Requisitos de *Software*

Prof.ª Ma. Jessica Oliveira

Católica
Universidade Católica de Brasília

Aula 08 – 12/05/2025

Métodos Ágeis e sua influência na Engenharia de Requisitos.

Universidade Católica de Brasília

Requisitos Tradicionais.

- Nos modelos tradicionais de desenvolvimento de software, como o modelo cascata, os requisitos são definidos de forma completa e detalhada no início do projeto.
- Essa abordagem pressupõe que todas as necessidades do cliente podem ser antecipadas e documentadas antes do início do desenvolvimento.
- Os requisitos são geralmente registrados em documentos extensos, como especificações de requisitos de software (SRS), e qualquer alteração posterior é tratada por meio de processos formais de controle de mudanças.

Requisitos Ágeis.

- Em contraste, os métodos ágeis reconhecem que os requisitos podem evoluir ao longo do tempo, à medida que o entendimento do produto e das necessidades do cliente se aprofunda.
- Assim, os requisitos ágeis são tratados como artefatos dinâmicos e iterativos, que são continuamente refinados e priorizados com base no feedback constante dos *stakeholders*.
- Essa abordagem promove uma maior flexibilidade e adaptabilidade no desenvolvimento de *software*.

Aspecto	Requisitos Tradicionais	Requisitos Ágeis
Documentação	Extensa e detalhada.	Leve e evolutiva.
Estabilidade	Estática após aprovação inicial.	Dinâmica e adaptável.
Responsável	Analista de Requisitos.	Equipe toda, com foco no *Product Owner*.
Validação	Formal e pontual.	Contínua e iterativa
Ferramentas	SRS, Casos de Uso.	Histórias de Usuário, Backlog.

Engenharia de Requisitos no Scrum.

Católica
Universidade Católica de Brasília

Introdução.

- O Scrum é um framework ágil que enfatiza a entrega incremental de valor por meio de ciclos curtos de desenvolvimento chamados *Sprints*.
- Embora o Scrum não defina explicitamente um papel de Analista de Requisitos, as atividades de engenharia de requisitos estão incorporadas nas responsabilidades dos membros da equipe Scrum, especialmente do *Product Owner* (PO).

Papel do *Product Owner* (PO).

- O PO é responsável por maximizar o valor do produto resultante do trabalho da equipe de desenvolvimento.
- Isso inclui a gestão eficaz do *Product Backlog*, que é a lista ordenada de tudo o que é necessário no produto.
- O PO atua como a principal interface entre os stakeholders e a equipe de desenvolvimento, garantindo que as necessidades do cliente sejam compreendidas e priorizadas adequadamente.

Atividades de ER no Scrum.

- As atividades tradicionais de engenharia de requisitos são adaptadas no contexto do Scrum da seguinte forma:
  - **Elicitação:** realizada por meio de interações contínuas com os *stakeholders*, como reuniões, entrevistas e *workshops*.
  - **Análise e Especificação:** os requisitos são representados como Histórias de Usuário no *Product Backlog*, frequentemente acompanhadas de critérios de aceitação.
  - **Priorização:** o PO prioriza os itens do backlog com base no valor para o negócio, riscos e dependências.
  - **Validação:** a validação dos requisitos ocorre iterativamente durante as revisões de Sprint, onde os *stakeholders* fornecem *feedback* sobre os incrementos entregues.

Product Backlog, Refinamento e *Definition of Done* (DoD).

Product Backlog.

- É o artefato central do *framework* Scrum e representa a lista ordenada de tudo o que se espera de um produto, ou seja, **as funcionalidades**, **melhorias**, **correções** e **requisitos técnicos** que precisam ser desenvolvidos.
- Cada item dessa lista é chamado de *Product Backlog Item* (PBI) e pode ser representado por **histórias de usuário**, ***bugs***, **tarefas técnicas** ou ***spikes*** (investigações).
- O *backlog* é **dinâmico e evolutivo**. Isso significa que ele está em constante atualização, sendo refinado ao longo do tempo para refletir as reais necessidades dos stakeholders e do negócio.

Product Backlog.

- Essa atualização é feita com base em **aprendizados**, ***feedbacks*** e **mudanças de contexto**.
- A **priorização dos itens** do *backlog* é responsabilidade do ***Product Owner***, que precisa balancear valor de negócio, riscos, dependências técnicas, prazo, viabilidade e retorno sobre o investimento (ROI).
- Não se trata apenas de ordenar tarefas; trata-se de **definir a melhor estratégia de entrega de valor ao cliente**.

Refinamento do *Backlog*.

- O *Backlog Refinement* é uma atividade contínua em que o PO e a equipe de desenvolvimento colaboram para revisar e ajustar os itens do backlog.
- Isso pode incluir a decomposição de itens grandes em menores, a adição de detalhes e critérios de aceitação, e a reordenação dos itens com base em novas prioridades.

Definition of Done (DoD).

- É um acordo explícito dentro da equipe Scrum que define os critérios que um incremento deve atender para ser considerado completo.
- Isso pode incluir aspectos como testes realizados, documentação atualizada e revisão de código.
- A DoD promove uma compreensão compartilhada da qualidade e assegura a consistência nas entregas.

Especificação e Priorização Ágil de Requisitos.

Católica
Universidade Católica de Brasília

Histórias de Usuário (*User Stories*).

- Técnica comum para capturar requisitos no desenvolvimento ágil.
- Elas são descritas de forma simples e centrada no usuário, geralmente seguindo o formato:

"Como [tipo de usuário], eu quero [ação] para [benefício]."

