Introdução ao Projeto Guiado por Comportamento e Histórias de Usuários

SIN5005 — Tópicos em Engenharia de Software

Daniel Cordeiro

18 de setembro de 2019

Escola de Artes, Ciências e Humanidades | EACH | USP

Rotas & REST

REST (Representational State Transfer) — R. Fielding, 2000

- Ideia: URI são nomes de *recursos*, não páginas ou ações
  - autocontido: qual o recurso, o que fazer com ele?
  - respostas incluem *hyperlinks* que permitem descobrir novos recursos RESTful
  - "uma descrição *post hoc* (depois do fato) das funcionalidades que fizeram a Web ser tão bem sucedida"
- Um serviço (no sentido de SOA) cujas operações seguem esses princípios é chamado de serviço RESTful
- Idealmente, URIs RESTful dão nome às operações

O que é REST?

REST (*Representational State Transfer* ou Transferência de Estado Representacional) possui 3 definições populares:

O que é REST?

REST (*Representational State Transfer* ou Transferência de Estado Representacional) possui 3 definições populares:

1. Um *estilo arquitetural* para definição de sistemas fracamente acoplados
   - definido por um conjunto de restrições gerais (princípios)
   - a Web (URL/HTTP/HTML/XML) é uma instância desse estilo

O que é REST?

REST (*Representational State Transfer* ou Transferência de Estado Representacional) possui 3 definições populares:

1. Um *estilo arquitetural* para definição de sistemas fracamente acoplados
   - definido por um conjunto de restrições gerais (princípios)
   - a Web (URL/HTTP/HTML/XML) é uma instância desse estilo
2. A Web usada corretamente (ou seja, não como um protocolo de transporte)
   - HTTP foi construído usando princípios RESTful
   - serviços que são construídos usando padrões Web sem utilizá-los incorretamente
   - mais importante, HTTP é um protocolo de aplicação (e não de transporte)

O que é REST?

REST (*Representational State Transfer* ou Transferência de Estado Representacional) possui 3 definições populares:

1. Um *estilo arquitetural* para definição de sistemas fracamente acoplados
   - definido por um conjunto de restrições gerais (princípios)
   - a Web (URL/HTTP/HTML/XML) é uma instância desse estilo
2. A Web usada corretamente (ou seja, não como um protocolo de transporte)
   - HTTP foi construído usando princípios RESTful
   - serviços que são construídos usando padrões Web sem utilizá-los incorretamente
   - mais importante, HTTP é um protocolo de aplicação (e não de transporte)
3. Qualquer coisa que use HTTP e XML (que não seja SOAP)
   - XML–RPC foi a primeira tentativa para conseguir isso
   - viola REST porque não há interface uniforme

Estilos arquiteturais

Estilo arquitetural vs. Arquitetura

- Estilo arquitetural: princípios gerais que guiam a criação de uma arquitetura
- Arquitetura: projeto com a solução para um determinado problema dado um conjunto de restrições
- Estilos arquiteturais informam e guiam a criação de arquiteturas

Exemplo
O **Louvre** é um *projeto* arquitetônico cujo *estilo arquitetural* é o estilo **Barroco**.

REST não é arquitetura

- REST é um estilo arquitetural
  - capturado da Web *a posteriori*
  - alguns dos padrões e práticas Web não são perfeitamente RESTful
- A Web é um sistema de informação que segue o REST
- É possível projetar outros sistemas de informação RESTful
  - podem existir interfaces diferentes que sejam uniformes (mas que não usem métodos HTTP)
  - diferentes representações (que não HTML ou XML)
  - diferentes identificadores (que não URIs)

O estilo arquitetural REST

- Um conjunto de restrições que caracterizam uma arquitetura:
  1. Identificação de recursos
  2. Interface uniforme
  3. Mensagens autodescritivas
  4. Estado de aplicação determinado pela hipermídia
  5. Interações sem estado (*stateless*)
- Objetivos: escalabilidade, usabilidade, acessibilidade, facilidade de composição

