ENGENHARIA DE REQUISITOS

Sheila Reinehr

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Especificação de requisitos funcionais utilizando histórias de usuário

Objetivos de aprendizagem

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

- Reconhecer os elementos de um cenário de desenvolvimento ágil.
- Identificar os elementos de uma história de usuário.
- Aplicar histórias de usuário para especificar requisitos funcionais.

Introdução

A identificação e o registro dos requisitos de *software* constituem o primeiro, e o mais fundamental, passo para o desenvolvimento de um bom produto de *software*. Sua importância é reconhecida tanto por aqueles que praticam processos ágeis, como por aqueles que utilizam métodos mais tradicionais de desenvolvimento. As consequências de requisitos mal definidos são sentidas, principalmente, na forma de retrabalho, custos mais elevados de desenvolvimento e insatisfação do usuário.

Independentemente da forma que se escolha para realizar as atividades relacionadas a requisitos, um ponto é chave: o foco na entrega de valor para o cliente. A forma como esses requisitos serão registrados e gerenciados vai depender das diversas variáveis do contexto no qual o projeto está inserido, mas o foco deve ser sempre na entrega de valor. Esse deveria ser o mantra repetido e internalizado por todas as equipes de desenvolvimento do mundo, independentemente do processo de trabalho que elas adotam. Se não houver valor na entrega, ela nem deveria ser realizada.

Uma forma pela qual as equipes ágeis vêm trabalhando com os requisitos é o uso de histórias de usuário. Neste capítulo você vai estudar o ambiente ágil de desenvolvimento de *software,* vendo como identificar os elementos que compõem uma história de usuário e como aplicar as histórias de usuário.

1 Desenvolvimento ágil de *software*

Métodos ágeis vêm se consolidando como uma das formas mais utilizadas no desenvolvimento de *software*. Embora a sua origem anteceda os anos 2000, esses métodos ganharam mais força recentemente, especialmente devido à pressão do mercado por entregas mais rápidas e frequentes. Esse é o caso, por exemplo, do mercado de aplicativos para *smartphones*. A Apple Store tem mais de 100 milhões de usuários ativos que consomem mais de 2 milhões de aplicativos.

Os princípios de agilidade começaram a ser fortemente difundidos a partir do Manifesto para o Desenvolvimento Ágil de *Software*, ou, simplesmente Manifesto Ágil (BECK *et al.*, 2001). Esse documento foi produzido pelo esforço conjunto de diversos profissionais de *software*, que se reuniram em 2001 para discutir e organizar um conjunto de valores e princípios que passaram a ser considerados como a base do desenvolvimento ágil. Entre os profissionais estavam Alistair Cockburn (criador do método *Crystal Clear* e defensor da escrita de casos de uso eficazes); Kent Beck (criador do método XP — *eXtreme Programming*); e, Ken Schwaber e Jeff Sutherland (criadores do Scrum).

Apesar de divergirem entre si em alguns aspectos, e em alguns casos serem até mesmo concorrentes no mercado, esses profissionais chegaram à conclusão que partilhavam dos mesmos valores e princípios e chegaram a um consenso que está expresso no próprio documento gerado (BECK *et al.*, 2001).

Os valores ágeis, formalizados no Manifesto Ágil em 2001, estão representados na Figura 1. Como se pode observar, o foco está mais sobre as pessoas e a comunicação com o cliente do que sobre o formalismo de processos, planos e contratos. Porém, isso não quer dizer que esses metodologistas eram contra processos; na verdade, eles eram a favor de processos que agregassem valor ao produto final entregue ao usuário, e não processos que agregassem burocracias e documentações desnecessárias.

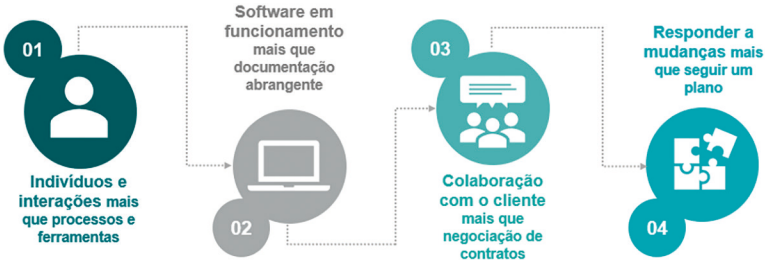

Figura 1. Valores dos métodos ágeis.
Fonte: Adaptada de Beck *et al.* (2001).

