DESENVOLVIMENTO DE SOFTWARE COM METODOLOGIAS ÁGEIS

Fabrício Leonard Leopoldino

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Lean software development (LSD)

Objetivos de aprendizagem

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

- Explicar a filosofia *lean* da Toyota e suas origens.
- Discutir os princípios da metodologia *lean software development* (LSD).
- Demonstrar o uso da metodologia *lean* no desenvolvimento de *software*.

Introdução

Em todo o mundo, gestores estão mudando as formas de gerenciar as suas empresas, com o objetivo de sobreviver em um ambiente cada vez mais competitivo, desde a produção de vassouras até a produção de naves espaciais. Se observarmos ao nosso redor, poderemos perceber diversas mudanças, principalmente, nos grandes centros comerciais, onde antes eram abarrotados de produtos e hoje muitos têm poucos produtos em suas prateleiras ou em seus estoques locais e, em diversas situações, quando buscamos por um determinado produto, somos direcionados aos seus portais na internet. E, com isso, a forma de produzir *software* também vem mudando, pois esse também é um produto que precisa ser desenvolvido.

Com o objetivo de auxiliar nesse processo de mudança, equipes de desenvolvimento estão adotando metodologias que contribuem para o enfrentamento de imprevistos durante o processo de desenvolvimento de um *software*, auxiliando o processo de gestão de projetos, que estimula a inspeção e os ajustes frequentes; essas metodologias são conhecidas como ágeis. As metodologias ágeis estimulam as atividades de uma equipe de desenvolvimento, a comunicação, a auto-organização, a atenção focada no cliente e, o melhor de tudo, que cada entrega deve ter valor, e não apenas entregar por entregar.

Neste capítulo, estudaremos o sistema *lean* de produção Toyota, o seu histórico e as suas características, a metodologia *lean* aplicada ao desenvolvimento de *software*, os seus princípios básicos, como aplicar a metodologia *lean* no processo de desenvolvimento de *software* e a metodologia de treinamento para desenvolvimento de *software* Coding Dojo.

1 Sistema Toyota de Produção

Com a necessidade de busca por melhores resultados, a palavra redução vem sendo incluída no dicionário de muitas pessoas ao redor do mundo. É nesse ponto que o Sistema Toyota de Produção, criado entre 1947 e 1975, no qual a sua base consiste na absoluta eliminação do desperdício no processo de produção, tem dois pontos como itens essenciais ao sistema (OHNO, 1988).

Just-in-time consiste no processo em que somente as partes necessárias para a montagem correta de um produto são utilizadas, sem a necessidade de fazer estoque de tais partes. Como exemplo, vamos supor que queremos fazer um bolo e a receita pede apenas três ovos. Nessa situação, não é necessário comprar quatro ou mais ovos, sendo que o quarto ovo indica desperdício para a produção desse bolo. Mas leve em conta que para isso funcionar os três ovos devem estar perfeitos.

Autonomação consiste no processo de automação com um toque humano, no qual a máquina é acoplada a um dispositivo de parada automática, e quando algum problema é detectado, a máquina para de funcionar, evitando, dessa forma, a produção de produtos defeituosos. Esse sistema recebe o nome de *jidoka* (controle autônomo de prevenção de defeitos) (POPPENDIECK; POPPENDIECK, 2010).

O criador desse sistema foi o engenheiro de produção Taiichi Ohno e, segundo ele, o Sistema Toyota de produção não é apenas um sistema de produção, mas um sistema gerencial adaptado à era atual de mercados globais e de sistemas de computadores de informações de alto nível (OHNO, 1988).

No Sistema Toyota de Produção, a habilidade individual e o trabalho em equipe são bem valorizados. Buscando uma relação prática com os pilares do sistema *just-in-time* e da autonomação, podemos fazer uma analogia com uma equipe de desenvolvimento de *software* ágil, na qual a autonomação corresponde à habilidade e ao talento individual que cada desenvolvedor tem, ao passo que o *just-in-time* corresponde ao trabalho da equipe envolvida para estabelecer um objetivo comum preestabelecido.