Identificação de recursos

- Dê nome a tudo aquilo que você quiser mencionar
- “aquilo” aqui pode se referir a qualquer coisa:
  - produtos em uma loja *online*
  - categorias que são usadas para agrupar produtos
  - clientes, os quais espera-se que irão comprar algo
  - carrinhos de compras, onde os clientes armazenarão seus produtos
- O estado de uma aplicação é representado também como um recurso
  - os links “próximo” em um processo de submissão de múltiplas páginas
  - resultados paginados (à la Gooooooogle) com URIs identificando as páginas seguintes

Interface uniforme

- Um mesmo conjunto pequeno de operações se aplica a *tudo*
- Um pequeno conjunto de *verbos* se aplica a um grande conjunto de *substantivos*
  - verbos são universais e não algo inventado para cada aplicação
  - se muitas aplicações precisarem de novos verbos, a interface uniforme pode ser estendida
  - as linguagens naturais funcionam da mesma forma (novos verbos dificilmente são incorporados a uma língua)
- Identifica as operações que são candidatas à serem otimizadas
  - `GET` e `HEAD` são operações seguras
  - `PUT` e `DELETE` são operações idempotentes
  - `POST` pode ter várias possibilidades e, portanto, pode causar efeitos colaterais
- Constrói funcionalidades baseadas nas propriedades úteis dessas operações

Mensagens autodescritivas

- Recursos são entidades abstratas (elas não podem ser utilizadas *per se*)
  - *Identificação de recursos* garantem que eles são identificados claramente
  - eles podem ser acessados usando uma *Interface Uniforme*
- Recursos são acessados via *representações de recursos*
  - representações de recursos devem ser suficientes para distinguir um certo recurso
  - comunica qual tipo de representação é usada
  - o formato para a representação pode ser negociado entre os envolvidos
- Representações de recursos podem ser baseados em restrições diferentes
  - XML e JSON podem representar o mesmo modelo para diferentes usuários
  - qualquer que seja a representação, ela deve permitir o uso de *links*

Estado de aplicação determinado pela hipermídia

- Representação de recursos possuem links para recursos identificados
- Recursos e o estado podem ser usados por links navegáveis
  - links tornam recursos interconectados navegáveis
  - sem navegação, a identificação de novos recursos é algo específico do serviço
- Aplicações RESTful *navegam* ao invés de *chamar*
  - representações contém informações sobre formas diferentes de percorrer os links
  - a aplicação navega para o próximo recurso dependendo da semântica do link
  - a navegação pode ser delegada já que todos os links usam identificadores

Interações sem estado (stateless)

- Essa restrição não significa "**Aplicações** sem Estado"!
  - em muitas aplicações RESTful, estado é uma parte essencial
  - a ideia de RESTful é evitar transações longas *nas aplicações*
- "Sem estado" significa mover o estado para os clientes ou recursos
  - consequência mais importante: evitar manter o estado da aplicação no lado do servidor
- O estado do recurso é gerenciado no servidor
  - é o mesmo para todos os clientes que interagem com o serviço
  - quando um cliente mudar o estado do recurso, outros clientes também verão a mudança

Interações sem estado (stateless)

- O estado do cliente é gerenciado no cliente
  - é específico para cada cliente e, portanto, deve ser mantido no cliente
  - pode afetar o *acesso* aos recursos do servidor, mas não afeta os recursos em si
- Problemas com segurança se tornam mais importantes quando o estado está no cliente
  - clientes podem (tentar) enganar o servidor mentindo sobre seu estado
  - manter o estado do cliente no servidor é caro (mas pode valer a pena)

O que é a Web?

