Desenvolvimento de Software *Lean*

Curso de Verão 2007 - IME/USP

www.agilcoop.org.br

Danilo Sato & Alfredo Goldman

AgilCoop

Introdução

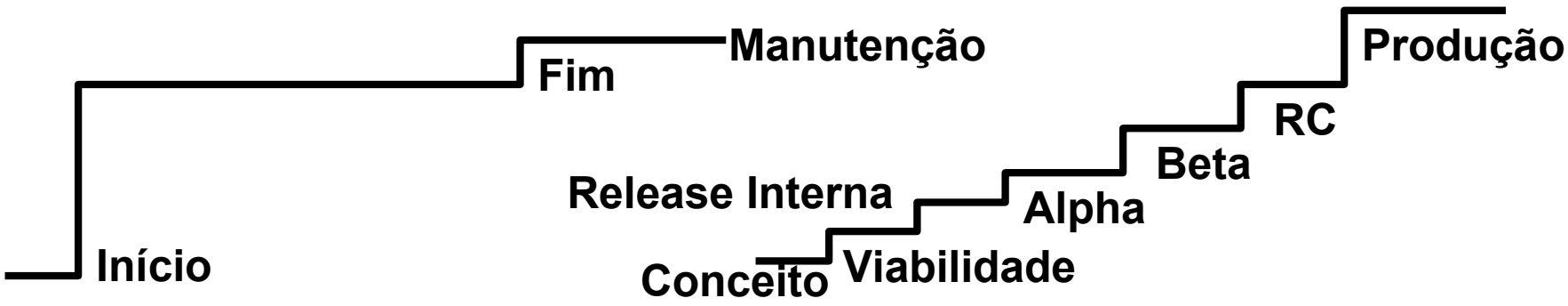
"Desenvolvimento de software é uma cadeia com diversos elos"

-- Kent Beck

- Qual o papel do software?
- Como o software se encaixa no plano mais amplo (*big picture*)?

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*:

Produção / Manufatura

Produção / Manufatura +
Desenvolvimento de Produtos

Cadeia de Suprimento

Desenvolvimento de Software

Origens do Pensamento *Lean*

- O Sistema de Produção da Toyota:
  - Taiichi Ohno:
    - Fluxo “Just-In-Time”
      - Complexidade vs. Economia de Escala
    - *Autonomation*
      - *“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

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

Elimine Desperdícios

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

Trabalho Incompleto

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

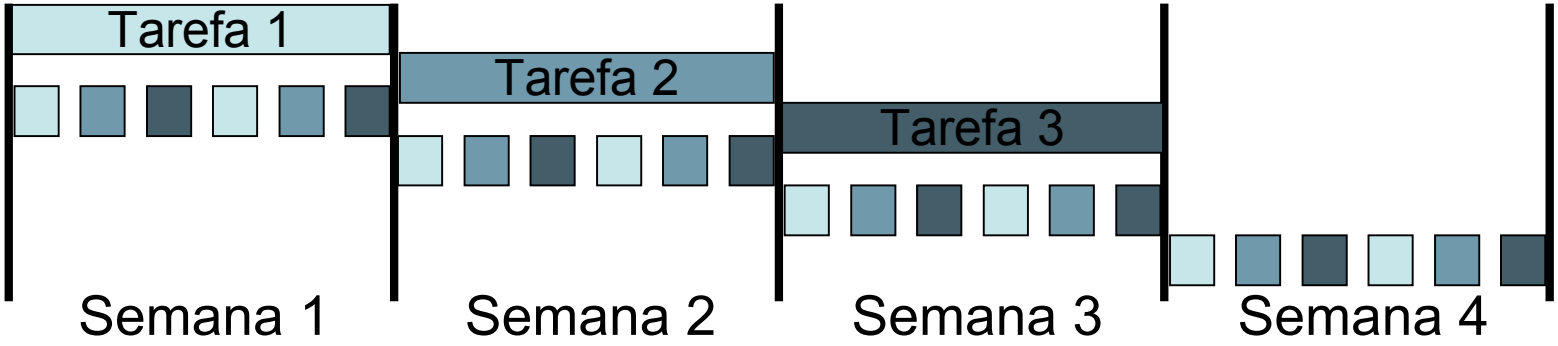
Processos a mais

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

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

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

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*

Atrasos

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

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)

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)

Inclua a Qualidade no Processo

"Inspecionar para previnir defeitos é bom; Inspecionar 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

Inclua a Qualidade no Processo

- Prevenindo defeitos com vários tipos de teste:

Crie Conhecimento

"Não existe bala de prata"
-- Fred Brooks

- Metáfora: criar vs. preparar uma receita
- Incentive o compartilhamento de conhecimento tácito
- Buscar um processo "padrão" engessa
- O processo deve ser continuamente melhorado

Crie Conhecimento

- Método científico (***P**lan-**D**o-**C**heck-**A**ct*):
  - Enquadre o problema
  - Procure pela raíz do problema
  - Proponha uma solução
  - Implemente a solução
  - Verifique os resultados
  - Analise e adapte seus padrões
- Mito: Predições criam previsibilidade

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

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 primeiro meses
  - Motores híbridos viraram item opcional

Entregue Rápido

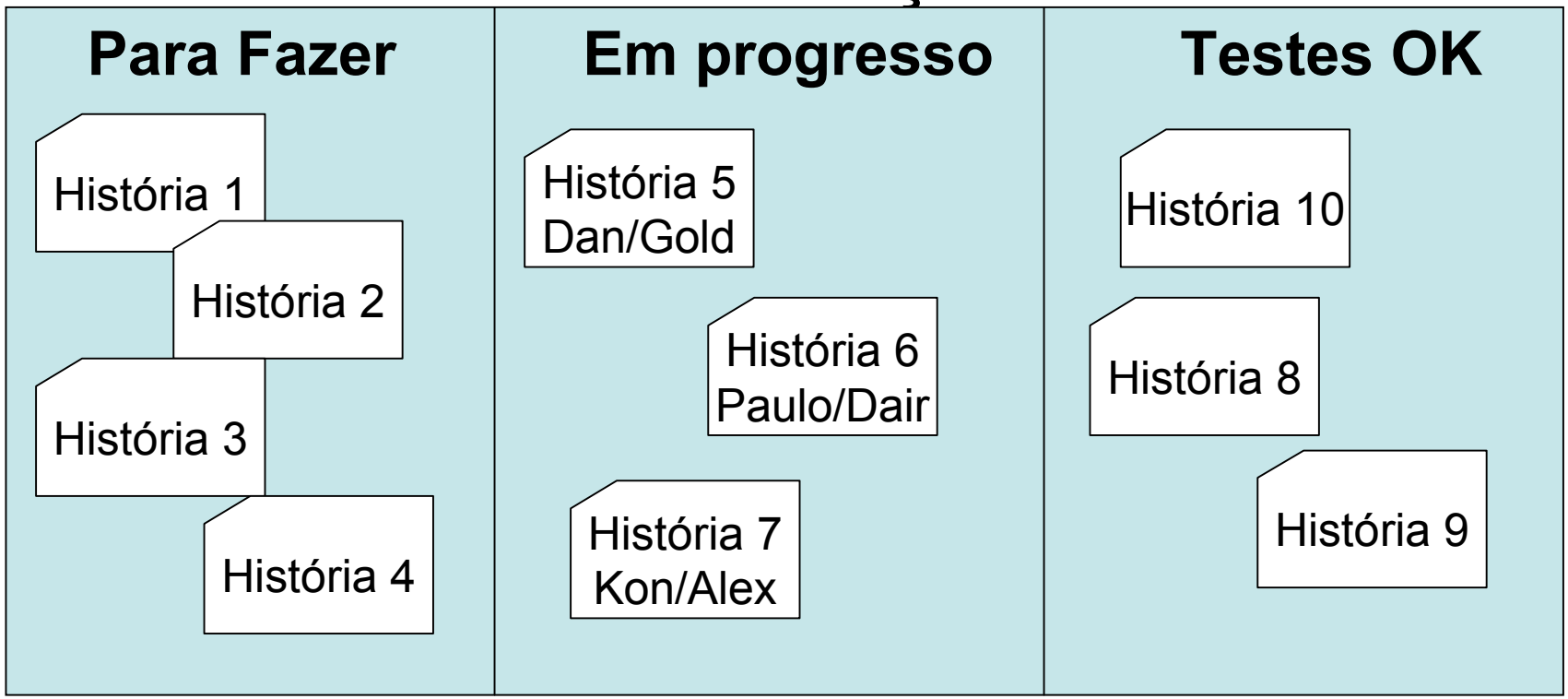
"A moral da história é que devemos encontrar uma maneira de entregar software tão rápido, que nossos clientes não tenham tempo de mudar de idéia"

-- Mary Poppendieck

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

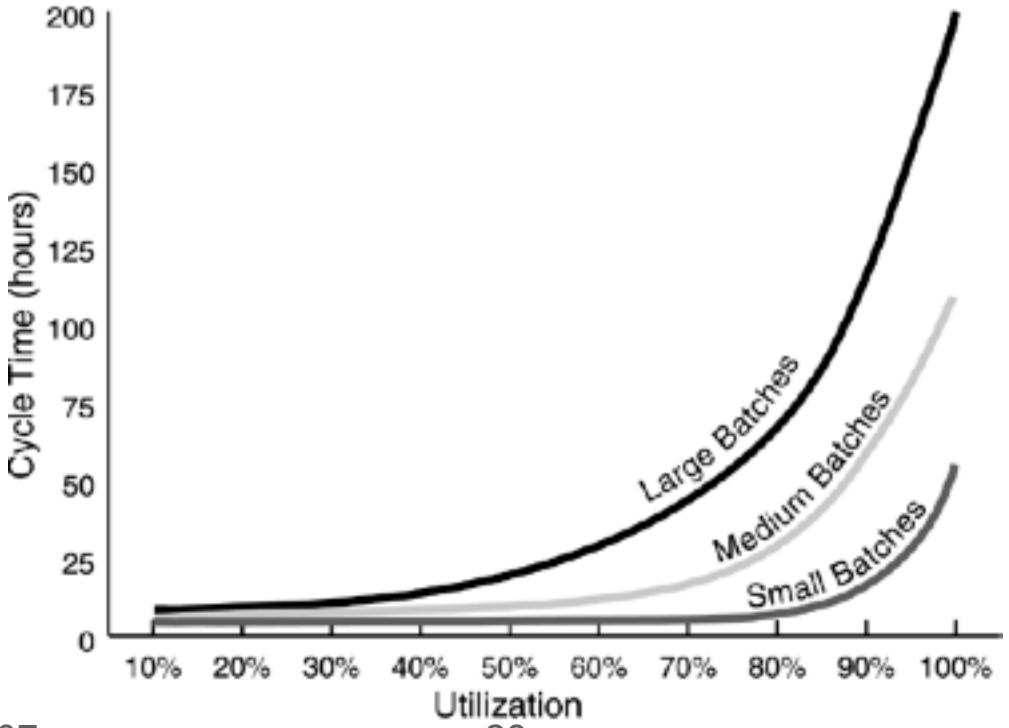
Entregue Rápido

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



Entregue Rápido

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

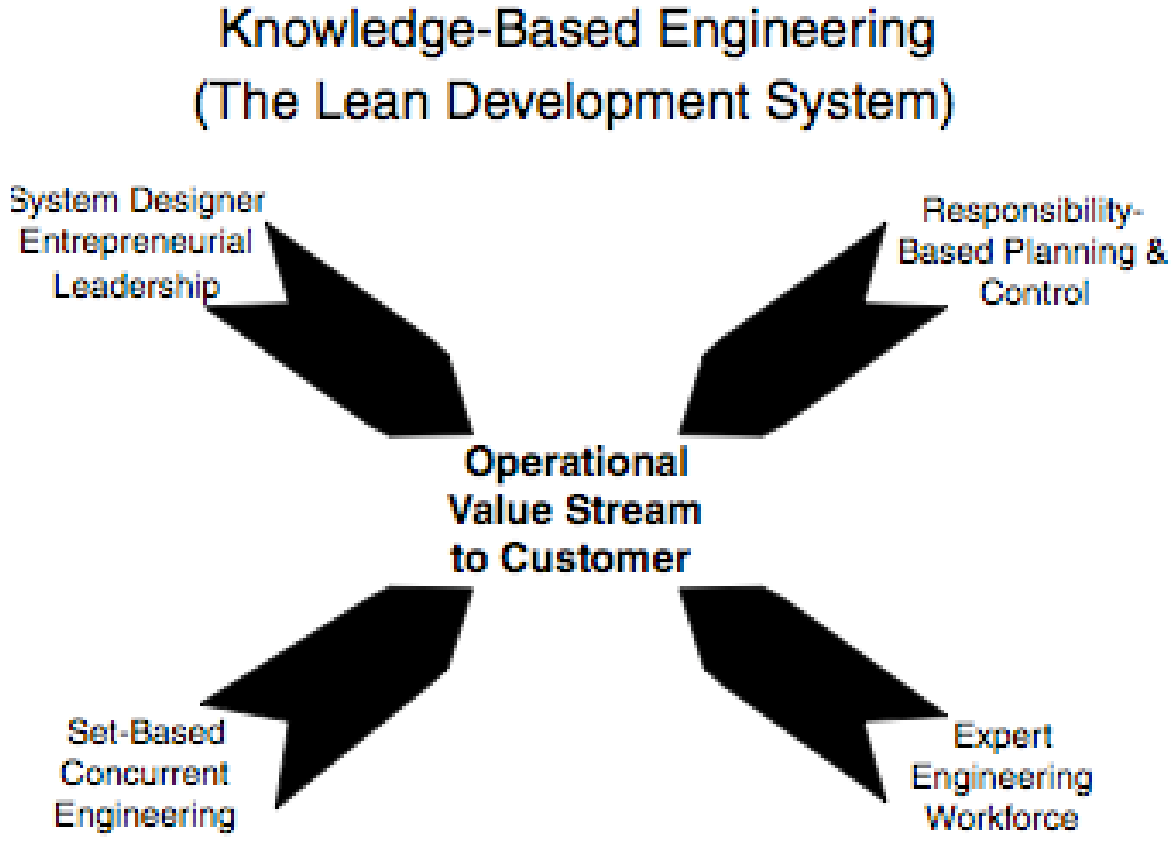


Entregue Rápido

- 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*

Respeite as Pessoas

- Desenvolvimento de Produtos na Toyota



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

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

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

Respeite as Pessoas

"A verdadeira inovação da Toyota é sua habilidade em usufruir da inteligência dos trabalhores 'comuns'"

-- Gary Hamel

- Programas de Qualidade:
  - CMM, CMMI, 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

Otimize o Todo

- Exemplo: Zara
- Círculo vicioso #1 no desenvolvimento de sotware:
  - 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

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

Otimize o Todo

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

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

Otimize o Todo

- Diminua o número de métricas de desempenho
- Meça para cima:
  - Medidas no nível mais alto que direcionam para o comportamento correto
  - Estabeleça uma base para fazer *trade-offs*
- Tempo de ciclo +Mapa de Fluxo de Valor
- ROI + Modelo de Lucros e Perdas
- Satisfação do Cliente + Entendimento das suas necessidades

Recapitulando…

1. Elimine Desperdícios
2. Inclua a Qualidade no Processo
3. Crie Conhecimento
4. Adie Comprometimentos
5. Enregue 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 Yout Job”, Keynote Speech at Third International Conference on Extreme Programming (XP2002), 2002
- Online:
  - leandevelopment @ Yahoo Groups
  - www.poppendieck.com