ENGENHARIA DE REQUISITOS

Sheila Reinehr

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Especificação dos requisitos não funcionais de *software*

Objetivos de aprendizagem

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

- Identificar os requisitos não funcionais de um projeto de *software*.
- Especificar os requisitos não funcionais de *software*.
- Selecionar indicadores para avaliar os requisitos não funcionais de *software*.

Introdução

Com a intensa proliferação de produtos tecnológicos no dia a dia das pessoas, as exigências do mercado ficaram mais altas. Todos somos rotineiramente expostos a diversos *softwares*, especialmente na forma de aplicativos nos *smartphones*. Com a mesma facilidade com que são baixados novos *apps*, eles também são deletados. Essa grande exposição a vários tipos de produto torna o usuário cada vez mais exigente, seja com a qualidade da interface, seja com a praticidade do uso, seja com a satisfação que as funcionalidades lhe proporcionam. Negligenciar esses requisitos de qualidade é deixar de lado um dos maiores fatores de conquista do usuário.

Requisitos não funcionais esquecidos ou não implementados adequadamente são uma das maiores causas de abandono de produtos de *software*. Há mais do que um conjunto de funcionalidades em um *software* de sucesso. Quando um *software* falha em entregar requisitos como bom desempenho, facilidade de uso, segurança e mais uma série de outros atributos de qualidade, é muito provável que ele tenha vida curta no mercado.

Quando determinados requisitos não funcionais são descobertos tardiamente no ciclo de desenvolvimento de *software*, pode não ser mais possível resolver o problema, ou pode ser muito caro resolvê-lo. Se decisões arquiteturais forem tomadas sem levar em consideração os requisitos não funcionais (de escalabilidade, por exemplo), pode ser que não seja possível implementar esse requisito sem que uma grande mudança seja feita. Isso pode inviabilizar a continuidade de um projeto ou de um produto de *software*, devido aos altos custos. Em casos extremos, pode causar até mesmo perda de vidas.

Neste capítulo você vai ver como identificar requisitos não funcionais de produtos de *software*. Vai também ler sobre como descrever esses requisitos e, por fim, sobre como definir indicadores que permitam sua verificação e validação.

1 Identificação de requisitos não funcionais de *software*

Certamente você já ouviu falar que requisitos não funcionais, também chamados de requisitos de qualidade, são mais difíceis de serem elicitados e especificados do que requisitos funcionais. Será que isso é verdade? Na maioria das vezes, sim. É comum que os *stakeholders* responsáveis por fornecer os requisitos não enxerguem, pelo menos à primeira vista, os requisitos não funcionais, e eles podem permanecer invisíveis por muito tempo.

Além disto, diferentes tipos de *software* terão diferentes tipos de requisitos não funcionais. *Softwares* com intensa interação com os usuários certamente terão requisitos distintos daqueles de *softwares* que estão embarcados em equipamentos. *Softwares* que desempenham funções críticas também terão demandas distintas de *softwares* que podem falhar. Dessa forma, o contexto de negócio e o tipo de aplicação irão influenciar sobremaneira os requisitos de qualidade aplicáveis.

Em sistemas interativos, ou seja, que são baseados no diálogo com o usuário, alguns requisitos não funcionais são fundamentais, como, por exemplo, requisitos relacionados à usabilidade. Usabilidade depende fortemente do contexto e do público-alvo. Se os usuários forem pessoas que têm intimidade com a tecnologia, é uma coisa; se os usuários forem pessoas leigas em tecnologia, é outra coisa bem diferente.

Se o *software* for disponibilizado sem treinamento, os requisitos de usabilidade aumentam. Por exemplo, *sites* que vendem produtos pela *web*. Não é esperado que as pessoas tenham que fazer um curso para aprender a usá-los, é esperado que elas acessem o *site* e consigam concluir sua compra com sucesso. O projeto da interface tem que ajudar o vendedor a não perder uma venda por problemas, por exemplo, de navegação no *site*.

Exemplo

Wiegers e Beatty (2013) citam o exemplo de um novo *software* com interface gráfica que foi instalado em uma equipe de *call center* em substituição ao sistema anterior, que era baseado em caracteres e teclas de atalho. Os usuários tinham uma grande produtividade no sistema anterior porque eram experientes e navegavam rapidamente com as teclas de atalho. Ao mudar para o novo sistema, o uso do *mouse*, intercalado com o uso do teclado, tirou-lhes a produtividade. Resumindo: o novo sistema teve que ser remodelado para contemplar a possibilidade da navegação com teclas de atalho.

Se estivermos falando de um sistema para o segmento bancário, teremos uma infinidade de requisitos não funcionais, mas certamente a segurança estará entre os de maior prioridade. Oferecer uma interface fácil de operar é importante também neste segmento, mas desde que os aspectos de segurança tenham sido adequadamente tratados.

Em sistemas de missão crítica relacionados a aspectos que envolvem vidas, por exemplo, haverá certamente requisitos não funcionais de os sistemas estarem livres de riscos. Este é o caso, por exemplo, de veículos autônomos. Uma operação incorreta pode custar a vida de pedestres ou de passageiros do próprio veículo. O mesmo ocorre para sistemas que podem oferecer riscos financeiros ou ambientais.

Existem os requisitos que não são diretamente visualizados pelos clientes e usuários, mas que são uma preocupação da equipe desenvolvedora — são os requisitos relacionados à qualidade interna do produto, como manutenibilidade, ou seja, facilidade de dar manutenção. Quantas vezes a equipe recebe um chamado relatando um *bug* que demora dias para ser localizado. Por que isto ocorre? Porque a arquitetura da aplicação não favorece a manutenção, dificultando a agilidade na solução de problemas ou na implementação de melhorias e evoluções.