- Web = URL + HTTP + (HTML | XML)
- a Web RESTful usa métodos HTTP como sua interface uniforme
  - a web não-RESTful usa **GET**/**POST** e chamadas RPC
  - uma “web RESTful diferente” usa WebDAV
- Imagine sua aplicação sendo utilizada em “10 navegadores”
  - os recursos com os quais ela interage devem ser identificados e “linkados”
  - o tamanho de fonte preferido do usuário pode ser modelado como o estado do cliente
- Imagine sua aplicação sendo utilizada em “10 abas do navegador”
  - não faz diferença contanto que o estado do cliente seja baseado na representação
  - cookies são compartilhados entre as janelas do mesmo navegador

Identificação de recursos na Web

- Essencial para implementar um mecanismo de Identificação de Recursos
- URIs são um esquema de identificação universal legível para identificar “coisas”
  - muitos esquemas de identificação não podem ser lidos por humanos (strings binárias ou hex)
  - muitos sistemas baseados em RPC não possuem esquema de identificação universal para seus objetos
- É importante fazer com que todas as coisas possam ser identificadas unicamente
  - isso evita a necessidade de identificadores não universais usados dentro de um escopo
  - permite referenciar todas as coisas exatamente do mesmo jeito
- Coisas identificadas por URI devem possuir um modo de interação bem definido

Processamento de URI

- O processamento de uma URI não é tão trivial como parece
  - regras de normalização e escape de caracteres não é trivial
  - muitas implementações não estão corretas
  - se tornam ainda mais complicadas quando se tratam de um *Internationalized Resource Identifier* (IRI)
- URIs não são apenas strings
  - URIs são strings com um conjunto considerável de regras implícitas
  - implementar essas regras não é trivial
  - ... mas é crucial

Aplicações RESTful falam HTTP

- HTTP é essencial para a implementação de Interfaces Uniformes
    - HTTP define um conjunto pequeno de métodos para interagir com recursos identificados com uma URI
- Má utilização de HTTP torna uma aplicação não-RESTful
    - ela perde a capacidade de ser utilizada apenas aderindo aos princípios de REST
    - perceber que você precisa de uma linguagem de descrição de interface é um sinal de que há algo errado
- Estender o uso de HTTP transforma sua aplicação em uma aplicação RESTful mais especializada
    - pode ser apropriado quando mais operações são necessárias
    - mas reduz muito o número de potenciais clientes

Métodos HTTP

- *Métodos seguros* podem ser ignorados ou repetidos sem causar efeitos colaterais
  - segurança aritmética: $41 \times 1 \times 1 \times 1 \times 1 \cdots$
  - na prática “sem efeitos colaterais” significa “sem efeitos colaterais significativos”
- *Métodos idempotentes* o resultado devolvido por uma requisição ou por *n* requisições é o mesmo
  - $41 \times 0 \times 0 \times 0 \times 0 \cdots$
- Métodos inseguros ou que não sejam idempotentes devem ser tratados com cuidado
- HTTP tem dois métodos seguros: GET e HEAD
- HTTP tem dois métodos idempotentes: PUT e DELETE

O que é identificado por uma URI?

- Essencial para implementar Mensagens autodescritivas
  - também devem permitir “estado de aplicação determinado pela hipermídia”
- Identificação de recursos fala sobre um *recurso abstrato*
  - recursos nunca são trocados ou processados diretamente
  - todas as interações usam representação de recursos
- Representações dependem de vários fatores:
  - a natureza do recurso
  - as funcionalidades do servidor
  - as funcionalidades da mídia de comunicação
  - as funcionalidades do cliente
  - requisitos e restrições do domínio da aplicação
  - negociação do “melhor” tipo de representação
- Representações são identificados pelo seu tipo de mídia (MIME type)

O básico do básico sobre config/routes.rb

Rotas: conectam URIs ao código

1. Rails recebe a requisição `get /patients/17`
2. então pergunta ao roteador se essa requisição casa com uma ação de controlador, se o primeiro casamento for algo como: `get '/patients/:id', to: 'patients#show'`
3. a requisição será despachada para a ação `show` do controlador `patients`

Veja guia básico em
`http://guides.rubyonrails.org/routing.html`

CRUD em um recurso RESTful

- Rails permite definir recursos que serão acessíveis usando as operações CRUD.
- Ao invés de criar rotas para `index`, `show`, `new`, `edit`, `create`, `update` e `destroy`, você define uma *resourceful route* com uma linha de código:

  ```
  resources :photos
  ```

