ENGENHARIA DE REQUISITOS

Sheila Reinehr

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Verificação de requisitos de *software*

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Reconhecer a importância da verificação de requisitos de *software*.
- Aplicar *checklists* de verificação de requisitos funcionais e não funcionais de *software*.
- Criar casos de teste para a verificação de requisitos funcionais e não funcionais de *software*.

Introdução

Todos sabemos que o mercado tem se tornado cada vez mais exigente com relação à qualidade dos produtos de *software*. Isso se dá em parte porque existe uma grande quantidade de *softwares* disponíveis para resolver diferentes tipos de problemas do cotidiano. As lojas virtuais oferecem uma infinidade de aplicativos para celular, tanto para iOS quanto para Android, que estão disponíveis a apenas um toque de distância. Assim como é fácil baixar e começar a usar, também é fácil desinstalar e escolher outro aplicativo para substituí-lo.

Essa intensa exposição à tecnologia na vida pessoal torna os usuários mais exigentes inclusive com os *softwares* que ele utiliza no trabalho. Nesse ambiente, entretanto, não é tão simples substituir um produto por outro. Não se instala e desinstala um *software* corporativo, como, por exemplo, um sistema de gestão integrada, com a mesma facilidade com que se substitui um aplicativo para acompanhar exercícios físicos pelo *smartwatch*.

E como garantir que os produtos de *software* tenham a qualidade esperada e proporcionem uma experiência extraordinária ao usuário? A resposta não é trivial e envolve todas as etapas do ciclo de desenvolvimento de *software*, desde as primeiras conversas sobre o escopo do projeto até a entrega de uma versão ou do produto final ao usuário. Não importa que tipo de método estejamos utilizando para desenvolver, o objetivo é o mesmo: fazer uma entrega de qualidade.

Diferentemente da manufatura, que emprega muitas etapas automatizadas, o desenvolvimento de *software* é uma atividade essencialmente cognitiva, que apresenta desafios muito mais complexos, pois erros humanos podem ser cometidos durante todas as fases. E o que podemos fazer para reduzir os efeitos nocivos desses erros? Utilizar técnicas de verificação e validação ao longo das atividades de todo o ciclo de vida. E isto começa pela etapa de requisitos.

Neste capítulo você vai estudar a verificação de requisitos de *software* e sua importância para o ciclo de desenvolvimento. Vai ver como utilizar *checklists* de verificação de requisitos funcionais e não funcionais e também como criar casos de teste para a verificação de requisitos funcionais e não funcionais.

1 Importância da verificação de requisitos de *software*

A engenharia de requisitos é a especialidade da engenharia de *software* que trata de todos os temas relacionados aos requisitos. Podemos compreender, de acordo com o SWEBOK (Software Engineering Body of Knowledge), que a engenharia de requisitos é "uma forma sistemática de tratar os requisitos" (BOURQUE; FAIRLEY, 2014, documento *on-line*).

Etapas da engenharia de requisitos

Ainda de acordo com o SWEBOK (BOURQUE; FAIRLEY, 2014, documento *on-line*): "A área de conhecimento de Requisitos de Software (KA) preocupa-se com a obtenção, análise, especificação, e validação de requisitos de *software* bem como o gerenciamento de requisitos durante todo o ciclo de vida do produto de *software*".

Vamos relembrar essas etapas referentes à engenharia de requisitos por meio da Figura 1. Considere o semicírculo mais externo como as atividades da engenharia de requisitos e o semicírculo interno como o desenvolvimento em si, ou seja, a implementação. Permeando todo o processo está o gerenciamento de requisitos.

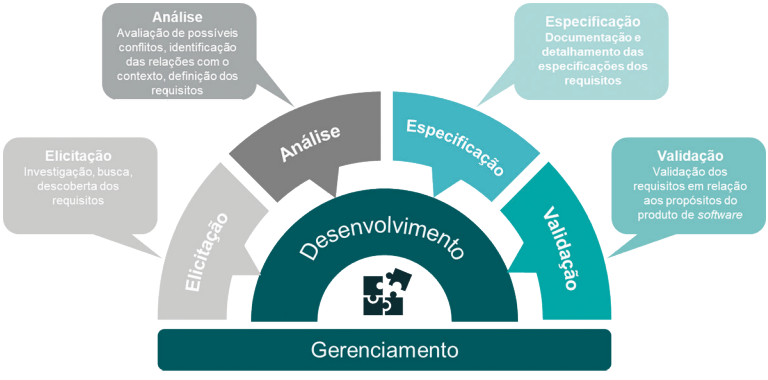

Figura 1. Etapas da engenharia de requisitos.

Conforme você pode observar na Figura 1, a primeira etapa se refere à **elicitação de requisitos**. A elicitação de requisitos é "[…] o primeiro estágio para construir uma compreensão sobre o problema que o *software* deve resolver" (BOURQUE; FAIRLEY, 2014, documento *on-line*). Nela são identificadas as fontes de informação e são selecionadas as técnicas de elicitação mais adequadas, de acordo com o contexto do projeto. Após a seleção, é possível aplicar essas técnicas para obter os requisitos, sejam eles funcionais ou não funcionais.

Embora estejamos utilizando a expressão "etapa de elicitação", é bom lembrar que isso não significa que é necessário que elicitar o conjunto completo de requisitos em um mesmo momento e nem que seja preciso chegar a níveis similares de detalhamento para cada requisito. Pode-se trabalhar de forma iterativa e incremental, como nas *sprints* dos métodos ágeis, nas quais os requisitos do *product backlog* vão sendo detalhados à medida que são necessários para alocação à *sprint*.