Fique atento

Taiichi Ohno nasceu em 12 de fevereiro de 1912, em Dalian, China, e depois frequentou a Escola Técnica de Nagoya. Após a formatura, ele começou a trabalhar na Toyoda Spinning e Automatic Loom Works, uma das primeiras empresas da família Toyoda. Esta família realizou um concurso para criar um novo logotipo para a empresa e o *design* vencedor usou a palavra Toyota — com o "t" no lugar do "d" — assim nasceu a Toyota Motor Company. Em 1943, Ohno era o engenheiro de produção da Toyota Motor Company. Ele subiu rapidamente entre as fileiras da Toyota. Em 1949, tornou-se gerente de oficina de máquinas e progrediu como diretor em 1954, diretor gerente em 1964, diretor gerente sênior em 1970 e vice-presidente executivo em 1975 (OHNO, 1988).

No Sistema Toyota de Produção, o principal objetivo é reduzir custos e, consequentemente, menos custos geram maiores lucros ou podem ser compartilhados com os clientes, oferecendo produtos com melhores preços. Mas, para conseguir essa redução, é necessário um sistema de gestão que desenvolva a habilidade humana até a sua mais plena capacidade de produção para melhorar a criatividade e a eficácia na utilização das instalações e dos equipamentos, com o objetivo de eliminar todo o desperdício (OHNO, 1988).

Durante o desenvolvimento do Sistema Toyota de Produção, Taiichi Ohno identificou o que ele chamou de "os sete desperdícios" (KEMP, 2018), listados a seguir:

1. tempo de espera sem valor agregado;
2. superprodução — fazendo demais;
3. processamento excessivo ou adição de atividades sem valor;
4. transporte desnecessário;
5. movimentos ou movimentos excessivos;
6. inventário;
7. baixa qualidade e defeitos.

Saiba mais

Em setembro de 2006, o então diretor sênior de engenharia da Corbis (empresa fundada por Bill Gates), David J. Anderson, decide projetar um sistema *kanban* que substituiria a então abordagem existente para a atualização de aplicativos baseado no *kanban* criado pelo engenheiro japonês Taiichi Ohno (BERNARDO, 2014).

O Sistema Toyota de Produção é regido pelas quatro regras implícitas (MARTINS; LAUGENI, 2015) listadas a seguir.

- Todo trabalho deve ser altamente especificado no seu conteúdo, sequência, tempo e resultado.
- Toda relação cliente-fornecedor (interno e externo) deve ser direta, com um canal definido e claro para enviar pedidos e receber respostas.
- O fluxo de trabalho e processo para todos os produtos e serviços deve ser simples e direto.
- Qualquer melhoria deve ser feita pelo método científico, sob a coordenação de um orientador e no nível mais baixo da organização.

O Sistema Toyota de Produção passou a ser chamado de *lean production* (produção enxuta) e muitas empresas buscaram aplicar essa metodologia; algumas sem sucesso por acharem a metodologia contraintuitiva e outras com grande sucesso. Os princípios pregados pela metodologia *lean* (enxuta) foram adotados para diversas áreas, como as cadeias de fornecimento, o desenvolvimento de produtos e o desenvolvimento de *software*.

2 *Lean software development*

Uma metodologia de desenvolvimento de *software* é um conjunto de atividades que auxiliam no desenvolvimento de um *software*, em que o resultado dessa atividade é um produto que reflete a forma como o processo foi conduzido (KOSCIANSKI; SOARES, 2007). As equipes de desenvolvimento devem se preocupar em entregar um *software* de qualidade no menor tempo possível. Para Sommerville (2011), o processo de produção de *software* acontece de diferentes formas, seja para novos ou para os já em desenvolvimento, mas

sempre as fases de análise de requisitos, projeto, implementação e testes devem estar incluídas (Figura 1).