Histórias de Usuário (*User Stories*).

- Para garantir a eficácia dessa técnica, ela deve atender aos critérios INVEST:
    - Independente (*Independent*);
    - Negociável (*Negotiable*);
    - Valiosa (*Valuable*);
    - Estimável (*Estimable*);
    - Sucinta (*Small*);
    - Testável (*Testable*).

Técnicas de priorização.

- No contexto ágil, a priorização eficaz dos requisitos é crucial para maximizar o valor entregue.
- Duas técnicas comuns são:
  - **MoSCoW:** classifica os requisitos em quatro categorias: *Must have* (deve ter), *Should have* (deveria ter), *Could have* (poderia ter) e *Won't have* (não terá agora).
  - **Planning Poker:** uma técnica de estimativa em grupo que utiliza cartas numeradas para promover discussões e alcançar consenso sobre o esforço necessário para implementar cada requisito.

MoSCoW.

- ***Must have* (deve ter):** requisitos **obrigatórios**. Sem eles, o sistema não funciona ou não atende ao objetivo do projeto. São prioritários e inegociáveis.
- ***Should have* (deveria ter):** requisitos **importantes**, **mas não essenciais** para a primeira entrega. Eles agregam valor significativo, mas podem ser adiados.
- ***Could have* (poderia ter):** são **desejáveis**, mas de menor impacto. Se houver tempo e recursos, podem ser incluídos.
- ***Won't have* (por agora):** requisitos que **não serão incluídos na versão atual**. Isso não significa que foram descartados, apenas que não são prioridade agora.

MoSCoW.

- Exemplo prático em um aplicativo de agendamento de consultas:

  - *Must*: agendar, desmarcar e visualizar consultas.
  - *Should*: receber notificações por e-mail.
  - *Could*: personalizar o tema da interface.
  - *Won't*: acesso via reconhecimento facial.

Vamos para um desafio?

Católica
Universidade Católica de Brasília

Organização inicial:

- Dividam-se em dois grupos e, em cada um, deve conter:
  - 1 PO;
  - 1 Scrum Master;
  - 3 desenvolvedores.
- Cada grupo representará uma equipe Scrum trabalhando em um mesmo produto base, mas com liberdade para personalizar o público-alvo e as funcionalidades.

Produto simulado: Gerenciador de Vida Acadêmica.

Aplicativo voltado para estudantes universitários, cujo objetivo é facilitar a organização de tarefas, provas, trabalhos e outras atividades acadêmicas.

Católica
Universidade Católica de Brasília

Exploração do Produto e Adaptação ao Público-Alvo.

- Cada grupo escolhe um perfil específico de estudante para o qual o aplicativo será desenvolvido (ex.: calouro, estudante EAD, mãe universitária, aluno noturno, etc.).
- Com base nesse público, os grupos devem discutir rapidamente quais funcionalidades esse app deve ter para resolver problemas reais.

Criação das Histórias de Usuário.

- A equipe (liderada pelo PO) elabora de 5 a 7 histórias de usuário;
- Para cada história, os membros devem adicionar, pelo menos, quatro critérios de aceitação, respondendo: “Como saberei que essa história está pronta?”
- Exemplo:
  - História: “Como estudante noturno, eu quero receber lembretes automáticos das datas de provas para não me esquecer nos dias corridos.”
  - Critérios de Aceitação:
    - O usuário pode ativar/desativar lembretes.
    - Os lembretes incluem data, horário e disciplina.
    - O sistema envia a notificação no dia anterior e no mesmo dia da prova.

Refinamento do Backlog.

- As histórias são revisadas:
  - Estão seguindo o padrão INVEST?
  - Alguma está grande demais e precisa ser dividida?
  - Estão suficientemente claras para serem implementadas?
- Os Devs e o SM participam sugerindo melhorias ou fazendo perguntas ao PO, simulando o ambiente colaborativo do Scrum.

Priorização com Técnica Ágil.

- Classifiquem todas as histórias criadas na etapa anterior dentro das categorias MoSCoW
- A equipe precisa justificar ao menos uma escolha em cada categoria. Exemplo: *“Colocamos a funcionalidade X como Must porque sem ela o app não tem função básica.”*
- Durante essa etapa, o Scrum Master atua como facilitador da discussão, garantindo que todos sejam ouvidos.
- O PO é o responsável por bater o martelo final na categorização, assumindo seu papel de priorizador estratégico do produto.

Definição da *Definition of Done*.

- O grupo define os critérios mínimos para que qualquer história seja considerada “pronta” no projeto.
- Exemplo de DoD:
  - Todos os critérios de aceitação foram atendidos.
  - Código testado e revisado.
  - Interface aprovada pelo PO.
  - Funcionalidade documentada.

Apresentação dos resultados.

- Cada grupo apresenta:
  - 2 a 3 histórias de usuário com critérios de aceitação;
  - Técnica de priorização aplicada e justificativas;
  - Sua *Definition of Done*.
- Deve ser submetido no AVA todos os materiais desenvolvidos pela equipe em cada etapa da atividade.

Dúvidas?

jessica.oliveira@p.ucb.br

Católica
Universidade Católica de Brasília

Referências.

- SOMMERVILLE, I. **Engenharia de Software.** 9ª ed. Pearson, 2011.
- PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: Uma Abordagem Profissional.** 8ª ed. AMGH, 2016.
- SCHWABER, K.; SUTHERLAND, J. **The Scrum Guide™**. 2020. Disponível em: https://scrumguides.org/
- AMBLER, S. W. **Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process.** Wiley, 2002.