A segunda etapa se refere à **análise de requisitos**, que é o momento em que nos aprofundamos nos requisitos e buscamos identificar inconsistências e conflitos entre os requisitos. Identificamos também as fronteiras do sistema e com quem ele deve interagir. Segundo Vazquez e Simões (2016), o objetivo da elicitação é encontrar as peças do quebra-cabeças e o objetivo da análise é montá-lo.

A terceira etapa é a **especificação de requisitos**. Como o próprio nome sugere é o momento em que nos dedicamos a documentar os requisitos de modo que possam ser utilizados pelas etapas seguintes do desenvolvimento, evitando ambiguidades e retrabalho. O nível necessário de especificação varia de acordo com a criticidade do *software* para o ambiente onde ele será utilizado, da experiência da equipe desenvolvedora, dos riscos envolvidos etc. Níveis diferentes de especificação podem ser necessários para cada requisito do produto. Em sistemas nos quais o *software* é apenas um componente da solução, é comum que as especificações sejam mais robustas e complexas, envolvendo diversos tipos de documentos: definição do sistema, especificação de requisitos do sistema e especificação de requisitos do *software*.

Finalmente, a quarta etapa é chamada genericamente de **validação de requisitos**, embora, na verdade, ela englobe tanto a validação quanto a **verificação de requisitos**.

Fique atento

Verificação e validação são termos diferentes! De acordo com a norma internacional ISO/IEC/IEEE 12207 (ISO/IEC/IEEE, 2017, p. 10–11):

- **Validação** é "a confirmação, por meio do fornecimento de evidência objetiva, que o requisito foi atendido para um uso ou aplicação pretendidos específicos".
- **Verificação** é a "confirmação, por meio do fornecimento de evidência objetiva, que os requisitos especificados foram atendidos".

Ainda, de acordo com o CMMI-DEV 2.0 (CMMI INSTITUTE, 2018, p. 525), a diferença entre os termos é a seguinte:

- **Validação** "demonstra que a solução vai atender seu uso pretendido no ambiente alvo, isto é, 'você está construindo a coisa certa'".
- **Verificação** "endereça que o produto de trabalho ou a solução refletem adequadamente os requisitos especificados, isto é, 'você está construindo certo'".

De acordo com Bourque e Fairley (2014, p. 1-11),

> [...] os requisitos podem ser **validados** para assegurar que o engenheiro de *software* compreendeu os requisitos; é também importante **verificar** que o documento de requisitos está em conformidade com os padrões da empresa e que ele é compreensível, consistente e completo.

É comum dizermos que a verificação analisa se o produto foi construído certo, ou seja, de forma correta; e que a validação analisa se foi construído o produto certo, ou seja, aquele que os *stakeholders* desejavam (CMMI INSTITUTE, 2018).

Saiba mais

De acordo com o modelo CMMI-DEV 2.0 (CMMI INSTITUTE, 2018), a verificação e a validação na engenharia de *software* envolvem simulação, estudos de rastreabilidade, revisões ou auditorias funcionais, revisões ou auditorias físicas, revisões por pares, demonstrações, protótipos, revisões formais, testes de módulo, regressão e integração de sistema.

Finalmente, a próxima etapa se refere ao **gerenciamento de requisitos**. Nesta etapa vamos nos preocupar com como tratar os requisitos ao longo do seu ciclo de vida e como analisar e gerenciar as solicitações de mudanças que, inevitavelmente, ocorrerão.

Neste capítulo vamos nos concentrar na etapa de validação de requisitos, mais especificamente nas atividades de verificação de requisitos. Atividades de validação propriamente ditas não serão tratadas neste capítulo.

Importância da verificação de requisitos

À medida que os *softwares* foram se tornando mais complexos, mais integrados e mais críticos para todas as atividades cotidianas, mais importante ficou a garantia do seu correto funcionamento. Essa maior proximidade com a tecnologia vem tornando os usuários cada vez mais exigentes.

A grande questão é que há muitos pontos no ciclo de desenvolvimento nos quais erros que podem levar a falhas gigantescas de *software*. Isso ocorre porque as atividades envolvidas são de ordem cognitiva. A cada troca de mãos do requisito, ou seja, a cada troca de responsabilidade, um novo defeito pode estar sendo introduzido, como naquela brincadeira infantil do telefone sem fio.

Se no início do projeto o analista de requisitos não envolver todos os *stakeholders* relevantes para atuar como fonte de informação de requisitos, pode ser que um requisito muito importante não seja descoberto e permaneça invisível até as fases mais avançadas, levando ao retrabalho e a custos adicionais de desenvolvimento. Esta é a primeira passagem de bastão do requisito, ou seja, é a primeira transição. Ele sai do ambiente dos *stakeholders* e passa para o ambiente do projeto, via processo de elicitação de requisitos conduzido pelo analista de requisitos.

Caso os requisitos, uma vez elicitados, não estejam descritos de forma clara e não ambígua, eles podem ser implementados de forma incorreta. Esta é a segunda passagem de bastão do requisito. Ele sai das mãos do analista de requisitos e passa para as mãos da equipe técnica, que pode ser composta por arquitetos, projetistas e desenvolvedores, que irão projetar a arquitetura da aplicação, as interfaces e desenvolver o código.

Por fim, uma terceira passagem de bastão, ou seja, troca de responsabilidade, acontece quando a equipe de testes define os casos de teste que irão avaliar se o produto foi construído de acordo com o que estava especificado. Nesse caso o requisito sai das mãos do analista de requisitos e da equipe técnica e vai para as mãos dos testadores. Aqui podem acontecer problemas de interpretação que levam à construção de casos de teste ineficazes para o teste do produto.

