ENGENHARIA DE REQUISITOS

Sheila Reinehr

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Fundamentos da engenharia de requisitos

Objetivos de aprendizagem

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

- Discutir os problemas advindos dos requisitos em projetos de *software*.
- Reconhecer a importância da engenharia de requisitos no ciclo de vida de um *software*.
- Identificar as etapas da engenharia de requisitos.

Introdução

É inquestionável a presença da tecnologia em nosso dia a dia. Produtos de *software* fazem parte de praticamente todas as atividades humanas, sejam elas pessoais, sejam profissionais. Quando um *software* não funciona de forma adequada, sentimos o impacto por meio de transtornos, como a falta de sincronização de semáforos nas grandes cidades, o atraso e o cancelamento de voos, a impossibilidade de fazer uma compra *on-line*, a indisponibilidade de serviços bancários, e mais uma infinidade de contratempos. Falhas em *software* também podem tirar vidas.

Uma das principais fontes de problemas em produtos de *software* está relacionada aos requisitos. Requisitos mal compreendidos, mal especificados e mal gerenciados podem comprometer o desempenho de produtos de *software* e representam uma das maiores causas de fracasso em projetos de tecnologia da informação.

Neste capítulo, você vai ler sobre a importância do *software* na sociedade moderna e os impactos provenientes de problemas relacionados a requisitos de *software*. Verá também a importância da engenharia de requisitos no ciclo de desenvolvimento de *software* e sua influência nas atividades de desenvolvimento. Por fim, você vai saber como a engenharia de requisitos pode ajudar a eliminar, ou pelo menos minimizar, os problemas relacionados a requisitos.

1 Problemas advindos dos requisitos de *software*

Produtos de *software* estão presentes em todos os momentos de nossas vidas. Diariamente, mesmo sem nos darmos conta, entramos em contato com diferentes tipos de *software*. Acordamos despertados por uma funcionalidade de nossos *smartphones* e olhamos a previsão do tempo disponibilizada por outro aplicativo. Saímos para o exercício matinal acompanhados do *smartwatch*, que registra todo o nosso desempenho, incluindo o tempo, o percurso e até mesmo os batimentos cardíacos. Retornamos e aquecemos o café da manhã no micro-ondas, cuja programação automática é feita por um *software* embarcado no eletrodoméstico. Em seguida pedimos o transporte, utilizando mais um aplicativo no celular.

Ao chegar no trabalho, um mundo de diferentes sistemas de *software* nos apoia no dia a dia da empresa (*email*, mensagens instantâneas, programas de gestão, de controle de produção, de *marketing*, de vendas e por aí vai). Ao final do dia, cansados, voltamos do trabalho ouvindo uma música para relaxar usando o *streamer* de músicas e aproveitamos para, no caminho, pedir uma refeição no sistema de entrega de comida e acionar o ar condicionado de casa, para que esteja na temperatura ideal quando chegarmos. Ainda dá para marcar aquele encontro com os amigos para o final de semana, usando o aplicativo de troca de mensagens. Checamos nosso saldo no banco e verificamos se o débito automático da conta de luz foi realizado. Após o jantar, que pagamos com a máquina de cartão de crédito por aproximação do celular, encerramos o dia assistindo nossa série preferida no programa de *streaming* de vídeo e damos uma espiadinha nas redes sociais. O despertador já está programado para o próximo dia.

Praticamente nada disso seria possível sem o uso da tecnologia e dos sistemas de *software* que viabilizam esses serviços. Atualmente, todos os segmentos da atividade humana são permeados por *software* — e serão cada vez mais. Imagine, por um momento, se todo *software* do planeta parasse de funcionar. Quais seriam os impactos? Que atividades você realiza hoje que não poderia realizar? Que serviços essenciais deixariam de ser prestados? Algumas dessas facilidades estão tão enraizadas em nossas vidas que parece que elas sempre estiveram lá.

Com tantas novas tecnologias a cada dia, construir *software* se tornou uma atividade cada vez mais complexa. Se, por um lado, não temos mais as restrições de memória e de poder de processamento que tínhamos na década de 1960, a variedade e a complexidade dos elementos envolvidos atualmente

trazem novos desafios para o desenvolvimento de *software*. Além disso, os ambientes de negócios se tornaram mais sofisticados e complexos. O resultado é que desenvolver projetos de *software* não é uma atividade trivial, e os desafios começam na parte inicial do ciclo de vida: os **requisitos**.

Primeiramente, vamos definir requisitos. O *Glossário padrão de terminologia de engenharia de software* do *Institute of Electrical and Electronic Engineers* (IEEE, 1990) define requisito como:

1. Uma condição ou capacidade necessária para um usuário resolver um problema ou alcançar um objetivo.
2. Uma condição ou capacidade que deve ser atendida ou tida por um sistema ou componente do sistema para satisfazer a um contrato, padrão, especificação ou outro documento formalmente imposto.
3. Uma representação documentada de uma condição ou capacidade conforme estabelecido em 1 e 2.