- Isso cria 7 rotas diferentes automaticamente:

Verbo HTTP	Path	Controller#Action	Usado para
GET	/photos	photos#index	mostra uma lista com todas as fotos
GET	/photos/new	photos#new	devolve um form HTML para criar uma nova foto
POST	/photos/new	photos#create	cria uma nova foto
GET	/photos/:id	photos#show	mostra uma foto específica
GET	/photos/:id/edit	photos#edit	devolve um form HTML para editar uma foto
PATCH/PUT	/photos/:id	photos#update	atualiza uma foto específica
DELETE	/photos/:id	photos#destroy	apaga uma foto específica

Projeto Guiado por Comportamento e Histórias de Usuários

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?

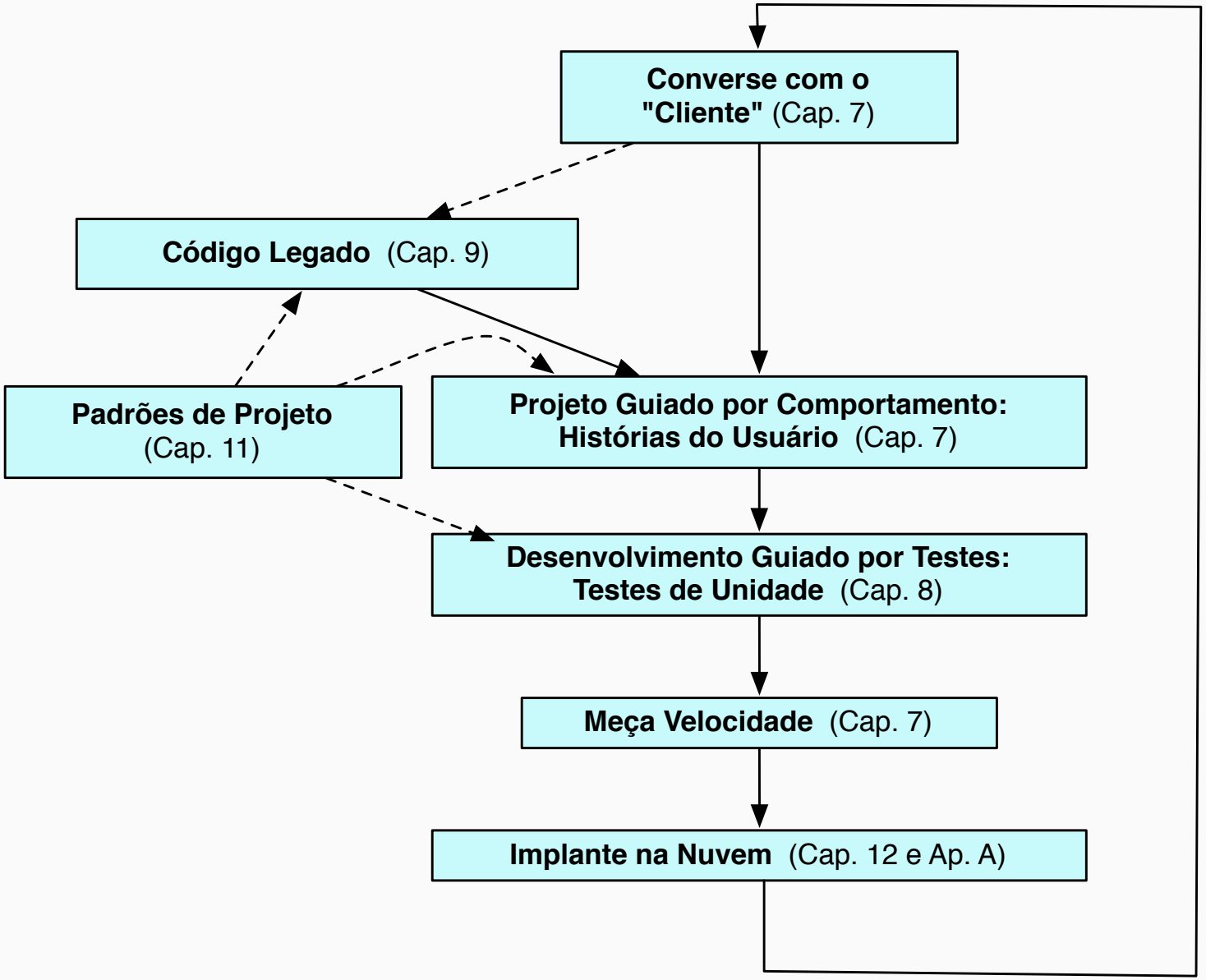
- Não fazem o que o cliente quer
- Ou os projetos atrasam
- Ou estouram o orçamento
- Ou são difíceis de manter e evoluir
- Ou todas as anteriores 😛