Não podemos nos esquecer da última troca de bastão, que irá ocorrer quando o produto for implantado no ambiente-alvo e utilizado pelos seus usuários para cumprir as funções para as quais ele foi especificado.

Todas essas diversas trocas de responsabilidade podem levar a erros que se refletirão no ambiente de operação e podem trazer consequências diversas, como o retrabalho e seus custos associados ou falhas na operação, que podem gerar riscos de ferimentos ou de prejuízos financeiros. Ao longo da história da computação, diversos foram os casos de problemas não detectados em fases iniciais do desenvolvimento e que foram propagados até gerar grandes desastres. Vários deles você certamente já ouviu falar.

Exemplo

Diversos são os exemplos de falhas de grande impacto ocasionadas por problemas em *software*, muitos deles oriundos da etapa de requisitos. Vamos relembrar alguns exemplos famosos:

- O projeto de construção do **aeroporto de Denver**, nos Estados Unidos, subestimou a complexidade do algoritmo de controle das esteiras de bagagem, gerando dificuldade de integração entre o *hardware* e o *software*, e levando a prejuízos na ordem de U$ 1,1 milhão ao dia para a cidade de Denver (CALLEAM CONSULTING, 2008).
- Em 2019, a Boeing teve que recolher toda a sua frota de 387 aviões do modelo **Boeing 737 MAX**, que estava em 59 companhias áreas, devido a um problema no novo *software* (BOEING..., 2019).
- O foguete **Ariane 5** explodiu em pleno ar devido a uma diferença de tamanho de campos de dados entre ele e o seu antecessor, cujo *software* foi reaproveitado. O prejuízo foi de US$ 370 milhões (HINKEL, [201-?]).

Muitas dessas falhas poderiam ter sido evitadas se procedimentos adequados de qualidade tivessem sido adotados ao longo do ciclo de vida, conforme ilustra a Figura 2, por meio das atividades de Verificação 1 a 5, representadas pelos triângulos. Note que está representado apenas um ciclo de vida genérico e bastante simplificado, que pode ser uma *sprint*, por exemplo.

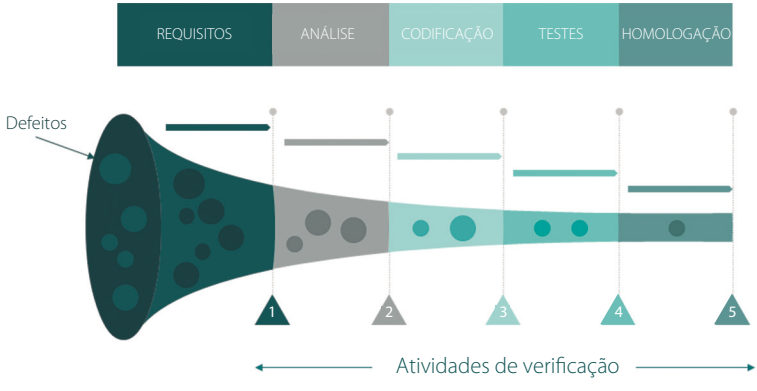

Figura 2. Defeitos inseridos e removidos ao longo do ciclo de desenvolvimento.

Como você pode observar, os defeitos estão presentes desde o início. Se não tivéssemos introduzido as atividades de verificação intermediárias, teríamos chegado à fase de testes com a quantidade inicial de defeitos (início do cone na figura). Possivelmente, esses defeitos ainda teriam sido amplificados, considerando que um defeito em um requisito pode implicar em defeitos em outros requisitos também.

Quando introduzimos a atividade de Verificação 1, os defeitos provenientes da etapa de requisitos são reduzidos. Note que não estamos aqui falando que todos os requisitos do produto deveriam estar definidos, ou seja, não estamos nos referindo a um ciclo cascata. Estamos nos referindo ao conjunto de requisitos que deveriam estar esclarecidos para uma versão ou *sprint*.

Ao encerrarmos a fase de análise, é inserida a atividade de Verificação 2 e, novamente, os defeitos são reduzidos, de modo a passar para a codificação um requisito que já esteja adequado para implementar. Se as atividades de Verificação 1 e Verificação 2 não tivessem sido realizadas, a codificação receberia todos os defeitos provenientes das fases anteriores.

Ao encerrarmos a codificação, esperamos que o desenvolvedor tenha realizado os testes de unidade ou testes unitários, que também são uma atividade de verificação. Ao final dessa fase, a atividade de Verificação 4 pode ser instituída tanto por meio de revisões de código quanto por meio de atividades de teste integrado.

Ao chegarmos na homologação, que é realizada pelo usuário, é esperado que esses defeitos tenham sido significantemente reduzidos. Note que, neste modelo fictício, se as atividades intermediárias não tivessem sido inseridas, a homologação receberia a quantidade inicial de defeitos, possivelmente ainda ampliada por outros inseridos como decorrência destes.

Diversas técnicas podem ser aplicadas para evitar que os erros se propaguem ao longo das fases, funcionando como uma espécie de filtro, como vimos na Figura 2. Quanto maior a quantidade de filtros e quanto mais finos eles forem, menos erros de propagarão. O problema é que esses filtros têm um custo, o chamado **custo da qualidade**, que deve ser cuidadosamente analisado para determinar o ponto de equilíbrio ideal.

Fique atento

Cada ponto de verificação precisa prover insumos para que o processo de desenvolvimento como um todo possa ser melhorado e a **causa raiz** dos erros possa ser identificada, de modo que no futuro os **defeitos possam ser prevenidos**.