Os valores ajudam a construir o *mindset* ágil da equipe para produzir *softwares* melhores. Esse *mindset* se refere às atitudes que o time deve ter diante das tarefas que ele precisa desempenhar (STELLMAN; GREEN, 2017). No entanto, no dia a dia, nem sempre é tão fácil perceber como esses valores estão presentes. Para isso, o Manifesto Ágil definiu um conjunto de 12 princípios ágeis que ajudam a tornar mais palpáveis esses valores nas tarefas do dia a dia.

Saiba mais

Se quiser conhecer um pouco mais sobre o Manifesto para o Desenvolvimento Ágil de *Software*, você pode pesquisar a versão original do manifesto na *web* — é bem fácil de encontrar. No *site* oficial do manifesto, é possível conhecer sua história, valores e os 12 princípios ágeis, bem como a biografia dos profissionais que participaram desse encontro. Esses continuam sendo os princípios norteadores de todas as metodologias ágeis atualmente em vigor.

O método SCRUM

Para compreender melhor os ambientes ágeis, optamos por utilizar como exemplo o método SCRUM, uma vez que ele é considerado hoje o método mais utilizado no mundo ágil.

O SCRUM é um *framework* para desenvolver, manter e entregar produtos complexos (SCHWABER; SUTHERLAND, 2017). A Figura 2 representa o ciclo de desenvolvimento utilizado no SCRUM. Ela nos será útil para explicar as características desses ambientes.

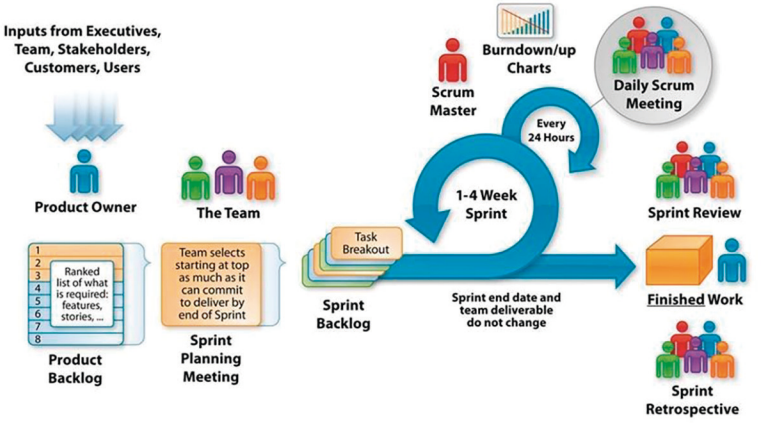

Figura 2. O ciclo de desenvolvimento SCRUM.

Fonte: Neon Rain Interactive (2020, documento *on-line*).

O princípio básico do SCRUM é de que o desenvolvimento de *software* deve ser dividido em entregas menores, de uma a quatro semanas, denominadas *sprints*. O desenvolvimento é todo organizado em torno das *sprints*. Ao final de uma *sprint*, é gerado um item potencialmente entregável para o cliente.

O *backlog* é o repositório de todas as requisições dos clientes e é gerenciado pelo *Product Owner* (PO, ou dono do produto). É o PO quem prioriza o conteúdo do *backlog* que será alocado à *sprint*. Só pode ser alocado à *sprint* um item do *backlog* que já esteja em um nível de detalhe suficiente para que a equipe possa implementar. Um item de *backlog* geralmente é uma história de usuário, mas pode ser qualquer solicitação do cliente.

Quem planeja a *sprint* é a equipe, estimando os itens do *backlog* e discutindo suas dificuldades para implementar. A ideia é que a data final da *sprint*, uma vez definida, não seja alterada, para que o cliente possa receber a entrega na data originalmente planejada. Ninguém tem autoridade de alterar a *sprint*, apenas a equipe de desenvolvimento. Isso previne a inserção de itens não previstos no decorrer da *sprint*.

O SCRUM também prevê alguns eventos, denominados **cerimônias**, que são reuniões com propósito específico e duração máxima estipulada, e são listadas a seguir:

- ***Sprint Planning Meeting* (Reunião de Planejamento da *Sprint*):** planejamento de como será realizada a *sprint*. É realizada pelo time SCRUM e seu esforço não deve ultrapassar 8 horas para uma *sprint* de um mês.
- ***Daily SCRUM Meeting* (Reunião Diária do SCRUM):** constitui a principal ferramenta de acompanhamento e comunicação. São reuniões curtas que acontecem todos os dias para que a equipe avalie o que fez, o que fará e o que não está andando. No XP elas são chamadas de *standup meetings* (literalmente reuniões em pé).
- ***Sprint Review* (Revisão da *Sprint*):** visa a reunir equipe e os principais *stakeholders* para avaliar o que estava planejado, o que foi realizado e quais as perspectivas para as próximas *sprints*. Tem a duração máxima de 4 horas para uma *sprint* de um mês.
- ***Sprint Retrospective* (Retrospectiva da *Sprint*):** o objetivo é que o time SCRUM faça uma avaliação interna de como as coisas ocorreram dentro da *sprint*. A partir daí são propostas as melhorias para as próximas *sprints*. Dura no máximo 3 horas para uma *sprint* de um mês.