Independentemente de a empresa ter processos mais burocráticos e formais de desenvolvimento ou utilizar métodos ágeis, requisitos mal compreendidos e mal gerenciados são causa frequente de inúmeros prejuízos e insatisfações. Diversos são os exemplos de falhas de *software* que podem ser ligadas a problemas relacionados a requisitos e levar a milhões em prejuízos. Vamos dar uma olhadinha em algumas delas?

- Na década de 1990, o Aeroporto de Denver, no Colorado, havia sido projetado para ser o maior *hub* de aviação dos Estados Unidos. Um dos pontos mais críticos estava relacionado aos requisitos que o sistema de gerenciamento de bagagens deveria atender para que o tempo em solo das aeronaves pudesse ser limitado a apenas 30 minutos. A complexidade técnica para implementar esse requisito foi subestimada, uma vez que o algoritmo de roteamento das bagagens não era trivial. Isso levou a um atraso de 16 meses na inauguração e a um prejuízo de 1,1 milhão de dólares ao dia para a cidade de Denver (CALLEAM CONSULTING, 2008).
- Em 1996, o foguete europeu Ariane 5 explodiu em pleno ar apenas 66 segundos após ter sido lançado. A causa principal foi o requisito de reutilizar o *software* de seu antecessor, o Ariane 4, que tinha alguns campos internos com tamanhos diferentes do Ariane 5. O requisito de compatibilidade entre os modelos não foi corretamente analisado e o prejuízo estimado foi de 370 milhões de dólares (HINKEL, [201-?]).

- Em 1999, a sonda espacial climática da NASA que iria para Marte errou o ponto de inserção orbital no planeta, realizando a manobra com um erro de 100 km, o que ocasionou a destruição do equipamento e gerou um prejuízo estimado de 125 milhões de dólares. A causa foi um problema no sistema de conversão de medidas — enquanto uma equipe estava usando o sistema métrico decimal, a outra estava usando o sistema imperial (ISBELL; SAVAGE, 1999).
- Em 2019, a Boeing foi obrigada a suspender toda a frota de 387 aviões do modelo Boeing 737 Max, que voavam para 59 companhias aéreas em todo o mundo. O fato ocorreu depois que dois acidentes aéreos tiraram a vida de 346 pessoas em menos de 5 meses. A causa foi atribuída a problemas encontrados no novo *software* de controle de voo que havia sido implantado (BOEING…, 2019).

Esses são apenas alguns dos inúmeros exemplos ocorridos, cuja consequência pode ser a perda de vidas ou o prejuízo financeiro, ou ambos. No dia a dia das empresas de *software*, a consequência mais comum dos problemas em requisitos é o aumento do retrabalho, ou seja, a necessidade de refazer coisas que já estavam prontas, levando a custos não previstos inicialmente. Quanto mais longe do início do projeto ou da *sprint* se percebe o problema, mais caro é para consertar. Quanto mais atividade já tiver sido realizada sobre o requisito (especificação, implementação, testes, entrega), mais difícil e trabalhoso é consertar e maiores são os impactos negativos para o projeto ou para a *sprint*.

Saiba mais

Sprint é o termo que denomina um ciclo de desenvolvimento em uma empresa que utiliza o método ágil Scrum. Uma *sprint* tem uma duração de duas a quatro semanas e visa à entrega de um item possivelmente liberável e que agrega valor ao cliente (SCHWABER; SUTHERLAND, 2018).

Os problemas estão relacionados ao fato de que um requisito pode:

- estar incompleto;
- estar em um nível de detalhe insuficiente para as etapas seguintes do ciclo de desenvolvimento;
- conter ambiguidades ou imprecisões que levem os membros da equipe a interpretá-lo de forma diferente do que o esperado, levando a erros nas fases seguintes do ciclo de desenvolvimento;
- ser incompatível com outro requisito;
- ser tecnicamente inviável;
- ser difícil de testar e validar;
- ter priorização conflitante sob a ótica dos diversos *stakeholders*.

E por que esses problemas acontecem com os requisitos? Existem diversos motivos pelos quais esses problemas podem acontecer, e isso varia de acordo com o tipo de projeto, com o contexto de negócio, com as características da equipe e das tecnologias envolvidas. Um projeto simples, cujo contexto de negócios é dominado pela equipe técnica, vai ter menos riscos associados a requisitos do que um projeto que envolva uma equipe menos experiente ou um negócio muito inovador e sem histórico anterior.