Outro contexto específico é o das *startups* de tecnologia. É comum que elas comecem seu negócio baseadas em uma ideia inovadora dos sócios que se esforçam para colocar de pé um MVP (*minimum viable product* ou produto mínimo viável). Depois de um tempo, esse protótipo se transforma no produto que vai para o mercado. Como o tempo exigido pelo mercado é curto, a *startup* evolui o produto sem muito planejamento, afinal ela precisa escalar! Se os cuidados com a arquitetura e os requisitos de escalabilidade tiverem sido tomados, tudo certo; mas esse não é o cenário mais comum, e o produto evoluído tem dificuldade para escalar e para sofrer manutenções (BERG *et al*., 2018). É comum o código estar no formato de monolito e a manutenção ser dificultada. O que era fácil, quando a empresa e a equipe eram pequenas, se torna insustentável com o crescimento. Neste momento, os requisitos não funcionais se tornam críticos e passam a ameaçar a existência da própria empresa.

Nem todos os requisitos de qualidade serão viáveis para implementação em um produto de *software* (WIEGERS; BEATTY, 2013). Depois de elicitados, certamente haverá uma análise de prioridade, e aqueles que forem identificados como mais prioritários irão direcionar as escolhas da equipe, seja em termos de alocação de pessoas, seja em termos de escolhas tecnológicas e arquiteturais.

Fique atento

Nem todos os produtos de *software* são sistemas de informação; vários deles, especialmente hoje, são sistemas ditos **embarcados**, ou seja, aqueles em que o *software* está dentro de produtos de *hardware*, como sistemas mais complexos. Sistemas dessa natureza envolvem dispositivos mecânicos, elétricos e eletrônicos, como sensores, atuadores, controladores, motores, baterias etc. Nesses casos, requisitos relacionados ao desempenho podem se tornar críticos.

Nos sistemas baseados em IoT (*Internet of Things*, ou Internet das Coisas), por exemplo, um ponto crucial é o consumo de energia. Algoritmos para esses casos devem ser otimizados e devem consumir pouca energia para processar.

Como identificar os requisitos não funcionais

Assim como os requisitos funcionais, os requisitos não funcionais podem ser obtidos pelo contato direto com os *stakeholders* em reuniões, entrevistas, *brainstormings*, ou por meio de questionários. Podem ainda ser descobertos quando se conduz uma observação no ambiente de execução. Nesses momentos, é possível, para o observador, identificar requisitos não funcionais relacionados ao uso do sistema.

Muitas vezes, os requisitos precisam ser obtidos a partir da análise de documentos, leis, regulamentações, guias etc. Nesses casos, os requisitos não funcionais podem aparecer de forma explícita ou implícita na documentação. Com a Lei Geral de Proteção de Dados Pessoais (LGPD), por exemplo, muito requisitos não funcionais foram explicitados. A consequência é que funcionalidades adicionais precisam ser implementadas em todos os sistemas do mundo. Uma delas, a mais óbvia, é alertar ao usuário, logo na instalação do produto, quais são as políticas de segurança e compartilhamento de dados que são aplicadas pelo fornecedor, exigindo que ele concorde com elas.

Saiba mais

Para saber mais sobre a Lei nº 13.709, de 14 de agosto de 2018, mais conhecida como Lei Geral de Proteção de Dados Pessoais (LGPD), consulte o *site* da Casa Civil do Governo Federal ou o Diário Oficial da União (DOU). Para obter informações sobre a lei internacional equivalente, pesquise por General Data Protection Regulation (GDPR).

Não existe uma fórmula mágica para fazer com que os requisitos não funcionais surjam, mas as atividades sugeridas por Wiegers e Beatty (2013, p. 265), apresentadas na Figura 1, podem ajudar. Elas foram baseadas em um documento disponibilizado *on-line* pela empresa Clarrus (2010).

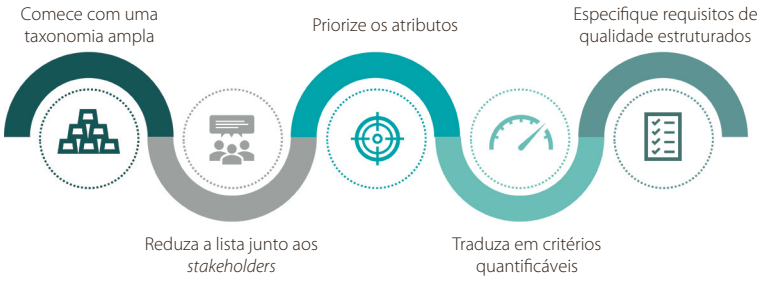

Figura 1. Elicitação de requisitos não funcionais.
Fonte: Adaptada de Wiegers e Beatty (2013).

Como se pode observar na Figura 1, os autores sugerem que a descoberta dos requisitos não funcionais comece partindo de uma **lista abrangente de atributos de qualidade**. Naturalmente, nem todos os tipos de *software* necessitarão de todos os atributos. Além disso, nem sempre se tem todos os recursos disponíveis para implementá-los, mas é melhor começar com uma lista maior do que correr o risco de deixar algum atributo importante de lado.

Como sugestão para a elaboração dessa lista, você pode utilizar as características e subcaracterísticas de qualidade que estão presentes na família de normas internacionais ISO/IEC 25000, mais especificamente o modelo de qualidade em uso e o modelo de qualidade de produto, que podem ser encontrados na norma internacional ISO/IEC 25010 (ISO/IEC, 2011).

