# User stories

# *User Stories*

- Nos métodos ágeis, uma maneira bastante utilizada para descrever os requisitos são as *user stories* (estórias de usuário).
- Uma *user story* é uma descrição concisa de um requisito do ponto de vista do usuário.
- Ela deve ser simples o bastante para que possa ser escrita em um pequeno cartão.
- A *user story* não descreve tudo que há de ser feito, mas serve como lembrete do que deverá ser feito.

# *User Stories*

Uma *user story* é composta por três partes:

- cartão (descrição);
- conversa (detalhes);
- confirmação (testes).

# *User Stories* - Cartão

O cartão define três aspectos da *user story*:

- quem? $\rightarrow$ interessado;
- o quê? $\rightarrow$ necessidade;
- por quê? $\rightarrow$ resultado.

# *User Stories* - Cartão

"Eu, enquanto professor, desejo postar aulas com vídeo e outros materiais didáticos."

Eu, enquanto <**PAPEL**>, desejo <**fazer algo**>.

# *User Stories* - Cartão

A *user story* também pode conter uma razão ou justificativa, como:

"Eu, enquanto professor, desejo postar aulas com vídeo e outros materiais didáticos **para que meus estudantes surdos possam estudar em casa**."

A justificativa em negrito não é obrigatória, mas ela pode ajudar os desenvolvedores em alguns contextos onde a funcionalidade não é tão clara.

# *User Stories* - Cartão

- A escrita a *story* deve ser focada na solução e não no problema.
- A *story* deve ser sempre do ponto de vista do usuário.
- O gerente de produto (ou cliente) deve estar atento a quem de fato é o usuário da funcionalidade e qual benefício este obterá.

# *User Stories* - Cartão

- As estórias de usuário são colocadas no *Product Backlog* (se você usar Scrum).
- Deve-se evitar *stories* negativas, pois é difícil implementar o que o sistema não deve fazer.

# *User Stories* - Cartão

"Eu, enquanto usuário, não quero que o sistema grave e transmita minhas informações a servidores externos."

É melhor reescreve-la de uma maneira positiva:

"Eu, enquanto usuário, desejo controlar a informação que é grava e transmitida a servidores externos para que eu garanta que minha informação pessoal não é compartilhada."

# *User Stories* - Cartão

Segundo Mike Cohn, são boas práticas para escrita de *user stories*:

- escrever primeiro estórias ligadas aos objetivos dos usuários;
- escrever em voz ativa;
- o cliente deve escrever a estória [1];
- não numerar os cartões;
- cartões são apenas um lembrete do que será discutido;
- dividir estórias grandes ou complexas;

[1] sob supervisão, obviamente

# *User Stories* - INVEST

Bill Wake, famoso autor sobre métodos ágeis, criou o acrônimo INVEST para criação de boas *user stories*:

- I - independente;
- N - negociável;
- V - valiosa;
- E - estimável;
- S - pequena (*small*);
- T - testável.

# *User Stories* - Independente

- As *user stories* devem ser independentes entre si.
- Se uma *story* de alta prioridade depende de uma estória de baixa prioridade, então inverte-se a lógica de priorização, pois a estória menos valiosa deverá ser implementada primeiro.

# *User Stories* - Independente

Outro problema da dependência entre *user stories* é em relação a estimativas. Tenha-se as seguintes *stories*:

- Eu, enquanto Cliente, desejo pagar pela compra usando um cartão Visa.
- Eu, enquanto Cliente, desejo pagar pela compra usando um cartão MasterCard.
- Eu, enquanto Cliente, desejo pagar pela compra usando um cartão American Express.

# *User Stories* - Independente

- As três *stories* são bastante similares e possuem complexidades quase idênticas.
- A primeira será implementada em, talvez, 3 dias.
- As demais, em 1 dia cada.
- As estimativas para a segunda e terceira *stories* não corresponderão à realidade.

# *User Stories* - Independente

Baseado em uma sugestão de Mike Cohn para problema similar, tem-se o seguinte reagrupamento das *user stories*:

- Eu, enquanto Cliente, desejo pagar pela compra usando uma bandeira de cartão.
- Eu, enquanto Cliente, desejo pagar pela compra usando outras duas bandeiras de cartão.

# *User Stories* - Negociável

- O cartão da *user story* não deve detalhar o requisito por completo.
- Ele serve apenas como um lembrete gentil do que deverá ser feito.
- No planejamento da iteração e em sua execução que devem ser discutidos os detalhes.