Como métodos ágeis tentam evitar tais falhas?

Revisão do Ciclo de Vida Ágil

- 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



Projeto Guiado por Comportamento

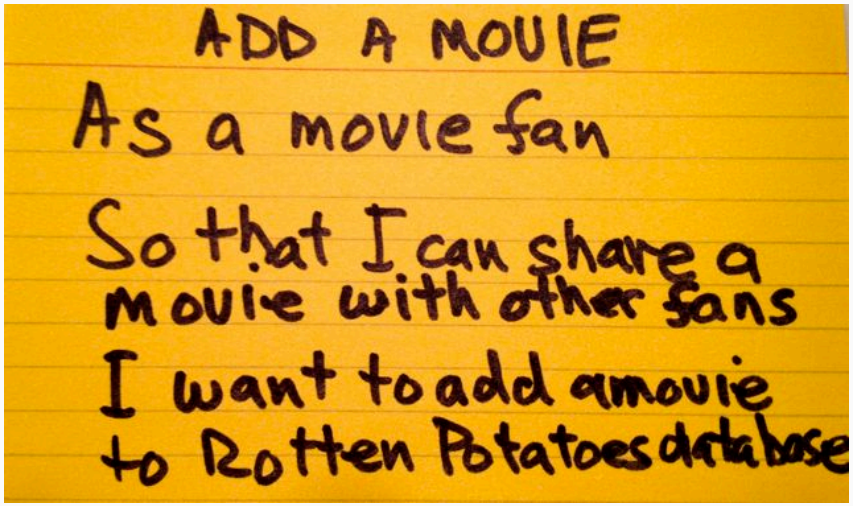
- BDD (*Behavior-Driven Design*) faz as perguntas certas sobre o comportamento do app antes e durante o desenvolvimento para diminuir falhas de comunicação (validação vs. verificação)
- Requisitos são escritos como histórias de usuários; descrições simplificadas de como o app será usado
- BDD se concentra 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

Histórias de usuário

- 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)



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

Backlog do produto

- 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

Relacionado: spike

- 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!

Pergunta

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

Produtividade e Ferramentas

- 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?

Medindo produtividade

- 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

Mais sobre Pontos

- 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 ≥ 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

Mais sobre Pontos

- $\geq 5 \Rightarrow$ 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

Pivotal Tracker

- 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*

Pivotal Tracker: funcionalidades vs. tarefas

- 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

Quadro branco da equipe no ciberespaço

- 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

Pergunta

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

Histórias vs. Camadas

☺

- 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

Histórias de usuário SMART

Ao criar histórias de usuário

Como distinguir uma boa de uma má história de usuário?

- Ela tem o tamanho adequado?
- Ela é muito fácil (ou muito difícil)?
- Vale a pena?