É possível também usar a relação compilada por Wiegers e Beatty (2013), apresentada no Quadro 1.

Em seguida, deve-se engajar todos os *stakeholders* relevantes do produto para que possam **selecionar os atributos de qualidade** que são importantes para o projeto. Em um primeiro momento, pode ser que eles selecionem todos, ou quase todos os atributos da lista. Isso pode gerar uma lista inviável de ser implementada, seja pelas restrições de contexto, seja por conflitos entre os próprios atributos. Cabe ao analista de requisitos conduzir os *stakeholders* durante essas decisões.

Quadro 1. Alguns atributos de qualidade

Qualidade externa	Breve descrição
Disponibilidade	Extensão na qual os serviços do sistema estão disponíveis quando e onde são necessários
Instalabilidade	Quão fácil é instalar, desinstalar e reinstalar corretamente a aplicação
Integridade	A extensão na qual o sistema protege contra imprecisão e perda de dados
Interopera-bilidade	Quão fácil o sistema pode interconectar e trocar dados com outros sistemas e componentes
Desempenho	Quão rápido e previsivelmente o sistema responde às entra-das do usuário e outros eventos
Confiabilidade	Por quanto tempo o sistema roda antes de apresentar uma falha
Robustez	Quão bem o sistema responde a uma condição inesperada de operação
Proteção	Quão bem o sistema protege contra ferimentos e danos
Segurança	Quão bem o sistema protege contra acesso não autorizado à aplicação e seus dados
Usabilidade	Quão fácil é para as pessoas aprenderem, lembrarem e utilizarem o sistema
Qualidade interna	**Breve descrição**
Eficiência	Quão eficientemente o sistema usa os recursos do computador
Modificabilidade	Quão fácil é manter, modificar, melhorar e reestruturar o sistema
Portabilidade	Quão facilmente o sistema pode ser colocado em operação em outros ambientes operacionais
Reusabilidade	Em que extensão os componentes podem ser usados em outros sistemas
Escalabilidade	Quão facilmente o sistema pode crescer para tratar mais usuários, transações, servidores e outras extensões
Verificabilidade	Quão prontamente desenvolvedores e testadores podem confirmar que o *software* foi implementado corretamente

Fonte: Adaptado de Wiegers e Beatty (2013).

O próximo passo é realizar a priorização desses atributos. Uma forma de realizar essa atividade é usando uma tabela que combina os diversos atributos e faz análise sobre eles, dois a dois. Isso é importante para ajudar o analista de requisitos a focar as atividades de elicitação sobre os atributos relevantes, não gastando grande esforço em atributos que possam ser descartados. O Quadro 2 apresenta um exemplo adaptado de Wiegers e Beatty (2013) que reflete o que foi proposto por Clarrus (2010).

Quadro 2. Exemplo de quadro para priorização de atributos de qualidade

Qualidade externa	***Score***	**Disponibilidade**	**Integridade**	**Desempenho**	**Confiabilidade**	**Robustez**	**Segurança**	**Usabilidade**	**Verificabilidade**
Disponibilidade	1		^	<	^	^	^	^	^
Integridade	7			<	<	<	<	<	<
Desempenho	1				^	^	^	<	^
Confiabilidade	3					^	^	<	^
Robustez	4						^	^	<
Segurança	5							^	<
Usabilidade	3								^
Verificabilidade	4								

Fonte: Adaptado de Wiegers e Beatty (2013).

A tabela funciona de forma muito simples. Para cada par de atributos você faz a seguinte pergunta: se for para escolher um dos dois, qual é o mais importante? Ao entrar com um sinal de menor (<) indica que o atributo da linha é mais importante. Ao entrar com o sinal circunflexo (^) indica que o atributo da coluna é mais importante.

O exemplo apresentado no Quadro 2 é fictício. A leitura seria a seguinte: entre o atributo "disponibilidade" e "integridade", o símbolo aponta para cima, ou seja, para a "integridade". Isso significa que o atributo "integridade", para este contexto, é mais importante do que "disponibilidade". A lista de atributos é um recorte da lista abrangente apresentada no passo um e que foi refinada no passo dois. Esta matriz é útil para a tomada de decisão em momentos em que seja necessário priorizar um requisito em detrimento de outro.

A coluna do *score* é obtida pela soma de vezes em que o atributo é selecionado como prioridade. No caso do exemplo fictício do Quadro 2, nota-se que o atributo "integridade" foi mais importante do que todos os outros sete atributos, recebendo o *score* 7. Em seguida, vem o atributo "segurança", com 5 pontos, e assim por diante.

O objetivo do uso dessa tabela é obter uma visão consolidada sobre a prioridade dos requisitos não funcionais. É possível que elas sejam conflitantes entre os diversos *stakeholders*, e, nesse momento, cabe ao analista de requisitos resolver esses conflitos junto aos *stakeholders*.

A próxima etapa é traduzir os atributos em **critérios quantificáveis**. Geralmente os *stakeholders* usam palavras ambíguas para expressar os requisitos não funcionais, como amigável, rápido, confiável, robusto e assim por diante. Entretanto essas expressões geram interpretações imprecisas e, geralmente, são impossíveis de validar. Para chegar a critérios mais objetivos, será necessário que o analista de requisitos faça perguntas que os *stakeholders* saibam responder e na linguagem que eles entendem. Em vez de perguntar quão robusto o sistema deve ser, o analista deveria perguntar quantos usuários simultâneos estão previstos, por exemplo.

A última etapa é **estruturar os requisitos de qualidade**. Wiegers e Beatty (2013) recomendam que seja utilizado sempre o mnemônico SMART (*specific*, *measurable*, *attainable*, *relevant and time-sensitive*), ou seja, o requisito deve ser específico, mensurável, atingível, relevante e sensível ao tempo.

