ENGENHARIA DE REQUISITOS

Sheila Reinehr

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Validação de requisitos de *software*

Objetivos de aprendizagem

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

- Reconhecer a importância do envolvimento dos *stakeholders* na validação de requisitos de *software*.
- Projetar a validação de requisitos funcionais e não funcionais de projetos de *software*.
- Tratar os resultados da validação de requisitos de projetos de *software*.

Introdução

É inquestionável que os usuários têm se tornado cada vez mais exigentes com relação à qualidade dos produtos de *software* que utilizam, quer seja no ambiente profissional, quer seja na vida pessoal. A intensa exposição a uma grande variedade de soluções tecnológicas fez com que todos nós nos tornássemos mais críticos e menos tolerantes com as falhas decorrentes da tecnologia.

Temos experiências que são extremamente satisfatórias e outras, nem tanto. Já não temos mais paciência quando o aplicativo do banco nos diz que uma transação está temporariamente indisponível e que devemos tentar mais tarde, ou quando um *site* de compras *on-line* não consegue finalizar a transação com a operadora de cartão de crédito. Queremos uma interface agradável para o nosso gosto pessoal e simples de usar; queremos disponibilidade e tempo de resposta imediato. Queremos tudo isso porque já tivemos experiências desse tipo e desejamos que elas se repitam em todos os *softwares* com os quais temos contato.

No entanto, quem se dedica ao desenvolvimento de *software* sabe que atender a todas essas expectativas não é trivial e, muitas vezes, nem é possível. E os problemas começam nas primeiras fases, quando ainda estamos elicitando os requisitos. É comum surgirem primeiro os requisitos funcionais, que são os mais fáceis de identificar, e, posteriormente, os não funcionais, que muitas vezes ficam implícitos e invisíveis até que o usuário tenha contato com o produto.

Quanto mais cedo envolvermos no processo os diversos *stakeholders*, especialmente os usuários ou os seus representantes, maiores serão as chances de sucesso. Esse envolvimento começa com as atividades de elicitação, mas deve prosseguir com atividades constantes de validação, ou seja, de revisão crítica sobre aquilo que estamos construindo. E isso não pode ocorrer apenas na data final de entrega, pois, nesse momento, o estrago já terá sido feito e a insatisfação do usuário e o retrabalho serão inevitáveis.

Neste capítulo você vai ler sobre a importância do envolvimento dos *stakeholders* no processo de validação, desde os primeiros estágios do desenvolvimento. Vai também estudar como projetar a validação de requisitos funcionais e não funcionais de projetos de *software* e tratar os resultados provenientes da validação.

1 Importância dos *stakeholders* na validação de requisitos de *software*

Para compreendermos o processo de validação de requisitos, é importante relembrarmos as atividades envolvidas na engenharia de requisitos, ilustradas na Figura 1 e descritas no SWEBOK (*Software Engineering Body of Knowledge v3*):

> 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* (BOURQUE; FAIRLEY, 2014, documento *on-line*, tradução nossa).

Considere o semicírculo mais externo da Figura 1 como sendo as atividades da engenharia de requisitos e, o semicírculo interno, o desenvolvimento em si, ou seja, a implementação. Transversal a todo o processo está o gerenciamento de requisitos.

Figura 1. Etapas da engenharia de requisitos.

A disposição em forma de semicírculo é para destacar que não se trata de um processo cascata, no qual uma fase sucede a outra, mas, sim, de um processo iterativo e incremental, permitindo que as interações com o usuário ocorram em todos os momentos. Isso significa que existe uma **atividade** de elicitação de requisitos, e não uma fase de elicitação. O mesmo vale para a análise, a especificação e a validação — lembrando que as atividades de validação referenciadas no SWEBOK também englobam as atividades de verificação.

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, tradução nossa):

- **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 modelo de maturidade CMMI-DEV 2.0 (CMMI INSTITUTE, 2018, p. 525, tradução nossa), 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'".

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

O CMMI Institute (2018) define que validar os requisitos significa assegurar que a solução que será desenvolvida vai se comportar conforme o esperado no ambiente-alvo. Isso é importante para evitar os custos com retrabalho e para aumentar a satisfação do usuário ao receber uma solução que, efetivamente, atende às suas necessidades.

Neste capítulo vamos nos concentrar nas atividades relacionadas à validação de requisitos. Atividades de verificação de requisitos não serão aqui tratadas.

Conceitos de validação de requisitos

O que vem à sua mente quando se fala em validação de requisitos? É provável que você pense que validar seja o usuário testar o produto ou incremento do produto que está sendo liberado para identificar se ele atende às suas necessidades. Embora teste de *software* seja uma das técnicas de validação, ela não é a única. Podemos realizar atividades de validação desde muito cedo no ciclo de vida, mesmo antes de termos algum código implementado.