Histórias SMART

- ***S**pecific* (Específica)
- ***M**easurable* (Mensurável)
- ***A**chievable* (Realizável) — idealmente, implementável em uma iteração
- ***R**elevant* (Relevante) — “os 5 porquês”
- ***T**imeboxed* (com Duração Fixa) — saiba quando desistir

Específica & Mensurável

- Cada cenário é testável
  - implica conhecimento de uma boa entrada e que existem resultados esperados para ela
- Contraexemplo: “UI deve ser amigável para o usuário”
- Exemplo: Dada/Quando/Então
  1. *Dada* alguma(s) condição(ões) inicial(is),
  2. *Quando* eu realizar uma ação X
  3. *Então* uma ou mais coisas específicas deveriam acontecer

Realizável

- Complete em uma iteração
- Se não puder entregar a funcionalidade em uma iteração, entregue subconjuntos das histórias
    - Sempre almeje ter código funcionando no final de uma iteração
- Se $<$ 1 história por iteração, você precisa melhorar a estimativa de pontos por história

Relevante: “valor de negócio”

- Descubra o valor de negócio, ou desista da história
  - protege a receita
  - aumenta a receita
  - gerencia custo
  - melhora o valor da marca
  - torna o produto memorável
- Você pode incluir histórias que não possuem um valor de negócio óbvio?

Os 5 “porquês” para encontrar a relevância

“Mostrar os amigos do Facebook que irão em um evento”

Como um gerente de bilheteria
Para fazer com que meu cliente aproveitem melhor o espetáculo
Quero mostrar quais de seus amigos irão a um dado concerto

1. Por que adicionar um recurso no Facebook? Como gerente, acho que venderemos mais tíquetes

Os 5 “porquês” para encontrar a relevância

“Mostrar os amigos do Facebook que irão em um evento”

Como um gerente de bilheteria
Para fazer com que meu cliente aproveitem melhor o espetáculo
Quero mostrar quais de seus amigos irão a um dado concerto

1. Por que adicionar um recurso no Facebook? Como gerente, acho que venderemos mais tíquetes
2. Por que se importar se o público aproveita o espetáculo? Eu acho que venderemos mais tíquetes

Os 5 “porquês” para encontrar a relevância

“Mostrar os amigos do Facebook que irão em um evento”

Como um gerente de bilheteria
Para fazer com que meu cliente aproveitem melhor o espetáculo
Quero mostrar quais de seus amigos irão a um dado concerto

1. Por que adicionar um recurso no Facebook? Como gerente, acho que venderemos mais tíquetes
2. Por que se importar se o público aproveita o espetáculo? Eu acho que venderemos mais tíquetes
3. Por que vender mais tíquetes? Porque assim teremos mais dinheiro.

Os 5 “porquês” para encontrar a relevância

“Mostrar os amigos do Facebook que irão em um evento”

Como um gerente de bilheteria
Para fazer com que meu cliente aproveitem melhor o espetáculo
Quero mostrar quais de seus amigos irão a um dado concerto

1. Por que adicionar um recurso no Facebook? Como gerente, acho que venderemos mais tíquetes
2. Por que se importar se o público aproveita o espetáculo? Eu acho que venderemos mais tíquetes
3. Por que vender mais tíquetes? Porque assim teremos mais dinheiro.
4. Por que o teatro precisa de mais dinheiro? Para que não vá a falência no próximo ano

Os 5 “porquês” para encontrar a relevância

“Mostrar os amigos do Facebook que irão em um evento”

Como um gerente de bilheteria
Para fazer com que meu cliente aproveitem melhor o espetáculo
Quero mostrar quais de seus amigos irão a um dado concerto

1. Por que adicionar um recurso no Facebook? Como gerente, acho que venderemos mais tíquetes
2. Por que se importar se o público aproveita o espetáculo? Eu acho que venderemos mais tíquetes
3. Por que vender mais tíquetes? Porque assim teremos mais dinheiro.
4. Por que o teatro precisa de mais dinheiro? Para que não vá a falência no próximo ano
5. Por que se importar se o teatro irá a falência? Porque senão perderei meu emprego!