Existem projetos que têm um cliente bem definido e usuários com perfis conhecidos. Nesse contexto, os requisitos podem ser obtidos pelo contato direto com o cliente ou com os próprios usuários. Os requisitos, nesse caso, podem falhar por problemas de comunicação entre as partes, nos quais cada uma tem uma perspectiva diferente e até mesmo um vocabulário diferente. É comum ouvir que os usuários não sabem o que querem, mas, na verdade, por mais que eles saibam, sua perspectiva pode mudar pelo simples fato de entrarem em contato com a equipe, por experimentar a primeira versão de um protótipo ou mesmo pelo fato de que suas necessidades mudam ao longo do tempo. Os efeitos da mudança vão depender de diversos fatores, mas o seu tratamento vai depender fortemente do grau de confiança entre as partes. Quanto maior o grau de confiança, mas fácil é de se gerenciar as mudanças nos requisitos e suas consequências para o projeto, para o produto e para o relacionamento entre as partes.

Há projetos que são inovadores e visam a criar um produto de *software* que não existe ainda, como é o caso das *startups* de tecnologia. Nesse cenário, de onde podem vir os requisitos? Eles podem ser resultantes da própria equipe que está criando o produto, com base no que elas enxergam que o mercado

precisa. Por esse motivo, para esses casos, um MVP (*minimum viable product* ou produto mínimo viável) é criado para analisar se a ideia pode ou não ir adiante. Quanto mais cedo surge falha, mais cedo a ideia é pivotada e um novo ciclo de desenvolvimento é iniciado, com novos requisitos.

Fique atento

Quanto maior a capacidade da equipe de perceber os requisitos que o mercado deseja para o produto, maiores são as chances de sucesso da *startup*.

Há casos ainda em que o projeto visa a uma atualização tecnológica de um produto, na qual a própria versão anterior do produto pode ser a fonte de requisitos. Nesses casos, a qualidade dos requisitos obtidos depende fortemente da qualidade do sistema anterior e da habilidade da equipe em extrair esses requisitos.

Algumas das causas mais frequentes dos problemas relacionados a requisitos, de acordo com Wiegers e Beatty (2013), são:

- os objetivos de negócio, a visão e o escopo do projeto nunca foram definidos claramente;
- os clientes estavam muito ocupados para dedicar mais tempo trabalhando com os analistas ou os desenvolvedores nos requisitos;
- o time não pode interagir diretamente com usuários representativos para entender suas necessidades;
- os clientes argumentaram que todos os requisitos eram críticos, por isso eles não estabeleceram prioridades;
- os desenvolvedores encontraram ambiguidades e informações faltantes quando foram codificar, então, eles precisaram adivinhar certas informações;
- as comunicações entre o cliente e os desenvolvedores focaram na aparência da interface do usuário e não no que os usuários necessitavam alcançar com o *software*;
- os clientes nunca aprovaram os requisitos;
- os usuários aprovaram os requisitos para um *release* ou iteração e então mudaram eles continuamente;

- o escopo do projeto aumentou à medida que as mudanças de requisitos foram aceitas, mas o cronograma se desviou porque nenhum recurso adicional foi fornecido e nenhuma funcionalidade foi removida;
- as mudanças solicitadas nos requisitos foram perdidas; ninguém sabia o *status* de solicitações de mudanças específicas;
- os clientes requisitaram certa funcionalidade e os desenvolvedores a desenvolveram, mas ninguém nunca utilizou;
- ao final do projeto, a especificação foi satisfeita, mas o usuário e os objetivos de negócio não.

Relembrando o que Frederick Brooks já nos alertava em 1987 (BROOKS, 1987, p. 13):

> A parte mais difícil ao se construir um sistema de *software* é decidir precisamente o que construir. Nenhuma outra parte do trabalho conceitual é tão difícil quanto estabelecer os requisitos técnicos detalhados, incluindo a interface com as pessoas, máquinas e outros sistemas de *software*. Nenhuma outra parte afeta tanto o sistema resultante se for feita de forma errada. Nenhuma outra parte é tão difícil de consertar depois.

2 Ciclo de vida de *software* e engenharia de requisitos

O ciclo de vida de um produto de *software* pode ser compreendido como um conjunto de etapas pelas quais o produto passa ao longo de sua existência, conforme ilustra, de forma simplificada, a Figura 1.

Figura 1. Ciclo de vida de um produto de *software*.

Concepção

Inicialmente, a necessidade de um novo produto de *software* é identificada e a viabilidade de sua construção é analisada. Como produtos de *software* podem ter diversas naturezas, a fonte dessa necessidade pode variar enormemente, especialmente hoje, quando a tecnologia permeia todas as nossas atividades, como vimos no início deste capítulo.