Quando um defeito é detectado, ele precisa retornar para ser consertado, e isso é um retrabalho. Por isso é tão importante analisar os resultados provenientes das atividades de verificação. Portanto, *insights* sobre como melhorar o processo para evitar os erros constituem uma das saídas mais valiosas da análise de um processo de verificação.

A definição dos pontos onde essas intervenções são necessárias dependerá, primeiramente, da criticidade que uma falha representa para determinado sistema. Se a consequência for a perda de vidas, certamente os investimentos em qualidade serão todos compensados. Se a consequência for apenas uma pequena parada de um sistema que não tenha consequências críticas, então é possível que não se invista tanto nessas atividades de qualidade, priorizando esforços em outros pontos mais críticos.

Como realizar a verificação de requisitos

Os requisitos podem ser verificados de diversas formas. A primeira delas, naturalmente, é a leitura crítica dos artefatos, que é realizada pelo próprio autor, ou seja, o analista de requisitos, analisando pontos de ambiguidade, de inconsistência e de falta de completude nas definições. Para isso, ele pode ou não se apoiar em um *checklist,* que é um instrumento que funciona como uma espécie de lembrete, para que nenhum ponto seja esquecido.

Outras formas envolvem a revisão por pares, ou seja, a revisão que é realizada, como o próprio nome diz, pelos pares do analista de requisitos. Esses pares podem ser outros analistas de requisitos ou pessoas técnicas da equipe. São consideradas revisões por pares as listadas a seguir (CMMI PRODUCT TEAM, 2010):

- inspeções;
- *walkthroughs* estruturados;
- refatoração deliberada;
- programação em pares (*pair programming*).

Da mesma forma que a autoavaliação, a avaliação por pares pode ou não ser apoiada por um *checklist*. Na próxima seção veremos como elaborar e utilizar um *checklist*.

Outra forma, e a mais utilizada, são os testes de *software*. Testes são realizados para identificar erros que já estão materializados no código. Abordaremos esse assunto mais adiante neste capítulo.

2 *Checklists* de verificação de requisitos de *software*

Conforme apresentado anteriormente, a norma internacional ISO/IEC/IEEE 12207:2017(E) (ISO/IEC/IEEE, 2017, p. 82) define a finalidade do processo de verificação como "[…] prover evidência objetiva que o sistema ou elemento do sistema atende completamente seus requisitos e características especificados".

A mesma norma define que:

> […] o processo de verificação identifica anomalias (erros, defeitos e falhas) em qualquer item de informação (requisitos de sistema/*software* ou descrição de arquitetura), elementos implementados, ou processos de ciclo de vida usando métodos, técnicas, padrões e regras apropriados (ISO/IEC/IEEE, 2017, p. 76).

Existem diversas formas de fazer a verificação dos requisitos. Uma delas é a **revisão por pares**, que nada mais é do que um par realizando a revisão sobre o trabalho do colega, com o objetivo de identificar inconsistências ou anomalias. Um par é um profissional de *expertise* equivalente ou superior a do(s) autor(es) do artefato, que pode contribuir tecnicamente, fazendo uma avaliação objetiva do artefato.

Artefato é o resultado da realização de uma atividade, podendo ser um documento, um diagrama ou o próprio código. Revisões por pares podem ser aplicadas a qualquer artefato do projeto. Vamos aqui discutir a sua aplicação para a verificação dos artefatos relacionados às etapas de requisitos, mas o raciocínio utilizado se aplica a qualquer outro tipo de artefato do ciclo de vida de desenvolvimento, incluindo o código.

As revisões por pares podem acontecer no formato de **inspeções, *walk-throughs* estruturados**, **auditorias** ou outras formas de revisão. Sua aplicação vai depender do grau de formalismo desejado para a revisão. Inspeções são consideradas revisões mais formais e têm um formato próprio para acontecer.

As revisões também podem ser feitas seguindo diferentes abordagens. Podem ser baseadas em ***checklists***, em **cenários** de execução, com foco na **perspectiva** de algum usuário, com foco em uma **funcionalidade** específica ou ser feita até mesmo de forma *ad hoc*.

Saiba mais

Um dos requisitos não funcionais importantes referentes à qualidade interna de um produto de *software* é a **verificabilidade**. Verificabilidade se refere a "[...] quão bem os componentes de *software* ou o produto integrado podem ser avaliados para demonstrar se as funções do sistema funcionam conforme o esperado" (WIEGERS; BEATTY, 2013, p. 286).

Para identificar esses requisitos, você pode utilizar como dica as seguintes perguntas:

- Como nós poderemos confirmar que cálculos específicos estão fornecendo os resultados esperados?
- Existe alguma parte do sistema que não leva a resultados determinísticos, de forma que poderia ser difícil de determinar se está funcionando corretamente?
- É possível ter conjuntos de dados de teste que tenham uma alta probabilidade de revelar quaisquer erros nos requisitos e sua implementação?
- Que relatórios de referência ou outras saídas podem ser utilizadas para verificar se o sistema está produzindo os resultados corretamente?

Artefatos de requisitos

Se estamos focando em revisões dos requisitos, a pergunta é: que artefatos devem ser revisados? Os requisitos, uma vez elicitados, podem ser representados de diversas formas, dependendo do contexto do projeto, da organização desenvolvedora, das exigências do contratante, da experiência da equipe desenvolvedora e outros.