Os 5 “porquês” para encontrar a relevância

“Mostrar os amigos do Facebook que irão em um evento”

Como um gerente de bilheteria
Para fazer com que meu cliente aproveitem melhor o espetáculo
Quero mostrar quais de seus amigos irão a um dado concerto

1. Por que adicionar um recurso no Facebook? Como gerente, acho que venderemos mais tíquetes
2. Por que se importar se o público aproveita o espetáculo? Eu acho que venderemos mais tíquetes
3. Por que vender mais tíquetes? Porque assim teremos mais dinheiro.
4. Por que o teatro precisa de mais dinheiro? Para que não vá a falência no próximo ano
5. Por que se importar se o teatro irá a falência? Porque senão perderei meu emprego!

Os 5 “porquês” para encontrar a relevância

“Mostrar os amigos do Facebook que irão em um evento”

Como um gerente de bilheteria
Para fazer com que meu cliente aproveitem melhor o espetáculo
Quero mostrar quais de seus amigos irão a um dado concerto

1. Por que adicionar um recurso no Facebook? Como gerente, acho que venderemos mais tíquetes
2. Por que se importar se o público aproveita o espetáculo? Eu acho que venderemos mais tíquetes
3. Por que vender mais tíquetes? Porque assim teremos mais dinheiro.
4. Por que o teatro precisa de mais dinheiro? Para que não vá a falência no próximo ano
5. Por que se importar se o teatro irá a falência? Porque senão perderei meu emprego!

Temos certeza de que o valor de negócio está aparente para pelo menos um *stakeholder*!

Com duração fixa

- Pare a história quando ela exceder o tempo/orçamento
  - Desista dela ou a divida em histórias menores ou reagende o que ainda falta a ser feito
- Faz com que não subestimamos a duração do projeto
- Pivotal Tracker acompanha a velocidade, o que ajuda a não subestimar

Pergunta

Qual funcionalidade abaixo é a menos SMART?

1. Usuário pode procurar por um filme pelo título
2. Dado que eu tenho um cupom para ver um filme gratuitamente, eu quero usar o cupom com um filme elegível antes que ele expire
3. Ao adicionar um filme, 99% das páginas de adicionar filmes deveriam aparecer em menos de 3 segundos
4. Como um cliente, eu quero ver os 10 filmes mais vendidos, listados por preço, de forma que eu possa comprar os mais baratos primeiro

Estimativa de custos Ágil

Estimativa de custos Ágil

- No mundo real precisamos estimar os custos antes do cliente poder concordar com o projeto
- Se não há planejamento e cronogramas em métodos ágeis, como podemos estimar o custo de um projeto?

Modelo do Pivotal Labs

- O Pivotal Labs ensina o método Ágil pro cliente
- Usando método Ágil, Pivotal nunca promete entregar as funcionalidades X, Y e Z até a data D
- Ao invés disso, promete empregar recursos para trabalharem do modo mais eficiente possível até a data D
    - o cliente trabalha continuamente com a equipe para definir as prioridades até a data D
- Mas ainda precisamos de uma estimativa pro projeto

Estimativa Ágil da Pivotal Labs

1. 1 hora no telefone para explicar o método
   - esforço conjunto, comprometimento que o cliente deve ter, ...
2. Cliente visita a Pivotal por 1,5 horas (“scoping”)
   - cliente traz designer, desenvolvedor (tudo para esclarecer o que o cliente quer que seja feito)
   - Pivotal traz dois engenheiros para fazer perguntas (tentando identificar o que causaria incerteza na estimativa)
3. Engenheiros levam ½ hora para estimar as semanas
   - maior vs. menor incerteza: 18–26 vs. 20–22 semanas
4. Propõe um custo em função do tempo e recursos necessários

Pergunta

Qual afirmação relacionada a estimativa de custos é verdadeira (PL = Pivotal Labs)?

1. Como praticantes de métodos ágeis, PL não usa contratos
2. Como praticantes de programação pareada, PL estima o custo por 1 par (que será atribuído pela PL para completar o projeto)
3. O custo proposto é estimado considerando-se o tempo e materiais que serão utilizados por um dado número de semanas
4. Como mostram os estudos, 84–90% dos projetos terminam no prazo e dentro do orçamento; gerentes planeje-e-documente prometem aos clientes um conjunto de funcionalidades por um custo e prazo que são acordado previamente