Embora nas fases iniciais de desenvolvimento os testes não possam ser executados sobre o produto rodando, porque ele ainda não existe, testes conceituais podem identificar problemas nos requisitos logo no início, revelando erros, ambiguidades e omissões (WIEGERS; BEATTY, 2013).

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.

As atividades de validação de requisitos são realizadas para identificar que (WIEGERS; BEATTY, 2013, tradução nossa):

- os requisitos de *software* descrevem, de forma precisa, as capacidades e propriedades do sistema que vão satisfazer às diversas necessidades dos *stakeholders*;
- os requisitos de *software* estão corretamente derivados dos requisitos de negócios, requisitos de sistema, regras de negócio e outras fontes;
- os requisitos estão completos, viáveis e verificáveis;
- todos os requisitos são necessários e o conjunto completo dos requisitos é suficiente para atender aos objetivos de negócios;
- todas as representações dos requisitos são consistentes umas com as outras;
- os requisitos proveem uma base adequada para prosseguir para o projeto (*design*) e a construção.

Quanto mais cedo no ciclo de vida identificarmos os defeitos nos requisitos, menos custoso será para corrigir. Defeitos que são inseridos nas fases iniciais do desenvolvimento vão se propagando em termos de custos adicionais no desenvolvimento de *software*. Se um requisito for compreendido de forma incorreta, ele resultará em desdobramentos também incorretos, como os requisitos detalhados, as especificações de requisitos, a codificação, os casos de teste e, por fim, o produto integrado e entregue ao cliente. A cada etapa que ele permanece não descoberto, novos custos são adicionados.

A Figura 2 ilustra um ciclo genérico de desenvolvimento, no qual os defeitos de requisitos estão representados pelas bolinhas no início do cone. Como você pode observar, ao incluir as atividades de validação ao longo do ciclo de vida (pontos 1, 2, etc.), esses defeitos tendem a diminuir, prevenindo os custos com retrabalho e a insatisfação do usuário. Se essas atividades não tivessem sido realizadas, é possível que a quantidade de defeitos inicial estivesse ainda presente no momento da homologação. Poderia ser ainda que ela tivesse se ampliado, uma vez que um defeito em um requisito pode implicar em defeitos em vários outros requisitos que dele dependem.

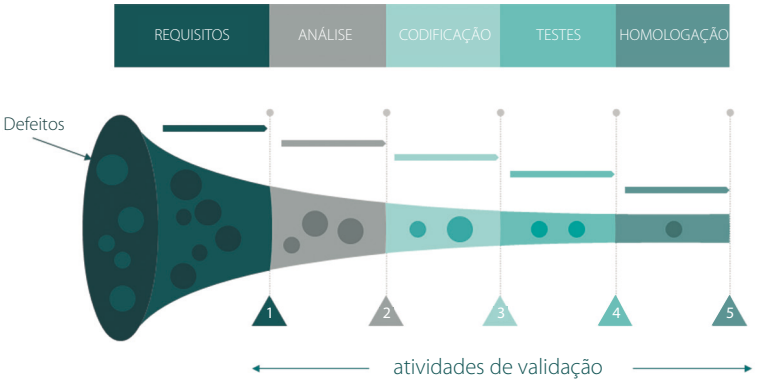

Figura 2. Defeitos inseridos e removidos ao longo do ciclo de desenvolvimento.

Embora a Figura 2 possa sugerir um ciclo cascata, novamente não é essa a nossa intenção. O que queremos representar aqui é que um requisito compreendido de forma errada será levado para as demais atividades e será implementado também de forma errada. Quanto mais tempo esse defeito permanecer escondido, mais custos de manutenção serão incorridos no futuro. Se o único momento em que o usuário vier a avaliar esse requisito for na hora da homologação, então o requisito terá que voltar para as origens, onde será novamente analisado, implementado e testado, gerando custos adicionais provenientes do retrabalho e inicialmente não previstos no projeto.

Quem deverá fazer a validação dos requisitos?

Independentemente da forma como se opte por realizar as atividades de validação, é inquestionável a necessidade de envolvimento do usuário desde o início. Compreender as classes de usuários é o primeiro passo para identificar as fontes de informação que serão a base para a elicitação de requisitos. E por que estamos falando de elicitação de requisitos em um capítulo que foca a validação de requisitos? Porque é na elicitação que tudo começa. Quando não envolvemos, de forma adequada e contínua, todos os *stakeholders* do projeto desde o início, a chance de construirmos o produto correto será muito pequena, ao passo que a chance de termos custos elevados de retrabalho e altos índices de insatisfação será grande.

Aquelas classes de usuários que foram identificadas na etapa de elicitação serão agora utilizadas para a validação dos requisitos. Geralmente, essas classes de usuários são as que terão alguma interface com o negócio que está sendo tratado pelo *software*. Quando o setor for muito afetado por leis ou regulamentações que precisam ser atendidas, uma classe de usuários deverá poder validar os requisitos de ordem legal.

Para apoiar a identificação dos *stakeholders* externos e internos que farão a validação, vamos relembrar no Quadro 1 as questões que Leffingwell (2011) recomenda que sejam feitas para identificá-los.

Quadro 1. Questões de apoio para identificar fontes de informação

***Stakeholders* externos**	***Stakeholders* internos**
Quem utilizará diretamente o sistema? Quem utilizará os resultados daqueles que usam o sistema? Quem será responsável por dar suporte ao sistema? Com quais outros sistemas o nosso sistema irá interagir? Que interfaces devem ser fornecidas? Quem pode fornecer orientações sobre as funcionalidades e os itens de qualidade do sistema (usabilidade, confiabilidade, desempenho e manutenibilidade)?	Quem precisa ser consultado sobre o escopo do projeto? Quem contribuiu para o orçamento e o cronograma? Quem gerencia o relacionamento de negócios entre as equipes e o cliente? Quem irá determinar como e quando o sistema deve ser entregue para os clientes? Quem pode afetar ou apoiar politicamente o projeto? Que parceiros dependem do sistema? Quem se preocupa com o processo que será usado no desenvolvimento do sistema?

Fonte: Adaptado de Leffingwell (2011).

Neste momento podemos ainda revisitar as personas que foram definidas para representar as classes de usuários. Leffingwell (2011) define uma **persona primária** como alguém que interage com o *software* e que necessita de uma interface projetada especificamente para ela. Uma **persona secundária** é um usuário que utiliza o *software* com uma interface projetada para outro tipo de usuário. Persona é um representante hipotético, genérico, de uma classe de usuários (WIEGERS; BEATTY, 2013). No momento da validação, é necessário identificar quem fará a validação como representantes das personas.

Validação de produtos de *software* × validação de serviços

Como temos dito, a validação vai depender do tipo de projeto. Quando estamos falando de projetos de *software* que habilitam negócios inovadores, como aqueles desenvolvidos pelas *startups*, por exemplo, é possível que a percepção sobre qualidade do produto de *software* se confunda com a percepção sobre a qualidade do serviço prestado. Isso é bastante comum em produtos que se confundem com serviços.

Vamos tomar como exemplo os aplicativos para chamar um transporte (como o Uber, o 99 ou similares). Se você pedir um carro e o aplicativo não conseguir localizar um motorista nas proximidades, ele vai indicar um motorista mais distante, que vai demorar mais tempo para chegar. Você se sentirá insatisfeito, mas esse não é um problema proveniente do produto de *software*, mas sim do serviço que é prestado por meio do produto de *software*. Você pode optar por desinstalar aquele aplicativo e utilizar outro, mas o motivo do abandono não terá sido o aplicativo em si, mas o serviço.

Por outro lado, considerando ainda o caso anterior, se você não conseguir localizar onde você adiciona uma parada intermediária no seu trajeto, então esse sim será um problema de usabilidade do produto de *software*, que pode afetar a forma como você percebe a qualidade do serviço como um todo. Esse também pode ser um motivo para você desinstalar o aplicativo e passar a utilizar outro, mas, nesse caso, o motivo terá sido o *software*, e não o serviço.

Uma forma menos evidente de diferença entre o defeito do *software* e do serviço seria, por exemplo, o oferecimento de opções de pagamento diferentes. Se a opção de pagamento no aplicativo de transporte for apenas via cartão de crédito isso pode ser um problema de *software* (não foi implementada a opção do pagamento em dinheiro, embora tivesse sido solicitada pelo demandante do *software*) ou pode ser um problema do serviço (o negócio não prevê que seja feito o pagamento em dinheiro, permitindo apenas receber via cartão de crédito). Conseguiu perceber a diferença?

Validar produtos de *software*, portanto, não pode ser confundido com validar o serviço que ele presta. Essa linha é tênue e nem sempre é fácil identificar claramente essa separação. Por isso os procedimentos de validação devem ser cuidadosamente planejados para que possam ser realizados de maneira eficaz na descoberta antecipada de defeitos no produto.

2 Planejamento da validação de requisitos

Conforme apresentado anteriormente, a norma internacional ISO/IEC/IEEE 12207:2017(E) (ISO/IEC/IEEE, 2017, p. 89, tradução nossa), define a finalidade do processo de validação da seguinte maneira: "[…] fornecer evidência objetiva de que o sistema, quando em uso, cumpre seus objetivos de negócio ou de missão e os requisitos dos *stakeholders*, alcançando seu uso pretendido no seu ambiente operacional pretendido". A mesma norma complementa que "[…] esse processo provê as informações necessárias para que as anomalias identificadas possam ser resolvidas pelos processos técnicos apropriados onde essa anomalia foi criada" (ISO/IEC/IEEE, 2017, p. 89, tradução nossa).

Para sistemas de *software*, a norma estabelece que a finalidade do processo de validação é a seguinte:

- confirmar que os requisitos para um uso específico de um produto de trabalho foram atendidos (normalmente chamado de validação de *software*);
- atingir a confiança (especialmente de um adquirente ou um cliente) que o produto entregue atende às necessidades dos *stakeholders* (normalmente chamado de teste de aceitação de *software*).

Veja que nesse último item aparece a figura do adquirente ou cliente. Essa é uma observação importante, pois nem sempre o cliente que adquire o produto é necessariamente o usuário do produto. Nesse caso o cliente representa os interesses do usuário do produto.

Processo de validação

O processo de validação pode ser compreendido como um conjunto de três atividades, conforme ilustra a Figura 3. A primeira atividade é **preparar para a validação**, ou seja, realizar o planejamento de como será conduzida a validação. em seguida, temos a atividade de **realizar a validação**, que, como o próprio nome sugere, refere-se às atividades de execução da validação e de registro dos seus resultados. por fim, temos a atividade de **gerenciar os resultados da validação**, na qual os resultados da validação são analisados para que planos de ação possam ser executados.

Figura 3. Atividades do processo de validação.
Fonte: Adaptada de ISO/IEC/IEEE (2017).

Preparar a validação envolve tarefas específicas relacionadas à definição das estratégias que serão adotadas para a validação, como, por exemplo:

- que produtos de trabalho (artefatos) serão validados;
- quais são as restrições que podem existir para que a validação seja executada;
- quais itens são prioritários;
- que métodos e técnicas serão empregados na validação, bem como os respectivos critérios de aceitação;
- que sistemas ou ferramentas serão necessários para permitir a validação e como eles serão obtidos.

A preparação da validação é uma tarefa tão importante quanto a sua realização. Ao identificarmos a necessidade de validação de um artefato (documento ou o próprio código), é importante refletir sobre como acontecerá essa validação, quem será envolvido, que materiais serão necessários, qual o escopo da validação, que técnica será utilizada para validar. Para cada tipo de técnica, um tipo diferente de planejamento será necessário.

Quando a validação estiver concluída, é preciso **gerenciar os resultados**. Se o produto estiver de acordo com o esperado, basta formalizar o aceite e dar por encerrado o processo de validação, seguindo com o projeto/versão para as demais atividades. Caso anomalias tenham sido encontradas, elas deverão ser analisadas e as ações necessárias deverão ser identificadas, priorizadas e executadas. Nova rodada do processo de validação pode ser requerida, nesses casos. Discutiremos isso na próxima seção.

Fique atento

Embora estejamos nos referindo sempre ao termo **processo de validação**, vale aqui lembrar que não se trata de uma burocratização das atividades, mas de **conceitos que são importantes** quando realizamos a validação, independentemente do método de desenvolvimento que está sendo utilizado.

Em **ambientes ágeis de desenvolvimento**, onde é esperado que o cliente se faça presente de forma mais contínua, as atividades de validação podem ocorrer simultaneamente às atividades de elicitação de requisitos. Isso normalmente se dá por meio da discussão sobre as histórias de usuário e seus critérios de aceitação, que ocorrem de forma continuada ou em *workshops* de validação.

Como realizar a validação de requisitos

Os requisitos podem ser validados de diversas formas, desde a inspeção cuidadosa dos artefatos de requisitos até o teste do produto em relação aos requisitos inicialmente identificados e às necessidades dos usuários. O modelo de melhoria de processos de desenvolvimento CMMI-DEV v1.3 (CMMI PRODUCT TEAM, 2010) identifica as seguintes formas de validação:

- discussões com os usuários finais (em um contexto de inspeções formais, por exemplo);
- demonstrações de protótipos;
- demonstrações funcionais (por exemplo: sistema, unidades de *hardware*, *software*, documentação de serviços, interfaces de usuário);
- pilotos com materiais de treinamento;
- testes de produtos e componentes de produtos por usuários finais e outros *stakeholders* relevantes;
- entregas incrementais de produtos funcionais e potencialmente aceitáveis;
- análises de produtos e componentes de produtos (simulações, modelagens, análises de usuários).

Saiba mais

Quando as atividades de validação envolverem componentes de *hardware*, considere incluir modelagem para validar a forma, a adequação e a função dos projetos mecânicos; modelagem térmica; análises de manutenibilidade e confiabilidade; demonstrações ao longo do tempo; e simulações de projeto elétrico de componentes eletrônicos e mecânicos (CMMI PRODUCT TEAM, 2010, documento *on-line*).

Técnicas de validação

Existem diversas formas de fazer a validação dos requisitos. Vamos destacar aqui a **inspeção dos artefatos gerados** e o **teste do produto ou componente do produto**. Para cada uma delas, um planejamento será necessário. Se estivermos nos referindo a uma inspeção nos requisitos a ser realizada pelos *stakeholders*, então os elementos envolvidos serão os próprios requisitos e quem realizará a validação. Nesse caso, talvez tenhamos apenas que prever materiais de apoio, que podem incluir um computador para registro dos resultados ou notinhas adesivas tipo Post-it®.

No entanto, se estivermos falando de um **teste no produto ou componente do produto**, então um ambiente próprio precisará ser montado e configurado, o que envolve, possivelmente, *hardware*, *softwares* de apoio, bases de dados, etc. Normalmente, esses ambientes são chamados de ambientes de homologação, pois visam a reproduzir, com a maior fidelidade possível, o ambiente-alvo onde o *software* será executado quando estiver entregue.

A validação utilizando o conceito de revisões pode acontecer no formato de **inspeções formais**, ***walkthroughs* estruturados**, **auditorias** ou outras formas de revisão. Sua aplicação vai depender do grau de formalismo desejado para a validação. Inspeções são consideradas revisões mais formais e têm um formato próprio de funcionamento.

Neste momento, você deve estar se perguntando se isso não seria a mesma coisa que realizar uma verificação, uma vez que as técnicas são as mesmas. Embora as técnicas possam ser as mesmas, existe uma grande diferença: a validação pressupõe que o usuário, ou o seu representante, participe do processo. Lembre-se de que o foco da validação é garantir que estamos construindo o que o usuário deseja, ou seja, o produto certo. Vamos discutir um pouco mais sobre isso.

As revisões com foco na validação podem ser feitas seguindo diferentes abordagens. Elas até podem ser baseadas em ***checklists***, mas isso não será suficiente sob a ótica de quem utilizará o produto. Embora usar um *checklist* seja uma boa forma de apoiar uma verificação, ele não se mostra tão adequado para apoiar a validação.

Exemplo

Vamos tomar como exemplo um item que tipicamente está presente em um *checklist* de verificação de requisitos: "O requisito está especificado de forma completa e que possibilita que o desenvolvedor o implemente?". Esse é um item que não poderá ser avaliado pelo usuário, ele não sabe se o requisito está no nível adequado para o desenvolvedor implementar. Outro exemplo: "O requisito é viável técnica e financeiramente de ser implementado, de acordo com as restrições do projeto?". O usuário não terá elementos para julgar se ele é viável, o máximo que ele poderá contribuir nesse caso é dizer que um requisito é mais importante do que o outro, levando à uma priorização de requisitos em caso de recursos escassos.

Para o usuário, é mais eficiente que a validação utilize **cenários** de execução, com foco na **perspectiva** de um grupo específico de usuários ou com foco em uma **funcionalidade** específica, quando ela for mais complexa.

Caso você tenha tido alguma experiência com ambientes que usam métodos ágeis, já deve estar relacionando a validação com o **mapeamento de histórias de usuário**. E está correto! O mapeamento de uma história de usuário é uma forma de realizar a validação de uma solução, desde que o usuário ou o representante dele faça parte desse mapeamento.

A validação utilizando técnicas de **teste de *software*** envolve o planejamento da exposição do usuário aos requisitos implementados. Inspeções e cenários podem ser considerados como uma espécie de teste conceitual. No entanto, teste, no conceito mais estrito do termo, significa expor o usuário ao produto, ou parte do produto implementado, mesmo que na forma de uma versão preliminar ou de um protótipo.

Saiba mais

Em ambientes ágeis que utilizam a abordagem *lean startup*, existe um conceito muito forte que é a exposição do usuário, o mais cedo possível, a um produto mínimo viável, mais conhecido pela sua sigla **MVP**, que vem do inglês *minimun viable product*. Isso significa que você precisa compreender em profundidade a dor do usuário e apresentar uma pequena solução para que ele possa validar se aquilo vai atendê-lo.

Acompanhe um exemplo no livro de Jeff Patton, *User Story Mapping* (em inglês), que descreve a história de um sistema de acesso à biblioteca de uma faculdade nos Estados Unidos e como eles usaram os conceitos de *lean startup* para desenvolver uma solução mais eficiente. (PATTON, 2014).

Segundo Patton (2014), testar uma solução envolve mais do que buscar os defeitos, uma vez que um *software* sem defeitos não garante a satisfação do usuário. Para Patton (2014, p. 214, tradução nossa), é preciso que você "Coloque as suas soluções em frente às pessoas que irão comprar ou usar o seu produto. Não espere que elas sejam um sucesso da primeira vez. Faça iterações e melhore-as".

Testes em ambientes ágeis

É possível que você já tenha visto alguma vez a matriz que descreve os tipos de testes em ambientes ágeis proposta por Leffingwell (2011), ilustrada na Figura 4. Na realidade, embora o foco seja nos testes em ambientes ágeis, se observarmos mais detidamente, veremos que ela se aplica a qualquer ambiente, mesmo aqueles que não compartilham dos princípios ágeis.

Os quadrantes superiores, que apresentam a **face de negócios**, são os que se dedicam às atividades de validação, pois envolvem os testes de aceitação das funcionalidades e sistema. Isso pode ser observado no quadrante Q2 (Testes funcionais) e no quadrante Q3 (Teste de aceitação de sistema).

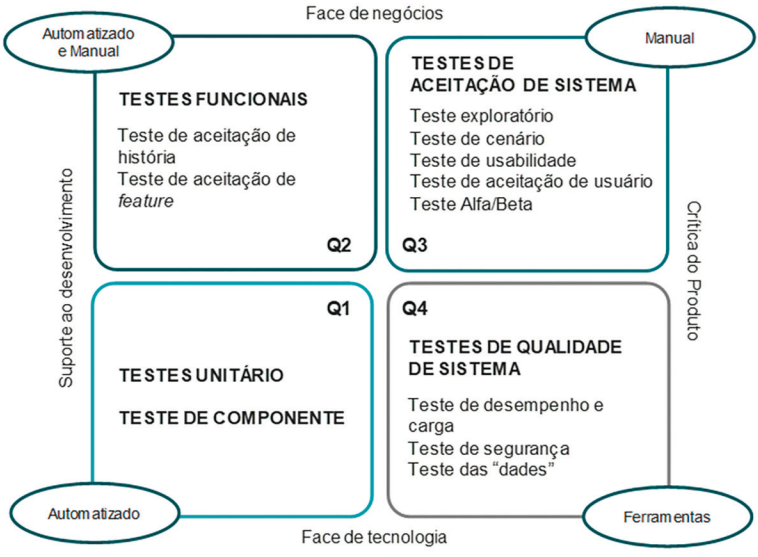

Figura 4. A matriz de testes ágeis.
Fonte: Adaptada de Leffingwell (2011).

O **teste de aceitação de histórias** e o **teste de aceitação de *features***, representados no quadrante Q2 da Figura 4, são testes realizados pelos desenvolvedores com base nos critérios que são escritos como complemento às histórias de usuário.

Vale aqui relembrar que as histórias de usuário são descritas normalmente levando em consideração os 3 Cs: **C**artão, **C**onversa e **C**onfirmação (LEFFINGWELL, 2011). O **cartão** deve conter a declaração principal da história, que deve ter os seguintes elementos: quem + o que + porque, ou seja, quem é o papel que executará a função, o que ele precisa e por que ele precisa. A **conversa** deverá ser o elemento principal no relacionamento entre a equipe e os *stakeholders*, ela é a base da validação contínua dos requisitos. Por fim, a **confirmação**, que se refere aos critérios de aceitação que serão usados para validar as histórias de usuário.

Já o quadrante Q4 apresenta os **testes de aceitação de sistema**, que são tipicamente realizados pelos *stakeholders* e que constituem a validação dos requisitos no produto implementado e integrado. São eles:

- teste exploratório;
- teste de cenário;
- teste de usabilidade;
- teste de aceitação do usuário;
- teste alfa/beta.

Assim como os requisitos funcionais, também os requisitos não funcionais podem ser validados desde sua elicitação. A diferença é que é mais fácil para os *stakeholders* definirem os requisitos funcionais do que os requisitos não funcionais, especialmente em termos quantitativos. Por esse motivo, é importante que os procedimentos de elicitação para esses requisitos sejam baseado em *checklists*, que ajudam a não esquecer de algum item importante para o produto. Uma boa dica nesse caso é olhar para a família de normas ISO/IEC 25.000.

Saiba mais

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

3 Tratamento dos resultados da validação de requisitos

Quando uma rodada de validação de requisitos é encerrada, ela gera resultados que precisam ser analisados de forma conjunta entre os *stakeholders* do projeto e a equipe de desenvolvimento, de modo a gerar ações corretivas. Os resultados podem estar relacionados à identificação destes pontos:

- erros nos requisitos existentes;
- mudança nos requisitos existentes;
- novos requisitos;
- conflitos entre requisitos.

Um **erro em um requisito** ocorre quando ele foi identificado de forma incorreta pelo analista de requisitos ou quando foi escrito de forma incorreta. Isso pode ser decorrente de suposições que o analista de requisitos fez e que não estavam adequadas, ou pode ser que ele simplesmente tenha cometido alguma falha na hora da escrita. Independentemente da causa, o erro deve ser corrigido antes que possa seguir para a implementação. Quanto mais cedo o erro for detectado, menos oneroso ele será para o projeto.

No momento da validação, **mudanças nos requisitos existentes**, ou até mesmo **novos requisitos**, podem surgir, seja em decorrência do desdobramento de requisitos existentes, seja como consequência da melhor compreensão do produto pelos *stakeholders*. Em ambos os casos isso representa esforço adicional de desenvolvimento, assim como custos e recursos adicionais. Isso pode gerar um impacto no cronograma do projeto ou da versão, bem como no orçamento inicialmente estabelecido.

Outro aspecto que pode ser identificado durante a validação de requisitos é o conflito entre os requisitos. Isso pode ser decorrente de visões diferentes entre *stakeholders* ou pode ser proveniente de um único *stakeholder*. Conflitos denotam requisitos que são mutuamente exclusivos e que precisam ser tratados, pois não podem seguir dessa forma para a implementação.

Quanto mais cedo esses problemas forem detectados, mais fácil e mais barato será corrigi-los. Quando o cliente está distante, ou até mesmo ausente do desenvolvimento, a validação pode ser adiada para fases posteriores, quando o produto já estará construído, e então os custos para reparar esses problemas serão maiores.

Um plano de ação idealmente leva em consideração o seguinte:

- impacto da mudança sobre o requisito afetado;
- impacto da mudança sobre outros requisitos afetados relacionados;
- restrições de custo e prazo impostas ao projeto ou à versão/*sprint*;
- riscos da alteração;
- riscos da não alteração.

Como resultado, decisões deverão ser tomadas. Quando se tratar de um erro, ele terá, obrigatoriamente, que ser corrigido antes que prossiga para a implementação. No entanto, quando se tratar de uma melhoria ou de um novo requisito, pode ser que a opção seja pela não implementação ou pelo adiamento da alteração para versões posteriores do produto.

Em ambientes ágeis, é comum que o *product owner* seja o responsável por essa decisão, junto com o time de desenvolvimento. Em ambientes mais tradicionais essa decisão será tomada pelo cliente juntamente com o gerente de projetos, com o apoio do analista de requisitos.

Fique atento

Tão importante quanto identificar os problemas nos requisitos o mais cedo possível é identificar sua **causa raiz**. A identificação das causas vai permitir a adoção de ações preventivas, com vistas a evitar que o mesmo erro venha a se repetir no projeto corrente e em projetos futuros.

Quando a causa raiz não é identificada, é possível que, em algum momento posterior, os mesmos erros sejam repetidos. Ações decorrentes da identificação dessas causas incluem prover treinamento para a equipe de desenvolvimento, realizar melhorias no processo de engenharia de requisitos, adquirir ferramentas de apoio ao processo de engenharia de requisitos ou até mesmo substituir membros da equipe.

Tratamento de conflitos

Quando for observado o conflito entre as visões de dois ou mais *stakeholders*, então esse conflito deverá ser gerenciado, de acordo com Pohl e Rupp (2015), considerando as etapas ilustradas na Figura 5.

Figura 5. Fases da resolução de conflitos.

Conflitos podem surgir em qualquer etapa da engenharia de requisitos. Uma vez realizada a atividade de **identificação do conflito**, vem a etapa de **análise do conflito**. Um conflito pode surgir pelos seguintes motivos (POHL; RUPP, 2015, p. 105–106):

- **Conflito de dados:** quando existe divergência entre os dados fornecidos por *stakeholders* diferentes. Por exemplo, um *stakeholder* pode achar que o tempo de resposta adequado para determinada funcionalidade é de menos de 1 segundo, enquanto outro pode achar que 1 segundo é inviável tecnicamente.
- **Conflito de interesses:** quando existe uma diferença de expectativas entre dois ou mais *stakeholders*. Por exemplo, um acha que os custos do projeto devem ser mantidos no nível mínimo, enquanto outro acha que a qualidade deve ser a prioridade.
- **Conflito de valores:** quando existe uma diferença entre os valores pessoais dos *stakeholders*, que podem ser provenientes de culturas diferentes. Isso pode ocorrer, por exemplo, quando um dos *stakeholders* defende o uso de tecnologias *open source* enquanto outro defende o uso de tecnologias proprietárias.
- **Conflito de relacionamento:** quando existem personalidades fortes e antagônicas. Isso pode ocorrer quando dois *stakeholders*, geralmente de mesma posição hierárquica, conflitam sobre um requisito e um deles deseja impor a sua opinião sobre o outro.
- **Conflito estrutural:** quando existe diferença de perspectivas entre pessoas de níveis hierárquicos diferentes. Pode ser, por exemplo, que um nível hierárquico superior não ache que um subordinado tenha competência para especificar aquele tipo de requisito.

Dificilmente os conflitos são tão claramente classificáveis, pois é comum que um seja consequência, ou esteja atrelado ao outro. Por exemplo, um conflito de relacionamento pode ser decorrente de um conflito de valores.

Após a análise vem a **resolução do conflito**, que constitui uma das principais etapas da validação de requisitos. Se uma resolução deixar uma das partes insatisfeita, então, o comprometimento dessa parte com o projeto pode ser reduzido e até mesmo inviabilizado, ou seja, o *stakeholder* pode se tornar um detrator do projeto e, possivelmente, um obstáculo ao seu sucesso. Independentemente da natureza do conflito, é importante envolver os diversos *stakeholders* do projeto na busca por uma alternativa de resolução.

Existem diversas técnicas que podem ser utilizadas para resolver conflitos, entre elas (POHL; RUPP, 2015, p. 106–108):

- **Acordo:** acontece quando se busca uma solução de consenso entre as partes.
- **Comprometimento:** ocorre quando uma solução alternativa é buscada pelas partes sem conflito. A solução criada é nova e não as mesmas que haviam sido propostas.
- **Votação:** ocorre quando são apresentadas as soluções alternativas e é feita uma votação. Essa não é a melhor solução, pois haverá vencedores e perdedores e, quando isso ocorre, o lado perdedor pode se tornar um obstáculo.
- **Definição de variantes:** ocorre quando se define uma forma de parametrização no sistema, para acomodar visões diferentes de requisitos.
- **Decisão superior (*overruling*):** ocorre quando a decisão é tomada pelo nível hierárquico superior aos conflitantes. Essa também é uma situação que deve ser evitada, pois traz como consequência o sentimento de ter havido um ganhador e um perdedor.
- **Considere todos os fatores (CAF):** lista-se os diversos atributos do requisito que está em conflito e, após análise desses atributos, chega-se a uma decisão sobre o conflito.
- **Mais-menos-interessante:** características positivas e negativas de uma solução são listadas, de modo que elas possam ser comparadas. O termo interessante é usado para denotar quando uma solução parece ser interessante, mas precisa ser melhor investigada.
- **Matriz de decisão:** ocorre quando se lista as alternativas em conflito nas colunas e os critérios de seleção nas linhas. Cada atributo é pontuado para cada uma das alternativas e, no final, apura-se o que teve a maior pontuação.

Por fim, é realizada a **documentação do conflito**, na qual todos os conflitos e decisões são documentados. Isso é importante para poder manter o padrão de decisões de conflitos ao longo do projeto, bem como servir de fonte para consulta sobre as decisões estabelecidas.

Prevenção de problemas em requisitos

Ao identificar as causas dos problemas em requisitos, mudanças na forma de trabalho das pessoas devem ser introduzidas para a prevenção de sua futura repetição. No entanto, de acordo com Wiegers e Beatty (2013, tradução nossa), podem surgir barreiras para que mudanças nos processos de requisitos aconteçam. A maioria delas não é de ordem técnica, mas sim pessoal. Entre elas estão as seguintes:

- falta de reconhecimento dos problemas que as práticas atuais de requisitos causam;
- falta de tempo — todos já estão muito ocupados;
- pressão do mercado ou da gerência para entregar rapidamente;
- falta de comprometimento da gerência para investir no processo de engenharia de requisitos;
- ceticismo quanto ao valor da engenharia de requisitos;
- relutância em seguir um processo de engenharia de requisitos ou de desenvolvimento mais estruturado;
- políticas e cultura corporativa enraizada;
- conflitos entre *stakeholders*;
- membros da equipe inadequadamente treinados e habilidosos;
- papéis e responsabilidades do projeto não claros;
- falta de propriedade e responsabilidade pelas atividades de requisitos.

Saiba mais

Wiegers e Beatty (2013, p. 561–574) apresentam um conjunto de sintomas que podem ser identificados no desenvolvimento e que são provenientes dos requisitos. Para cada sintoma, a causa provável é apresentada, bem como uma possível solução, de acordo com a categoria do problema: processo, produto, planejamento, comunicação, elicitação, análise, especificação, validação, gestão e gerenciamento de mudanças.

Referências

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 maio 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 maio 2020.

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

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.

POHL, K.; RUPP, C. *Requirements engineering fundamentals:* a study guide for the certified professional for requirements engineering exam, foundation level, IREB Compliant. 2. ed. Santa Barbara: Rock Nook, 2015.

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