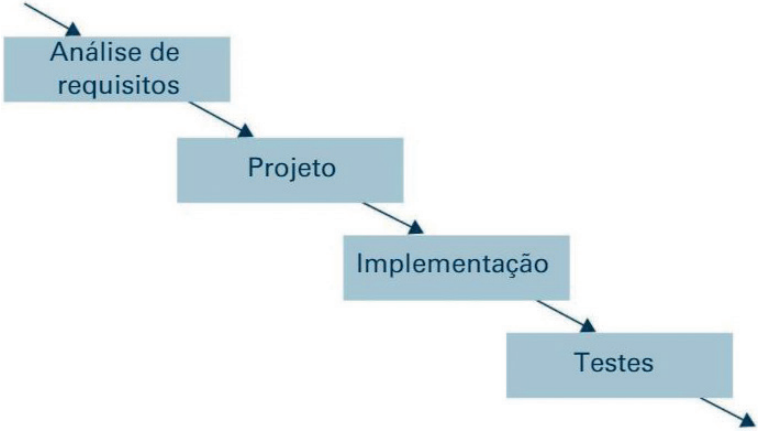

Figura 1. Fases tradicionais de desenvolvimento do ciclo de vida do *software*.
Fonte: Brookshear (2013, p. 267).

Uma metodologia de desenvolvimento de *software* ágil consiste na realização das atividades de levantamento de requisitos, planejamento, projeto, implementação e testes, de fabricação rápida por meio de uma série de incrementos de funcionalidades, interações de curtos espaços de tempo e entregas aprovadas pelo cliente. Esse tipo de metodologia é utilizado para produzir, rapidamente, *software* que entreguem valor ao cliente.

Quando se deseja utilizar métodos ágeis no desenvolvimento de um produto de *software*, a equipe deve ter em mente que o *software* deverá entregar um *software* de qualidade no menor tempo possível, reduzindo a burocracia por meio de poucos artefatos e de iterações de execução de atividade em pequenos espaços de tempo, sendo possível realizar as alterações solicitadas pelo cliente, tornando, dessa forma, um projeto leve.

Criado no Japão, o Sistema Toyota de Produção, que também pode ser conhecido como *lean manufacturing*, aplicado na produção de carros, surgiu logo após a Segunda Guerra Mundial na fábrica da empresa automobilística Toyota. Nessa época, a indústria japonesa tinha uma produtividade muito baixa e sofria com a falta de recursos, o que consequentemente a impedia de adotar o modelo de produção em massa (OHNO, 1988). A filosofia da metodologia

lean busca produzir melhor, com redução de recursos e tempo, incorporando maior valor ao cliente.

O termo desenvolvimento de *software lean* originou-se em um livro com o mesmo nome, escrito por Mary Poppendieck e Tom Poppendieck, em 2003. O livro reafirma os princípios enxutos tradicionais, bem como um conjunto de 22 ferramentas e compara as ferramentas às práticas ágeis correspondentes. O envolvimento dos Poppendiecks na comunidade de desenvolvimento de *software agile*, incluindo palestras em várias conferências *agile*, resultou em tais conceitos, sendo mais amplamente aceitos dentro da comunidade *agile*. A metodologia *lean* é distribuída em sete princípios (POPPENDIECK; POPPENDIECK, 2010):

1. eliminar o desperdício;
2. integrar qualidade;
3. criar conhecimento;
4. adiar comprometimentos;
5. entregar rápido;
6. respeitar as pessoas;
7. otimizar o todo.

Fique atento

Princípios são verdades subjacentes que não mudam no tempo ou espaço, enquanto práticas são a aplicação dos princípios a uma situação particular. As práticas podem e devem diferir conforme você muda de um ambiente para o próximo, e elas também mudam à medida que uma situação evolui (POPPENDIECK; POPPENDIECK, 2010).

Sete princípios do desenvolvimento *lean* de *software*

Depois de conhecermos a metodologia *lean* desde o seu histórico até os princípios, fica a pergunta: como podemos implementar isso no processo de desenvolvimento de *software*? Não existe uma fórmula mágica para isso, mas Poppendieck e Poppendieck (2010) sugerem duas formas de começarmos a implementar os princípios do *lean*: o famoso "aprenda fazendo", no qual escolhemos um conjunto de práticas já consolidadas que levarão ao entendimento

dos princípios ou o "entenda antes de fazer", em que entendemos primeiro o princípio e, a partir dele, criamos práticas que resolvem melhor o problema dentro de um determinado contexto.