O SCRUM define um papel central relacionado aos requisitos, que é a figura do ***Product Owner***. O PO é uma pessoa, e não um grupo de pessoas, mas ele pode até representar um grupo de pessoas ou um comitê. Sua função é gerenciar o *backlog* do produto. Outro papel é o **SCRUM Master**, cuja atribuição é a remoção dos impedimentos que atrapalham a realização da *sprint*. Ele não é um gerente de projetos, pois a ideia é que a equipe seja autogerenciada e autônoma. Ele é também um membro da equipe que realiza atividades técnicas. O papel do **time SCRUM** é ocupado por desenvolvedores e testadores.

Os requisitos geralmente são expressos no formato de histórias do usuário, que são os itens que compõem o *backlog* do produto. As histórias de usuário, uma vez compreendidas e detalhadas, serão a base para as estimativas que a equipe irá realizar para compor o seu planejamento da *sprint*. O *backlog* da *sprint* contém os itens que foram acordados para a *sprint*, além do plano para o seu desenvolvimento. Vamos conversar mais sobre histórias do usuário na próxima seção.

O que caracteriza um ambiente ágil é a presença de características comuns, que são o foco no cliente e no que agrega valor para esse cliente, a entrega frequente de valor para esse cliente (entregas mais curtas e rápidas), a comunicação aberta e transparente tanto dentro time de desenvolvimento quanto com o cliente.

O caso das *startups* de *software*

As *startups* de *software*, ou simplesmente *startups*, convivem em um ambiente dinâmico, caótico e coberto de incertezas, vivendo sob a constante pressão do mercado e dos concorrentes, ao mesmo tempo em que precisam lidar com a permanente escassez de recursos. Um projeto falho pode colocar sua existência em risco, e a maioria delas deixa de existir em um prazo de dois anos (GIARDINO *et al*., 2016).

Diversos são os fatores que contribuem para a falha das *startups* em sua fase inicial. A maioria desses fatores é de natureza interna, muito mais do que de natureza externa, como desenvolvedores inteligentes, mas inexperientes, negligenciando aspectos importantes de engenharia de *software*; o produto muitas vezes não é apenas um produto, mas uma família de produtos, levando a custos mais elevados de desenvolvimento; o produto não tem um dono, o que dificulta a tomada de decisão sobre as funcionalidades; não existe um planejamento estratégico para o desenvolvimento do produto; as tecnologias envolvidas podem não suportar a evolução do produto etc. (CROWNE, 2002).

Outros fatores surgem posteriormente, quando o produto já está no mercado e a *startup* enfrenta dificuldades para crescer: conflitos entre os membros que estavam desde o início e os novos membros; o *software* se torna complexo e instável; torna-se difícil gerenciar os requisitos; as expectativas sobre o produto se tornam muito altas, entre outros (CROWNE, 2002).

Devido às características das *startups*, focadas em entregas rápidas e falhas igualmente rápidas, o processo de gerenciamento do ciclo de vida mais utilizado é o processo ágil e suas variantes *lean* (GIARDINO *et al*., 2014). Mesmo assim, em geral elas não seguem rigorosamente processo algum, preferindo prototipar rapidamente de modo a testar um produto mínimo viável (MVP). Qualidade de *software* não é uma de suas grandes prioridades (GIARDINO *et al*., 2014) e as consequências aparecem posteriormente, quando os problemas no desenvolvimento gerados pela ausência de engenharia podem chegar a inviabilizar a evolução do produto e o crescimento da empresa (BERG *et al*., 2018), ou, até mesmo, levá-la ao fracasso. Processos com menos cerimônia e mais reativos funcionam no início, mas tendem a ser insuficientes à medida que o produto ganha mercado e necessita de robustez, escalabilidade, *performance* e foco no consumo de energia (BERG *et al*., 2018).

Não queremos, contudo, dizer que se deva adotar processos burocráticos e cheios de cerimônia, mas devemos, sim, buscar a implementação da filosofia ágil, mesclada com a visão da perenidade da organização.

