MÉTODOS ÁGEIS E SAAS

ENGENHARIA DE SISTEMAS DE INFORMAÇÃO

Daniel Cordeiro

11 de agosto de 2017

Escola de Artes, Ciências e Humanidades | EACH | USP

MANIFESTO ÁGIL, 2001

"Estamos descobrindo maneiras melhores para se desenvolver software e de ajudar as pessoas a fazê-lo. Ao longo desse trabalho viemos a valorizar mais:

- **Indivíduos e interações** *ao invés de processos e ferramentas*
- **Software que funciona** *ao invés de documentação extensa*
- **Colaboração com o cliente** *ao invés de negociação de contratos*
- **Resposta a mudanças** *ao invés de seguir um plano*

Isto é, enquanto há valor nos itens da direita, nós damos mais valor os itens da esquerda."

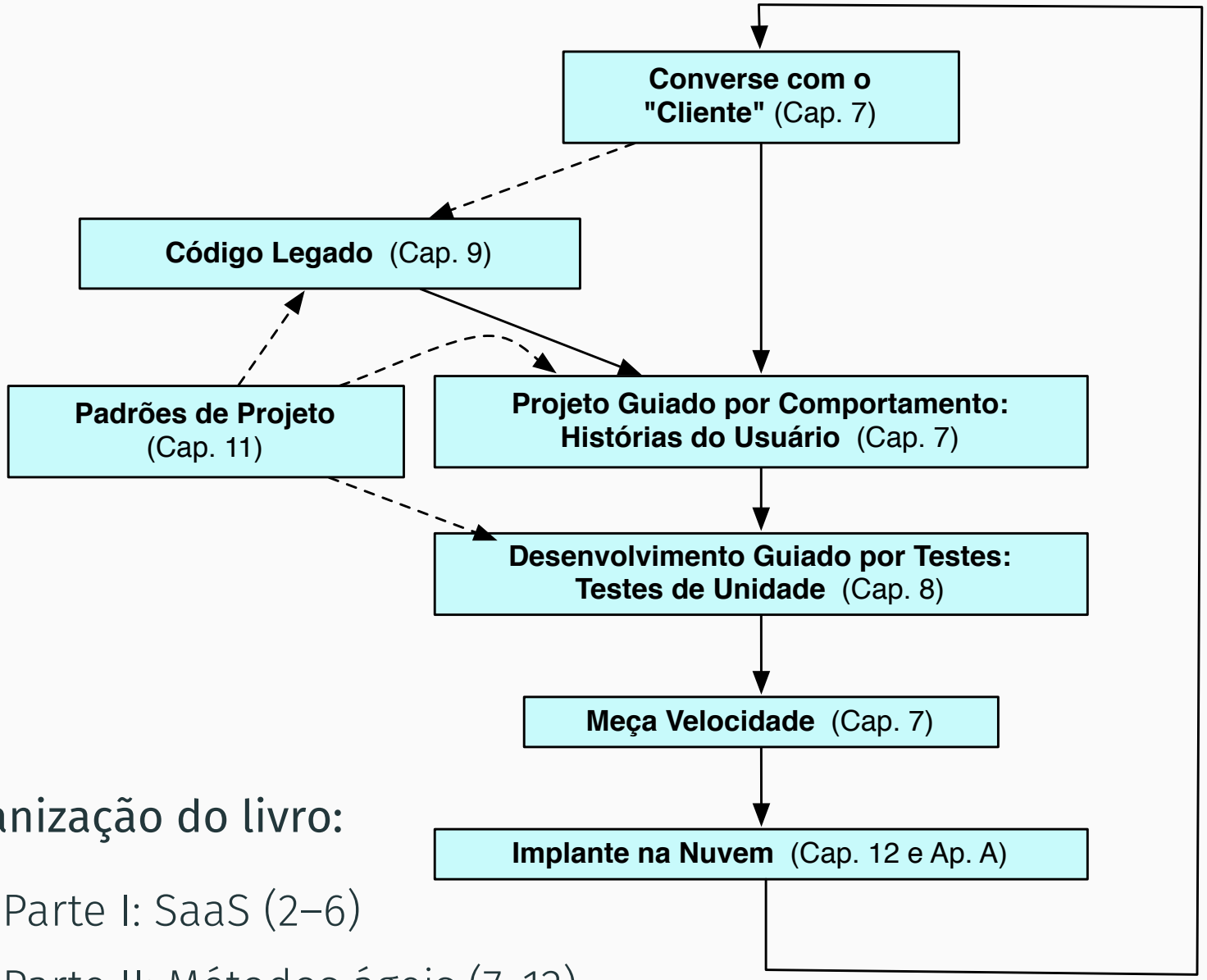
Programação extrema (XP), uma versão de método ágil

- Se iterações curtas são boas, faça que sejam as menores possíveis (semanas vs. anos)
- Se simplicidade é bom, sempre faça a coisa mais simples possível que possa fazer o trabalho
- Se testar é bom, teste o tempo todo. Escreva o código de teste antes mesmo de escrever o código que será testado
- Se revisões de código são boas, reveja código continuamente programando em pares, revezando quem ficará de olho no código do outro

CICLO DE VIDA ÁGIL

- Abrace mudanças como sendo um fato da vida: melhoria contínua vs. fases
- Desenvolvedores refinam continuamente um protótipo incompleto, mas funcional, até que o cliente esteja feliz com o resultado. *Feedback* do cliente em todas as iterações (cada 1–2 semanas)
- Métodos ágeis enfatizam que *Test-Driven Development* (TDD) diminuem os defeitos, *User Stories* validam os requisitos dos clientes e *Velocity* mede o progresso

Iterações de métodos ágeis / organização do livro-texto



Organização do livro:

- Parte I: SaaS (2–6)
- Parte II: Métodos ágeis (7–12)

MÉTODOS ÁGEIS: ANTES E AGORA

- Controverso em 2001

> *O Manifesto Ágil é uma tentativa a mais de minar a disciplina de engenharia de software. (...) Na profissão da engenharia de software existem engenheiros e hackers. (...) Me parece que isso não é nada mais do que uma tentativa de legitimar o comportamento do hacker. (...) A profissão de engenharia de software apenas mudará para melhor quando os clientes se recusarem a pagar por um software que não cumpre o esperado. (...) Mudar de uma cultura que encoraje a mentalidade do hacker para uma que seja baseada em práticas de engenharia de software previsíveis apenas ajudará a transformar a engenharia de software em uma disciplina de engenharia respeitada.*
>
> — *Steven Ratkin, "Manifesto Elicits Cynicism",* IEEE Computer, *2001*

MÉTODOS ÁGEIS: ANTES E AGORA

- Controverso em 2001
- Aceito em 2013
  - um estudo de 2012 com 66 projetos verificou que a maioria usava métodos Ágeis, mesmo com times distribuídos

SIM: PLANEJE-E-DOCUMENTE

NÃO: MÉTODOS ÁGEIS

1. É necessário criar uma especificação?
2. Os clientes estarão indisponíveis durante o desenvolvimento?
3. O sistema a ser construído é muito grande?
4. O sistema a ser construído é muito complexo (ex: sistema de tempo real)?
5. O sistema terá uma vida útil muito longa?
6. Você usa ferramentas de softwares pobres?
7. O time está geograficamente distribuído?
8. A cultura do time (seu modo de pensar e planejar) é orientado à documentação?
9. O time possui habilidades de programação fracas?
10. O sistema a ser construído está sujeito a regulações e normas?

Qual dessas afirmações é verdadeira?

- Uma grande diferença entre métodos Ágeis e P-e-D é que Ágil não usa requisitos
- Uma grande diferença entre métodos Ágeis e P-e-D é medir o progresso em relação a um plano
- Você pode construir SaaS usando métodos Ágeis, mas não usando P-e-D
- Uma grande diferença entre métodos Ágeis e P-e-D é a construção de protótipos e as interações com os clientes durante o processo de desenvolvimento

FALÁCIAS E ARMADILHAS

FALÁCIAS E ARMADILHAS

- Falácia: o ciclo de vida Ágil é o melhor para o desenvolvimento de software
  - Ágil é uma boa escolha para alguns tipos de software, especialmente SaaS
  - Mas não é boa escolha para a NASA; código sujeito a regulamentações
- Em cada tópico do livro veremos como aplicar métodos Ágeis na prática, mas também veremos como seria a perspectiva de Planeje-e-Documente
  - obs: você irá se deparar com novas metodologias de desenvolvimento ao longo da sua carreira, então prepare-se para aprender outras no futuro

FALÁCIAS E ARMADILHAS

- Armadilha: ignorar o custo do *design* de software
  - como o custo de fabricação de um software é ≈ 0, alguns podem acreditar que também não existe custo para modificá-lo do jeito que um cliente quiser
  - ignora o custo de *design* e teste
- Será que não ter custo de fabricação de software/dados não é a mesma ideia que justifica cópias piratas de software/dados? Ninguém deveria pagar pelo desenvolvimento, mas só pela fabricação?

RESUMO: ENGENHARIA DE SOFTWARE É MAIS DO QUE PROGRAMAR

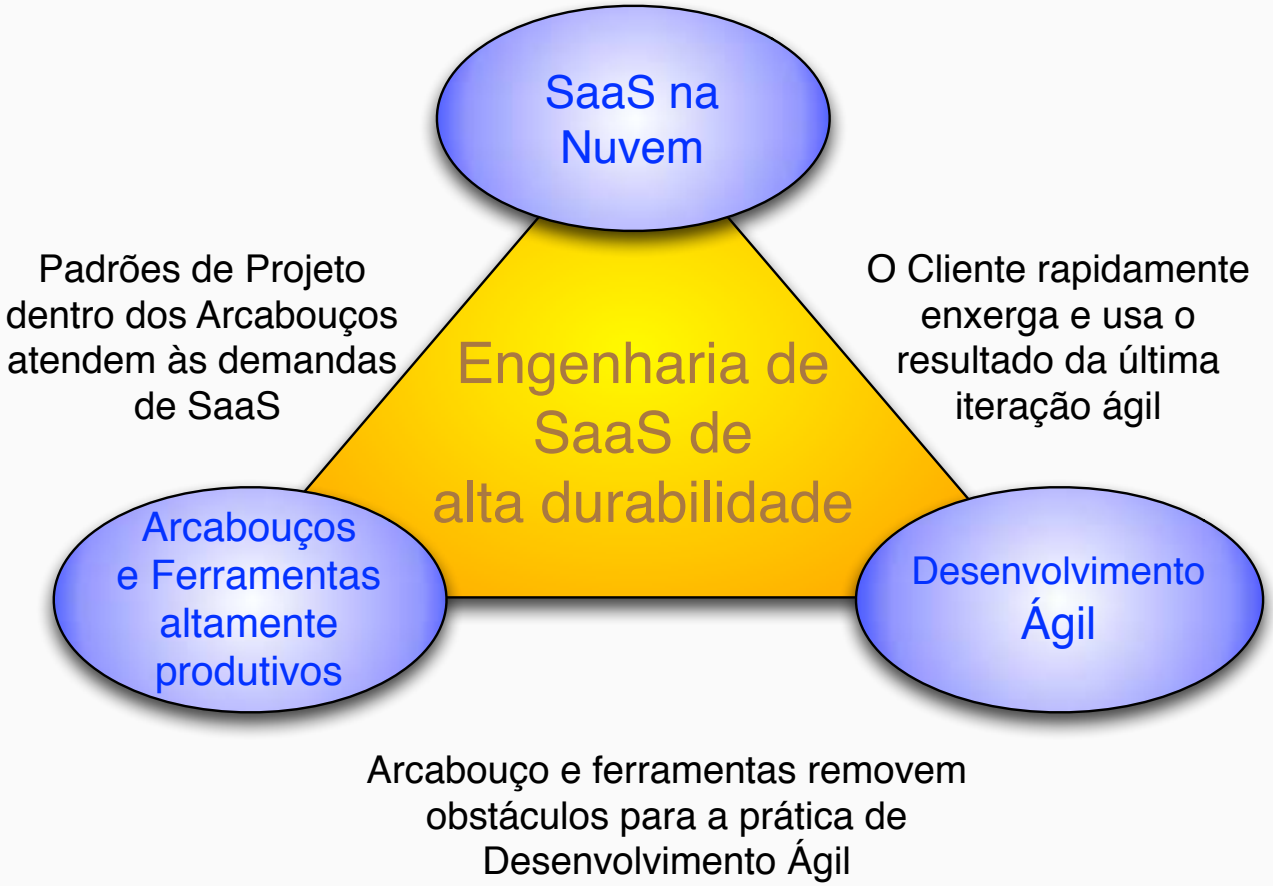


Figura 1: O triângulo da virtude da engenharia SaaS é formado a partir de três jóias da coroa da engenharia de software, (1) SaaS na Nuvem, (2) Desenvolvimento Ágil e (3) Arcabouços e ferramentas de alta produtividade.

SOBRE O PROJETO

- Times de 5–6 alunos
- Ideia: projetar, planejar, desenvolver, testar e implantar um software como um serviço
- Recomendação: usar Rails para o *backend* e HTML5+JavaScript para o *frontend*; outras linguagens/plataformas/arcabouços podem ser usados, porém você **deve** ter:
  - um arcabouço para os testes de Unidade & funcionais (recomendação: RSpec para Ruby/Rails e Jasmine para JavaScript)
  - uma forma de medir a cobertura dos testes (Coveralls e Travis CI)
  - um arcabouço de teste completo que possa expressar testes que corresponderão às histórias dos usuários (Cucumber e Capybara)
  - um arcabouço para a medição da qualidade do código, que ajude a identificar mal cheiros de projetos, problemas no estilo do código, etc. (CodeClimate)
  - arcabouço de SaaS para o lado do servidor (Rails)
  - arcabouço de SaaS para o lado do cliente (HTML5+JavaScript)

ITERAÇÃO 0 I

A iteração 0 compreende (sugiro que a ordem dos itens seja seguida):

1. Criar um projeto no GitHub
2. Adicionar *todos os integrantes* do grupo como colaboradores
3. Todos os membros devem conseguir fazer um clone do repositório e executar o projeto em seu ambiente de desenvolvimento local (recomendação: use Linux/MacOS ou Cloud9)
4. `rake spec` e `rake features` deve rodar sem erros (mesmo que ainda não haja testes escritos)
5. Integrar o repositório com Travis CI, o que significa ter o Travis CI reportando `Build: passing`
6. Integrar o repositório com o CodeClimate

ITERAÇÃO 0 II

7. Um projeto público no Pivotal Tracker configurado para esse projeto, com todos os membros do time adicionados como colaboradores
8. Um `README.md` no repositório do projeto que inclua todos os itens abaixo:
   - "CodeClimate badge" mostrando o GPA do projeto
   - "Travis CI badge" mostrando o status do *build* no *branch master* (deve ser: `passing`)
   - Link para a app implantada no Heroku
   - Link para o projeto no Pivotal Tracker
   - Integrantes do projeto e uma pequena descrição de qual problema o software de vocês resolverá

Prazo:

Domingo, 20 de agosto de 2016