Vamos considerar neste capítulo duas formas de especificar requisitos: a abordagem orientada a casos de uso e a abordagem orientada e histórias de usuário. Além disso, vamos considerar também a lista de requisitos, que pode ser utilizada nas duas abordagens. Dessa forma, os seguintes artefatos serão considerados neste capítulo:

- Lista de requisitos funcionais e não funcionais.
- Casos de uso:
  - Diagrama de casos de uso.
  - Especificação de casos de uso.
- Histórias de usuário:
  - Cartão da história de usuário.
  - Critérios de aceitação das histórias de usuário.

Checklist para revisão de artefatos de requisitos

Um *checklist* pode ser considerado como uma lista contendo pontos de atenção para apoiar os revisores. Geralmente, ela é constituída dos itens que anteriormente já foram considerados problemáticos e que a equipe deseja evitar. O ideal é que cada artefato tenha seu *checklist* com itens próprios.

Vamos começar com alguns critérios que podem ser usados para analisar a **lista de requisitos**, que pode conter tanto os requisitos funcionais quanto os requisitos não funcionais. O Quadro 1 apresenta uma sugestão para este *checklist*. Itens identificados como "não" representam pontos de atenção que deverão ser priorizados e tratados pelo analista de requisitos. É sugerido que o revisor complemente as informações dos itens assinalados como "não", de modo a facilitar as correções.

Quadro 1. *Checklist* para a lista de requisitos

Atributo	Definição	Sim	Não	Não se aplica
Completo	O requisito está especificado de forma completa e que possibilita que o desenvolvedor o implemente?			
Correto	O requisito reflete o que o usuário, cliente ou seus representantes desejam?			
Único	O requisito descreve uma única capacidade, característica, restrição ou atributo de qualidade?			

(Continua)

(Continuação)

Quadro 1. Checklistt para a lista de requisitos

Atributo	Definição	Sim	Não	Não se aplica
Viável	O requisito é viável técnica e financeiramente de ser implementado, de acordo com as restrições do projeto?			
Necessário	O requisito tem um motivo de existir, que é representado pelo seu relacionamento a uma fonte de informação e a um objetivo de negócio?			
Priorizado	O requisito tem uma prioridade atribuída para que possa ser alocado a uma versão do *software*?			
Não ambíguo	O requisito não contém ambiguidades que levem os *stakeholders* a interpretá-lo de forma diferente?			
Verificável	O requisito é possível de ser verificado posteriormente quanto à sua implementação?			
Conforme	O requisito está em conformidade com os padrões de especificação estabelecidos, se houver?			

Fonte: Adaptado de Wiegers e Beatty (2013) e ISO/IEC/IEEE (2018).

O diagrama de casos de uso é um diagrama que ajuda o analista de requisitos a se comunicar com os usuários. Embora seja composto por elementos simples, o diagrama tem um grande poder de comunicação. O Quadro 2 apresenta uma sugestão de *checklist* para verificação do **diagrama de casos de uso**. Itens identificados como "não" representam pontos de atenção que deverão ser priorizados e tratados pelo analista de requisitos.

Quadro 2. *Checklist* para o diagrama de casos de uso

Atributo	Definição	Sim	Não	Não se aplica
Atores (escopo)	Todos os atores estão representados no diagrama?			
Atores (formato)	Todos os atores estão representados por um boneco palito e um texto com a identificação do ator?			
Relacionamentos de generalização entre atores (representação)	O relacionamento de generalização entre os atores é representado por uma seta vazada que aponta do ator filho para o ator pai?			
Relacionamentos de generalização entre atores (significado)	O relacionamento de generalização expressa corretamente a intenção de herança entre os atores envolvidos?			
Casos de uso (representação)	Todos os casos de uso são representados por elipses e identificados por meio de um verbo no infinitivo seguido de um complemento?			
Casos de uso (escopo)	Todos os casos de uso que representam as funcionalidades que os *stakeholders* desejam estão representados?			
Relacionamentos dos casos de uso com atores	Todos os casos de uso estão ligados a pelo menos um ator?			

(Continua)

(Continuação)

Quadro 2. *Checklist* para o diagrama de casos de uso

Atributo	Definição	Sim	Não	Não se aplica
Relacionamentos de <*include*> entre casos de uso	Os relacionamentos de <*include*> estão representados por setas pontilhadas que partem do caso de uso principal para o caso de uso incluído?			
Relacionamentos de <*extend*> entre casos de uso	Os relacionamentos de <*extend*> estão representados por setas pontilhadas que partem do caso de uso que estende para o caso de uso principal?			
Relacionamentos de generalização entre casos de uso (representação)	O relacionamento de generalização entre os casos de uso é representado por uma seta vazada que aponta do caso de uso filho para o caso de uso pai?			
Relacionamentos de generalização entre casos de uso (significado)	O relacionamento de generalização expressa corretamente a intenção de herança entre os casos de uso envolvidos?			

O diagrama de casos de uso sozinho não é suficiente como especificação de requisitos funcionais, portanto é necessário complementar com um documento de especificação de casos de uso (que pode também ser feito dentro de uma ferramenta de modelagem).

O Quadro 3 apresenta uma sugestão de *checklist* para verificação das **especificações de casos de uso**. Alguns itens se referem à especificação e outros se referem à consistência entre o diagrama de casos de uso e as suas especificações correspondentes. Itens identificados como "não" representam pontos de atenção que deverão ser priorizados e tratados pelo analista de requisitos.

Quadro 3. *Checklist* para a especificação de casos de uso

Atributo	Definição	Sim	Não	Não se aplica
Identificação do caso de uso	O caso de uso tem um identificador único e um texto contendo um verbo no infinitivo e um complemento?			
Ator principal	O ator principal está identificado corretamente?			
Ator secundário	O ator secundário (se houver) está identificado corretamente e participa do caso de uso juntamente com o ator principal?			
Pré-condição	As pré-condições (se houver) estão descritas corretamente?			
Fluxo principal	Todos os passos do fluxo principal estão numerados sequencialmente e descrevem claramente o diálogo entre o ator e o sistema?			
Fluxos alternativos (descrição)	Todos os fluxos alternativos (se houver) estão numerados sequencialmente e descrevem claramente o diálogo entre o ator e o sistema?			
Fluxos alternativos (retorno)	Todos os fluxos alternativos (se houver) definem qual é o próximo passo a ser executado ao seu término?			

(Continua)

(Continuação)

Quadro 3. *Checklist* para a especificação de casos de uso

Atributo	Definição	Sim	Não	Não se aplica
Fluxos alternativos (chamadas)	Para todos os fluxos alternativos (se houver) estão identificados os pontos de chamada no fluxo básico?			
Fluxos de exceção (descrição)	Todos os fluxos de exceção (se houver) estão numerados sequencialmente e descrevem claramente o diálogo entre o ator e o sistema em uma condição de exceção?			
Fluxos de exceção (retorno)	Todos os fluxos de exceção definem qual é o próximo passo a ser executado ao seu término?			
Fluxos de exceção (chamadas)	Para todos os fluxos de exceção estão identificados os pontos de chamada, seja no fluxo básico, seja nos fluxos alternativos?			
Pós--condições	As pós-condições (se houver) estão descritas corretamente?			

Quando a equipe de desenvolvimento utiliza métodos ágeis, é comum que os requisitos funcionais sejam expressos como **histórias de usuário**. As histórias de usuário são compostas por cartões, com uma declaração da necessidade do usuário e por critérios de aceitação que normalmente detalham a história ou especificam os requisitos não funcionais. O responsável pelos itens que compõem o *product backlog*, e que irão gerar as histórias, é o *product owner* (PO). Cabe a ele representar os interesses do cliente ou do usuário junto à equipe de desenvolvimento. As histórias de usuário geralmente são escritas de forma conjunta, entre o PO e a equipe de desenvolvimento.

A principal característica dos times ágeis é a comunicação. Os integrantes são estimulados a se reunir em *workshops* de esclarecimento de histórias e seguem a máxima dos 3 Cs: cartão + conversa + confirmação (critérios de aceitação). Portanto, é comum que as revisões já aconteçam ao longo do desenvolvimento das próprias histórias, durante esses *workshops* (PATTON, 2014).

No entanto, nem todos os ambientes ágeis funcionam tão bem. É bastante comum que as histórias de usuário sejam, na verdade, chamados que chegaram pelo Help Desk, foram sendo armazenados em um *backlog* de solicitações de mudança e foram alocados à *sprint* sem antes terem passado por uma conversa para aprofundar o seu entendimento. Nesses casos, a aplicação de um *checklist* pode apoiar a equipe na melhoria da qualidade da compreensão dessas histórias.

O Quadro 4 apresenta uma sugestão para o *checklist* de histórias de usuário. Note que esses critérios são muito similares aos critérios da lista de requisitos. É sugerido que o revisor complemente as informações dos itens assinalados como "não", de modo a facilitar as correções a serem realizadas pelo autor do artefato.

Quadro 4. *Checklist* para histórias de usuário

Atributo	Definição	Sim	Não	Não se aplica
Formato	A história de usuário está escrita no padrão: "Como (nome do papel), eu quero (…) de modo que (…)"?			
Completa	A história de usuário está especificada de forma completa e que possibilita que o desenvolvedor o implemente?			
Correta	A história de usuário reflete o que o usuário, cliente ou seus representantes desejam?			
Única	A história de usuário descreve uma única capacidade, característica, restrição ou atributo de qualidade?			

(Continua)

(Continuação)

Quadro 4. *Checklist* para histórias de usuário

Atributo	Definição	Sim	Não	Não se aplica
Viável	A história de usuário é viável técnica e financeiramente de ser implementada, de acordo com as restrições do projeto ou da *sprint*?			
Neces-sária	A história de usuário tem um motivo de existir, que é repre-sentado pelo seu relaciona-mento a uma fonte de informa-ção e a um objetivo de negócio?			
Priorizada	A história de usuário tem uma prioridade atribuída para que possa ser alocada a uma *sprint?*			
Não ambígua	A história de usuário não contém ambiguidades que levem os *stakeholders* a interpretá-la de forma diferente?			
Verificável	A história de usuário é possível de ser verificada posteriormente quanto à sua implementação?			

Melhoria da qualidade das revisões

As revisões por pares são momentos preciosos para promover a melhoria na qualidade das especificações de requisitos. Para que as revisões sejam proveitosas e se ganhe em produtividade, Wiegers (2006) apresenta as dicas a seguir.

- **Eduque os revisores:** explique para os revisores o que é esperado deles e qual será o processo usado na revisão. Aprender a dar *feedback* faz parte desse aprendizado. O foco da revisão deve estar sobre o artefato, e não sobre a pessoa.
- **Não sobrecarregue os revisores:** não espere que o documento esteja 100% completo. Ele pode ir sendo revisado aos poucos. É mais fácil conseguir 30 minutos de uma pessoa do que uma tarde toda.

- **Construa parcerias colaborativas com os representantes dos usuários e com outros membros da equipe do projeto:** a voz do cliente é muito importante na revisão dos requisitos, embora esta seja uma atividade de validação e não de verificação. Envolva o usuário ou representantes do usuário.
- **Convide os revisores certos:** envolva desenvolvedores e testadores também no processo de revisão, pois eles podem ter *insights* que os analistas de requisitos não têm. São visões diferentes que contribuem para a revisão. Eles podem usar a técnica de *perspective based reading*, ou leitura baseada em perspectiva (é uma leitura orientada sob a ótica específica daquele papel).
- **Faça os revisores analisarem os entregáveis apropriados:** nem todos os revisores terão que avaliar todos os artefatos de requisitos. Encaminhe o artefato para o revisor que mais pode contribuir com aquele artefato especificamente.
- **Projete para facilitar a revisão:** represente os requisitos de forma a facilitar a revisão.
- **Inspecione todos os entregáveis de requisitos:** embora as revisões informais sejam úteis, as inspeções formais são mais profundas e descobrem mais detalhes, uma vez que fazem com que os inspetores realmente analisem os artefatos.
- **Enfatize encontrar os maiores erros:** os requisitos faltantes são os mais difíceis de serem detectados. Invista tempo para este tipo de requisito.

3 Casos de teste de requisitos de *software*

A técnica mais utilizada para realizar a verificação de artefatos de desenvolvimento de *software* é o teste. Testes de *software* são utilizados para avaliar o código implementado, seja ele no formato de um componente isolado, seja no formato de um produto integrado.

Não é nosso objetivo neste capítulo o aprofundamento nas diversas técnicas de teste de *software*, mas apresentar os casos de teste como forma de refinamento e verificação de requisitos, tanto em ambientes tradicionais quanto em ambientes ágeis.

Testes em ambientes tradicionais

Geralmente, os casos de teste são escritos como apoio para que a equipe de testes possa realizar o seu trabalho. Existe uma variedade de formatos de escrita de casos de teste, que vão desde *templates* no formato de planilhas eletrônicas até ferramentas automatizadas, com robôs programados para executar os testes.

Independentemente da forma como o teste seja implementado, é importante compreender o que está envolvido no planejamento dos testes. Se voltarmos à Figura 2, veremos que existe uma etapa de testes representada logo após a codificação. Esse é um exemplo apenas didático e simplificado. Na realidade, os testes são realizados desde a fase de codificação, como forma do próprio desenvolvedor avaliar o seu trabalho. Em seguida, o produto segue para a equipe de testes, quando a empresa tem uma. E, finalmente, passa para o ambiente de homologação, onde iniciam os testes que envolvem o usuário ou um representante do usuário. Nesse caso, passamos a chamar de validação, e não mais de verificação, conforme as definições que já foram apresentadas anteriormente neste capítulo.

Fique atento

O grande benefício do desenvolvimento dos testes concomitantemente aos requisitos, ou logo após esses, é que os testes ajudam a esclarecer pontos obscuros dos requisitos. Refinamentos são possíveis sobre os requisitos, uma vez que se busca as formas para testá-lo. Visão similar foi incorporada pelos métodos ágeis, como se verá adiante.

Para a especificação dos casos de teste, o *template* apresentado no Quadro 5 pode ajudar. Inicialmente, deve ser identificada a origem do requisito. Neste caso, temos um exemplo usando casos de uso. Em seguida deve vir o identificador único do caso de teste, depois temos uma coluna onde são registrados os passos que o testador deve executar e qual é o resultado esperado.

Se tudo der certo e o sistema prover a resposta esperada, ele indica isto marcando um X na coluna ✓. Caso a execução não tenha ocorrido com sucesso, então ele marcará um X na coluna que aparece como ✗ na planilha. Nesse segundo caso, ou seja, o sistema não apresentou o resultado esperado, o testador deve reportar os detalhes para que o desenvolvedor possa corrigir.

Quadro 5. *Template* para casos de teste

ID do caso de uso	ID do caso de teste	Passos	Resultado esperado	✓	✗	N/A
<inserir o id do caso de uso>	<inserir o id do caso de teste>	<inserir os passos detalhados para a realização do caso de teste>	<inserir o resultado esperado ao final da execução do passo>			
EXEMPLO						
UC-CAD-001	CT-CAD-001	1 – Selecionar função Cadastrar Cliente	Tela de Cadastro de Cliente aberta	×		
		2 – Preencher CPF incorreto	Msg de erro exibida: "CPF inválido"		×	
		(…)				

Casos de teste são considerados ativos do processo de desenvolvimento e fazem parte do produto. A cada nova alteração em uma funcionalidade, os casos de teste precisam ser revisados para que continuem coerentes com as implementações. No caso de testes automatizados, é necessário que o código do teste seja refeito.

Testes em ambientes ágeis

Em ambientes ágeis de desenvolvimento de *software*, de acordo com Leffingwell (2011), é comum pensar de forma mais holística e sistemática, em que histórias (requisitos), implementação (código) e validação (testes de aceitação, testes unitários e outros) estejam juntos. Leffingwell (2011) apresenta uma matriz que descreve os tipos de teste em ambientes ágeis, ilustrada na Figura 3.

No quadrante Q1 estão os testes realizados pelo desenvolvedor, geralmente automatizados e em grande volume. São os **testes unitários e de componentes**. No quadrante Q2 estão os **testes funcionais**, que testam o funcionamento das histórias de usuário. Esses testes podem ser automatizados ou manuais. No quadrante Q3 se encontram os **testes de aceitação** ou testes de sistema; geralmente são testes manuais, que envolvem usuários e testadores em ambientes que simulam o ambiente real. No quadrante Q4 estão os **testes de qualidade** do sistema; geralmente envolvem o uso de ferramentas para os testes de desempenho e de carga.

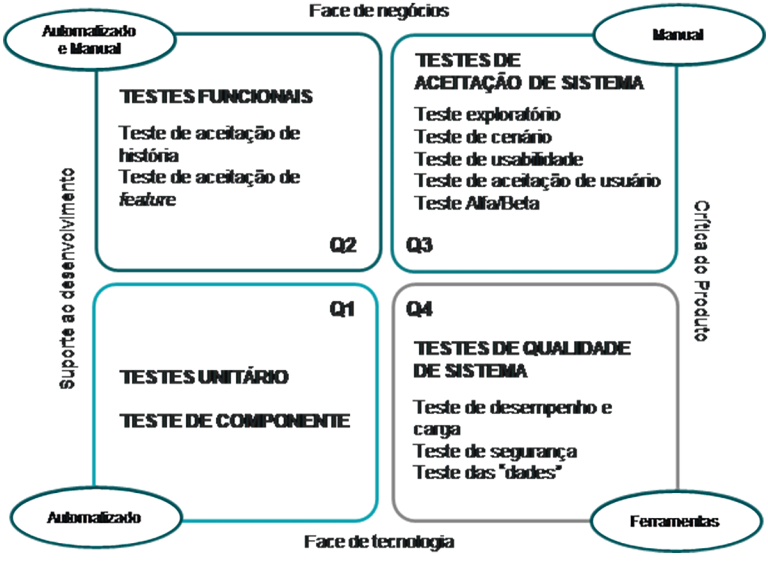

Figura 3. A matriz de testes ágeis.
Fonte: Adaptada de Leffingwell (2011).

Os **testes de aceitação de histórias**, representados no quadrante Q2 da Figura 3, são testes realizados pelos desenvolvedores. Trata-se de testes funcionais para garantir que as histórias de usuário entregam o que era pretendido. Geralmente, o time ágil passa grande parte do tempo definindo esses testes, pois esta é uma forma de refinar o comportamento esperado da história.

É comum, nestes ambientes, que a história seja uma afirmação curta, escrita no formato padronizado que expressa quem + o que + porquê. Para que ela possa ser adequadamente compreendida, os testes de aceitação da história são escritos e devem conter todos os detalhes para que o desenvolvedor possa implementá-la.

Exemplo

Considere a seguinte história de usuário (LEFFINGWEL, 2011):

Como cliente, eu sempre vejo o preco atual da energia refletido no meu portal e nos meus dispositivos de modo que eu sei que os meus custos de uso da energia são precisos e refletem qualquer mudanca de preco.

Os seguintes testes de aceitação da história serão derivados:

1. *Verificar que o preco atual é sempre usado e que os números calculados são exibidos corretamente no portal e em cada dispositivo (ver anexos).*
2. *Verificar que o preco e os números calculados são atualizados corretamente quando o preco muda.*
3. *Verificar que o campo "preco atual" em si é atualizado de acordo com a agenda.*
4. *Verificar as mensagens de informacão/erro quando houver erro na precificacão (ver mensagens de erro aprovadas no anexo)*

Saiba mais

Para se aprofundar nos temas relacionados a teste de *software*, o capítulo 22 do livro *Engenharia de Software: uma abordagem profissional*, de Pressman e Maxim (2016), que apresenta estratégias e teste de *software*, é uma leitura que vale a pena.

Referências

BOEING 737 Max Lion Air crash caused by series of failures. *BBC*, 2019. Disponível em: https://www.bbc.com/news/business-50177788. Acesso em: 29 abr. 2020.

BOURQUE, P.; FAIRLEY, R. *SWEBOK*: guide to the software engineering body of knowledge: version 3. USA: IEEE Computer Society, 2014. Disponível em: https://www.computer.org/education/bodies-of-knowledge/software-engineering. Acesso em: 29 abr. 2020.

CALLEAM CONSULTING. *Denver Airport baggage handling system case study*. [*S. l.*], 2008. Disponível em: http://calleam.com/WTPF/wp-content/uploads/articles/DIABaggage.pdf. Acesso em: 29 abr. 2020.

CMMI INSTITUTE. *CMMI model V2.0*. Pittsburgh: CMMI Institute, 2018.

CMMI PRDUCT TEAM. *CMMI® for Development, Version 1.3*. Hascom AFB: Software Engineering Institute, 2010. Disponível em: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf. Acesso em: 29 abr. 2020.

HINKEL, J. N. *Ariane 5*: um erro numérico (overflow) levou à falha no primeiro lançamento. São José dos Campos, [201-?]. Disponível em: https://www.ime.uerj.br/~demoura/Especializ/Ariane/. Acesso em: 29 abr. 2020.

ISO/IEC/IEEE. *ISO/IEC/IEEE 12207:2017(E)*: systems and software engineering: software life cycle processes. Switzerland: ISO, 2017.

ISO/IEC/IEEE. *ISO/IEC/IEEE 29148:2018*: systems and software engineering: life cycle processes: requirements engineering. Switzerland: ISO, 2018.

LEFFINGWELL, D. *Agile software requirements*: lean requirements practices for teams, programs, and enterprises. Upper Saddle River: Adisson-Wesley, 2011.

PATTON, J. *User story mapping*: discover the whole story, build the right product. Sebastopol: O´Reilly, 2014.

VAZQUEZ, C. E.; SIMÕES, G. S. *Engenharia de requisitos*: software orientado ao negócio. Rio de Janeiro: Brasport, 2016.

WIEGERS, K. E. *More about software requirements*: thorny issues and practical advice. 3. ed. Redmond: Microsoft Press, 2006.

WIEGERS, K. E.; BEATTY, J. *Software requirements*. 3. ed. Redmond: Microsoft Press, 2013.

Leitura recomendada

PRESSMANN, R.; MAXIM, B. *Engenharia de software*: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS