Métodos Ágeis de Desenvolvimento de Software e a Programação eXtrema (XP)

Academia de Ensino Superior - AES

Danilo Sato e Dairton Bassi
Departamento de Ciência da Computação

www.agilcoop.org.br

AgilCoop

Boa Noite!

Documento de Especificação da Palestra

- Objetivo:
  - Apresentar os Métodos Ágeis de Desenvolvimento de Software e a Programação Extrema (XP)
- Autores:
  - Danilo Sato:
    - Bacharel Ciência da Computação no IME/USP
    - Mestrando em Ciência da Computação no IME/USP
    - Membro da AgilCoop
    - Consultor de Desenvolvimento na Assebléia Legislativa do Estado e SP
    - Consultor de Desenvolvimento no Ikwa
  - Dairton Bassi:
    - Bacharel Ciência da Computação no IME/USP
    - Mestrando em Ciência da Computação no IME/USP
    - Membro Fundados da AgilCoop
    - Diretor de Tecnologia do Ikwa

Histórico de Revisões

Revisão	Autor	Observações
Versão 0.1	Alfredo	Rascunho
Versão 0.2	Fabio	Versão Revisada
Versão 0.3	Alfredo	Versão Extendida Rascunho
Versão 1.0	Fabio	Versão Aprovada
Versão 1.1	Danilo	Novas Idéias
Versão 1.2	Dairton	Outras idéias
Versão 1.3	Paulo	Resolução de Conflito de idéias

Referência

Definição de Métodos Ágeis

Dkfjhasdkflajshdl fkjashd sdkjfahsdlfh dkfjahsl dkjfhas dhfla ksjdhfaklsjd laksdjhf alskjdf askjdfh askjdflaksjdhf aklsjdh alskjdhf asdjfhasdj fasdsjfhla sdjfhlas jdhflasdjhflasjd falsjkdhflkajsdhfasdjf ajsdgfawefouiwewio u d fkd jfaljwhlwefhldfjnalsedh wieuhfnei fuhwf eiur hweifh aidfnifnwoiefn eifh wiefiw eiu eif weinf efuqwei ncndfn wife jwoiefefion weufnweifnw l eunwc efio uweuiwhe oiuwefhqoiw euh ioweuh oweiuh weufow eoweufh oweiuf weiowf howiuhowe fiuwehfo e ohwiefhoiuh wieuhoweiweofih eoufhowefhwoeiuryh wefdo eofuiqw rhoew woeiruyuowe werruweiro iurweyr fhwiuoerh fhhrfhroiqer woe weorweo we rhrfh ruyhfru efweo9ie erh roq eruh wewijfhla sdjfhlas jdhflasdjhflasjd falsjkdhflkajsdhfasdjf ajsdgfawefouiwewio u d fkd jfaljwhlwefhldfjnalsedh wieuhfnei fuhwf eiur hweifh aidfnifnwoiefn eifh wiefiw eiu eif weinf efuqwei ncndfn wife jwoiefefion weufnweifnw l eunwc efio uweuiwhe oiuwefhqoiw euh ioweuh oweiuh weufow eoweufh oweiuf weiowf howiuhowe fiuwehfo e ohwiefhoiuh wieuhoweiweofih eoufhowefhwoeiuryh wefdo eofuiqw rhoew woeiruyuowe werruweiro iurweyr fhwiuoerh fhhrfhroiqer woe weorweo we rhrfh ruyhfru efweo9ie erh roq eruh wewijfhla sdjfhlas jdhflasdjhflasjd falsjkdhflkajsdhfasdjf ajsdgfawefouiwewio u d fkd jfaljwhlwefhldfjnalsedh wieuhfnei fuhwf eiur hweifh aidfnifnwoiefn eifh wiefiw eiu eif weinf efuqwei ncndfn wife jwoiefefion weufnweifnw l eunwc efio uweuiwhe oiuwefhqoiw euh ioweuh oweiuh weufow eoweufh oweiuf weiowf howiuhowe fiuwehfo e ohwiefhoiuh wieuhoweiweofih eoufhowefhwoeiuryh wefdo eofuiqw rhoew woeiruyuowe werruweiro iurweyr fhwiuoerh fhhrfhroiqer woe weorweo we rhrfh ruyhfru efweo9ie erh roq eruh wewieurw eryoerui wefhnounqweur whernncvqei e weuro weor e riweierywuie wiehkjld we