# *User Stories* - Negociável

Os cartões devem possuir apenas a *user story* e, no máximo, um par de notas que servirão de lembrete para as discussões como no exemplo abaixo:

- *Story*: Eu, enquanto Cliente, desejo pagar pela compra usando cartão de crédito.
- Nota: serão aceitos cartão da bandeira Discover?
- Nota para UI: não colocar campo "bandeira", pois pode ser descoberto através dos primeiros números do cartão.

# *User Stories* - Valiosa

- As *user story* devem ter valor para os usuários ou clientes.
- Deve-se lembrar, que, do ponto de vista ágil e moderno, o importante é o produto a ser desenvolvido e o valor que ele agrega aos usuários e clientes.

# *User Stories* - Estimável

Estimar uma estória de usuário é reconhecer a complexidade da estória. Há três fontes principais para dificuldade, ou impossibilidade, de se estimar uma estória:

- os desenvolvedores não possuem conhecimento do domínio;
- os desenvolvedores não possuem conhecimento técnico;
- a *story* é muito grande.

# *User Stories* - Estimável

- No caso de falta de conhecimento do domínio, deve-se conversar com os criadores da estória.
- No caso de falta de conhecimento técnico, pode-se realizar um *spike*, onde aos desenvolvedores é dado uma janela de tempo para conhecerem sobre a tecnologia.
- Se a *story* for muito grande, então é necessário dividi-la.

# *User Stories* - Pequena (*Small*)

- O tamanho de uma *story* de usuário é um problema.
- A *story* deve ser, a princípio, pequena.
- Entretanto, se muito pequena, pode levar a uma quantidade muito grande de requisitos.

# *User Stories* - Pequena (*Small*)

Uma *story* grande é chamada de épico. Os épicos são geralmente em dois tipos:

- *story* composta;
- *story* complexa.

# *User Stories* - Pequena (*Small*)

A *story* composta é um *epic* (épico) que agrega diversas estórias de usuário. Por exemplo:
"Eu, enquanto usuário, desejo autenticar-me no sistema através de credenciais externas."

Pode ser quebrado em:

- "Eu, enquanto usuário, desejo autenticar-me no sistema através das minhas credenciais no Google."
- "Eu, enquanto usuário, desejo autenticar-me no sistema através das minhas credenciais no Facebook."
- "Eu, enquanto usuário, desejo autenticar-me no sistema através das minhas credenciais na Apple."
- "Eu, enquanto usuário, desejo autenticar-me no sistema através das minhas credenciais no Microsoft."

# *User Stories* - Pequena (*Small*)

- A *story* complexa é um *epic* com uma ou mais partes com grau elevado de complexidade.
- Isso ocorre quando o *epic* deve estender algum algoritmo complexo ou criar um totalmente novo.
- Geralmente são fortes candidatos a *spike*.

# *User Stories* - Testável

- As *user stories* devem ser prefencialmente testáveis do ponto de vista de testes automatizados.
- Em alguns casos, especialmente em requisitos não funcionais, os testes não poderão ser automatizados.
- Ainda assim, estes devem ser a ínfima minoria.

# *User Stories* - Conversas

- As conversas entre time de desenvolvimento, gerente de produto (ou cliente) e demais interessados serve para negociar os detalhes das *user stories*.
- Elas são necessárias porque os cartões em si não bastam para especificar os detalhes das *stories*.
- São estabelecidos também os critérios de aceitação das *stories*, ou seja, as regras de como a funcionalidade deve se comportar.

# *User Stories* - Confirmação

- A confirmação compreende os critérios e testes de aceitação.
- Os critérios de aceitação são similares às regras de negócio.
- Um critério de aceitação deve ser expresso por um enunciado pequeno e de fácil entendimento.

# *User Stories* - Confirmação

Para a *story*: "Eu, enquanto Comprador, quero utilizar meu cartão de crédito no pagamento das minhas compras."
Tem-se os seguintes critérios de aceitação (que poderão ficar no verso do cartão):

1. somente podemos aceitar cartões de crédito com bandeiras com que temos convênio.
2. somente podemos aceitar cartões de crédito com data de expiração no futuro.
3. somente podemos aceitar cartões de crédito com número e nome do dono válidos.

# *User Stories* - Confirmação

- Cada critério de aceitação pode possuir uma série de testes de aceitação.
- Um teste de aceitação serve para verificar se as saídas são corretas, ou seja, se, sob ponto de vista de negócios, a funcionalidade, de fato, realiza o que se propõe de acordo com as regras de negócio.
- Obviamente, nem todos os critérios de aceitação podem ter testes de aceitação como, por exemplo, "o botão deve ser azul".

# *User Stories* - Confirmação

Para o critério de aceitação: "somente podemos aceitar cartões de crédito com bandeiras com que temos convênio."[2]
Tem-se os seguintes testes de aceitação:

- Comprador utiliza cartão de crédito Visa
  - Aceitou = correto.
  - Recusou = errado, deve ser corrigido!
- Comprador de Livros utiliza cartão de crédito Amex
  - Aceitou = errado, deve ser corrigido!
  - Recusou = correto.

[2] Exemplo adaptado de K21.

# *Definition of Done* (DoD)

- Tão importante quanto definir as *user stories* é definir quando elas estão de fato prontas.
- Para isto, existe a *Definition of Done* (DoD).
- Uma *user story* é considerada pronta (*done*) quando atinge completamente uma série de requisitos e parâmetros.

# *Definition of Done* (DoD)

Derek Huerter sugere que a DoD de uma *user story* seja composta por:

- atendimento aos critérios de aceição;
- testes unitários bem sucedidos;
- código revisado;
- testes funcionais bem sucedidos;
- atendimento à restrições (requisitos não-funcionais);
- aceitação pelo gerente de produto, *product owner* ou cliente.

# *Definition of Done* (DoD)

- A DoD não deve ser colocada nunca em segundo plano.
- Pelo contrário, ela é um dos pilares de manutenção da qualidade do produto final e do processo ágil.
- Quanto mais pobre uma DoD, mais risco é assumido.

# Requisitos não-funcionais

- As *user stories* parecem representar muito bem as funcionalidades do *software*.
- Entretanto, há um conjunto de requisitos que provocam restrições nos mais diversos aspectos tanto ao *software* quanto ao seu desenvolvimento.
- Esta classe de requisitos é chamada na literatura de **requisitos não-funcionais**.

# Requisitos não-funcionais

Há pelo menos três abordagens sugeridas pelas mais diversas fontes em como ligar com requisitos não-funcionais em um ambiente de desenvolvimento ágil:

- escrever a restrição como uma *user story*;
- colocar a restrição como um critério de aceitação de uma *user story*;
- colcoar a restrição como parte da *Definition of Done* (DoD).

# Requisitos não-funcionais como *user stories*

Para escrita de requisitos não-funcionais como *user stories* tem-se os exemplos:

"Eu, enquanto usuário, desejo ser notificado quando minha conexão cair para que possa saber as razões e tentar solucionar o problema. (performance)"

"Eu, enquanto usuário, desejo ser informado que alguns elementos não foram mostrados devido à minha conexão lenta para que eu tenha noção do problema e possa resolvê-lo. (performance)"

# Requisitos não-funcionais como critérios de aceitação

- A escrita de requisitos não-funcionais é bastante mais simples e talvez até mais direta.
- Quando um requisitos não-funcional se aplicar a apenas uma *user story*, é forte indicativo de que seja um bom critério de aceitação.

# Requisitos não-funcionais como critérios de aceitação

- *Story*: Eu, enquanto cliente do banco, desejo realizar pagamentos e transferências via PIX.
- Critério de aceitação: A confirmação do pagamento/transferência não deve ultrapassar 5 segundos. (performance)
- Critério de aceitação: O pagamento/transferência deverá ser aceito pelo Banco Central. (segurança)
- Critério de aceitação: O usuário necessitará de apenas 3 cliques + digitações para executar o/a pagamento/transferência. (usabilidade)

# Requisitos não-funcionais como DoD

- Quando há requisitos não-funcionais que abrangem diversas *stories*, pode-se coloca-los como critérios da DoD destas *stories*.
- No documento de requisitos (*product backlog*, se usado o Scrum), pode-se colocá-los à parte, porém de modo não fiquem escondidos.

# Requisitos não-funcionais como DoD

São exemplos de requisitos não-funcionais a serem colocados na DoD.

- Deve rodar em Windows, Mac e Linux. (usabilidade)
- A interface deve ser responsiva (*desktop*, *smart phone* e *tablet*). (usabilidade)

# Referências


- Sommerville, Ian. Software Engineering - Global Edition. 10ed. 2016. Pearson Education.
- Sommerville, Ian. Engineering Software Products: An Introduction to Modern Software Engineering. 1ed. 2021. Pearson Education.
- Cohn, Mike. User Stories Applied: For Agile Software Development. 1ed. 2004. Addison Wesley.
- K21 Global. Como é a user story? https://k21.global/pt/blog/como-e-a-user-story
- Huerter, Derek. Definition of Done. https://www.leadingagile.com/2017/02/definition-of-done/
- Core BTS. Agile User Story Splitting by Non-Functional Requirements. https://corebts.com/blog/agile-user-story-splitting-non-functional-requirements/
- Saboe, Dave. Lightning Cast: Non-Functional Requirements in Agile. https://masteringbusinessanalysis.com/lightning-cast-non-functional-requirements-in-agile/