Fique atento

Se o testador não puder testar um requisito, então o requisito não está bom o suficiente.

Identificação de requisitos não funcionais em ambientes ágeis

Em ambientes de desenvolvimento ágil de *software*, a identificação dos requisitos não funcionais geralmente ocorre por meio de um *workshop* para discussão das histórias de usuário. As histórias de usuário vão focar nos requisitos funcionais, mas o seu complemento, que são os critérios de aceitação, irão incluir critérios referentes aos requisitos funcionais e também aos requisitos não funcionais. E é aí que eles, os requisitos não funcionais, entram em cena.

Patton (2014) apresenta algumas dicas para a condução de *workshops* para a discussão das histórias do usuário e dos critérios de aceitação, acompanhe:

- O *workshop* é, basicamente, uma conversa apoiada por diversas figuras e dados, ao final do qual a equipe deve chegar à conclusão de como será validado o que foi decidido construir.
- Antes do *workshop* é importante comunicar à equipe quais histórias do usuário serão discutidas.
- O ideal é um *workshop* pequeno, entre três a cinco pessoas. Patton (2014) recomenda que seja "do tamanho de conversa de jantar".
- Cerifique-se de ter incluído as pessoas certas: (1) alguém que entenda o usuário, geralmente um analista de negócios, o *product owner* ou alguém com conhecimento em experiência do usuário (UX); (2) desenvolvedores que entendam de código e que saibam analisar a viabilidade técnica do que está sendo proposto; (3) um analista de teste que fará as perguntas difíceis, quando a maioria será otimista. Outras pessoas podem ser incluídas, mas lembre-se de que muitas pessoas juntas podem tornar o *workshop* improdutivo.
- Mergulhe fundo para compreender: exatamente quem é o usuário; exatamente como ele vai usar o *software*; exatamente como ele deve se parecer (interface); exatamente como o *software* deve se comportar por trás da interface (regras de negócio e validação de dados); basicamente como o *software* será construído — para permitir estimativas.
- Entre em acordo sobre o que será construído, respondendo a duas perguntas: o que vamos checar para verificar o que foi construído; como vamos demonstrar o *software* quando formos revisar juntos.
- Converse e documente tudo o que ficou decidido, pode ser com desenhos, anotações e fotografias.
- Fale por meio de exemplos para materializar a história. Seja específico sobre situações e dados.
- Divida ou esvazie histórias, se necessário.

Utilizar *workshops* para definir os requisitos não funcionais, ou seja, os critérios de aceitação, pode ser uma excelente escolha. No entanto, dependendo da característica das pessoas envolvidas, pode não dar muito certo. Saiba como reconhecer quando o *workshop* não está funcionando verificando estes pontos (PATTON, 2014):

- Ninguém participa: alguém descreve o que é requerido e os outros escutam.
- Quando o foco fica apenas nos critérios de aceitação e não se discute a história e quem faz o que, e por quê.
- Quando se falha em identificar opções, sejam funcionais, sejam técnicas.

Fique atento

Identificar os requisitos não funcionais de *software* não exige uma técnica especial, mas sim uma atitude especial. É preciso que o analista de requisitos, ou papel equivalente, esteja atento para explorar, junto aos diversos *stakeholders*, quais são os pontos críticos do sistema e para onde a equipe de desenvolvimento deve envidar os seus esforços. Um *checklist* contendo os principais requisitos de qualidade pode ajudar na identificação desse tipo de requisito. A família de normas internacionais ISO/IEC 25000 pode ajudar.

2 Especificação de requisitos não funcionais de *software*

Quando falamos de requisitos funcionais logo nos vem à mente a especificação utilizando os diagramas de caso de uso e as especificações de caso de uso. Embora essas sejam duas ferramentas muito poderosas para a especificação de requisitos, elas não se aplicam aos requisitos não funcionais. Elas são boas para explicitar as funcionalidades.

Para as especificações de requisitos não funcionais, usamos alguma forma de documentação suplementar, que visa a explicitar os detalhes desses requisitos, de modo que possam seguir para as demais fases do desenvolvimento.

O Quadro 3 apresenta alguns exemplos de especificação de requisitos não funcionais, extraídos de Wiegers e Beatty (2013). Note que todos eles têm em

comum a característica de buscar trazer mais precisão para a declaração do requisito.

Quadro 3. Exemplos de especificação de requisitos não funcionais

Tipo	Identificador	Descrição
Disponibilidade	AVL-1	O sistema deve estar pelo menos 95% disponível nos dias de semana das 6:00 h às 24:00 h e pelo menos 99% disponível nos dias de semana entre 15:00 h e 17:00 h.
Instalabilidade	INS-1	Um usuário sem treinamento deve ser capaz de executar com sucesso a instalação inicial da aplicação em 10 minutos.
Integridade	INT-1	Depois de executar o *backup* de arquivos, o sistema deve verificar a cópia do *backup* em relação ao original e relatar qualquer discrepância.
Desempenho	PER-1	A autorização de um saque no caixa eletrônico não deve demorar mais do que 2 segundos.
Segurança	SEC-1	O sistema deve bloquear a conta do usuário depois da quarta tentativa de *login* sem sucesso consecutiva em um período de 5 minutos.
Usabilidade	USE-1	Todas as funções do menu Arquivo devem ser teclas de atalho definidas que usam a tecla Ctrl pressionada simultaneamente com outra tecla. Comandos do menu que também aparecerem no MS-Word devem usar as mesmas teclas de atalho *default* que o MS-Word usa.

Fonte: Adaptado de Wiegers e Beatty (2013).

Requisitos não funcionais nos ambientes ágeis

Quando uma empresa utiliza métodos ágeis, os requisitos são especificados no formato de histórias de usuário. De acordo com Patton (2014, p.91, tradução nossa), "Histórias foram assim nomeadas devido a como é esperado que elas

sejam usadas, não devido à forma como você está tentando escrevê-las". Uma história do usuário nada mais é do que um cenário de uso de um produto por um tipo específico de usuário.

Segundo Patton (2014), Roy Jeffrey, um dos criadores do XP, estabeleceu os 3 Cs da história de usuário: **C**artão, **C**onversa e **C**onfirmação, representados da Figura 2. O **cartão** é onde a história do usuário é escrita. A **conversa** refere-se ao fato de que a história nasce da intensa comunicação entre a equipe de desenvolvimento e o usuário. E, por fim, a **confirmação** representa os critérios de aceitação da história, que é onde os requisitos não funcionais são especificados. Também pode ser que você encontre lugares em que os requisitos não funcionais sejam registrados no próprio cartão, como se fosse uma história independente.

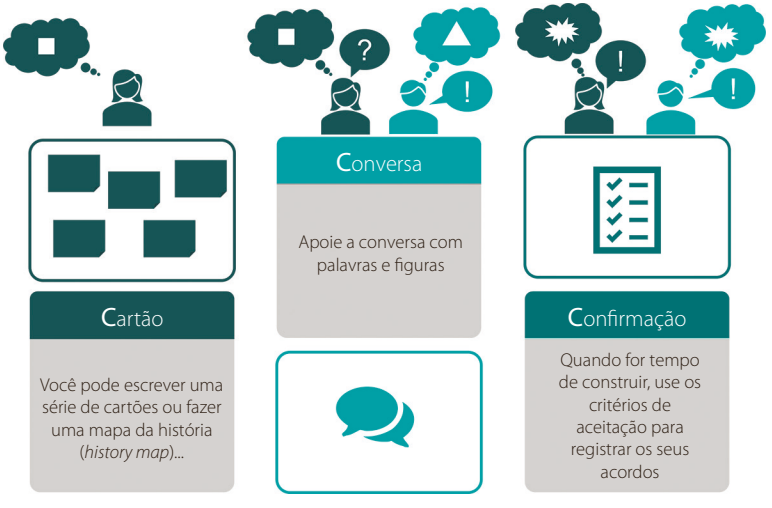

Figura 2. Os 3 Cs da história de usuário.

Fonte: Adaptada de Patton (2014).

3 Indicadores para avaliar requisitos não funcionais de *software*

Uma das formas de analisar se chegamos a um resultado é comparar o valor obtido com o valor planejado. Isso vale para qualquer aspecto da vida pessoal e profissional. Medidas fazem parte de nossa vida desde que nascemos: nasceu

dentro da faixa de peso ideal? Cresceu o suficiente no primeiro mês? Está no tamanho ideal para a idade?

Se você quiser saber se foi bem em uma prova, você compara o resultado ideal (o 10,0) com o resultado que você obteve — isso vai lhe mostrar o quanto você está distante do ideal. Se você quiser comparar o seu desempenho com o dos colegas, vai analisar o que você tirou em relação à média dos seus colegas. Se você estiver em um treinamento para fazer uma maratona, vai analisar quantos quilômetros consegue correr em determinado tempo. Se quiser emagrecer, vai estabelecer um peso ideal e um caminho para chegar lá.

Medições também fazem parte da nossa vida profissional: quanto tempo precisamos para entregar esta funcionalidade? Quantas pessoas são necessárias para este projeto? Quanto tempo falta para terminar o projeto? Qual é a produtividade média da equipe? Qual é a gravidade deste risco? Quanto tempo levaríamos para implementar todo o *backlog* do produto?

Só é possível controlar aquilo que se consegue medir. Só é possível avaliar se houver algum padrão para comparar. Só é possível planejar com certa precisão se houver medidas confiáveis nas quais se basear.

E como medir a qualidade de um produto de *software*? Como garantir se o que foi entregue é o que o usuário queria? Se estivermos falando de funcionalidades, podemos contar quantas funcionalidades foram entregues em relação a quantas funcionalidades foram solicitadas. Mas será que isso é suficiente? E se a funcionalidade não apoiar o usuário adequadamente em suas tarefas, conta como funcionalidade entregue com qualidade?

A grande questão é: como garantir que aquilo que vamos entregar atenderá às expectativas dos usuários do produto? Inicialmente, investindo na elicitação de requisitos, com especial atenção para os requisitos não funcionais. Em seguida, identificando indicadores que podem ser perseguidos pela equipe técnica para garantir que as necessidades dos usuários serão atendidas na medida certa.

Conceitos de medição

Existem termos que têm um significado específico quando se trata de medição. Vamos adotar aqui os conceitos utilizados na norma internacional ISO/IEC 15939 (ISO/IEC, 2007), que é a norma da ISO/IEC que descreve o processo de medição. Os termos relevantes se encontram no Quadro 4.

Quadro 4. Definição de termos para medição

Termo	Significado
Atributo	Propriedade ou característica de uma entidade que possa ser distinguida quantitativa ou qualitativamente por seres humanos ou meios automatizados
Dados	Coleta de valores atribuídos a medidas base, medidas derivadas e/ou indicadores
Entidade	Objeto que deve ser caracterizado medindo seus atributos
Escala	Conjunto ordenado de valores, contínuo ou discreto, ou um conjunto de categorias para as quais o atributo está mapeado
Função de medição	Algoritmo ou cálculo realizado para combinar duas ou mais medidas básicas
Indicador	Medida que fornece uma estimativa ou avaliação de atributos especificados derivados de um modelo com relação a necessidades de informação definidas
Medição	Conjunto de operações com o objetivo de determinar uma medida
Medida	Variável à qual um valor é atribuído resultante de uma medição
Medida básica	Medida definida em termos de um atributo e método para quantificá-lo
Medida derivada	Medida que é definida como uma função de dois ou mais valores de medidas básicas
Medir	Fazer uma medição
Procedimento de medição	Conjunto de operações, descrito especificamente, utilizado no desempenho de determinada medição de acordo com determinado método
Valor	Resultado numérico ou categórico atribuído a uma medida básica, medida derivada ou indicador

Fonte: Adaptado de ISO/IEC (2007).

A Figura 3, adaptada da norma ISO/IEC 15939 (ISO/IEC, 2007), apresenta os relacionamentos entre os elementos envolvidos na medição. Para que eles possam ser mais bem compreendidos, vamos utilizar um exemplo prático e fácil. Vamos supor que a **necessidade de informação** de um *stakeholder* seja identificar se a produtividade de sua equipe de desenvolvimento está compatível com a produtividade média do mercado.

Para atender a essa necessidade de informação vamos começar identificando as entidades envolvidas e quais são os atributos (características) dessas entidades que nós precisamos medir para atender à necessidade de informação. Nesse caso, a **entidade** pode ser a tarefa realizada e os **atributos** podem ser o tamanho da tarefa e o tempo necessário para desenvolver a tarefa.

Em seguida definimos qual é o **método de medição** envolvido. Neste caso precisamos saber como obter o valor do atributo de tamanho da tarefa e o valor do atributo do tempo. Se a tarefa for uma história de usuário, podemos utilizar o conceito de *story points* (pontos de história do usuário). Para o atributo tempo, podemos assumir que seja o tempo decorrido desde o início até o término da história do usuário. Portanto, nossas **medidas básicas** serão quantidade de pontos de história de usuário e tempo de desenvolvimento da história.

Muitas vezes temos que nos valer de métodos de medição mais subjetivos (que envolvem o julgamento humano). Devemos evitá-los e dar sempre preferência a métodos mais objetivos, ou seja, que envolvem uma medição (contagem) a ser realizada de forma manual ou automatizada.

A **função de medição** será a forma como vamos combinar as duas medidas básicas para gerar a **medida derivada**. Nesse caso vamos considerar que a medida derivada será a produtividade e que a função de medição será a razão entre a quantidade de pontos de história de usuário pelo tempo de desenvolvimento da história.

Em seguida estabelecemos o **modelo de análise**, que é a forma como vamos analisar a medida derivada que acabamos de obter. Neste nosso exemplo, podemos entender que seria a forma como vamos interpretar a produtividade obtida. Aqui estão envolvidos os critérios de decisão, que poderiam envolver o seguinte: qual é o valor médio da produtividade no mercado para este tipo de produto, com esta tecnologia? Essa seria a base para poder interpretar a produtividade que foi obtida usando nossa função de medição.

O que se obtém é um **indicador**, que pode nos dizer se a produtividade está abaixo do mercado, se a produtividade está na média do mercado ou se a produtividade está acima do mercado. Isso nos permite realizar a **inter-**

pretação da informação, que é a explicação relacionada com a informação quantitativa expressa pelo indicador, em uma linguagem que a pessoa que vai usar a medição compreenda.

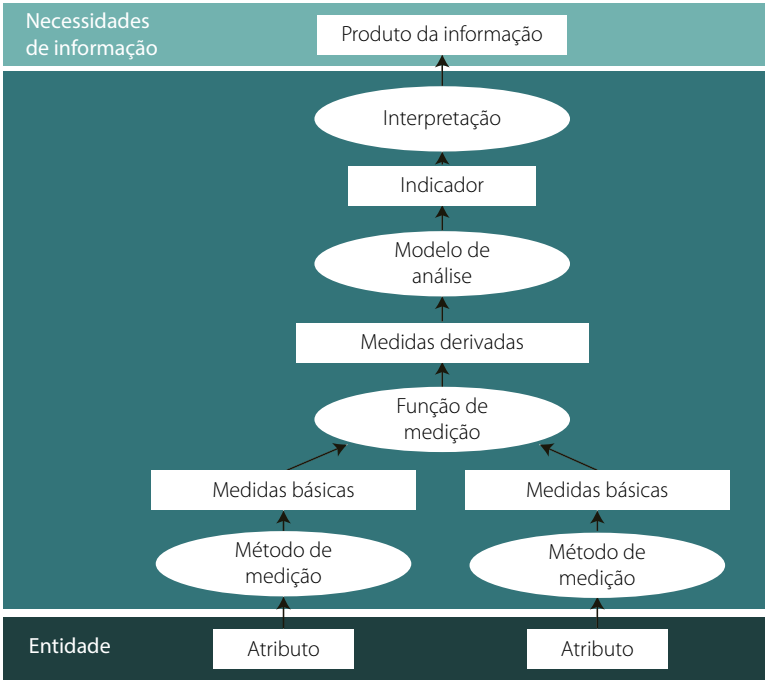

Figura 3. Modelo de Informação de Medição da ISO/IEC 15939.

Fonte: Adaptada de ISO/IEC (2007).

Saiba mais

A norma internacional ISO/IEC 15939 ISO/IEC (2007) apresenta mais detalhes sobre o processo de medição. Exemplos são fornecidos no Anexo A da norma, como forma de esclarecer os conceitos em contextos diferentes.

Como definir indicadores de qualidade de produto

Na seção anterior vimos os conceitos de medição, vimos como esses conceitos se relacionam e vimos também um exemplo aplicado a uma medição do cotidiano das empresas — a produtividade. Agora vamos ver como aplicar isso à qualidade do produto de *software*, ou seja, aos atributos que estão envolvidos em determinar se os requisitos não funcionais foram atendidos.

A família de normas internacionais da ISO/IEC que trata da qualidade dos produtos de *software* é a família 25000 (ISO/IEC, 2014). Entre esse conjunto abrangente de normas, podemos destacar as seguintes:

- ISO/IEC 25022: Medição da qualidade em uso.
- ISO/IEC 25023: Medição da qualidade de produto de sistemas e *software*.
- ISO/IEC 25024: Medição da qualidade de dados.

A ISO/IEC 25022 (ISO/IEC, 2016a) trata da medição de aspectos relacionados à qualidade em uso, ou seja, aqueles aspectos que são percebidos pelo usuário, quando ele usa o produto no ambiente-alvo. Seria, por exemplo, quando você utiliza um aplicativo no seu celular para pedir comida. Neste caso, haverá dois aspectos envolvidos na sua percepção de qualidade: um está relacionado ao serviço prestado e o outro, ao produto de *software* em si. E é bem comum confundir os dois!

Exemplo

Vamos imaginar que você queira pedir uma pizza individual e o aplicativo só mostre pizzas grandes. Esse certamente será um problema que vai deixar você muito insatisfeito, mas esse não é um problema do *app* que você está usando, mas sim do serviço que está sendo prestado via *app*. O problema é da pizzaria, que não oferece a pizza do tamanho que você precisa.

Agora, se na hora que você for fechar o pedido, tiver dificuldade para conseguir localizar onde pode trocar o endereço de entrega, mesmo essa funcionalidade existindo no *app*, então esse é um problema de usabilidade do aplicativo. Ele provavelmente não tem uma interface intuitiva, uma vez que você não consegue localizar uma funcionalidade. Esse sim é um requisito não funcional relacionado à usabilidade do produto e não ao serviço.

Mas há limites menos óbvios para o usuário entre o que é um problema do produto e o que é um problema do serviço. Vamos continuar no exemplo da pizza: imagine que só é oferecida a forma de pagamento com cartão de crédito. Este é um problema do serviço ou do produto de *software*? Na verdade, depende. Pode ser que o aplicativo ofereça diversas formas de pagamento e o estabelecimento comercial optou por configurar apenas a opção de pagamento em cartão de crédito. Neste caso, é uma limitação do negócio e não do aplicativo. Já se o aplicativo não permitir o cadastramento de outra forma de pagamento, por mais que o estabelecimento permita, então é uma limitação do aplicativo, que causa desconforto ao usuário.

Certamente você já deve ter experimentado diversos aplicativos no seu celular. Alguns você achou fáceis e agradáveis de utilizar e outros você abandonou o uso porque não gostou deles. Às vezes os motivos de não ter gostado são fáceis de identificar, por exemplo: achei a tela muito poluída e ruim de navegar, os controles não estavam fáceis de localizar, o aplicativo vivia travando, não me sentia seguro em informar meus dados bancários etc. Mas, às vezes, você não consegue saber com certeza porque não gosta deste ou daquele aplicativo, e é aí que está o problema: **utilizar um *software* é uma experiência que envolve diversas percepções**, muitas delas difíceis de precisar, pois são mais sutis e subjetivas.

O que tentamos fazer ao definir medidas para avaliar os requisitos não funcionais é retirar o máximo que conseguimos das subjetividades e tornar a avaliação mais precisa. A ISO/IEC 25022 (ISO/IEC, 2016a) nos ajuda apresentando as medidas que podem ser utilizadas para avaliar, de forma objetiva, a qualidade em uso de produtos de *software*.

A Figura 4 apresenta as características relacionadas à qualidade em uso, para as quais a norma apresenta as propostas de medidas.

Figura 4. Características de qualidade em uso tratadas na ISO/IEC 25022.
Fonte: Adaptada de ISO/IEC (2016a).

A norma propõe um grupo de medidas referentes à **efetividade**, que são aquelas que avaliam a precisão e a completude com que o produto apoia as atividades do usuário. A norma sugere que podem ser usadas medidas como as seguintes:

- percentual de tarefas que foram completadas sem ajuda, em relação ao total de tarefas que se tentou realizar;
- percentual de objetivos que foram atingidos sem ajuda;
- quantidade de erros cometidos pelo usuário ao realizar uma tarefa;
- percentual de tarefas que tiveram erros cometidos pelo usuário;
- proporção de usuários cometendo erros em tarefas.

É fácil imaginar que se o usuário comete erros realizando a tarefa no *software*, significa que o *software* não está ajudando o usuário a evitar esses erros. Se utilizarmos as medidas definidas acima, podemos identificar o tamanho do impacto que isso pode acarretar à experiência do usuário.

E como evitamos erros cometidos pelo usuário? Por exemplo, quando colocamos uma mensagem do tipo "Confirma a operação?", estamos dando ao usuário a oportunidade de conferir o que ele realizou e de voltar atrás. É uma forma de chamar a atenção do usuário em pontos críticos, onde um erro

pode ter consequências mais graves. Esse é um recurso muito usado quando estamos finalizando uma operação bancária, por exemplo.

Medidas de **eficiência** também são propostas pela norma. Essas são as medidas que avaliam o uso dos recursos para que o usuário atinja os seus objetivos: podemos usar medidas simples, como o tempo que o usuário leva para fazer uma tarefa, ou podemos usar medidas mais complexas, como as listadas a seguir:

- Eficiência em relação ao tempo — quantidade de tarefas que ele faz em determinado tempo (por exemplo, quantos novos cadastros ele consegue fazer por hora).
- Eficiência em relação ao custo — custo para realizar as tarefas.
- Proporção de tempo que o usuário está executando uma tarefa útil — tempo que o usuário realiza a tarefa real, em comparação ao tempo que ele gasta esperando o sistema se recuperar de um erro, por exemplo, ou o tempo que ele gasta pedindo ajuda.
- Ações desnecessárias.
- Consequências da fadiga — como a produtividade cai com o uso prolongado.

Uma medida muito interessante que pode ajudar em dois aspectos diferentes é a quantidade de ações desnecessárias que um usuário realiza para completar uma tarefa. Por exemplo, a quantidade de cliques errados que ele deu antes de conseguir chegar no item correto que ele precisava. Isso pode apontar uma ineficiência (tempo gasto em tarefas não úteis) e também um problema na usabilidade — se ele tem que clicar em diversos itens incorretos antes de achar o item correto, a interface não está intuitiva.

Um terceiro aspecto da qualidade em uso para o qual a norma define medidas é a **satisfação**. Ela visa a medir o grau de satisfação do usuário quando utiliza o *software*. Na norma, as medidas de satisfação estão divididas em subcaracterísticas, que são as seguintes: utilidade, confiança, prazer (experiência do usuário), conforto (ergonomia).

Utilidade pode ser medida da seguinte forma:

- medida de satisfação geral;
- medida de satisfação com uma funcionalidade específica;
- percentual de usuários que usa uma função em relação ao total que poderia utilizá-la;

- percentual de usuários reclamando do *software*;
- percentual de reclamações dos usuários em relação a uma funcionalidade específica em relação às reclamações totais.

As subcaracterísticas de **confiança**, **prazer (experiência do usuário)** e **conforto (ergonomia)** podem ser medidas por meio de uma pesquisa de satisfação junto aos usuários do produto.

A quarta característica que a norma nos ajuda a gerar medidas em relação à qualidade em uso é o fato de o *software* **ser livre de riscos** (do original, em inglês *freedom from risk*). Riscos podem estar relacionados às subcaracterísticas de riscos econômicos, riscos de saúde e segurança e riscos ambientais.

- Os **riscos econômicos** estão mais relacionados a aspectos de negócio, como o retorno sobre o investimento (ROI), o momento para iniciar o ROI, a lucratividade com o uso do produto e o nível de serviço dos usuários, quantos visitantes de um *website* se tornam clientes, vendas para um cliente e prejuízos decorrentes do produto.
- Os **riscos de saúde e segurança** se referem, como o próprio nome sugere, aos relatos de problemas de saúde ou de segurança decorrentes do uso do *software*. Esses indicadores são especialmente importantes quando estamos desenvolvendo sistemas de missão crítica que podem ocasionar perdas de vidas ou danos à sua segurança e saúde.
- Finalmente, os **riscos ambientais** se referem a possíveis danos ao ambiente que o produto de *software* possa ocasionar. Dependendo do tipo de sistema, isso pode estar relacionado à poluição, poluição auditiva ou aquecimento global, por exemplo.

A quinta e última característica que a norma aborda é referente à **cobertura do contexto.** Neste caso, são abordados dois aspectos: completude do contexto e flexibilidade. A medida de **completude do contexto** é mais específica e se refere ao quanto o contexto é coberto pelo *software.* Já as medidas sugeridas para a flexibilidade incluem o seguinte:

- contexto de uso flexível;
- flexibilidade do produto;
- independência de proficiência.

Fique atento

Apresentamos nesta seção um resumo de algumas das sugestões realizadas pela norma internacional ISO/IEC 25022 (ISO/IEC, 2016a). Para maiores detalhes, a norma deverá ser consultada.

Referências

BERG, V. *et al*. Software startup engineering: a systematic mapping study. *Journal of Systems and Software*, v. 144, p. 255–274, out. 2018. Disponível em: https://www.researchgate.net/profile/Ilias_Pappas/publication/325858070_Software_Startup_Engineering_A_Systematic_Mapping_Study/links/5c3dbfb3458515a4c726f927/Software-Startup-Engineering-A-Systematic-Mapping-Study.pdf. Acesso em: 24 abr. 2020.

CLARRUS. *Software quality attribute*: following all the steps. [*S. l.*], 2010. Disponível em: https://www.clarrus.com/documents/Software%20Quality%20Attributes.pdf. Acesso em: 24 abr. 2020.

ISO/IEC. *ISO/IEC 25000:2014*: systems and software engineering, systems and software quality requirements and evaluation (SQuaRE), guide to SQuaRE. Switezland: ISO, 2014.

ISO/IEC. *ISO/IEC 25022:2016*: systems and software engineering, systems and software quality requirements and evaluation (SQuaRE), measurement of quality in use. Switezland: ISO, 2016a.

ISO/IEC. *ISO/IEC 25023:2016*: systems and software engineering, systems and software quality requirements and evaluation (SQuaRE), measurement of system and software product quality. Switezland: ISO, 2016b.

ISO/IEC. *ISO/IEC 25024:2015*: systems and software engineering, systems and software quality requirements and evaluation (SQuaRE), measurement of data quality. Switezland: ISO, 2015.

ISO/IEC. *ISO/IEC/IEEE 15939:2017*: systems and software engineering, measurement process. Switzerland: ISO, 2007.

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

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

Leitura recomendada

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

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS