ESPECIFICAÇÃO DE SISTEMAS DE INFORMAÇÃO

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Priorização de requisitos de *software*

Whesslley Rimar Bezerra

OBJETIVOS DE APRENDIZAGEM

- Reconhecer a necessidade de priorizar os requisitos de *software*.
- Identificar os métodos e as técnicas para a priorização de requisitos.
- Aplicar métodos e técnicas para a priorização de requisitos.

Introdução

Na construção de um *software*, o desenvolvimento dos requisitos não acontece de uma única vez. Deve haver um planejamento equilibrado no qual, por meio de técnicas e métodos específicos, os requisitos sejam organizados e, sempre que necessário, priorizados.

Você deve dar muita atenção aos requisitos, pois eles são os principais indicadores de como o produto deve ser construído e de como o projeto como um todo deve ser conduzido. Além disso, as funcionalidades de um sistema são elaboradas com base no que está especificado nos requisitos do projeto.

Neste capítulo, você vai ver por que é importante priorizar requisitos de *software*. Você também vai conhecer algumas técnicas e ferramentas elaboradas para esse fim. Além disso, vai ler sobre a implementação desses recursos.

Por que priorizar requisitos?

Não há uma definição única para a priorização de requisitos. Ela pode estar ligada a múltiplos significados que se complementam e evidenciam a sua importância. Entre esses significados, considere os listados a seguir.

- Organização de requisitos, de modo a facilitar tomadas de decisão importantes dos *stakeholders* (partes interessadas no projeto).
- Ordem ideal, mas não exclusiva, para que os requisitos de um sistema sejam desenvolvidos, a fim de que o *software* e suas possíveis versões sejam construídas em tempo hábil, atendendo satisfatoriamente às necessidades dos clientes.
- Entregas de funcionalidades de um sistema feitas em uma sequência lógica e clara, favorecendo a entrega de valor ao cliente desde o início do projeto.

Há outras definições da priorização de requisitos na literatura. Vazquez e Simões (2016), por exemplo, afirmam que a priorização de requisitos está ligada à atribuição de importância relativa aos requisitos, maximizando o valor que será entregue pelo projeto. Assim, são tratados em primeiro lugar os aspectos mais relevantes para o desenvolvimento do *software*. Isso significa que todos os requisitos considerados mais urgentes e que agregam maior valor ao produto (*software*), ao cliente e aos demais *stakeholders* devem ser desenvolvidos em uma ordem coerente com o cenário.

Entretanto, a priorização de requisitos não é feita de maneira aleatória. É necessário um processo que disponha os requisitos em uma sequência, de modo a aumentar o valor do produto a cada momento do seu desenvolvimento. Contudo, algumas dúvidas podem surgir nesse contexto, como você pode ver a seguir.

- Como definir quais requisitos devem ser entregues primeiro?
- Quais funcionalidades agregam maior valor?
- O produto está realmente agregando valor ao cliente?
- Quais requisitos devem ser entregues em cada *release* do produto?

Esses são apenas exemplos de perguntas que podem ser feitas pela equipe envolvida no projeto de *software*. Outras questões podem surgir, o que depende do nicho de mercado e do tipo de *software* que está sendo desenvolvido pelo time. Além disso, dependendo do nível de maturidade da equipe, os

questionamentos podem gerar insegurança. Logo, a gestão do projeto deve estar preparada para trabalhar com a equipe de maneira que todos, ainda que desafiados, sintam-se confortáveis e motivados com o desenvolvimento do escopo do projeto.

Alguns critérios, entretanto, devem ser considerados nesse processo. No Quadro 1, a seguir, veja quais são esses critérios.

Quadro 1. Critérios para a priorização de requisitos

Critérios de priorização	Como priorizar os requisitos?	Quando utilizar?
Valor de negócio (benefício)	Análise de custo–benefício	Em projetos incrementais e de baixo orçamento
Risco	Análise de riscos de falhas no projeto	Para aumentar a probabilidade de sucesso do projeto
Custo	Análise de custos de implementação	Em requisitos que podem ter suas prioridades alteradas de acordo com o seu custo
Perda	Análise de perdas que podem surgir devido à não implementação	Quando houver risco de perder espaço para a concorrência, ou quando a empresa for afetada por alguma regulamentação
Dependência de outros requisitos	Análise da dependência que há entre requisitos que agregam muito valor e requisitos que agregam pouco valor	Em projetos incrementais e de baixo orçamento
Estabilidade	Análise da estabilidade dos requisitos, ou seja, verificação do quão valiosos ou úteis os requisitos são para os *stakeholders*	Quando ocorrer conflito de interesses entre os *stakeholders* do projeto
Sensibilidade temporal	Análise da sensibilidade de tempo	Quando houver oportunidades para o projeto, como períodos sazonais ou outra situação que possa evidenciar a necessidade dos requisitos

Fonte: Adaptado de Vazquez e Simões (2016).

A seguir, veja a descrição de cada um dos critérios elencados no Quadro 1.

- **Valor de negócio (benefício):** as partes interessadas no projeto devem analisar qual é a relação custo–benefício que a implementação dos requisitos vai proporcionar. Se o resultado da análise for positivo, o requisito será priorizado; caso contrário, não. Esse critério geralmente é utilizado em projetos em que o orçamento é muito limitado.
- **Risco:** esse critério se refere à análise de riscos do projeto, ou seja, são verificadas as principais ameaças. Assim, os requisitos que se enquadram nesse critério são priorizados para que ampliem as chances de sucesso do projeto como um todo.
- **Custo:** os custos de um requisito podem impactar diretamente a sua implementação. Se o custo de determinado requisito for alto e ele tiver menos relevância do que outros requisitos, é possível que permaneça aguardando na fila para ser implementado em outro momento, ou até mesmo que não seja levado adiante no projeto.
- **Perda:** um dos critérios que podem influenciar muito a priorização dos requisitos é a análise de perdas. Por meio dessa análise, é possível identificar quais são as possíveis perdas que a empresa vai ter por tomar a decisão de não implementar determinado requisito.
- **Dependência de outros requisitos:** esse critério é útil para avaliar se determinados requisitos devem ser implementados ainda que não agreguem tanto valor, uma vez que outros requisitos com maior valor agregado dependem deles para funcionar adequadamente.
- **Estabilidade:** esse critério considera quão estáveis são os requisitos mediante os conflitos de interesses que podem existir entre os *stakeholders*. Alguns *stakeholders* podem considerar os requisitos pouco úteis, enquanto outros os consideram muito valiosos.
- **Sensibilidade temporal:** alguns requisitos podem ser priorizados em função da sazonalidade ou por alterações de escopo decorrentes de eventos diversos.

A priorização de requisitos é considerada exaustiva por muitas empresas que trabalham com desenvolvimento de *software*. Contudo, para que um projeto seja bem-sucedido, ele deve passar por um levantamento adequado e detalhado dos seus requisitos. Isso significa que o planejamento de um *software* deve ser bem estruturado, a fim de que o resultado atenda a todas as necessidades especificadas pelo cliente.

Dessa forma, não basta saber quais são os requisitos necessários para o desenvolvimento do sistema. Se os requisitos não forem organizados por prioridade, o projeto pode sofrer com conflitos de interesse e, eventualmente, fracassar.

No processo de elicitação de requisitos, muitas mudanças podem ocorrer no escopo do projeto. Tais mudanças podem impactar diretamente a maneira como os requisitos são definidos e organizados. Logo, mantê-los em uma ordem coerente, que faça sentido para as partes interessadas e para a viabilidade do produto, pode não ser uma tarefa fácil, exigindo bastante esforço da equipe envolvida. Entretanto, essa tarefa é extremamente necessária para o sucesso do *software*.