Principais aplicações de Métodos Ágeis

Dkfjhasdkflajshdl fkjashd sdkjfahsdlfh dkfjahsl dkjfhas dhfla ksjdhfaklsjd laksdjhf alskjdf askjdfh askjdflaksjdhf aklsjdh alskjdhf asdjfhasdj fasdjfhla sdjfhlas jdhflasdjhflasjd falsjkd√hflkajsdhfasdjf ajsdgfawefouiwewio u d fkd jfaljwhlwefhldfjnalsedh wieuhfnei fuhwf eiur hweifh aidfnifnwoiefn eifh wiefiw eiu eif weinf efuqe fiuwehfo e ohwiefhoiuh wieuhoweiweofih eoufhowefhwoeiuryh wefdo eofuiqw rhoew woeiruyuowe werruweiro iurweyr fhwiuoerh fhhrfhroiqer woe weorweo we rhrfh ruyhfru efweo9ie erh roq eruh wewijfhla sdjfhlas jdhflasdjhflasjd falsjkdhflkajsdhfasdjf ajsdgfawefouiwewio u d fkd jfaljwhlwefhldfjnalsedh wieuhfnei fuhwf eiur hweifh aidfnifnwoiefn eifh wiefiw eiu eif weinf efuqwei ncndfn wife jwoiefefion weufnweifnw l eunwc efio uweuiwhe oiuwefhqoiw euh ioweuh oweiuh weufow eoweufh oweiuf weiowf howiuhowe fiuwehfo e ohwiefhoiuh wieuhoweiweofih eoufhowee fiuwehfo e ohwiefhoiuh wieuhoweiweofih eoufhowefhwoeiuryh wefdo eofuiqw rhoew woeiruyuowe werruweiro iurweyr fhwiuoerh fhhrfhroiqer woe weorweo we rhrfh ruyhfru efweo9ie erh roq eruh wewijfhla sdjfhlas jdhflasdjhflasjd falsjkdhflkajsdhfasdjf ajsdgfawefouiwewio u d fkd jfaljwhlwefhldfjnalsedh wieuhfnei fuhwf eiur hweifh aidfnifnwoiefn eifh wiefiw eiu eif weinf efuqwei ncndfn wife jwoiefefion weufnweifnw l eunwc efio uweuiwhe oiuwefhqoiw euh ioweuh oweiuh weufow eoweufh oweiuf weiowf howiuhowe fiuwehfo e ohwiefhoiuh wieuhoweiweofih eoufhowefhwoeiuryh wefdo eofuiqw rhoew woeiruyuowe werruweiro iurweyr fhwiuoerh fhhrfhroiqer woe weorweo we rhrfh ruyhfru efweo9ie erh roq eruh wewieurw eryoerui wefhnounqweur whernncvqei e weuro weor e riweierywuie wiehkjld we

Estão entendendo?

Podemos continuar?

O que é desenvolvimento de sofware ?

?

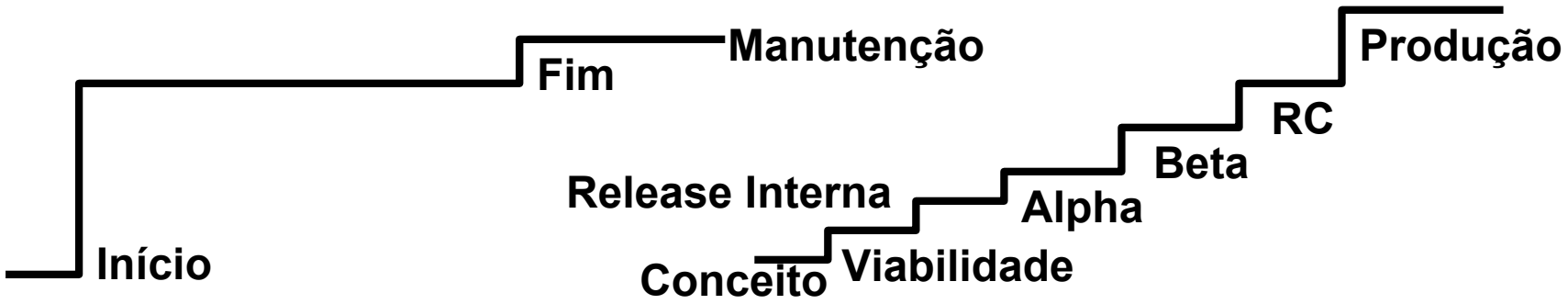
O que é desenvolvimento de sofware ?

- Por: Alistair Cockburn

Modelagem	(Jacobson)
Engenharia	(Meyer)
Disciplina	(Humphreys)
Poesia	(Cockburn)
Artesanato	(Knuth)
Arte	(Gabriel)

Desenvolver software é um processo reprodutível ?

- Processo empírico ou definido?
- Analogias:
  - Construção Cívil?
  - Preparar uma Receita?
  - Fábrica?
  - Obra de Arte?
- Gerenciado como produto ou projeto?



Novos ventos no mundo do Desenvolvimento de Software

- Sociedade demanda
  - grande quantidade de sistemas/aplicações
  - software complexo, sistemas distribuídos, heterogêneos
  - requisitos mutantes (todo ano, todo mês, todo dia)
- Mas, infelizmente,
  - não há gente suficiente para desenvolver tanto software com qualidade.

Problemas

- Com métodos tradicionais de desenvolvimento
  - Supõem que é possível prever o futuro
  - Pouca interação com os clientes
  - Ênfase em burocracias
    - (documentos, formulários, processos, controles rígidos, etc.)
  - Avaliação do progresso baseado na evolução da burocracia e não do código
- Com software
  - Grande quantidade de erros
  - Falta de flexibilidade

Como resolver esse impasse?

- Melhores Tecnologias
  - Padrões de Projeto (reutilização de idéias)
  - Componentes (reutilização de código)
  - Middleware (aumenta a abstração)
- Melhores Metodologias
  - Métodos Ágeis
- Colaboração e Comunicação

Métodos Ágeis de Desenvolvimento de Software

- Movimento iniciado por programadores experientes e consultores em desenvolvimento de software.
- Questionam e se opõem a uma série de mitos/práticas adotadas em abordagens tradicionais de Engenharia de Software e Gerência de Projetos.
- Manifesto Ágil:
  - Assinado por 17 desenvolvedores em Utah em fevereiro/2001.
  - http://agilemanifesto.org

O Manifesto do *Desenvolvimento Ágil de Software*

- ***Indivíduos e interações*** são mais importantes que *processos e ferramentas.*
- ***Software funcionando*** é mais importante do que *documentação completa e detalhada*.
- ***Colaboração com o cliente*** é mais importante do que *negociação de contratos*.
- **Adaptação a mudanças** é mais importante do que *seguir o plano inicial*.

Princípios do Manifesto Ágil

- Objetivo: satisfazer o cliente entregando, rapidamente e com freqüência, sistemas com algum valor.
  - Entregar versões funcionais em prazos curtos.
  - Estar preparado para requisitos mutantes.
  - Pessoal de negócios e desenvolvedores juntos.
  - Troca de informações através de conversas diretas.

Mais princípios do Manifesto Ágil

- Medir o progresso através de software funcionando.
- Desenvolvimento sustentável.
- Construir projetos com indivíduos motivados.
- Simplicidade é essencial.
- De tempos em tempos, o time pensa em como se tornar mais eficiente e ajusta o seu comportamento de acordo.

Em um projeto ágil ideal…. (1/2)

- O gerente de projeto concorda em prosseguir sem que todos os requisitos estejam bem definidos.
- Os desenvolvedores concordam em prosseguir sem ter todos os requisitos documentados.
- Os membros da equipe sabem que alguém vai ajudar quando ocorrerem problemas.

Em um projeto ágil ideal….(2/2)

- Os gerentes percebem que não precisam dizer à equipe o que fazer, ou garantir o que vai ser feito.
- A equipe percebe que ninguém vai dizer o que fazer, isto faz parte do trabalho da equipe.
- Não existe mais a impressão de divisão (*us and them*).

Enquanto isso, no mundo tradicional...

0. Levantamento de Requisitos

1. Análise de Requisitos

2. Desenho da Arquitetura

3. Implementação

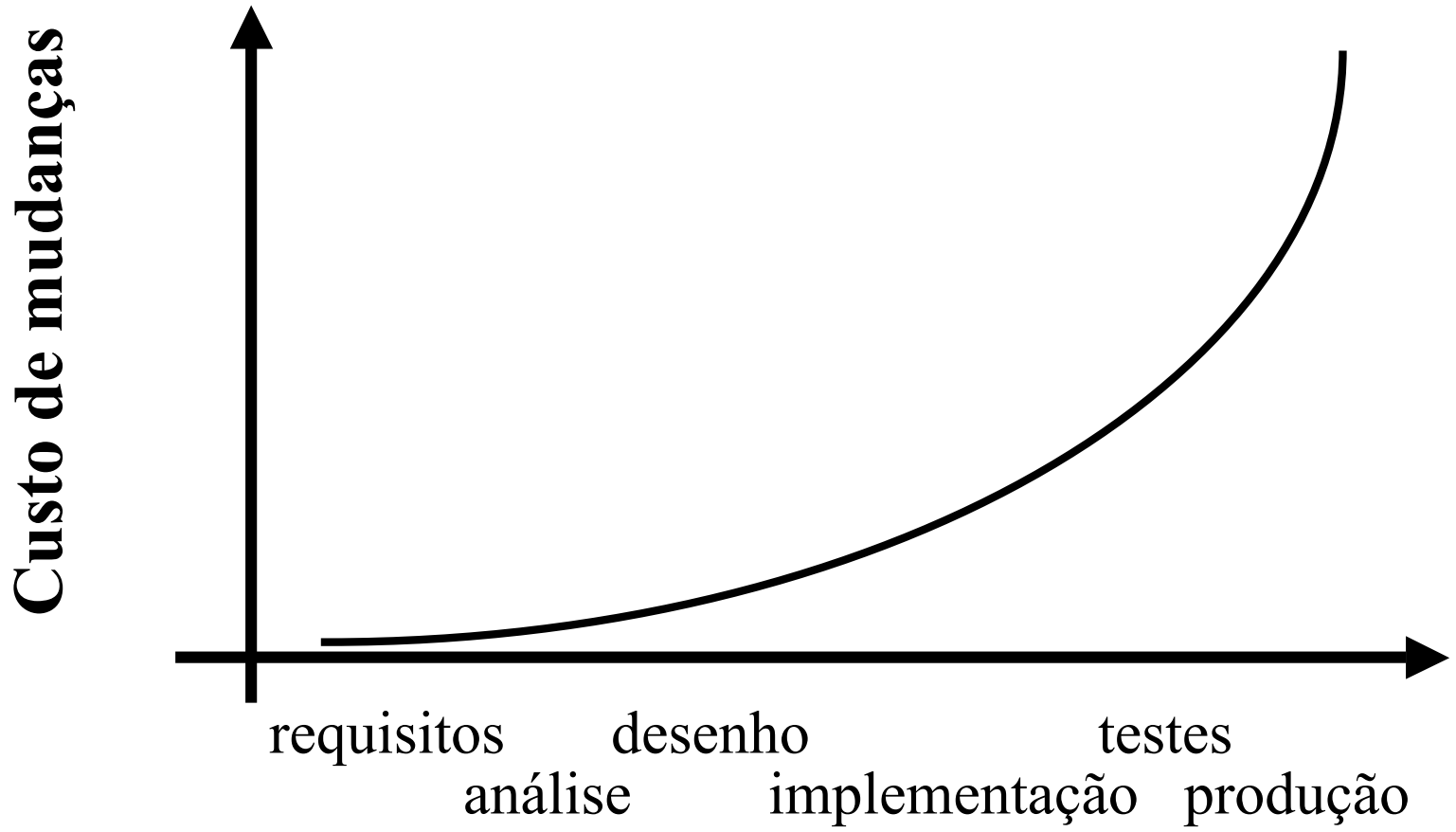
4. Testes

5. Produção / Manutenção

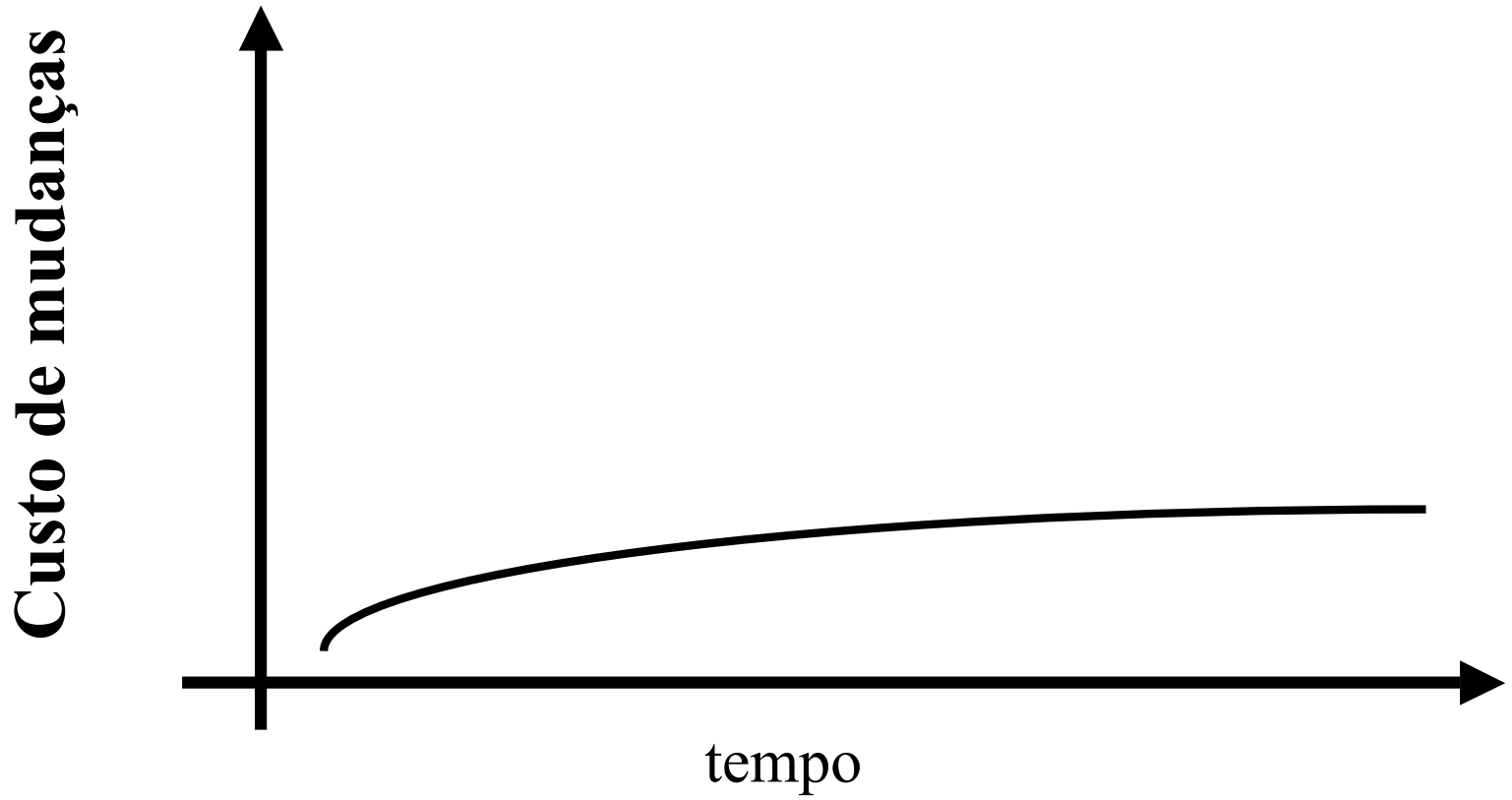
Premissas Básicas do Modelo Tradicional

- É necessário fazer uma análise de requisitos profunda e detalhada antes de projetar a arquitetura do sistema.
- É necessário fazer um estudo minucioso e elaborar uma descrição detalhada da arquitetura antes de começar a implementá-la.
- É necessário testar o sistema completamente antes de mandar a versão final para o cliente.

O que está por trás deste modelo?



E se a realidade hoje em dia fosse outra?



E se essa fosse a realidade?

- A atitude dos desenvolvedores de software seria completamente diferente:
  - Tomaríamos as grandes decisões o mais tarde possível.
  - Implementaríamos agora somente o que precisamos *agora.*
  - Não implementaríamos flexibilidade desnecessária.
  - Não anteciparíamos necessidades.

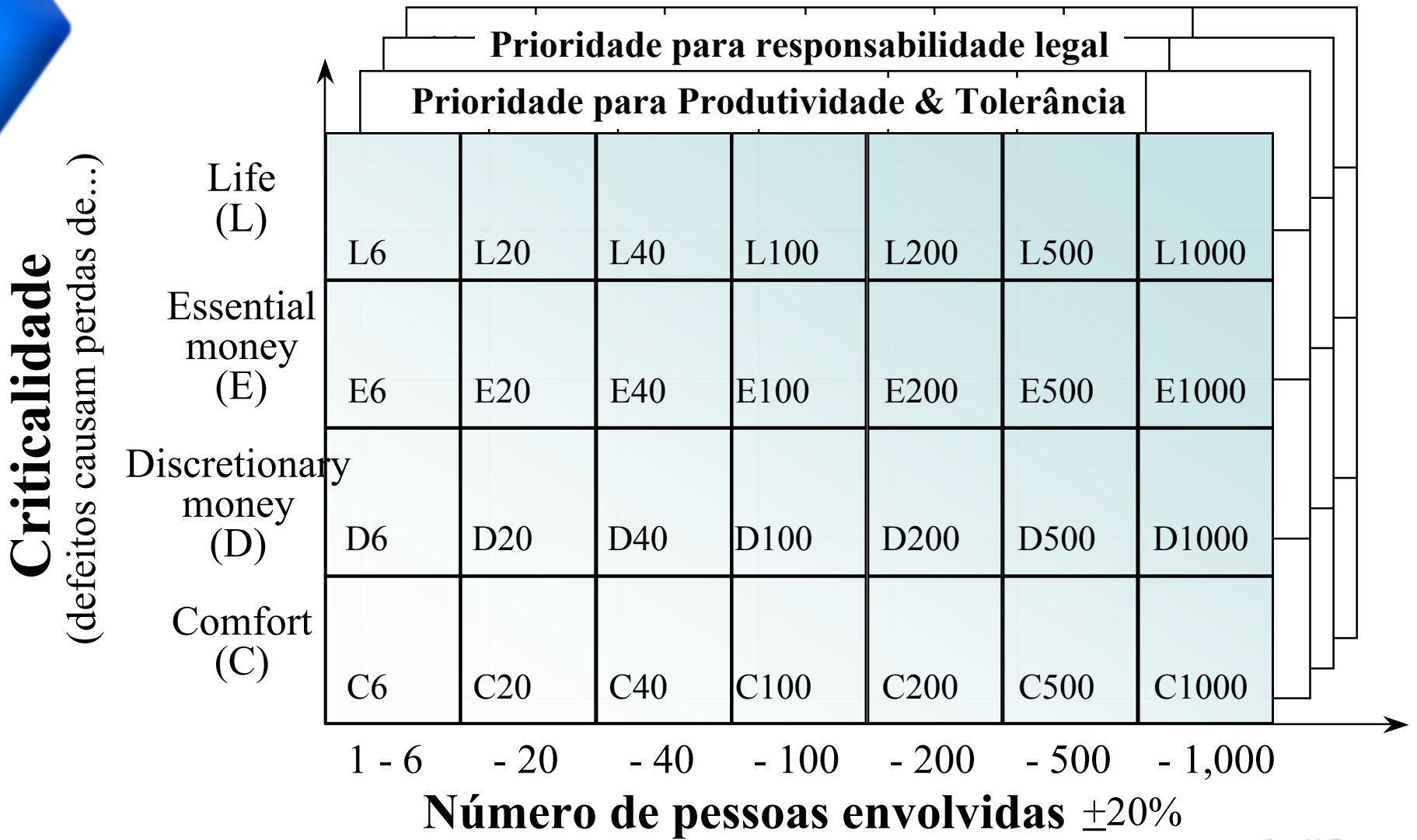
E essa é a nova realidade !
(pelo menos em muitos casos)

- **Orientação a Objetos**: facilita e cria oportunidades para mudanças.
- **Técnicas de Refatoração**.
- **Testes automatizados**: nos dão segurança quando fazemos mudanças.
- **Prática / cultura de mudanças**: aprendemos técnicas e adquirimos experiência em lidar com código mutante.

Principais Métodos Ágeis

- Programação eXtrema (XP)
- Crystal (uma família de métodos)
- Scrum
- Adaptive Software Development
- Feature Driven Development
- Lean Software Development
- etc.

Classificação de Projetos de Desenvolvimento de Software



Principais Características dos Métodos Ágeis

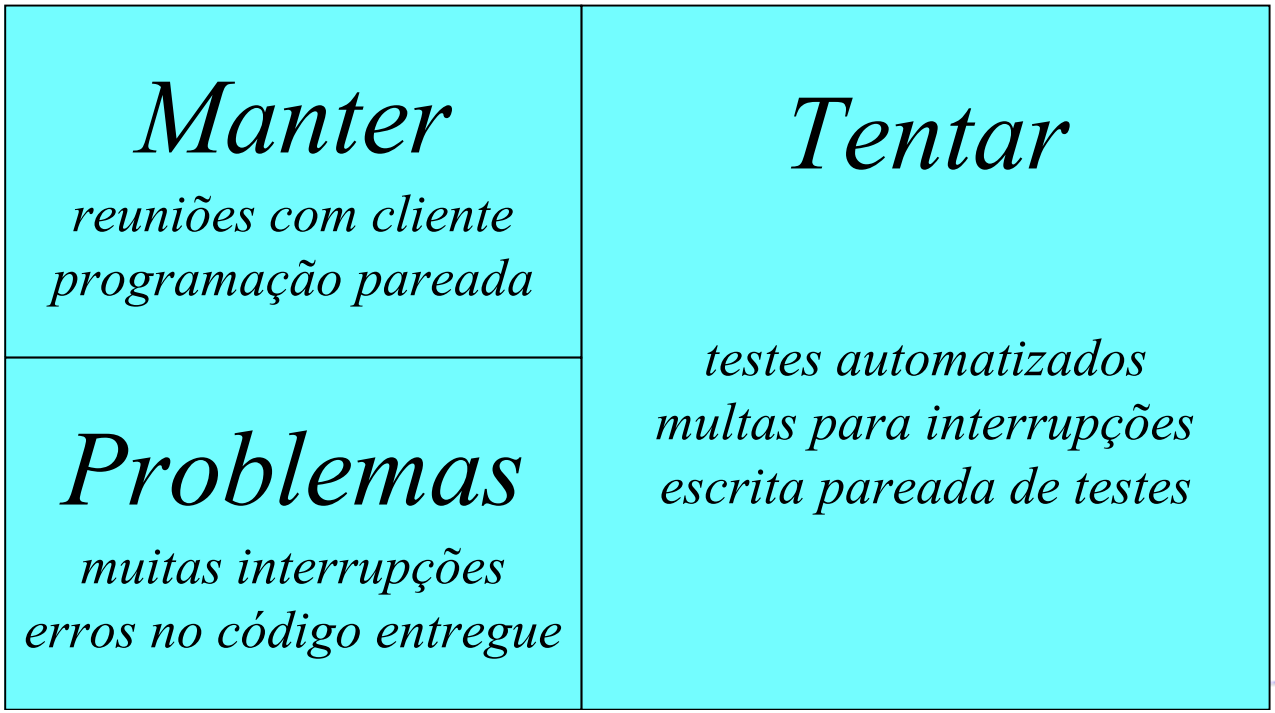
- Entregar valor
- Ênfase em comunicação
- Eliminar desperdícios
- Entrega freqüente de código funcionando
- Aproveitar as habilidades das pessoas
- Alta qualidade
- Melhorar constantemente

Adaptação da Metodologia

- Em cada caso, escolha a metodologia mais leve possível que pode fazer o que você precisa.
- Quanto maior o projeto (número de pessoas), maior burocracia será necessária e pior será a produtividade.
- *Retrospectivas* ao final de cada iteração.

Oficinas de Reflexão (*Reflection Workshops*)

- Perguntas:
  - O que aprendemos na última iteração?
  - O que podemos fazer melhor?
- Resultado:
  - pôster



Mais perguntas na reflexão

- O que fizemos de bom e de ruim?
- Quais são as nossas prioridades?
- O que mantivemos de mais importante?
- O que podemos mudar para a próxima vez?
- O que podemos adicionar/tirar?

- Após 2 ou 3 versões incrementais, a metodologia deve começar a convergir para uma metodologia tolerável para o projeto.

Programação eXtrema
XP

- Metodologia de desenvolvimento de software aperfeiçoada nos últimos 8 anos.
- Ficou popular a partir de 2000.
- Kent Beck.

Reações a XP

- Alguns odeiam, outros amam.

- Quem gosta de programar ama!
- Deixa o bom programador livre para fazer o que ele faria se não houvesse regras.
- Força o [mau] programador a se comportar de uma forma similar ao bom programador.

“XP não é pra mim”

- Você ainda pode aprender muito sobre desenvolvimento de software.
- Terá elementos para repensar a sua metodologia.
- No início se dizia:
  - “Ou você é 100% eXtremo ou não é eXtremo. Não dá prá ser 80% XP.”
- Hoje não é mais necessariamente assim.

Equipe de XP

- Papéis:
  - Programadores (sem hierarquia)
  - “Treinador” ou “Técnico” (*coach*)
  - “Acompanhador” (*tracker*)
  - Cliente

Os Valores de XP

- Comunicação
- *Feedback*
- Coragem
- Simplicidade
- Respeito

As 12 Práticas de XP (versão 2000)

1. Planejamento
2. Fases Pequenas
3. Metáfora
4. Design Simples
5. Testes
6. Refatoração
7. Programação Pareada
8. Propriedade Coletiva
9. Integração Contínua
10. Semana de 40 horas
11. Cliente junto aos desenvolvedores
12. Padronização do código

Princípios Básicos de XP

- *Feedback* rápido
- Simplicidade é o melhor negócio
- Mudanças incrementais
- Carregue a bandeira das mudanças / não valorize o medo (*Embrace change*)
- Alta qualidade do código

As 4 Variáveis do Desenvolvimento de Software

- **Tempo**
- **Custo**
- **Qualidade**
- **Escopo**

Um Projeto XP

- Fase de Exploração
  - termina quando a primeira versão do software é enviada ao cliente.
  - clientes escrevem “historias” (*story cards*).
  - programadores interagem com clientes e discutem tecnologias.
  - Não só discutem, **experimentam** diferentes tecnologias e arquiteturas para o sistema.
  - Planejamento: 1 a 2 dias.

Um Dia na Vida de um Programador XP

- Escolhe uma história do cliente.
- Procura um par livre.
- Escolhe um computador para programação pareada.
- Seleciona um “cartão de história” contendo uma tarefa claramente relacionada a uma característica (*feature*) desejada pelo cliente.

Um Dia na Vida de um Programador XP

- Discute modificações recentes no sistema
- Discute história do cliente
- Testes
- Implementação
- Projeto (*design*)
- Integração

Um Dia na Vida de um Programador XP

- Atos constantes no desenvolvimento:
  - Executa testes antigos.
  - Busca oportunidades para simplificação.
  - Modifica desenho e implementação incrementalmente baseado na funcionalidade exigida no momento.
  - Escreve novos testes.
  - Enquanto todos os testes não rodam a 100%, o trabalho não está terminado.
  - Integra novo código ao repositório.

Testes

- Fundamento mais importante de XP.
- É o que dá segurança e coragem ao grupo.
- Testes de unidades *(Unit tests)*
  - escritos pelos programadores para testar cada elemento do sistema (p.ex., cada método em cada caso).
- Testes de aceitação *(Acceptance tests)*
  - escritos pelos clientes para testar a funcionalidade do sistema.

Testes

- Testes das unidades do sistema
  - Tem que estar sempre funcionando a 100%.
  - São executados várias vezes por dia.
  - Executados à noite automaticamente.
- Testes de aceitação
  - Vão crescendo de 0% a 100%.
  - Quando chegam a 100% acabou a iteração.

O Código

- Padrões de estilo adotados pelo grupo inteiro.
- O mais claro possível.
  - nomes claros (*intention-revealing names*)
  - XP não se baseia em documentações detalhadas e extensas (perde-se sincronia).
- Comentários sempre que necessários.
- Comentários padronizados.
- Programação Pareada ajuda muito!

Programação Pareada

- Erro de um detectado imediatamente pelo outro.
- Maior diversidade de idéias, técnicas, algoritmos.
- Enquanto um escreve, o outro pensa em contra-exemplos, problemas de eficiência, etc.
- Vergonha de escrever código feio na frente do seu par.
- Pareamento de acordo com especialidades.
- Experimentos controlados mostraram que a qualidade do código produzido aumenta sem perda de velocidade
  [Laurie Williams]

Propriedade Coletiva do Código

- Modelo tradicional: só autor de uma função pode modificá-la.
- XP: o código pertence a todos.
- Se alguém identifica uma oportunidade para simplificar, consertar ou melhorar código escrito por outra pessoa, que o faça.
- Mas rode os testes!

Refatoração *(Refactoring)*

- Uma [pequena] modificação no sistema que não altera o seu comportamento funcional, mas que melhora alguma qualidade não-funcional:
  - simplicidade
  - flexibilidade
  - clareza

Exemplos de Refatoração

- Mudança do nome de variáveis
- Mudanças nas interfaces dos objetos
- Pequenas mudanças arquiteturais
- Encapsular código repetido em um novo método
- Generalização de métodos
  - `raizQuadrada(float x)⇒ raiz(float x, int n)`

Cliente

- Responsável por escrever “histórias”.
- Define prioridades do negócio.
- Decide quando as histórias estão prontas.
- Trabalha no mesmo espaço físico do grupo.
- Novas versões são enviadas para produção frequentemente.
- *Feedback* do cliente é essencial.
- Requisitos mudam.

Coach (treinador)

- Em geral, o mais experiente do grupo.
  - (eXPeriente em metodologia, não na tecnologia)
- Identifica quem é bom no que.
- Lembra a todos as regras do jogo (XP).
- Também faz programação pareada.
- Não desenha arquitetura, apenas chama a atenção para oportunidades de melhorias.
- Seu papel diminui à medida em que o time fica mais maduro.

Tracker (Acompanhador)

- A “consciência” do time.
- Coleta estatísticas sobre o andamento do projeto. Alguns exemplos:
  - Número de histórias definidas e implementadas.
  - Número de testes de unidade.
  - Número de testes funcionais definidos e funcionando.
  - Número de classes, métodos, linhas de código.
- Mantém histórico do progresso.

Quando XP Não Deve Ser Experimentada?

- Quando os valores não estão alinhados
- Quando o cliente não aceita as regras do jogo.
- Quando o cliente quer uma especificação detalhada do sistema antes de começar.
- Quando os programadores não estão dispostos a seguir (todas) as regras.
- Se (quase) todos os programadores do time são medíocres.

Quando XP Não Deve Ser Experimentada?

- Quando *feedback* rápido não é possível:
  - sistema demora 6h para compilar.
  - testes demoram 12h para rodar.
  - exigência de certificação que demora meses.

- Quando não é possível realizar testes (muito raro).

Conclusão:
Ser Ágil = Vencer Medos

- Escrever código.
- Mudar de idéia.
- Ir em frente sem saber tudo sobre o futuro.
- Confiar em outras pessoas.
- Mudar a arquitetura de um sistema em funcionamento.
- Escrever testes.

As 12 Práticas de XP (versão 2000)

1. Planejamento
2. Fases Pequenas
3. Metáfora
4. Design Simples
5. Testes
6. Refatoramento
7. Programação Pareada
8. Propriedade Coletiva
9. Integração Contínua
10. Semana de 40 horas
11. Cliente junto aos desenvolvedores
12. Padronização do código

Práticas de XP

- As práticas podem ser adaptadas com base nos valores e princípios:
  - Conserte XP quando a metodologia quebrar.
  - Mova as pessoas.
  - Daily Stand-up Meeting
  - Retrospectivas
  - Personas
  - Entre outras…

Conclusão

Todos podemos aprender com XP

- melhorando a qualidade do software
- melhorando a produtividade

Características Comuns dos Métodos Ágeis

- Coloca o foco:
  - Na entrega freqüente de sub-versões do software [funcionando] para o cliente.
  - Nos seres humanos que desenvolvem o software.
- Retira o foco de:
  - Processos rígidos e burocratizados.
  - Documentações e contratos detalhados.
  - Ferramentas que são usadas pelos seres humanos.

Maiores Informações

AgilCoop:

`www.agilcoop.org.br`

Dairton - dairton@ime.usp.br
Danilo - dtsato@ime.usp.br

Outros:

`www.agilealliance.org`