

# INTRODUÇÃO AO PROJETO GUIADO POR COMPORTAMENTO E HISTÓRIAS DE USUÁRIOS

ENGENHARIA DE SISTEMAS DE INFORMAÇÃO

---

Daniel Cordeiro

22 de setembro de 2017

Escola de Artes, Ciências e Humanidades | EACH | USP

## POR QUE PROJETOS DE SOFTWARE FALHAM?

- Não fazem o que o cliente quer

# POR QUE PROJETOS DE SOFTWARE FALHAM?

- Não fazem o que o cliente quer
- Ou os projetos atrasam

# POR QUE PROJETOS DE SOFTWARE FALHAM?

- Não fazem o que o cliente quer
- Ou os projetos atrasam
- Ou estouram o orçamento

# POR QUE PROJETOS DE SOFTWARE FALHAM?

- Não fazem o que o cliente quer
- Ou os projetos atrasam
- Ou estouram o orçamento
- Ou são difíceis de manter e evoluir

# POR QUE PROJETOS DE SOFTWARE FALHAM?

- Não fazem o que o cliente quer
- Ou os projetos atrasam
- Ou estouram o orçamento
- Ou são difíceis de manter e evoluir
- Ou todas as anteriores 😊

# POR QUE PROJETOS DE SOFTWARE FALHAM?

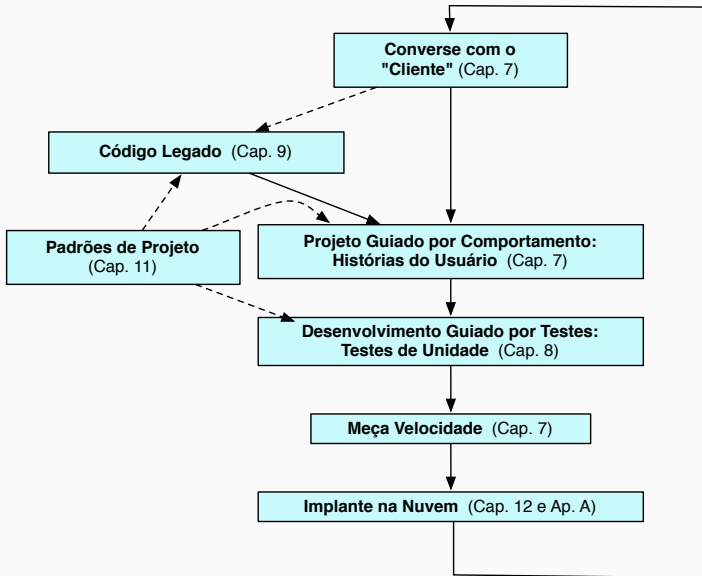
- Não fazem o que o cliente quer
- Ou os projetos atrasam
- Ou estouram o orçamento
- Ou são difíceis de manter e evoluir
- Ou todas as anteriores 😊

Como métodos ágeis tentam evitar tais falhas?

- Trabalhe próximo e continuamente com todos os interessados (*stakeholders*) para desenvolver os requisitos e testes
  - usuários, clientes, desenvolvedores, programadores de manutenção, operadores, gerentes de projeto, ...
- Mantenha um protótipo sempre funcionando enquanto desenvolve novas funcionalidades em todas as iterações
  - tipicamente cada 1 ou 2 semanas
  - ao invés de 5 grandes fases separadas por vários meses
- Verifique com os interessados qual o próximo passo, para validar que estamos desenvolvendo a coisa certa (vs. verificação)



# ITERAÇÕES DE MÉTODOS ÁGEIS / ORGANIZAÇÃO DO LIVRO-TEXTO

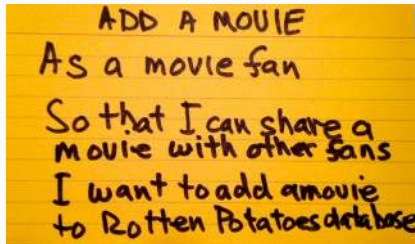


- BDD (*Behavior-Driven Design*) faz as perguntas certas sobre o comportamento do app **antes e durante o desenvolvimento** para diminuir falhas de comunicação (validação vs. verificação)
- Requisitos são escritos como **histórias de usuários**; descrições simplificadas de como o app será usado
- BDD se concentra no **comportamento** do app e não na **implementação** do app
  - mais para frente veremos como Desenvolvimento Guiado por Testes (TDD) ajuda a testar a implementação

- 1–3 frases em linguagem do dia a dia
  - cabem em uma ficha A7 (74x105mm)
  - escrito com/pelo cliente
- Formato “Connextra”:
  - nome da funcionalidade
  - **As a** [tipo do interessado]
  - **So that** [eu possa alcançar um objetivo]
  - **I want to** [fazer alguma coisa]
  - as 3 frases devem aparecer, em qualquer ordem
- Ideia: histórias de usuário podem ser formuladas para servirem como **testes de aceitação** antes do código ser escrito

## POR QUE CARTÕES 3X5 (OU FICHAS A7)?

- ideia da comunidade de IHC
- não são ameaçadores; todos os interessados participam do *brainstorming*
- fáceis de rearranjar; todos os interessados participam na priorização
- as histórias devem ser curtas e fáceis de serem mudadas durante o desenvolvimento (já que temos novos *insights* durante o desenvolvimento)



ADD A MOVIE  
As a movie fan  
So that I can share a  
movie with other fans  
I want to add a movie  
to Rotten Potatoes database

# STAKEHOLDERS DIFERENTES PODEM DESCREVER O COMPORTAMENTO DE FORMA DIFERENTE

- Ver quais dos meus amigos vão a um concerto:
  - **como um** espectador
  - **de forma que** eu possa aproveitar o concerto com meus amigos
  - **eu quero** ver quais dos meus amigos do Facebook irão a um dado concerto
- Mostrar os amigos do Facebook do chefe
  - **como um** gerente de bilheteria
  - **de forma que** eu possa induzir meu chefe a comprar os ingressos
  - **eu quero** ver quais de seus amigos no Facebook irão a um dado concerto

- Sistemas reais possuem centenas de histórias de usuários
- *Backlog*: histórias de usuário que ainda não foram completadas
- Priorize em ordem das histórias que retornarão o maior valor quando prontas
- Organize-as de forma que elas casem com as entregas do app

- Pequena investigação sobre alguma técnica ou problema
  - ex: um *spike* em algoritmos de recomendação
  - experimente, programe, faça funcionar
- Limite o tempo alocado para a tarefa
- Quando terminar, *jogue o código fora*
  - agora que você sabe qual a melhor abordagem, use-a direito!

Qual afirmação sobre BDD e histórias de usuário é falsa?

1. BDD foi projetado para ajudar com a validação (construir a coisa certa) em adição à verificação
2. Histórias de usuário devem incluir informação sobre decisões de implementação
3. Histórias de usuário em BDD assumem o mesmo papel da análise de requisitos em Planeje-e-Documente
4. A mesma funcionalidade pode aparecer em mais de uma história de usuário, na perspectiva de *stakeholders* diferentes



# PONTOS, VELOCIDADE E O PIVOTAL TRACKER

---

- Em métodos Ágeis a gente quer evitar fazer grandes planejamentos. Mas então como estimar tempo sem um plano?
- Histórias de usuário podem ser usadas para medir o progresso no projeto?
- O que uma boa ferramenta para medição de progresso em métodos Ágeis deveria fazer?

- Uma medida de produtividade de uma equipe: número médio de histórias / semana?
  - mas há histórias mais difíceis que outras...
- Ranqueie cada história de usuário, antes de mais nada, usando uma escala simples:
  - 1 para histórias triviais, 2 para histórias médias, 3 para histórias muito complexas
- **Velocidade:** número médio de *pontos* / semana

- Em times com mais experiência, uma escala de Fibonacci normalmente é usada: 1, 2, 3, 5, 8
  - (cada novo número é a soma dos 2 anteriores)
  - no Pivotal Labs, 8 é extremamente raro
  - conselho: no início, se  $\geq 5$ , então **divida a história!**
- Os times votam: cada um levanta os dedos, toma-se a média
  - se tiver divergência (2 e 5), o time deve discutir mais

- $\geq 5 \Rightarrow$  divida a história de usuário em histórias mais simples, agrupe em *epics* (epopéias)
- Não importa se a velocidade é 5 ou 10 pontos por iteração
  - o importante é que o time seja consistente
- A ideia é melhorar a autoavaliação e sugerir o número de iterações para cada conjunto de funcionalidades

- Priorize as histórias de usuário, colocando-as nos painéis *Current*, *Backlog* e *Icebox*
- Quando completar, mova para o painel *Done*
  - desenvolvedores apertam o botão *Finish*, envia para o *product owner*
  - o *product owner* experimenta a história e decide se aceita (*Accept*) ou rejeita (*Reject*)
- Você pode adicionar *Release points* lógicos, para saber quando um novo lançamento realmente ocorrerá (pontos restantes / velocidade)
- Epic (com painel próprio)
  - coloque histórias relacionadas juntas
  - ordene independentemente da história do usuário no *Backlog*

- Funcionalidades
  - histórias de usuários que fornecem valor de negócio para o cliente (ex: “adicionar um botão de confirmação na página de conclusão de compra”)
  - Vale pontos e, portanto, deve ser estimado
- Tarefas & bugs:
  - Histórias de usuário que são necessárias, mas que não tem um valor direto pro cliente (ex: “descobrir porque os testes estão tão lentos” ou “refatorar o subsistema de pagamentos”)
  - Não ganham pontos

- Tracker permite anexar documentos às histórias de usuário (ex: LoFi UI)
- Wiki do repositório no GitHub
- Documentos Google: criação e visualização colaborativa de desenhos, apresentações, planilhas e documentos de texto
- Campfire: serviço web com salas de bate-papo protegidos por senha
- Slack: uma mistura de alguns itens acima



Qual afirmação relacionada a Pontos, Velocidade e Tracker é verdadeira?

1. Quando se compara duas equipes, aquela com maior velocidade é a mais produtiva
2. Quando você não sabe como abordar uma história de usuário, dê 3 pontos para ela
3. Com Tracker, desenvolvedores pegam histórias de usuários e as marcam com *Accepted* quando terminarem
4. Tracker ajuda a priorizar e manter o controle sobre as histórias de usuário e seus status, calcula a velocidade e prediz o tempo de desenvolvimento do software



- Dividir o trabalho em histórias ajudou todos os membros da equipe a entenderem o app e a ficarem mais confiantes na hora de mudá-lo
- Tracker nos ajudou a priorizar as funcionalidades e estimar a dificuldade



- Nós dividimos por camadas (front-end vs. back-end vs. JavaScript, etc.) e foi difícil coordenar para colocar tudo para funcionar
- Foi mais difícil estimar se o trabalho havia sido dividido de forma justa... não estamos certos se nossa habilidade de estimar a dificuldade melhorou ou não com o tempo