Machado (2018), além de reforçar os critérios que você estudou anteriormente, adiciona novos fatores determinantes para a priorização de requisitos. Veja a seguir.

- **Relevância do requisito para o negócio:** esse item geralmente é definido pelo cliente e pela equipe técnica envolvida, que inclui, por exemplo, os analistas de sistemas. Avalia-se se o valor que o requisito agrega ao sistema em desenvolvimento é relevante para o ramo de atuação do cliente.
- **Relevância do requisito para a arquitetura do *software*:** esse fator é definido pelo profissional arquiteto de *software*. A ideia é verificar se o requisito é importante para a estrutura do *software* como um todo.
- **Riscos de projeto que devem ser mitigados:** requisitos que podem eliminar riscos (de natureza técnica ou não) devem ser priorizados, pois contribuem para manter o *software* competitivo no mercado, atendendo às necessidades dos clientes.
- **Dependências entre requisitos:** alguns requisitos podem compartilhar suas funcionalidades ou utilizar outros requisitos para funcionar corretamente. Requisitos que se enquadram nessa descrição podem ser priorizados se a equipe assim decidir.
- **Questões estratégicas:** outras situações podem influenciar a priorização de um requisito, como mudanças no escopo do projeto, interferências externas do mercado em que a empresa atua, mudanças na legislação e solicitações dos *stakeholders*.

Métodos e técnicas de priorização de requisitos

Alguns *frameworks* e técnicas podem ser utilizados para a priorização de requisitos. Você vai conhecer alguns deles adiante. No entanto, tenha em mente que não há uma técnica de priorização melhor do que as outras. O que há é a melhor técnica para cada cenário e cada momento específico. Assim, se uma técnica for utilizada em determinada situação e gerar bons resultados, ela não deve ser utilizada como referência exclusiva em outras situações. A equipe deve observar o contexto como um todo e verificar qual é a melhor abordagem para atender às suas necessidades, e isso pode ser alterado de projeto para projeto.

Modelo Kano

O modelo Kano foi idealizado por um pesquisador japonês chamado Noriaki Kano em meados da década de 1980. Ele é um método que visa a gerir a qualidade de produtos e serviços. Para isso, investiga como atender às necessidades do cliente a fim de garantir a sua satisfação.

O modelo Kano é representado em uma forma gráfica chamada "diagrama de Kano". Atualmente, ele é definido a partir de três categorias principais. A seguir, veja quais são essas categorias.

- **Atrativos (funções empolgantes):** esses requisitos podem ser considerados na implementação do *software*, porém não são essenciais. Como o próprio nome indica, estão em jogo requisitos que podem ser muito atrativos para o cliente se forem incluídos no *software*. Todavia, se tais requisitos não forem adicionados, o cliente muito provavelmente não perceberá sua ausência, pois não são vitais ao funcionamento do *software*. O ponto positivo desses requisitos é que eles aumentam consideravelmente a satisfação do cliente em relação ao sistema, já que geram grande empolgação por simplesmente estarem disponíveis para uso.

- **Unidimensionais ou de desempenho (*performance*):** esses requisitos melhoram o desempenho do sistema e, consequentemente, aumentam o engajamento e a satisfação do cliente em relação ao *software*. Em muitos casos, esses requisitos podem ser um diferencial significativo do produto em relação aos concorrentes.
- **Obrigatórios (funções básicas):** esses requisitos são obrigatórios para o *software* que está sendo desenvolvido. Portanto, invariavelmente devem ser priorizados pela equipe. É importante ressaltar que o desenvolvimento desses atributos/funcionalidades no sistema não significa necessariamente que o cliente estará satisfeito com o produto. Contudo, se esses requisitos não forem adicionados ao sistema, o cliente certamente ficará descontente, pois o *software* provavelmente não atenderá a todas as suas necessidades.

A equipe técnica organiza todos os requisitos com potencial de desenvolvimento seguindo alguns critérios que são avaliados concomitantemente. São eles: potencial de satisfação (encanto do cliente) e investimento para implementação. Na Figura 1, a seguir, observe como o modelo Kano é representado.

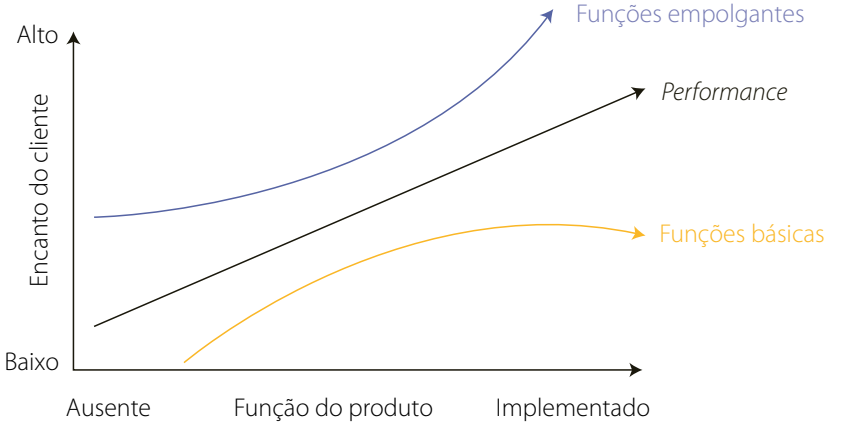

Figura 1. Diagrama de Kano.
Fonte: Adaptada de Carvalho (2019a).

Votação

Vazquez e Simões (2016) destacam que a priorização de requisitos pode ser feita por meio de uma votação. Essa é uma das maneiras mais simples de organizar requisitos por ordem de prioridade. De modo geral, os requisitos são colocados à mesa e os recursos são alocados a eles. Esses recursos podem ser: dinheiro, votos ou quaisquer outros elementos que auxiliem na decisão de quais requisitos devem ser priorizados. Os requisitos que tiverem o maior número de recursos acumulados serão priorizados. Essa técnica, contudo, não é muito utilizada na área de desenvolvimento de *software*.

MoSCoW

Muito utilizada em conjunto com metodologias ágeis, essa técnica de priorização é nomeada por um acrônimo formado por quatro categorias:

- *must have* (tenho que fazer);
- *should have* (devo fazer);
- *could have* (poderia fazer);
- *won't have* (não vou fazer).

O objetivo principal da técnica MoSCoW é encontrar um equilíbrio entre os *stakeholders* do projeto em relação aos requisitos do produto. Nessa técnica, os requisitos que recebem o valor *must have* têm mais importância do que os que recebem os valores *should have* e *could have*. Os requisitos classificados com o valor *won't have* são deixados de lado para serem implementados em outros momentos.

Os requisitos classificados como *must have* são os de maior prioridade. Portanto, devem obrigatoriamente ser implementados no *software*. Nessa categoria, enquadram-se aqueles requisitos que são essenciais para o funcionamento do *software* e que não podem ser deixados para depois.

Os requisitos classificados como *should have* são muito importantes, porém não são essenciais para o funcionamento. Ou seja, o sistema pode funcionar sem eles, porém desenvolvê-los aumentará o valor agregado do *software*. Por serem "não essenciais", esses requisitos podem ser implementados em versões futuras do *software* sem gerar impacto negativo.

Os requisitos classificados como *could have* são, em geral, opcionais e não impactam o andamento do projeto ou o funcionamento do *software*. Eles agregam valor, mas são como "um algo a mais" no projeto.

Os requisitos classificados como *won't have* são requisitos que não serão implementados ou desenvolvidos no momento, sendo deixados para o futuro. Entretanto, embora esses requisitos não sejam prioritários, eles podem mudar de *status* em algum momento. Afinal, no desenvolvimento de *software*, tanto os requisitos quanto as prioridades podem ser alterados a qualquer tempo.

Fique atento

Na prática, todos os modelos que você estudou são parecidos e têm um mesmo objetivo: priorizar requisitos agregando valor contínuo ao cliente. Logo, a escolha depende mais da afinidade da equipe com a técnica do que da ferramenta em si. Além disso, a equipe de desenvolvimento pode criar o seu próprio método de priorização de requisitos.

Onde aplicar os métodos e técnicas de priorização de requisitos?

Um dos grandes desafios enfrentados pelas empresas de desenvolvimento de *software* em todo o mundo consiste em entender como os requisitos de um sistema devem ser priorizados. Essa dificuldade não está ligada às ferramentas em si, que em geral são de simples entendimento. O problema começa quando as ferramentas precisam ser colocadas em prática. Em algumas situações, é necessária a mudança de mentalidade da equipe como um todo para que a atividade de priorização não seja feita sem critérios, e sim com base em questões técnicas, operacionais e de negócios.

As técnicas que você estudou anteriormente são muito conhecidas. Elas são implementadas em diversos nichos de mercado. Em todas elas, sabe-se que as intervenções dos *stakeholders* podem ser constantes e, em muitos casos, determinantes para a priorização ou não de um requisito. Nesse contexto, um questionamento que pode surgir é: onde aplicar os métodos e técnicas de priorização de requisitos?

A resposta a essa pergunta é simples: em praticamente qualquer área. Como o foco deste capítulo é o desenvolvimento de *software*, a seguir você vai conhecer mais uma técnica, o Scrum, que é utilizado em conjunto com metodologias de desenvolvimento ágil.

O Scrum, um dos *frameworks* mais conhecidos do desenvolvimento ágil, com seus papéis e ciclos de desenvolvimento bem definidos, constantemente atua com a priorização de requisitos. A seguir, veja quais são os papéis desempenhados no Scrum.

- ***Product Owner* (PO):** é o dono do produto.
- **Scrum *master*:** é o responsável por disseminar a cultura Scrum para todos os integrantes da equipe, mantendo-os alinhados.
- **Time de desenvolvimento:** é a equipe responsável pela execução do projeto.

O PO é a pessoa mais influente do projeto. Em regra, ele é o responsável por conhecer todo o escopo do projeto. Esse papel pode ser exercido por um dos *stakeholders* que conheça com profundidade o produto e as necessidades que ele precisa atender. Assim, o PO, o Scrum *master* e o time de desenvolvimento se reúnem para o planejamento da *sprint* (ciclo de desenvolvimento que dura de 2 a 4 semanas). Nessa reunião, o PO fica responsável por apresentar o escopo do projeto, e o Scrum *master* se responsabiliza por conduzir junto à equipe o *planning poker*, uma técnica utilizada no Scrum para definir a estimativa de esforço e a priorização do *backlog* (lista de tarefas).

Na prática, essa estratégia consiste em uma sequência de valores distribuídos em cartas (como de um baralho) que são entregues aos membros da equipe de desenvolvimento. Carvalho (2019b) pontua que os valores seguem a sequência de Fibonacci, com pequenas modificações: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40 e 100.

Na Figura 2, veja como se dá, na prática, a implementação do fluxo do *planning poker* para que o time meça a quantidade de esforço que será destinada aos requisitos, podendo, a partir de então, priorizá-los.

Uma característica clara do *planning poker* é que a priorização de requisitos se baseia na disponibilidade dos membros da equipe (quantidade de demandas que cada integrante tem sob sua responsabilidade) e no domínio técnico existente para o desenvolvimento do requisito.

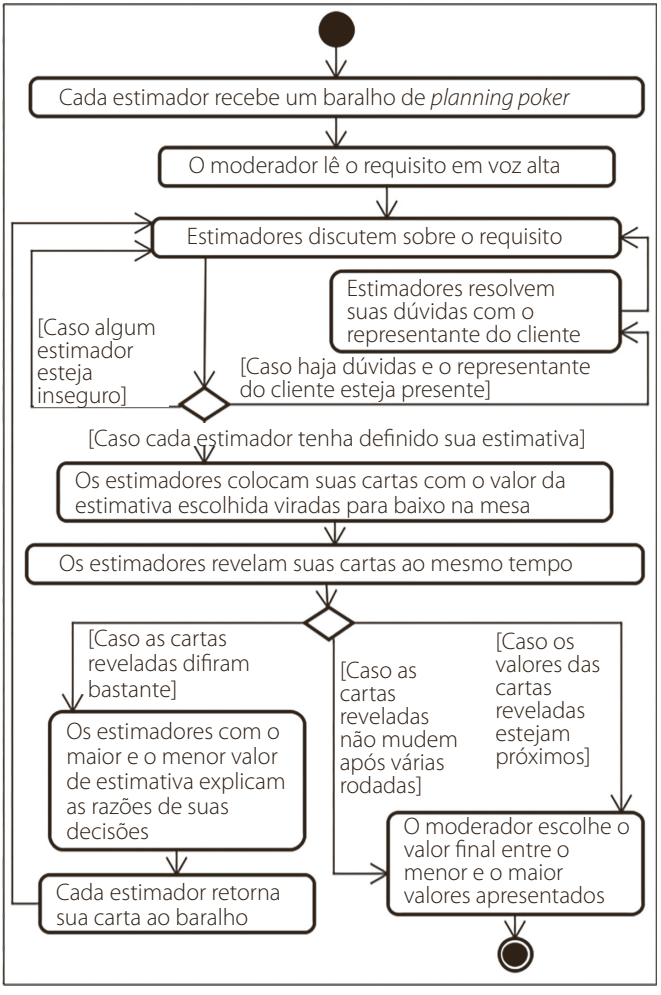

Figura 2. *Planning poker.*
Fonte: Tissot (2015, p. 32).

Aplicação do método MoSCoW

Para entender na prática o modelo MoSCoW, que você estudou anteriormente, considere a aplicação dele ao desenvolvimento do *software* descrito a seguir.

- ***Software*:** sistema de gestão de biblioteca.

- **Escopo do projeto:** construção de um sistema de gestão de biblioteca que ofereça os perfis: aluno, administrador e atendente. Além disso, de maneira geral, o sistema deve fazer a gestão de empréstimos de livros.
- ***Backlog* do sistema (simplificado):**
  - perfil de aluno — efetuar a reserva de um livro pela *web*, para não ter de ir até a biblioteca;
  - perfil de aluno — avaliar livros por estrelas (1 a 5) e adicionar comentários sobre livros lidos para que outros alunos possam ver;
  - perfil de administrador — visualizar livros (disponíveis para reserva, devolvidos dentro do prazo e não devolvidos);
  - perfil de administrador — solicitar empréstimos de livros entre bibliotecas para disponibilizar os volumes aos alunos;
  - perfil de atendente — consultar os dados dos alunos para contatá-los caso seja necessário;
  - perfil de atendente — consultar a localização de um livro para encontrá-lo rapidamente;
  - perfil de atendente — conversar instantaneamente com os alunos, por meio de ferramenta de *chat*.

Com o escopo definido, pergunta-se aos *stakeholders* como devem ser classificadas as suas preferências e necessidades, considerando as categorias do modelo MoSCoW (Quadro 2).

Quadro 2. Expectativas do cliente em relação ao *backlog* priorizadas com técnicas do modelo MoSCoW

Item do *backlog*	Expectativas do cliente (requisito detalhado)	MoSCoW
1	O perfil de aluno deve ter a possibilidade de fazer a reserva de um livro, sem impedimentos, via sistema *web*. Essa funcionalidade é considerada essencial.	*Must have*
2	O perfil de aluno deveria ter a possibilidade de fazer avaliações e adicionar comentários que possam ser visualizados por outros usuários do sistema.	*Should have*
3	O perfil de administrador deve ter a possibilidade de extrair relatórios com os livros disponíveis para reserva, devolvidos dentro e fora do prazo, ou ainda com os livros que não foram devolvidos.	*Must have*

(Continua)

(Continuação)

Item do *backlog*	Expectativas do cliente (requisito detalhado)	MoSCoW
4	O perfil de administrador poderia ter a possibilidade de solicitar livros a outras bibliotecas para disponibilizá-los aos alunos. Logo, o sistema poderia integrar diversas bibliotecas.	*Won't have*
5	O perfil de atendente deve ter a possibilidade de consultar os dados de contato dos alunos, a fim de se comunicar com eles sempre que necessário.	*Must have*
6	O perfil de atendente deve ter a possibilidade de identificar a localização de um livro para encontrá-lo com agilidade.	*Must have*
7	O perfil de atendente poderia ter acesso a uma ferramenta de *chat* para atender em tempo real às solicitações de livros provenientes dos alunos.	*Could have*

Como você viu no Quadro 2, os requisitos foram classificados com base na urgência e nas necessidades dos *stakeholders*. Os requisitos 1, 3, 5 e 6 foram classificados como *must have*, representando as funções essenciais do sistema de biblioteca. Por sua vez, o requisito 2 foi classificado como *should have*, ou seja, é uma funcionalidade muito desejada, mas não considerada essencial no primeiro momento; portanto, pode ser entregue posteriormente. Já o requisito 4 foi classificado como *won't have*, ou seja, não será implementado. Por fim, o requisito 7 foi classificado como *could have* e, apesar de ser uma funcionalidade que seria muito bem-vinda, não é urgente e pode ser implementada futuramente. Todavia, esse requisito pode representar um diferencial do *software* em relação a outros sistemas que oferecem as mesmas funcionalidades.

Como você viu ao longo deste capítulo, a priorização de requisitos é um processo completamente dinâmico, que pode passar por modificações constantes. Do mesmo modo, os requisitos também podem passar por alterações, assim como o próprio projeto pode sofrer mudanças em seu escopo. Portanto, sabendo desse dinamismo, a equipe deve conhecer a fundo todo o escopo do projeto e se manter organizada para que a construção do sistema ocorra sem grandes dificuldades.

Referências

CARVALHO, H. O framework de priorização: Modelo Kano. Vida de Produto, 12 set. 2019a. Disponível em: <https://vidadeproduto.com.br/framework-modelo-kano/#1_Features_basicas_minimos>. Acesso em: 18 set. 2020.

CARVALHO, T. O que é Planning Poker e como ele funciona com o Scrum. Voitto, 8 de nov. 2019b. Disponível em: https://www.voitto.com.br/blog/artigo/planning-poker. Acesso em: 22 set. 2020.

MACHADO, F. N. R. *Análise e Gestão de Requisitos de Software - Onde nascem os sistemas*. 3. ed. São Paulo: Érica, 2018. *E-book*.

TISSOT, A. A. Influência da revisão de atividades executadas para melhoria da acurácia na estimativa de software utilizando planning poker. 2015. 187 f. Dissertação (Mestrado em Engenharia de Software) - Programa de Pós-Graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná, Curitiba, 2015. Disponível em: http://riut.utfpr.edu.br/jspui/bitstream/1/1661/1/CT_PPGCA_M_Tissot,%20Andr%C3%A9%20Augusto_2015.pdf. Acesso em: 29 out. 2020.

VAZQUEZ, C. E.; SIMÕES, G. S. *Engenharia de Requisitos*: software orientado ao negócio. São Paulo: Brasport, 2016.

Leitura recomendada

COSTA, H. G. *Introdução ao método de análise hierárquica*: análise multicritério no auxílio à decisão. Niterói: H.G.C., 2002.

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*.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS