

Engenharia de Software

Seção 5 – Métodos Ágeis

Objetivos

O aluno deverá reconhecer os conceitos básicos dos Métodos Ágeis, tomando contato com alguns modelos de desenvolvimento.



Contexto histórico

- ✓ Em meados da década de 90, os processos baseados em modelo de desenvolvimento cascata (waterfall) sofriam muitos ataques.
- ✓ Desenvolvedores questionavam o “peso” destes processos, por serem pesadamente regulados e regimentados, microgerenciados e baseados em uma sequência de atividades muito engessada.
- ✓ Procuravam-se métodos mais “leves” para o desenvolvimento de software.

Contexto histórico (continuação)

- ✓ Algumas metodologias surgidas durante a década de 90 foram:
 - ✓ Scrum
 - ✓ Crystal Clear
 - ✓ Extreme Programming
 - ✓ Adaptive Software Development
 - ✓ Feature Driven Development
 - ✓ Dynamic Systems Development Model
- ✓ Estes métodos, inicialmente denominados “leves”, passaram também a se chamar “Ágeis”.

O Manifesto Ágil

Em fevereiro de 2001, 17 desenvolvedores se encontraram em um resort em Utah, EUA, e publicaram o Manifesto para Desenvolvimento Ágil de Software:

“Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:

- ✓ *Indivíduos e interações* mais que processos e ferramentas
- ✓ *Software em funcionamento* mais que documentação abrangente
- ✓ *Colaboração com o cliente* mais que negociação de contratos
- ✓ *Responder a mudanças* mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.” (www.agilemanifesto.org)

Princípios do Manifesto

Os signatários do Manifesto Ágil seguem a 12 princípios:

- ✓ A maior prioridade é satisfazer o cliente com a entrega contínua e adiantada de software com valor agregado.
- ✓ Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.

Princípios do Manifesto

(continuação)

- ✓ Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
- ✓ Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.



Princípios do Manifesto

(continuação)

- ✓ Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
- ✓ O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.

Princípios do Manifesto

(continuação)

- ✓ Software funcionando é a medida primária de progresso.
- ✓ Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.

Princípios do Manifesto

(continuação)

- ✓ Contínua atenção à excelência técnica e bom design aumenta a agilidade.
- ✓ Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.
- ✓ As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
- ✓ Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Por que agilidade?

- ✓ Em essência, métodos ágeis foram desenvolvidos para superar as fraquezas (reais ou aparentes) na engenharia de software convencional.
- ✓ Desenvolvimento ágil pode prover benefícios importantes, mas não é aplicável a todos os produtos, projetos, pessoas e situações.

Por que agilidade? (continuação)

- ✓ No mundo atual, é quase impossível prever como um sistema evoluirá com o tempo, pois as condições de mercado mudam, necessidades de clientes evoluem, e novas ameaças competitivas emergem sem aviso.
- ✓ Engenheiros de software precisam ser ágeis para responder a ambientes de negócios fluidos.

Por que agilidade? (continuação)

- ✓ Mas isso não quer dizer que tudo o que conhecemos de Engenharia de Software deva ser jogado fora...
- ✓ Segundo alguns autores, os métodos de engenharia de software tradicionais esqueceram-se da fragilidade das pessoas que constroem software.



Por que agilidade? (continuação)

- ✓ Engenheiros de software possuem diferentes estilos, níveis de habilidade, criatividade, etc. Os modelos tradicionais não levam estas diferenças em consideração, logo são da mesma forma frágeis.
- ✓ Métodos que enfatizam a tolerância no lugar da disciplina são mais fáceis de serem adotadas e sustentadas pelos desenvolvedores, mas podem ser menos produtivos.

Por que agilidade? (continuação)

Qualquer processo ágil de software é caracterizado de modo a atender a três suposições-chave:

1. É difícil prever antecipadamente quais requisitos de software vão **persistir** e quais serão modificados. Também é difícil prever como as prioridades do cliente vão mudar ao longo do desenvolvimento.
2. Para muitos tipos de software, o projeto e a construção são **intercalados**, para se comprovar os modelos de projeto.
3. Análise, projeto, construção e testes **não são tão previsíveis** (do ponto de vista do planejamento) quanto gostaríamos.

Processos ágeis

- ✓ Como criar um processo que possa gerenciar a imprevisibilidade?
 - ✓ Criando um processo adaptável a modificações rápidas do processo e de condições técnicas.
- ✓ Mas a adaptação contínua sem progresso resulta pouco. Como garantir o progresso?
 - ✓ Adaptando o processo incrementalmente, com feedback do cliente, catalisado por **protótipos operacionais**.
- ✓ **Incrementos de software** devem ser entregues em curtos períodos de tempo de modo que a adaptação se sincronize com as modificações.

Fatores humanos

- ✓ No desenvolvimento ágil, fatores pessoais são muito importantes, **moldando o processo às equipes**.
- ✓ Para que isto seja possível, várias características da equipe deverão estar presentes:
 - ✓ **Competência**: Habilidade e conhecimento do processo devem ser ensinadas a todos os membros da equipe.
 - ✓ **Foco comum**: Todos devem estar focados em uma meta – entregar incremento ao cliente – adaptando o processo às necessidades.



Fatores humanos (continuação)

- ✓ **Colaboração:** Para realizar as tarefas de engenharia de software, os membros da equipe devem colaborar – entre si, com o cliente e com o gerente.
- ✓ **Capacidade de tomada de decisão:** A equipe deve ter autonomia para tomar decisões de âmbito técnico e de projeto.



Fatores humanos (continuação)

- ✓ **Habilidade em resolver problemas vagos:** A equipe terá que lidar continuamente com ambiguidades e modificações.
- ✓ **Respeito e confiança mútua:** A equipe deve ser consolidada, de forma que o “todo seja maior que a soma das partes”.
- ✓ **Auto-organização:** A equipe se organiza para o trabalho a ser feito, organiza o processo para melhor acomodar o ambiente local e organiza o cronograma de trabalho para conseguir melhor entrega do incremento de software.

Modelos Ágeis

- ✓ Ao longo da história, sempre houve diversas metodologias e descrições de processo, métodos de modelagem e notações, ferramentas e tecnologias.
- ✓ Cada uma delas teve seu período de glória e depois foi eclipsada por algo mais novo e aparentemente melhor.
- ✓ O movimento ágil segue o mesmo caminho histórico, apresentando diversos modelos, todos eles satisfazendo, em maior ou menor grau, o Manifesto para o Desenvolvimento Ágil e seus princípios.

Extreme Programming (continuação)

- ✓ **Planejamento:** Criação de conjunto de histórias (**histórias de usuários**) , que descrevem as características e funcionalidades do software. Cada história, escrita pelo cliente, recebe por ele uma **prioridade**.
- ✓ A equipe avalia cada história e lhe atribui um **custo**, em semanas de desenvolvimento. Se for maior que três semanas, pede-se ao cliente para dividi-la.

Extreme Programming (continuação)

- ✓ Agrupam-se as histórias e decide-se como desenvolver o primeiro incremento:
 - ✓ Implementar todas as histórias, ou
 - ✓ Implementar primeiro as mais prioritárias, ou
 - ✓ Implementar primeiro as mais arriscadas.

Extreme Programming (continuação)

- ✓ **Projeto:** Segue o princípio KIS (Keep It Simple).
- ✓ Um projeto simples é sempre preferível a uma representação mais complicada. Histórias são implementadas como estão escritas. Projetos de funcionalidades que podem ser necessárias no futuro são desencorajados.

Extreme Programming (continuação)

- ✓ Usa-se cartões CRC (Classe – Responsabilidade – Colaboração), que são o único produto de projeto que é realizado.
- ✓ Se é encontrado um problema difícil de projeto, é encorajada a criação de um protótipo operacional daquela parte do projeto.

Extreme Programming (continuação)

- ✓ **Refactoring:** (Ou Refabricação) é o processo de alterar um sistema de software de tal modo que não altere o comportamento externo, mas aperfeiçoe a estrutura interna.
- ✓ Como o projeto XP praticamente não gera notações, e produz apenas cartões CRC e protótipos, o projeto é visto como um artefato provisório que pode e deve ser modificado à medida que a construção prossegue.
- ✓ O esforço necessário ao refactoring pode crescer sensivelmente à medida em que a aplicação também cresce.

Extreme Programming (continuação)


- ✓ **Construção:** Antes de codificar, são gerados testes unitários para exercitar as histórias projetadas.
- ✓ O desenvolvedor estaria, então, mais preparado para focalizar o que precisa ser implementado para passar no teste unitário.



Extreme Programming (continuação)

- ✓ Programação em pares: Duas pessoas trabalhando juntas em uma mesma estação de trabalho:
 - ✓ Solução rápida de problemas
 - ✓ Melhor qualidade de código
 - ✓ Mais foco no trabalho
- ✓ **Integração**; Realizada pelos pares ou por uma outra equipe.

Extreme Programming (continuação)

- ✓ **Testes:** Testes são criados de forma que possam ser automatizados.
 - ✓ São encorajados Testes de Regressão, do software completo, sempre que o código é modificado (o que é frequente).
- 
- A decorative wavy line in a light gray color, spanning the width of the slide and positioned near the bottom.

Extreme Programming (continuação)

- ✓ O teste de integração e validação ocorre diariamente, dando à equipe uma indicação contínua de progresso e levantando sinais de alerta caso o produto se esteja deteriorando.
- ✓ “Resolver pequenos problemas a cada intervalo de umas poucas horas leva menos tempo do que resolver grandes problemas perto da data de entrega.”

Desenvolvimento

Adaptativo de Software (continuação)

- ✓ DAS (ou *ASD – Adaptive Software Development*) foi proposto como uma técnica para a construção de sistemas e software complexos.
- ✓ O apoio filosófico do DAS concentra-se na colaboração humana e na auto-organização da equipe.
- ✓ **Especulação**: Planejamento do ciclo adaptativo, usando informações providas pelo cliente, restrições de projeto e requisitos básicos para determinar o conjunto de ciclos de versão (incrementos de software) que serão necessários para o projeto.

Desenvolvimento

Adaptativo de Software (continuação)

- ✓ **Colaboração:** Pessoal motivado trabalha junto de modo que multiplica seus talentos e resultados criativos além de seu número absoluto.
- ✓ Colaboração não é fácil. Exige:
 - ✓ Comunicação
 - ✓ Equipe “aglutinada”
 - ✓ Criatividade
 - ✓ Confiança

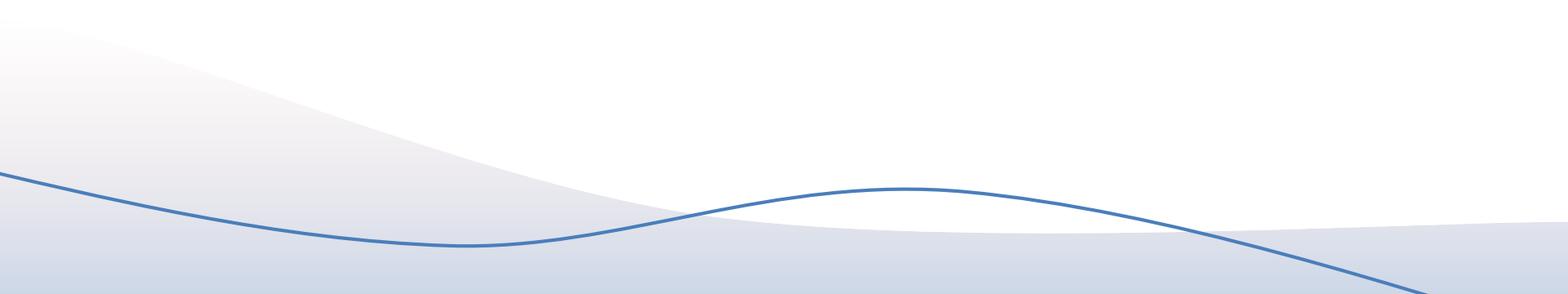
Desenvolvimento Adaptativo de Software (continuação)

- ✓ **Aprendizado:** À medida em que o trabalho progride, os desenvolvedores aprendem de três formas:
 - ✓ **Foco nos grupos:** clientes e/ou usuários finais fornecem feedback sobre os incrementos de software que são entregues.
 - ✓ **Revisões técnicas formais:** Os componentes são revisados pela equipe, que aprendem aperfeiçoando o produto.
 - ✓ **Pós-conclusão:** A equipe cuida do próprio desempenho e processo, com a intenção de aprender e depois aperfeiçoar sua abordagem.

DSDM

- ✓ DSDM (*Dynamic Systems Development Method* – Método de Desenvolvimento Dinâmico de Sistemas) é uma abordagem ágil que se baseia em uma prototipação incremental em um ambiente controlado.
- ✓ Semelhante a RAD, aplica o princípio de Pareto (80% do valor de uma aplicação pode ser entregue em 20% do tempo que levaria para desenvolver a aplicação inteira).

DSDM (continuação)

- ✓ A cada iteração, DSDM segue a regra dos 80%, ou seja, apenas um certo trabalho é necessário para que cada incremento facilite o avanço para o incremento seguinte.
 - ✓ Os detalhes restantes podem ser completados depois, quando mais requisitos forem conhecidos ou quando houver modificações.
- 

Ciclo de Vida DSDM

O Ciclo de Vida DSDM define três ciclos iterativos diferentes, precedidos por duas atividades adicionais:

- ✓ **Estudo de viabilidade:** estabelece requisitos básicos e restrições do negócio.
- ✓ **Estudo do negócio:** Estabelece requisitos funcionais e define a arquitetura básica da aplicação.

Ciclo de Vida DSDM (continuação)

- ✓ **Iteração do modelo funcional**: Produz protótipos incrementais que demonstram a funcionalidade.
- ✓ **Iteração de projeto e construção**: Revisita os protótipos garantindo que eles passaram por engenharia.
- ✓ **Implementação**: Coloca o último protótipo incremental no ambiente operacional.

