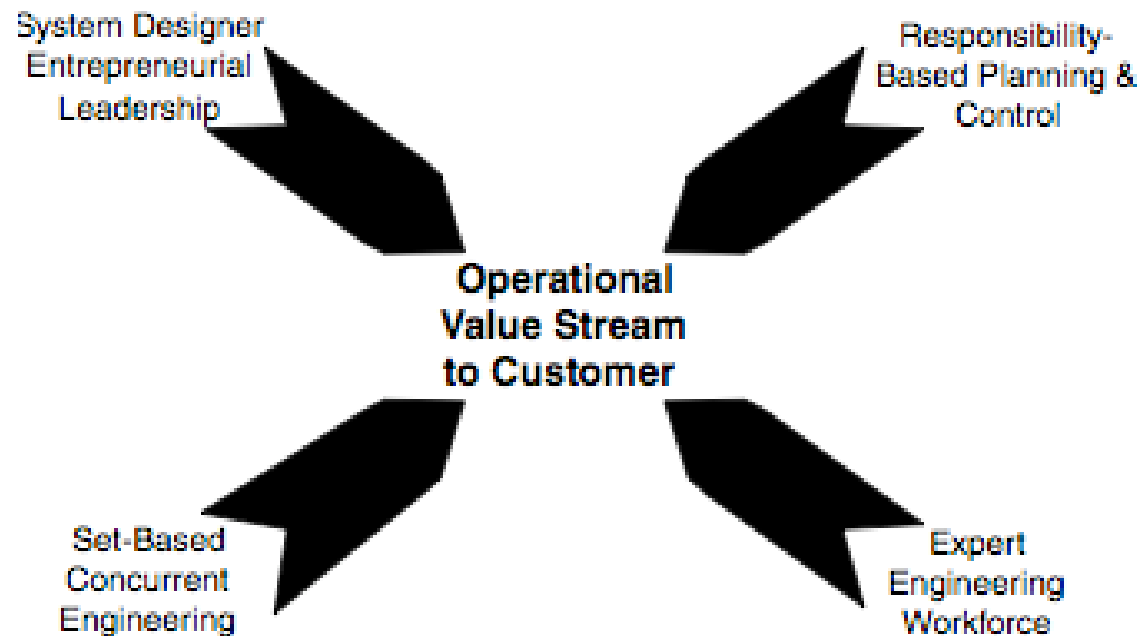
5. Entregue Rápido

- Reduzindo o tempo do ciclo:
 - Normalize a entrada de trabalho
 - Minimize o # de unidades “em-processo”
 - Minimize o tamanho das unidades “em-processo”
 - Estabeleça uma cadência regular
 - Limite o trabalho à capacidade
 - Use agendamento *pull*

6. Respeite as Pessoas

- Desenvolvimento de Produtos na Toyota

Knowledge-Based Engineering
(The Lean Development System)



An operational value stream emerges from the interaction of four cornerstone elements.

6. Respeite as Pessoas

- 3 pilares estão relacionados às pessoas:
 - Liderança
 - Força de trabalho com conhecimento
 - Planejamento e controle baseado em responsabilidade
- Liderança:
 - Grande conhecimento técnico
 - Grande conhecimento do cliente
- Times completos

6. Respeite as Pessoas

- Pessoas são recursos?
- Papel da gerência é distribuir tarefas e monitorar?
- Exemplo: Planta da GM - NUMMI
- Motivação:
 - Propósito
 - Participação (*belonging*)
 - Segurança
 - Competência
 - Progresso

6. Respeite as Pessoas

“A verdadeira inovação da Toyota é sua habilidade em usufruir da inteligência dos trabalhadores ‘comuns’”

-- Gary Hamel

- Programas de Qualidade:
 - CMM, CMMI, MPS.Br, Six Sigma, ISO, TQM...
 - Lançados com as melhores das intenções
 - Mal-implementados
 - Focam na parte burocrática
 - Processo definido por um grupo separado
- Mito: “O melhor jeito” existe

7. Otimize o Todo

- Exemplo: Zara
 - Design → Loja em 2 semanas
 - Pedidos entregues em 2 dias:
 - Em qualquer lugar do mundo
 - Prontos para venda (pendurados, com preço)

Resultados	Zara	Indústria
Novos itens introduzidos/ano	11.000	3.000
Itens vendidos no preço total	85%	60-70%
Itens não vendidos	<10%	17-20%
% vendas gasto em propaganda	0.3%	3-4%
% vendas gasto em TI	0.5%	2%

7. Otimize o Todo

- Círculo vicioso #1 no desenvolvimento de software:
 - Cliente pede nova funcionalidade, para ontem
 - Desenvolvedor ouve: Termine isso rápido!
 - Resultado: Mudanças feitas de qualquer jeito no código
 - Resultado: Complexidade do código aumenta
 - Resultado: Número de defeitos no código aumenta
 - Resultado: Tempo para adicionar funcionalidade cresce exponencialmente

7. Otimize o Todo

- Círculo vicioso #2 no desenvolvimento de software:
 - Equipe de testes sobrecarregada
 - Resultado: Testes bem após codificação
 - Resultado: Desenvolvedores não recebem *feedback* imediato
 - Resultado: Desenvolvedores criam mais defeitos
 - Resultado: Equipe de teste tem mais trabalho
 - ...
- Mito: Micro-otimização leva à Macro-otimização

7. Otimize o Todo

- É preciso olhar para o processo todo
- Não adianta resolver os sintomas
- É preciso resolver a causa
- 5 Porquês



7. Otimize o Todo

- Métricas:
 - Medir informação vs. Medir desempenho
 - Cuidado!
 - É fácil medir muitas coisas
 - É fácil medir as coisas erradas
 - *Earned Value* mede aderência ao plano:
 - Custo, escopo e cronograma
 - Mas e a satisfação do cliente e a qualidade?
 - Contra-exemplo:
 - Desenvolvedores: LOC/h
 - Testadores: # Defeitos encontrados

7. Otimize o Todo

- Diminua o número de métricas de desempenho
- Meça sempre um nível acima:
 - Obrigam a colaboração
 - Estabelecem uma base para fazer *trade-offs*
- Tempo de ciclo + Mapa de Fluxo de Valor
- ROI + Modelo de Lucros e Perdas
- Satisfação do Cliente (NPS) + Entendimento das suas necessidades

Resapitulando...

1. Elimine Desperdícios
2. Inclua a Qualidade no Processo
3. Crie Conhecimento
4. Adie Comprometimentos
5. Entregue Rápido
6. Respeite as Pessoas
7. Otimize o Todo

O Início de um Caminho a Trilhar

- Comece onde está
- Encontre sua maior restrição
- Visualize sua maior ameaça
- Avalie sua cultura
- Treine
- Resolva seu maior problema
- Remova acomodações
- Meça
- Implemente
- Repita o ciclo

Referências

- Livros:
 - Mary e Tom Poppendieck, “Lean Software Development: An Agile Toolkit”, Addison-Wesley, 2003
 - Mary e Tom Poppendieck, “Implementing Lean Software Development: From Concept to Cash”, Addison-Wesley, 2006
 - Jim Johnson, “ROI, It’s Your Job”, Keynote Speech at Third International Conference on Extreme Programming (XP2002), 2002
- Online:
 - leandevlopment @ Yahoo Groups
 - www.poppendieck.com