O que se percebe é que aspectos relacionados à engenharia de *software*, em geral, são negligenciados pelas *startups* no início do seu desenvolvimento, com destaque para os processos relacionados a requisitos, uma vez que o foco dessa fase inicial é ganhar rapidamente o mercado. Os requisitos acabam sendo os mais negligenciados, o que é um aspecto contraditório, uma vez que se ganha o mercado com um produto que atende justamente aos anseios desse mercado, ou seja, que atende aos requisitos.

Ao crescer e ganhar em escala, a *startup* sente os efeitos negativos da falta de engenharia de *software* do início e então precisa se voltar para dentro e organizar a casa. O primeiro passo geralmente é começar a implementar métodos ágeis com mais seriedade, focando aspectos antes negligenciados, como os requisitos, por exemplo. Nesses casos, é comum a utilização do SCRUM e das histórias de usuário.

Saiba mais

Para conhecer mais sobre como as *startups* têm lidado com as questões de engenharia de *software*, leia o artigo de Giardino *et al.* (2016). Nele os autores mapeiam o estado das *startups* em um modelo denominado *Greenfield Startup Model*, apontando as lacunas que existem na sua criação e seus efeitos sobre a *startup*.

2 Identificação dos elementos da história do usuário

Quando os métodos ágeis surgiram, no final da década de 1990, trouxeram uma forma diferente de pensar tanto a organização do time de desenvolvimento quanto a organização dos requisitos, que passaram a ser tratados como histórias de usuário. O conceito de história de usuário foi criado por Kent Beck, na metodologia ágil XP (*eXtreme Programming*).

Saiba mais

A definição original de história do usuário, dada por Kent Beck e Martin Fowler é: "A história é a unidade de funcionalidade em um projeto XP. Nós demonstramos o progresso entregando código integrado e testado, que implementa uma história. Uma história deveria ser compreendida pelos clientes, testável pelo desenvolvedor, de valor para o cliente e pequena o suficiente para que os programadores possam construir uma meia dúzia delas em uma iteração."

Fonte: Leffingwell (2011).

De acordo com Patton (2014), "Histórias foram assim nomeadas devido a como é esperado que elas sejam usadas, não devido à forma como você está tentando escrevê-las". Em essência, uma história do usuário é um cenário de uso de um produto por um tipo específico de usuário. Mais importante do que a história registrada em si é a comunicação com o cliente para chegar a um entendimento comum. No caso do XP, a história é escrita pelo próprio cliente. No caso do SCRUM isso é feito pelo PO, que é quem representa os interesses dos clientes e usuários.

Roy Jeffrey, um dos criadores do XP, estabeleceu os 3 Cs da história de usuário: **C**artão, **C**onversa e **C**onfirmação. Vamos ver cada um deles a seguir.

Cartão (*Card*)

Com o passar do tempo e o uso nas organizações, as histórias de usuário passaram a ter um formato mais padronizado, com afirmações escritas em primeira pessoa, contendo o papel (quem), a ação (o que) e o resultado da ação ou o valor de negócio da história (porque), conforme ilustrado na Figura 3.

Essa forma é chamada de **expressão da história do usuário** ou **voz do usuário** (LEFFINGWELL, 2011). Alguns denominam também de **cartão (*card*) da história**, uma alusão à sua escrita simples em cartões ou notas adesivas tipo Post-it®. Embora esses meios físicos possam ser usados, também é comum o uso de ferramentas de *software* para essa finalidade.

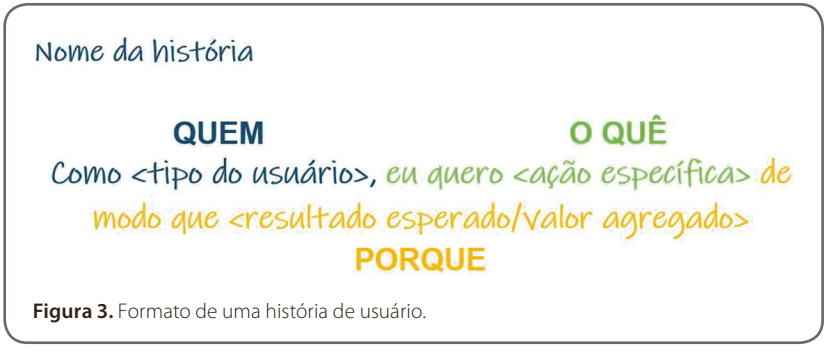

Figura 3. Formato de uma história de usuário.

Podemos entender que o **papel** (QUEM) representa quem se beneficiará da funcionalidade, e não quem está solicitando a funcionalidade, uma vez que podem ser *stakeholders* diferentes. A **atividade** (O QUÊ) descreve o que será a funcionalidade (ou seja, podemos entender como sendo o requisito funcional). O **resultado** (PORQUE) ou valor agregado, traduz a perspectiva da utilidade da funcionalidade no contexto de utilização, ou seja, qual é o seu valor de negócio. Representamos, portanto, três elementos: quem (papel), o quê (atividade), para quê (valor).

Exemplo

Vamos acompanhar o exemplo de um sistema de *check-in* de uma companhia aérea:

- *Como Passageiro, eu quero fazer meu check-in via celular, para não perder tempo na fila do aeroporto.*
- *Como Passageiro VIP, eu quero poder escolher assentos preferenciais, para viajar mais confortável.*
- *Como Atendente do Check-in, eu quero emitir etiquetas de bagagem, para despachar as malas dos passageiros.*
- *Como Atendente do Embarque, eu quero ler os cartões de embarque, para liberar o acesso dos passageiros ao avião*

Precisamos ficar atentos sobre os elementos da história de usuário. Você realmente deve se preocupar com o **quem**, ou seja, identifique quem serão os usuários e os clientes (pois, às vezes, não são a mesma pessoa). Não use o termo genérico usuário. Pense nos diversos perfis que podem ser usuários do produto. Se achar conveniente, faça um mapeamento de personas, para poder identificar os diversos perfis que irão interagir com o *software* (PATTON, 2014).

Patton ainda fala sobre o que realmente deve ser tratado na porção **o quê** da história. Ele recomenda que se fale sobre o que o *software* deve fazer, mesmo aquilo que aparentemente o usuário não percebe, como, por exemplo, obter a autorização da operadora do cartão de crédito para que uma compra seja realizada.

Finalmente, Patton (2014) nos diz que é importante falar sobre o **porquê**, pois existem muitas coisas escondidas atrás dos porquês. Continue escavando para entender por que é importante para o usuário, para a gerência do usuário, para o negócio.

O cartão pode ser compreendido como uma espécie de *token* que sumariza uma necessidade, mas os detalhes ainda serão definidos. Não podemos assumir que apenas o cartão irá conter tudo o que é necessário saber sobre a história para que ela possa ser implementada (LEFFINGWELL, 2011). Os detalhes virão na (C)onversa e nos (C)ritérios de aceitação.

É a partir das histórias do usuário que as tarefas serão derivadas, como, por exemplo, codificar a história, testar a história, documentar o *help* do usuário e assim por diante.

Conversa (*Conversation*)

Se voltarmos aos valores que embasam os métodos ágeis, vamos nos lembrar que um deles nos diz que devemos nos preocupar mais com a colaboração com o cliente do que com a negociação de contratos. Isso nos remete ao princípio fundamental da engenharia de requisitos: a comunicação. Nada substitui uma boa conversa (*conversation*) para que os requisitos sejam discutidos, interpretados, compreendidos. O objetivo da história de usuário é ser, justamente, o ponto de partida dessa conversa.

O que se espera dos *stakeholders* em ambientes ágeis é o seguinte (LEFFINGWEL, 2011):

- As decisões precisam ser tomadas pelos *stakeholders* no momento adequado em que são necessárias, especialmente as referentes ao escopo e às prioridades.
- A participação ativa dos *stakeholders* é necessária.
- As equipes precisam ter uma visão organizacional e enxergar a necessidade de integração com outros sistemas.
- Profissionais de produção e suporte precisam ser envolvidos desde o início.
- Profissionais que atuarão na manutenção do produto (caso seja um time diferente do time de desenvolvimento) deverão ser envolvidos também no início.

Aqui vale uma ressalva quanto aos que usam SCRUM. No SCRUM, o papel do PO é atuar na definição dos itens do *backlog,* bem como na sua priorização, mas isto não quer dizer que ele deva ser a única fonte de informação para a equipe de desenvolvimentos. Ele representa a voz de todos os demais *stakeholders*, e deve ser garantido que assim o seja.

Confirmação ou Critérios de aceitação (*Confirmation* ou *Acceptance Criteria*)

O "C" da confirmação (*confirmation*) se refere aos critérios de aceitação, ou seja, critérios complementares que ajudam a esclarecer como será validada a história. Podemos entender que são uma espécie de complemento da especificação da história, uma vez que a simplicidade do cartão não é suficiente para registrar tudo o que uma história significa. Alguns defendem, inclusive, que eles sejam escritos no verso do cartão da história, assumindo, claro, que um cartão real esteja sendo usado.

Critérios de aceitação não devem ser confundidos com testes funcionais ou testes unitários. Os casos de teste em si serão mais aprofundados e irão prever os vários tipos de caminhos dentro de um código. Alguns chamam esses critérios de testes de aceitação da história (LEFFINGWEL, 2011).

Os critérios de aceitação podem expressar fluxos alternativos que surgem ao longo da conversa com o usuário, bem como limites que estabelecem alguma fronteira.

Exemplo

Vamos acompanhar o exemplo da seguinte história de usuário de um sistema de *check-in* de uma companhia aérea:

- *Como Passageiro, eu quero fazer meu check-in via celular, para não perder tempo na fila do aeroporto.*

Os critérios de aceitação seriam:

- *Apresentar os assentos para seleção, destacando os assentos vazios*
- *Destacar os assentos especiais, permitindo compra na hora do check-in*
- *Apresentar a possibilidade de compra de bagagem extra no momento do check-in*

Boas histórias de usuário

Grande parte do esforço da equipe de desenvolvimento é empregado em compreender as histórias de usuário e em escrever critérios de aceitação para que elas possam ser testadas posteriormente. Algumas dicas para realizar adequadamente essa atividade foram projetadas por Bill Wake, que criou o acrônimo em inglês INVEST (LEFFINGWELL, 2011), conforme ilustra a Figura 4. Vamos ver a explicação de cada um desses itens na sequência.

Figura 4. Dicas para uma boa história de usuário.

Fonte: Adaptada de Leffingwell (2011).

Uma história de usuário deve ser **independente (*independent*)**, ou seja, ela pode ser identificada, discutida, implementada e até mesmo liberada para uso de forma independente de outras histórias. Este é um dos itens mais complexos, pois envolve o nível de granularidade da história de usuário. Se a história precisa ser independente, é preciso que seja possível liberá-la para uso também de forma independente dentro do prazo de uma *sprint*.

Saiba mais

Quando uma história do usuário é muito grande, ela é chamada de **épico**. Um épico pode ser dividido em duas ou mais histórias (COHN, 2004). Não há problema em ter um épico no *backlog* de produto. No entanto, para ser priorizado para fazer parte de uma *sprint*, ele deverá ser particionado em histórias independentes.

Toda história do usuário deve ser **negociável (*negotiable*)**. Isso quer dizer que ela não é um contrato imutável. O usuário pode mudar de ideia e a história poderá ser adaptada. Ela é mais uma forma de melhorar a comunicação entre o negócio e a equipe de desenvolvimento.

Um dos princípios básicos que permeiam os métodos ágeis é a entrega do maior valor possível para o cliente, dentro das limitações de tempo da *sprint*. Isso quer dizer que cada história deve ser **valiosa (*valuable*)**, ou seja, ela deve agregar valor para alguém. Se ela não agrega valor, ela não deveria estar alocada à *sprint*.

Uma história precisa ser **estimável (*estimable*)**. Ser estimável quer dizer que é possível estabelecer uma estimativa da complexidade da história para, no mínimo, saber se ela é compatível com a duração da *sprint* ou se precisa ser dividida. Não é nosso objetivo neste capítulo detalhar procedimentos de estimativa. O que importa para garantir que temos uma boa história é que a sua complexidade possa ser estimada e se possa alocá-la com confiança à *sprint*. Ao discutir com o time a complexidade de uma história, com vistas a estimá-la, podem vir à tona complexidades e riscos que não haviam sido anteriormente percebidos.

Uma história tem que ser **pequena (*small*)** o suficiente para caber em uma única *sprint*. A mentalidade ágil nos diz que o importante é liberar a história para o usuário em intervalos curtos (*sprints*) para que ela possa ser experimentada pelo usuário, de modo que a equipe de desenvolvimento possa receber rapidamente um *feedback*. Quando uma história é grande demais, ela é considerada um épico e deve ser particionada em histórias menores, que podem ser implementadas em uma *sprint*.

Toda história deve ser **testável (*testable*)**. A ideia é que uma história não deve entrar em uma *sprint* se ela não puder sair e, para sair, ela deve estar testada. Uma história só está pronta para o desenvolvimento se conhecermos os seus critérios de avaliação. Se não soubermos como avaliar uma história, ela não está pronta para ser implementada.

Mapeamento das histórias de usuário

Ao dar início ao mapeamento das histórias de usuário, certamente você vai se deparar com os diversos *stakeholders* que estão envolvidos em um projeto, cada um contribuindo com a sua visão sobre o projeto. Neste momento é possível que você perceba que apenas uma frase, escrita em um cartão ou Post-it®, contendo quem, o que e como não será suficiente.

Patton (2014) usa uma metáfora excelente para essas situações: histórias de usuário, escritas em cartões, são como os antigos cartões de livros das bibliotecas físicas — eles apresentam informações básicas sobre a obra, que permitem que possamos selecioná-la e localizá-la, mas o conteúdo está em outro lugar, ou seja, nos livros.

Haverá casos em que um cartão ou um Post-it® darão conta do recado, mas isto é raro. Na maioria dos casos, precisamos mais do que isso, especialmente porque as regras do negócio raramente são simples.

É bastante comum, atualmente, que as empresas estejam estruturadas no formato de grandes espaços abertos onde suas equipes sentam lado a lado, facilitando a comunicação e a colaboração. Tem sido também comum que as paredes dessas empresas sejam extensivamente utilizadas para expressar histórias, planejamentos, riscos, anotações. Geralmente são paredes de vidro ou paredes revestidas de materiais nos quais se pode escrever e apagar, como grandes quadros brancos. Nesses espaços reside o mapeamento das histórias de usuário. Graças à facilidade da tecnologia, uma foto, ou um vídeo, são suficientes para se guardar seu conteúdo para uso posterior. Esses registros em outros meios, além das notas adesivas e dos rascunhos nas paredes, são importantes para que se resgate o percurso posteriormente, quando necessário.

As duas principais razões pelas quais o mapeamento de histórias funciona, de acordo com Patton (2014), são estas: ele estabelece um entendimento comum por meio de palavras e figuras; e ele força a conversa na direção não apenas do que será construído, mas de quem vai utilizar o que será construído.

De acordo com Patton (2014), os seguintes passos devem ser utilizados para construir a história de usuário:

- **Delimite o problema:** para quem o produto está sendo desenvolvido e por que está sendo desenvolvido.
- **Mapeie a visão geral:** faça um mapa que seja abrangente em largura, mas raso em profundidade. Mapeie o mundo como ele é hoje, com as dores e alegrias do usuário.
- **Explore:** aprofunde, entendendo como as outras pessoas fazem a mesma coisa.
- **Gere uma estratégia de entrega:** sempre existe muita coisa para ser construída, então, priorize o que agrega mais valor.
- **Gere uma estratégia de aprendizagem:** construa o produto mínimo viável e teste com o público-alvo para aprender.
- **Gere uma estratégia de desenvolvimento:** divida o que deverá ser desenvolvido em o que deverá ser desenvolvido agora e depois. Desenvolva o que oferece os maiores riscos primeiro.

3 Aplicação de histórias de usuário

Histórias de usuário diferem de outras abordagens tradicionais de requisitos, de acordo com Leffingwell (2011), em diversos aspectos, acompanhe:

- Elas não são especificações detalhadas de requisitos, mas sim expressões de intenção negociáveis.
- Elas são curtas, fáceis de ler e compreensíveis por *stakeholders*, desenvolvedores e usuários.
- Elas representam pequenos incrementos de funcionalidade de valor, que podem ser implementadas em poucos dias ou semanas.
- Elas são relativamente fáceis de estimar, de modo que o esforço para implementação possa ser rapidamente estimado.
- Elas não são escritas em documentos grandes e pesados, mas organizadas em listas que podem ser arrumadas e rearrumadas mais facilmente à medida que novas informações são descobertas.
- Elas não são detalhadas no início do projeto, mas são elaboradas à medida que são necessárias (*just-in-time*), evitando especificações prematuras, atrasos no desenvolvimento, longo inventário de requisitos e uma declaração da solução muito restritiva.
- Elas requerem pouca ou nenhuma manutenção e podem ser facilmente descartadas depois de implementadas.
- As histórias de usuário, e o código criado rapidamente a partir delas, servem como entradas para documentação, que então é desenvolvida incrementalmente também.

É comum que você escute de uma empresa ágil que utilizar a abordagem de casos de uso reduz a agilidade da empresa, pois traz excesso de burocracia. Na realidade, é preciso compreender que a especificação de casos de uso pode ser descrita na profundidade que a equipe achar necessária para entender claramente o que deve ser construído, nem mais, nem menos. Se os casos de uso forem excessivamente detalhados, e a equipe tem um bom domínio do negócio, então eles se tornarão realmente enfadonhos e é possível que não tenham mesmo utilidade. No entanto, o outro lado também é preocupante, ou seja, ter apenas uma história do usuário registrada em um Post-it® no nível mais alto de um épico não será útil para uma equipe que não conhece a fundo o domínio de negócio.

Saiba mais

Para conhecer mais sobre a escrita efetiva de casos de uso proposta por Alistair Cockburn, consulte o livro *Escrevendo Casos de Uso Eficazes — Um Guia Prático para Desenvolvedores de Software* (COCKBURN, 2005). Nesse livro o autor aborda como escrever casos de uso, nos diversos níveis de abstração, discutindo como aplicar cada elemento desse tipo de especificação de requisitos funcionais. Vale a pena a leitura.

Aplicações, limitações e complementações

Tanto casos de uso quanto histórias do usuário são excelentes para elicitar requisitos de sistemas cujo controle está na mão do usuário, ou seja, sistemas com muitas interações por meio de interfaces com o usuário. No entanto, para os sistemas cuja complexidade não resida na interação propriamente dita, essas duas ferramentas podem não se aplicar tão bem. Esse é o caso de sistemas intensivos em processamento, como rotinas *batch*, processamento de grandes volumes de dados (aplicações de *business intelligence*, *big data*), algoritmos de inteligência artificial etc. Esse também é o caso de sistemas embarcados ou de tempo real (WIEGERS; BEATTY, 2013). Para esses casos, outras abordagens podem ser mais adequadas, como diagramas de máquinas de estado, diagrama de atividades, diagrama de sequência, diagrama de componentes, pseudocódigo etc.

À medida que você for ganhando mais experiência com a especificação de requisitos, vai perceber que pode ainda combinar diversas técnicas e agregar ou eliminar informações na medida da sua necessidade e da equipe de desenvolvimento. Pode ser que, em algum momento, para detalhar um fluxo de um caso de uso ou uma história de usuário, seja necessário, por exemplo, desenhar um diagrama de atividades ou um diagrama de sequência. Isso não é um problema e também não significa que você está deixando de ser ágil porque usou um diagrama UML; significa que você encontrou utilidade em usar as ferramentas de forma complementar.

A mensagem que fica é que o usuário deseja uma funcionalidade que resolva uma dor que ele sente. Para isso, você deve lançar mão da ferramenta da engenharia de requisitos que seja mais adequada àquele contexto, podendo ser um diagrama, um texto, um vídeo ou o que mais a sua imaginação de analista de requisitos criar. Qualquer ferramenta que apoie o diálogo para a compreensão dos requisitos é útil para o desenvolvimento de melhores produtos de *software*.

Referências

BECK, K. *et al. Manifesto para desenvolvimento ágil de software*. [*S. l.*], 2001. Disponível em: https://agilemanifesto.org/iso/ptbr/manifesto.html. Acesso em: 18 mar. 2020.

BERG, V. *et al.* Software startup engineering: a systematic mapping study. *Journal of Systems and Software*, v.144, p. 255–274, 2018.

COCKBURN, A. *Escrevendo casos de uso eficazes*: um guia prático para desenvolvedores de software. Porto Alegre: Bookman, 2005.

COHN, M. *User stories applied for agile software development*. Boston: Pearson Education, 2004.

CROWNE, M. Why Software Product Startups fail and what to do about it. *In*: IEEE INTERNATIONAL ENGINEERING MANAGEMENT CONFERENCE, 2002. *Anais*… Cambridge: IEEE, 2002. p. 338–343.

GIARDINO, C. *et al.* What do we know about software development in startups?. *IEEE Software*, v. 31, n. 5, p. 28–32, 2014.

GIARDINO, C. *et al.* Software Development in Startup Companies: The Greenfield Startup Model. *IEEE Transactions on Software Engineering*, v. 42, n. 6, p. 585–604, 2016.

LEFFINGWELL, D. *Agile software requirements:* lean requirements practices for teams, programs, and the enterprise. Upper Saddle River: Pearson Education, 2011.

NEON RAIN INTERACTIVE. *Agile Scrum for web development*. Denver, 2020. Disponível em: https://www.neonrain.com/agile-Scrum-web-development/. Acesso em: 18 mar. 2020.

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

SCHWABER, K.; SUTHERLAND, J. *Guia do SCRUM*: um guia definitivo para o SCRUM: as regras do jogo. [*S. l.*], 2013. Disponível em: https://www.Scrumguides.org/docs/Scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf. Acesso em: 18 mar. 2020.

STELLMAN, A.; GREENE, J. *Head first agile*: a brain-friendly guide. Sebastopol: O´Reilly, 2017.

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

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