Na fase de **concepção**, os primeiros requisitos aparecem na forma de objetivos de negócio de alto nível, do tipo: "Precisamos de um *software* para apoiar o gerenciamento do fluxo de solicitações". Às vezes esse objetivo vem atrelado a uma meta: "(…) de modo a reduzir o tempo de prestação do serviço a 6 horas". Ainda assim, não sabemos exatamente o que precisa ser construído, até que sejam detalhados os requisitos que representam as funcionalidades do produto, conhecidos como **requisitos funcionais de produto**. Então evoluímos para algo do tipo: "O sistema deve ter uma função que emita alertas toda vez que uma solicitação estiver parada por mais de 2 horas".

Nesse momento é também comum que surjam os requisitos de projeto e de processo. Os **requisitos de projeto** podem impor restrições ao projeto em si, e podem ser relacionados com os prazos, os recursos e o orçamento. Já os **requisitos de processo** podem imprimir restrições relacionadas à forma como o projeto será desenvolvido, como, por exemplo, um método específico de desenvolvimento.

De posse dessas informações iniciais, é possível avaliar a viabilidade de se dar início ou não ao Desenvolvimento. Um projeto pode ser cancelado antes mesmo de ter sido iniciado. Isso pode ocorrer por diversos motivos, entre eles o fato de o escopo não estar suficientemente claro para que se possa comprometer recursos para a sua execução.

Desenvolvimento

A fase de **desenvolvimento** compreende todas as atividades necessárias para a produção do *software*. Isso inclui aquelas relacionadas a requisitos, análise e *design*, implementação, testes, homologação e implantação, além de todas as atividades de suporte ao desenvolvimento e à gestão do desenvolvimento, conforme ilustra a Figura 2.

Figura 2. Etapas do desenvolvimento de *software*.

Fique atento

Embora o termo "etapas do desenvolvimento" esteja sendo utilizado, aqui ele não se refere a um modelo de ciclo de vida sequencial ou cascata. Essas etapas são mais parecidas com disciplinas, que podem se intercalar e se repetir à medida que o projeto avança, de forma iterativa e incremental. A Figura 2 ilustra essas etapas.

Quando um projeto de desenvolvimento de *software* tem início, é comum que um escopo esteja estabelecido, mesmo que esse escopo ainda esteja em um nível alto de granularidade. A partir daí, as atividades relacionadas a requisitos, que tiveram início na etapa de **concepção**, são intensificadas. Os detalhes referentes à engenharia de requisitos serão explorados na próxima seção.

Dependendo do tipo e criticidade do projeto, uma vez estabelecidos os requisitos, têm início as atividades de *design*. Essa etapa consiste em projetar a solução que implementará os requisitos. Esses modelos ajudam a aprofundar a compreensão do contexto e a delinear a solução que será implementada. Mesmo empresas que utilizam métodos ágeis costumam ter uma *sprint* zero, que se dedica a conceber a arquitetura da aplicação. Nesse momento é estabelecida a arquitetura do sistema, os componentes e como eles se relacionam.

Durante a etapa de análise e *design* podem emergir novos requisitos, derivados ou não daqueles que foram inicialmente elicitados. Isso pode ocorrer porque, ao modelar a solução, o analista de sistemas pode:

- identificar requisitos técnicos adicionais, advindos da arquitetura escolhida;
- perceber que determinados requisitos não têm viabilidade técnica de execução;
- identificar requisitos conflitantes;
- identificar requisitos incompletos ou imaturos para a implementação

Uma vez definidos os componentes no nível de detalhe compatível com as necessidades do projeto, iniciam as atividades de Implementação. Note que aqui pode haver um paralelismo de atividades, pois os componentes podem ir sendo especificados ao mesmo tempo em que outros já estão sendo implementados. Nessa etapa o desenvolvedor ainda pode detectar problemas nos requisitos, devido à inviabilidade técnica de implementar.

Ao concluir a codificação, o desenvolvedor realiza os testes individuais nas unidades pelas quais foi responsável pela implementação. Nesse momento são realizadas as atividades de verificação, usando os casos de teste manuais ou automatizados.

Há empresas que utilizam a abordagem DevOps, integrando o ciclo de desenvolvimento ao ciclo de operação em produção. Isso é feito utilizando os conceitos de integração contínua (CI, *continuous integration*) e entrega contínua (CD, *continuous delivery*) e implantação contínua (CD, *continuous deployment*). Entende-se por CI o fato de haver uma integração automática à medida que o desenvolvedor sobe o seu código para o repositório e este se integra a outras partes do código que foram desenvolvidas por outros desenvolvedores. Já a entrega contínua (CD) se refere a entregar uma solução integrada para o pessoal de implantação poder realizar a passagem para o ambiente de produção, e a implantação contínua (utiliza-se a sigla CD também) se refere à passagem automática para o ambiente de produção.

Saiba mais

Quer conhecer mais sobre DevOps? Consulte o livro *Jornada DevOps: unindo cultura ágil, Lean e tecnologia para entrega de software de qualidade*, organizado por Muniz *et al.* (2019).

Manutenção

A maior parte do tempo de vida de um produto de *software* se passa na etapa de Manutenção. A partir do momento que a primeira funcionalidade entra em produção e o produto começa a ser utilizado, já começam as requisições de mudanças, sejam elas por *bugs* encontrados pelos usuários, sejam por demandas de ordem legal/fiscal ou por melhorias e novas funcionalidades.

Na etapa de Manutenção, ocorre a maior parte das atividades de gerenciamento dos requisitos. Isso significa que a rastreabilidade é a ferramenta mais importante para a análise do impacto das solicitações de mudanças.

Descontinuação

Quando o produto de *software* não atende mais aos seus propósitos ele é descontinuado, ou seja, ele é retirado do ambiente de produção. Em alguns casos, basta solicitar a desinstalação do produto, como é o caso dos aplicativos que temos em nosso celular. Mas, na maioria das vezes, a descontinuação não é uma atividade tão simples.

Para descontinuar, por exemplo, um *software* de gestão do tipo ERP (*enterprise resource planning*, ou sistema de gestão empresarial), são necessários muitos anos, pois a transição para outro produto/fornecedor não é tão simples. Muitas vezes é necessário construir outro sistema para que se possa fazer a migração de dados e processos para o novo produto. Nesses casos, é importante compreender os requisitos da desativação.

Exemplo

O negócio no qual o *software* está inserido pode ter requisitos específicos relacionados a questões de ordem legal e/ou fiscal, como o arquivamento de dados e documentos por um longo período. Por exemplo, contratos de trabalho precisam ser guardados por tempo indeterminado, enquanto outros documentos requerem prazos de 5, 10, 20 e até 30 anos. Além disso, é preciso planejar a forma como esses dados serão recuperados em caso de necessidade. Esses são requisitos que podem estar presentes no projeto de desativação de um produto de *software*.

3 Etapas da engenharia de requisitos

Agora que já entendemos quais problemas podem vir dos requisitos e compreendemos a importância da engenharia de requisitos para lidar com essas questões, vamos conhecer quais são as atividades a serem realizadas nessa etapa do desenvolvimento de *software*.

Falar em etapas da engenharia de requisitos pode sugerir que estamos nos referindo a um modelo de desenvolvimento cascata, cheio de burocracias e distante da realidade atual das empresas. Na verdade, no contexto deste capítulo, estamos considerando que as etapas da engenharia de requisitos são atividades que devem ser realizadas no desenvolvimento de produtos de *software*, independentemente do modelo de ciclo de vida que está sendo utilizado. O momento de realização dessas atividades e os artefatos envolvidos poderão variar, mas a essência é a mesma — o que varia é a forma. O objetivo é um só: construir produtos de *software* melhores e mais confiáveis!

Como você pode observar na Figura 3, a engenharia de requisitos compreende duas grandes etapas: o desenvolvimento e o gerenciamento. No desenvolvimento de requisitos estão englobadas as atividades de elicitação, análise, especificação e validação, as quais veremos em mais detalhes a seguir.

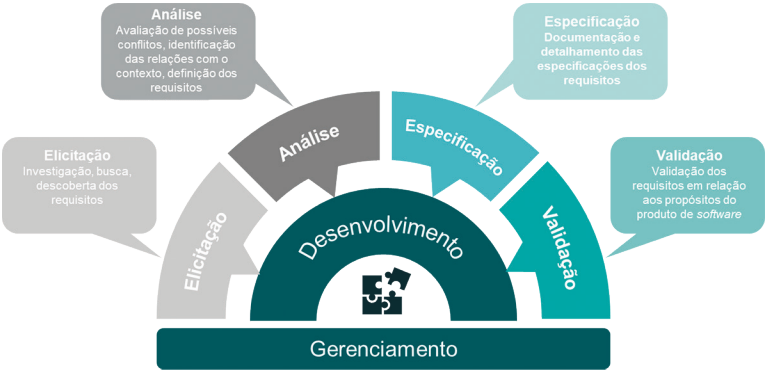

Figura 3. Etapas da engenharia de requisitos.

Elicitação de requisitos

A elicitação se refere à etapa de investigação dos requisitos. É o momento em que a equipe técnica precisa compreender o que deve ser feito. Antigamente, o termo utilizado era "levantamento de dados", no entanto, essa expressão está em desuso e, em seu lugar, usamos a expressão "elicitação de requisitos" para denotar algo mais forte, que remete a uma busca ativa pelos requisitos.

Isso significa que o analista de requisitos ou papel similar (analista de negócios, analista de sistemas, *product owner* ou equivalente) deve adotar uma postura proativa para a descoberta e o entendimento dos requisitos. Isso implica em utilizar diversas formas de elicitação, de acordo com o contexto em que o sistema está sendo desenvolvido. Outros termos associados a esta etapa podem ser: captura de requisitos, descoberta de requisitos ou aquisição de requisitos (BOURQUE; FAIRLEY, 2014).

Quando falamos da Elicitação de requisitos não estamos enfatizando que é necessário ter todos os requisitos extensivamente detalhados no início do projeto, ou que esta etapa ocorra uma única vez no início do projeto. Em projetos ágeis, os requisitos podem emergir em diversos momentos, compondo o *product backlog*. No entanto, é importante ter em mente que alguns requisitos, quando descobertos muito tardiamente no ciclo de vida de um projeto, podem implicar em grande quantidade de retrabalho e custos adicionais. Esse é o caso, por exemplo, dos requisitos que impactam fortemente sobre a arquitetura do produto — esses são mais difíceis de serem alterados posteriormente.

Fique atento

Product backlog (ou "*backlog* do produto") é o termo utilizado para denominar um conjunto de itens conhecidos que devem ser desenvolvidos para determinado produto. Quando itens do *product backlog* são priorizados para serem desenvolvidos em uma *sprint*, então, esse subconjunto é chamado de *sprint backlog*. O termo é comum em empresas que utilizam o método ágil Scrum (SCHWABER; SUTHERLAND, 2018).

O processo de elicitação exige uma intensa comunicação com os *stakeholders* do projeto. É no processo de comunicação que o analista de requisitos identifica o que realmente é imprescindível para o desenvolvimento do produto e não perde tempo detalhando o que não é importante ou que não precisa de detalhamento naquele momento. A comunicação ineficaz pode levar a requisitos incompletos e incorretos.

Há casos em que a comunicação com os *stakeholders* é direta e é possível aplicar técnicas como entrevistas e reuniões, pois existem fontes de informação humanas que compreendem o contexto de negócio e podem prover as informações necessárias para a definição dos requisitos. Delinear histórias do usuário ou mapear a jornada do usuário pode ajudar. Ambos se baseiam na narrativa de uma história de uso do sistema por um usuário.

No caso dos produtos inovadores, que estão sendo criados sem uma experiência anterior, ou que tenham um propósito disruptivo, é necessário que sejam utilizadas técnicas com o propósito de despertar a criatividade. Isto pode ser feito utilizando sessões de cocriação, como o *brainstorming* ou o *design thinking*.

Há ainda os momentos em que os requisitos são obtidos a partir de documentações como editais de licitação (no caso de contratação advinda do setor público), de normas e leis (no caso de sistemas que precisam atender a legislações específicas e órgãos regulamentadores), de documentação de outros sistemas anteriores e, até mesmo, no pior caso, do próprio código-fonte anterior.

O importante nas atividades de elicitação de requisitos é que o analista de requisitos seja capaz de compreender o contexto em que o projeto está inserido e que possa planejar de que forma irá realizar sua atividade de elicitar os requisitos. Pode ser empregando uma única técnica, pode ser empregando uma combinação de técnicas.

Fique atento

O papel do analista de requisitos é mergulhar no ambiente de negócio. O usuário não precisa de um *software*, ele precisa do serviço que o *software* presta, precisa de algo que resolva o seu problema de negócio. O mais importante é que o analista de requisitos tenha feito todos os esforços para engajar os *stakeholders* e fazer emergir os requisitos que farão do projeto um sucesso.

Análise de requisitos

A análise de requisitos é a etapa na qual nos concentramos em aprofundar o entendimento acerca dos requisitos, buscando possíveis conflitos que podem ser advindos das diferentes visões que os *stakeholders* possam ter sobre o requisito e sua prioridade no projeto de desenvolvimento. Entretanto, os problemas podem vir também de requisitos conflitantes entre si, especialmente no que se refere a **requisitos de qualidade**, também chamados de **requisitos não funcionais**. A complexidade de um requisito funcional, por exemplo, pode ser incompatível com a necessidade de desempenho exigida.

Faz parte da análise a decomposição de requisitos de alto nível em níveis apropriados de detalhe. Muitas vezes o requisito está expresso como um objetivo de negócio de alto nível e isso não é suficiente para as próximas etapas do desenvolvimento de *software*, sendo necessário decompô-lo em níveis mais detalhados.

Outro aspecto importante da análise dos requisitos diz respeito à definição dos atributos a ele associados, como o tipo do requisito, a volatilidade, o impacto sobre a arquitetura e o risco. Se um requisito, por exemplo, é volátil, ou seja, está mais sujeito a mudanças do que outros, pode ser que se decida por implementá-lo posteriormente, quando o seu entendimento esteja mais estabilizado. Da mesma forma, requisitos que tenham um grande impacto sobre a arquitetura da aplicação devem ser tratados com um cuidado redobrado, pois podem implicar em consequências para todos os demais requisitos, incluindo os requisitos de qualidade, como o desempenho da aplicação.

É importante lembrar que a extensão da análise de requisitos vai variar de acordo com a criticidade do *software* e qual o seu papel dentro do contexto no qual está inserido. Sistemas complexos, envolvendo, por exemplo *hardware* e *software*, como sistemas de *software* para a direção autônoma de veículos, precisam de uma etapa de análise de requisitos mais aprofundada, que pode até mesmo se confundir com o *design* e com as especificações arquiteturais do produto (o que envolve o sistema como um todo). Nos exemplos que vimos no início deste capítulo, pudemos observar como alguns requisitos técnicos podem levar sondas espaciais a se desintegrarem em pleno ar e aeronaves a serem retiradas de circulação após desastres aéreos provenientes do *software*. Em ambos os casos, com extensivos prejuízos materiais, mas, principalmente, com a perda de vidas humanas.

Especificação de requisitos

A especificação de requisitos é a etapa dedicada a representar os requisitos de uma forma que eles possam perdurar ao longo do tempo e possam ser verificados e validados posteriormente. Isso pode implicar em formatos diferentes de especificação que envolvem textos, diagramas e tabelas. Aqui cabe uma pequena discussão acerca do que é exatamente representar os requisitos e de que forma deveríamos realizar essa atividade.

Os métodos tradicionais de desenvolvimento, especialmente o cascata, com ciclos extensivos e exaustivos de especificação antes da codificação, já se mostraram ineficientes no passado e foram, gradualmente, substituídos. Há alguns anos que os métodos ágeis estão sendo mais amplamente adotados pela indústria, e muitos entendem que usar um método ágil significa não realizar qualquer tipo de "documentação". A palavra aqui vem entre aspas para denotar que o termo é de uso pejorativo no ambiente ágil, estando relacionado a uma ação que agrega pouco ou nenhum valor ao produto final. Agilistas mais radicais defendem que a única documentação fiel de um produto é o seu código. São dois extremos que, em geral, se antagonizam.

Vamos refletir por um instante: imagine que você estivesse no hospital e sua vida dependesse da precisão de um equipamento cirúrgico, operado, em grande parte, por um *software*. Será que você iria preferir que os projetistas tivessem documentado adequadamente os requisitos técnicos, para garantir que os envolvidos nas fases seguintes desenvolvessem o produto de forma precisa? Ou você estaria confortável em ter apenas uma história de usuário de alto nível como documentação desses requisitos?

A questão aqui é quanto de especificação é necessário antes de se dar início à implementação? A resposta é depende. Depende da complexidade do produto que está sendo desenvolvido, da criticidade de cada requisito, dos riscos de se ter um erro advindo de uma má compreensão, da experiência da equipe envolvida e até mesmo da fase em que o produto e a empresa se encontram.

É comum que empresas disruptivas de tecnologia, como as *startups*, iniciem suas atividades a partir de versões iniciais e pouco elaboradas do produto, o chamado MVP (em inglês, *minimum viable product*, isto é, produto mínimo viável). Nessa fase de sua existência, a empresa não se preocupa em especificar detalhadamente, pois seu foco é testar a viabilidade do negócio. Nesse momento a equipe é pequena, composta de duas a três pessoas, e a comunicação é direta e simples. No entanto, à medida que essa *startup* cresce e se torna uma grande empresa, não é mais possível continuar a desenvolver sem a preocupação com a especificação, pois a equipe também aumenta e a comunicação entre

os membros se torna mais complexa. Um erro no ambiente de produção pode afetar milhares de pessoas.

Fique atento

É preciso especificar os requisitos na medida certa para a situação. Nunca teremos os requisitos de forma perfeita. Precisamos ter os requisitos na forma necessária e suficiente para os objetivos do projeto e para prosseguir com as demais etapas do desenvolvimento, sejam elas denominadas de fases, sejam de etapas, iterações, estágios ou *sprints*.

Validação de requisitos

A especificação dos requisitos precisa ser validada entre os *stakeholders* e a equipe de desenvolvimento para garantir que existe uma compreensão correta e comum sobre os requisitos e que a equipe de desenvolvimento possui as condições de implementar um produto que irá satisfazer as necessidades do negócio (WIEGERS; BEATTY, 2013).

A validação pode ser realizada por meio de uma revisão sobre as especificações, que é feita por revisores designados para tal, com o apoio de um *checklist*, por exemplo. Há casos em que se conduzem *workshops* de validação, nos quais os diversos tipos de *stakeholders* são envolvidos.

Podem ainda ser usados protótipos funcionais que visam a confirmar os requisitos e até mesmo a apoiar a elicitação de novos requisitos. Nesse caso, é importante destacar a finalidade dessa validação. Existe o risco do processo de validação de um protótipo se resumir a analisar requisitos da interface com o usuário. Isso não será um problema se a interface com o usuário for um ponto crítico e que precise de muita atenção. Entretanto, não se pode perder de vista a validação das regras de negócio e dos demais tipos de requisitos, como os requisitos de qualidade.

Especificar um conjunto de casos de teste que possam ser utilizados para testar o produto também faz parte desta etapa. Esses casos de teste podem ser documentos ou casos automatizados, dependendo do nível de maturidade da organização. Testes podem ser utilizados posteriormente para verificar se o produto final cumpre o que estava na especificação (verificação) ou para validar se o produto faz aquilo que deveria fazer (validação).

Gerenciamento de requisitos

Permeando todo o ciclo dos requisitos, encontram-se as atividades relacionadas à fase de gerenciamento dos requisitos, que trata do estabelecimento de formas de rastrear os requisitos e facilitar a análise do impacto de uma solicitação de mudança para a tomada de decisão.

Pesquisa do PMI (Project Management Institute) aponta problemas no gerenciamento de requisitos (LARSON, 2014):

- Somente 49% das organizações têm recursos alocados adequadamente para o gerenciamento de requisitos.
- Somente 33% dos líderes das organizações valorizam o gerenciamento de requisitos como uma competência crítica.
- Somente 47% das organizações têm um processo formal de validação de requisitos.
- Dos recursos financeiros gastos em projetos e programas, 51% são desperdiçados devido ao gerenciamento de requisitos ineficiente.
- Dos objetivos não atendidos de projetos, 47% foram devido ao gerenciamento de requisitos ineficiente.

Para que os impactos de uma solicitação de mudança possam ser analisados de forma precisa, é necessário que se tenha conhecimento da fonte do requisito, de como os requisitos se relacionam entre si e de como eles se relacionam com os artefatos seguintes, como o código, por exemplo. Essa análise permite que tenhamos uma estimativa dos recursos e do tempo necessários para realizar a mudança solicitada, bem como para identificar quais artefatos precisam ser alterados. Isso permite que seja tomada uma decisão embasada sobre a solicitação de mudança.

Manter essa rastreabilidade entre os elementos não é uma atividade simples, especialmente em grandes produtos. Esta é uma atividade que permeia todo o ciclo de vida do produto e nasce quando os primeiros requisitos são identificados e especificados. Não é viável construir matrizes que precisem ser mantidas de forma manual, usando planilhas ou outro tipo de tabela; isso só é possível por meio de ferramentas automatizadas. Pior que não ter uma matriz de rastreabilidade é ter uma matriz desatualizada.

Referências

BOEING 737 Max Lion Air crash caused by series of failures. *BBC*, Estados Unidos, 1990. Disponível em: https://www.bbc.com/news/business-50177788. Acesso em: 30 dez. 2019.

BOURQUE, P.; FAIRLEY, R. (ed.). *SWEBOK:* guide to the *software* engineering body of knowledge version 3. Washington: IEEE Computer Society, 2014.

BROOKS, F. No silver bullet: essence and accidents of *software* engineering. *Computer*, v. 20, n. 4, p. 10–19, 1987.

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: 30 dez. 2019.

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: 30 dez. 2019.

IEEE. *IEEE Std 610.12-1990*: IEEE Standard Glossary of *Software* Engineering Terminology. Los Alamitos: IEEE Computer Society Press, 1990.

ISBELL, D.; SAVAGE, D. Mars climate orbiter failure board releases report, numerous NASA actions underway in response. *In:* MARS POLAR LANDDER. Washington, 1999. Disponível em: https://mars.jpl.nasa.gov/msp98/news/mco991110.html. Acesso em: 30 dez. 2019.

LARSON, E. I still don"t have time to manage requirements: my project is later than ever. *In*: PMI® GLOBAL CONGRESS, 2014, Phoenix, AZ. *Proceedings*... Newtown Square, PA: Project Management Institute, 2014. Disponível em: https://www.pmi.org/learning/library/poor-requirements-management-source-failed-projects-9341. Acesso em: 30 dez. 2019.

MUNIZ, A. *et al*. *Jornada DevOps*: unindo cultura ágil, Lean e tecnologia para entrega de *software* de qualidade. Rio de Janeiro: Brasport, 2019.

SCHWABER, K.; SUTHERLAND, J. *Guia do SCRUM:* um guia definitivo para o SCRUM: as regras do jogo. [*S. l.*], 2018. Disponível em: https://www.scrumguides.org/index.html. Acesso em: 30 dez. 2019.

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

Leitura recomendada

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

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS