Desenvolvimento Ágil

Fernando Magno Quintão Pereira

8 de Novembro de 2010

Questão 1

Considere as seguintes contraposições:

- Iterações entre indivíduos $\times$ ferramentas e processos.
- O software que funciona $\times$ boa documentação.
- Colaboração com os clientes $\times$ contratos bem estabelecidos.
- Resposta rápida às contingências $\times$ plano bem estruturado de trabalho.

1. Quais os prós e contras do lado esquerdo e direito de cada um destes pontos?
2. Qual a metodologia de desenvolvimento de software adotada no lado direito?
3. E qual a metodologia de desenvolvimento adotada no lado esquerdo?

Questão 2

Em suas próprias palavras, o que é agilidade no desenvolvimento de *software*?

Questão 3

1. Porque processos iterativos facilitam a gerência de mudanças na especificação do *software*?
2. É possível que um processo possa ser terminado em apenas uma iteração, e ainda ser chamado ágil?

Princípios do Desenvolvimento Ágil: 1–6

1. *Software* deve ser liberado frequentemente, seja semanalmente, seja mensalmente. O tempo de liberação deve ser o mais curto possível.
2. Mudanças nos requisitos são esperadas e bem-vindas.
3. Desenvolvedores e clientes devem trabalhar juntos, se possível diariamente, durante o projeto.
4. Dado o suporte tecnológico necessário, os desenvolvedores devem ter flexibilidade para levar o projeto adiante.
5. Encontros face-a-face são a melhor forma de disseminar e sincronizar informações entre o time de desenvolvimento.
6. Boas práticas de projeto e programação devem ser conhecidas e aplicáveis sempre que possível.

Princípios do Desenvolvimento Ágil: 7–12

7. A métrica básica de sucesso é o *software* que funciona.
8. Desenvolvedores precisam prover testes que indiquem que o *software* está funcionando.
9. O desenvolvimento do *software* deve ser sustentável, isto é, patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante durante um tempo indefinido.
10. Simplicidade – a arte de maximizar a quantidade de trabalho evitada – é essensial.
11. Os times de desenvolvimento de *software* devem ser capazes de se auto-organizerem.
12. Os times devem ser capazes de se auto-ajustarem durante o processo de desenvolvimento, isto é, em intervalos regulares o time precisa refletir sobre como se tornar mais efetivo.

Questão 4

Seria possível citarmos mais algum princípio que ajudasse os desenvolvedores de *software* a serem ainda mais ágeis e flexíveis?

Questão 5

1. Porque requisitos mudam tanto? Será que as pessoas não sabem de fato o que elas querem?
2. Alguém poderia citar algum exemplo da "vida real"?

Encontros face a face

Métodos de desenvolvimento que se encaixam neste paradigma "ágil" em geral advogam encontros face-a-face entre os desenvolvedores. Por exemplo, o método *scrum* advogoa encontros diários de 15 minutos, em que são respondidas as três perguntas

1. O que você fez desde o último encontro?
2. Que obstáculos você tem encontrado.
3. O que você planeja fazer até o próximo encontro?

Questão 6

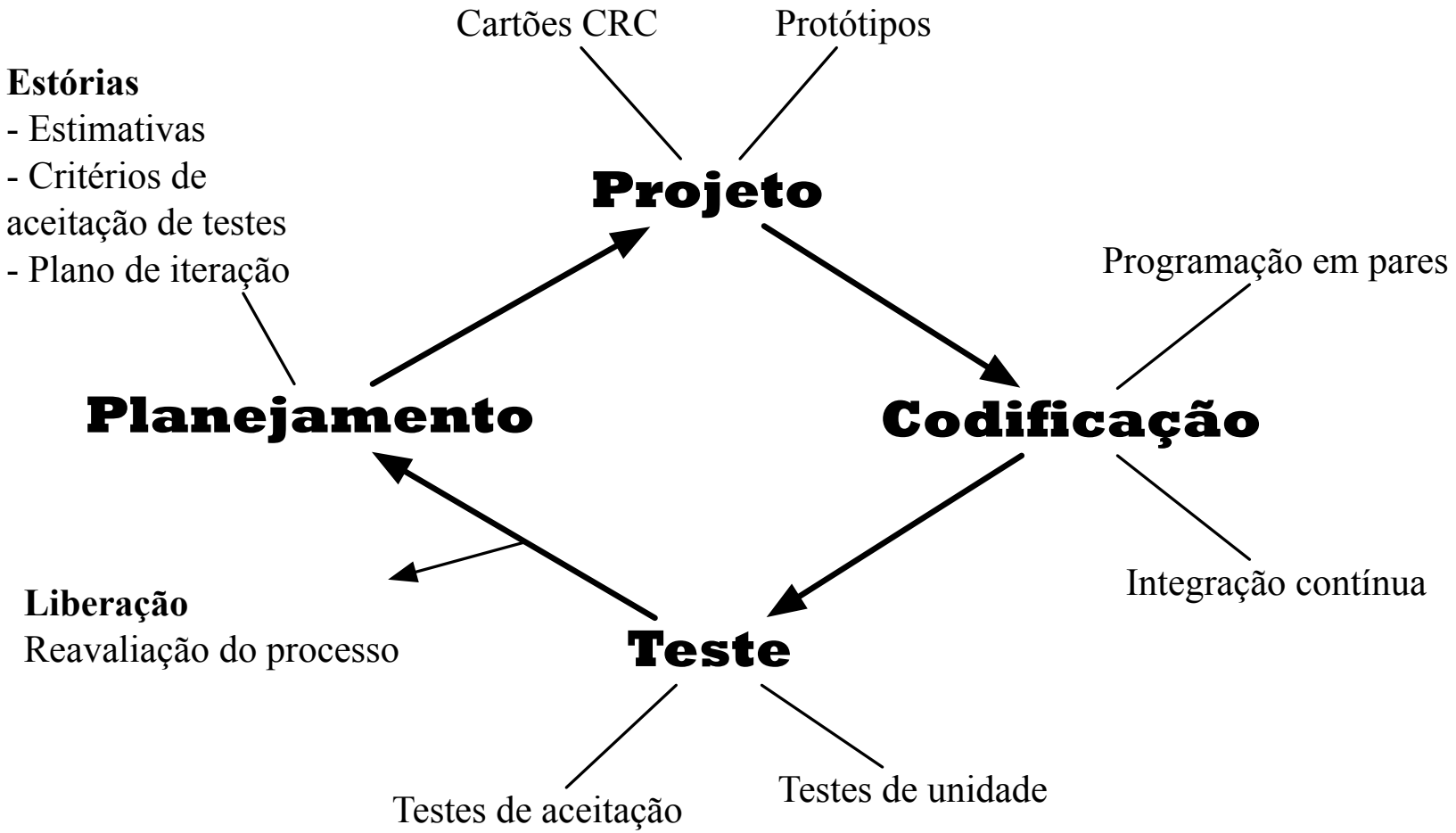
1. Qual a importância de encontros face-a-face?
2. É possível o desenvolvimento cooperativo de *software* se as partes cooperantes estão separadas geograficamente?
3. Alguém poderia compartilhar as práticas de reuniões que acontecem em seu próprio local de trabalho?

Questão 7

Refactoring é especialmente importante em desenvolvimento ágil.

1. O que é *refactoring*?
2. Vocês podem listar algumas técnicas de refatoração?
3. Porque esta importância especial?

X-treme Programming



Questão 8

1. O que são cartões CRC?
   1.1 Que tipo de diretrizes seguir para definir quais as classes compõem a aplicação e quais as responsabilidades de cada classe?
2. Qual é a vantagem da prototipagem?
3. E qual é o perigo da prototipagem?
4. O que é programação em pares?
5. A programação em pares não desperdiça um programador?

Questão 9

X-programming é um conceito relativamente novo, tendo surgido como tal em 1999. Esta metodologia de desenvolvimento de *software* recebeu muitas – muitas mesmo – críticas.

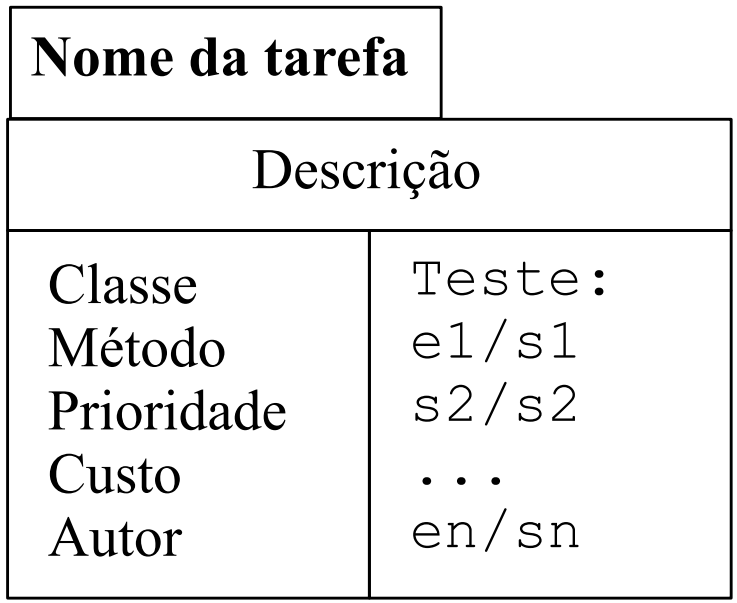
- Quais as possíveis desvantagens de *X-programming*?

Ainda assim, *X-programming* possui muitos fãs e proponentes, tanto na indústria quanto na academia.

- Alguém já participou de rodadas de *X-programming* em alguma empresa?

Task Oriented Programming

1. Desenhe um diagrama de classes básico, que descreva os componentes principais do sistema.
2. Crie tarefas a partir dos métodos projetados.
   - 2.1 Retire um cartão de tarefa da pilha.
   - 2.2 Resolva aquele cartão de tarefa.
   - 2.3 Adicione novas tarefas à pilha.
   - 2.4 Atualize o diagrama de classes com os componentes criados.



Exemplo

Determinar se mensagem é SPAM

Dada uma mensagem de e-mail, determine a probabilidade desta mensagem ser um SPAM.

Classe: `SpamFilter`
Método: `probSpam`
Prioridade: alto
Custo: alto
Autor: Fernando

```
Teste:
file:m1.txt/1.0
file:m2.txt/>0.5
file:m3.txt/<0.3
file:m4.txt/0.0
```

Questão 8

A diretoria de *Toy Inc.* está preocupada com o grande volume de spams que seus funcionários estão recebendo. Produza um filtro anti-spam para a companhia. Este filtro é um programa que recebe uma mensagem de e-mail, e separa esta mensagem em suas várias partes: remetente, mensagem, data, etc. Após analisar a mensagem, o filtro retorna um número real, entre 0 e 1, que indica a probabilidade de que aquela mensagem seja um spam. Há várias formas de se detectar um spam:

- A mensagem contém palavras chaves que ocorrem com freqüência em spams.
- A mensagem já foi vista antes, e já foi marcada como spam por um usuário do sistema.
- A mensagem provém de um remetente que é notório por enviar spams.