Mas, como citado, não existe uma fórmula mágica para isso. Quais das duas abordagens pode apresentar um melhor resultado? Poppendieck e Poppendieck (2010) indicam a junção das duas, pois copiar as práticas sem entender os princípios traz consigo um longo histórico de problemas. Quem nunca tentou copiar uma receita da Internet e no final o produto não condisse com que o estava descrito na receita? No entanto, quando entendemos os princípios, copiar práticas já consolidadas em situações parecidas pode ser o caminho mais eficiente.

Então, trazendo isso para o nosso dia a dia, se iniciarmos um novo projeto adotando boas práticas usadas em outro projeto, e as aplicarmos no projeto sem entendermos o porquê aquilo foi feito, certamente dará certo. Além disso, se precisarmos de uma adaptação, é possível que não consigamos fazer.

Para podermos implementar o *lean* com sucesso, precisamos entender um pouco mais sobre os princípios descritos por Poppendieck e Poppendieck (2010).

Princípio 1: eliminar o desperdício — no desenvolvimento de *software*, remova tudo o que não ajuda a entregar valor para o projeto, como, por exemplo, funcionalidades que o usuário não utilizará.

Princípio 2: integrar qualidade — evitar o problema e buscar a qualidade durante a produção em vez de procurar erros depois. Para isso, podemos utilizar metodologias de testes como o desenvolvimento guiado por comportamento, que é uma técnica de desenvolvimento ágil que encoraja a colaboração entre desenvolvedores, setores de qualidade e pessoas não técnicas ou de negócios em um projeto de *software*, relacionando-se com o conceito de verificação e validação. Lembre-se: a metodologia *lean* valoriza as pessoas.

Princípio 3: criar conhecimento — garantir que o conhecimento sobre o produto de *software* seja criado durante o seu desenvolvimento em vez de ter uma lista de requisitos e/ou um *layout* recomendando como deve ser o resultado da aplicação antes do início de seu desenvolvimento. Segundo Alan MacCormack, professor de Administração de Howard, existem quatro pilares que garantem uma boa aceitação no mercado: *releases* breves com um conjunto mínimo de funcionalidades para clientes avaliarem e darem *feedback*; *builds* diários e *feedback* rápido de testes de integração; uma equipe e/ou um líder com experiência e instintos para tomar boas decisões; e uma arquitetura modular

que dê suporte à habilidade de adicionar facilmente novas funcionalidades (POPPENDIECK; POPPENDIECK, 2010).

Princípio 4: adiar comprometimentos — comprometimentos são decisões tomadas que, por algum motivo, não podem voltar atrás. São decisões permanentes que serão adotadas dali para frente, dando certo ou não. Por isso, deve-se tentar trabalhar sempre com decisões planejadas que possibilitem colocar em prática, medir, aprender e validar. Lembre-se: estamos trabalhando com um produto volátil. Contudo, existem três passos básicos para identificar quando estamos assumindo um comprometimento ou tomando uma decisão planejada: identificar quando estamos fazendo uma escolha, identificar se a escolha é reversível e quanto tempo precisamos para voltar atrás da decisão que está sendo tomada.

Princípio 5: entregar rápido — precisamos descobrir como entregar tão rápido que nossos clientes não tenham tempo para mudar de ideia. Porém, é importante ressaltar que a velocidade não serve de nada se não andar junto com um nível muito alto de qualidade. Para conseguirmos ter um processo de produção rápido e uma boa disciplina na linha de produção, precisamos nos preocupar com algumas coisas, como: muita organização, planos e processos detalhados, forma de trabalho padronizada, documentação do *workflow* e descrição de trabalho específico. O objetivo é conseguir ter um projeto tão padronizado que a curva de aprendizado seja mínima quando precisarmos transferir as pessoas para outro projeto, indo ao encontro do princípio criar conhecimento.

Princípio 6: respeitar as pessoas — é o principal princípio de todos os sete da metodologia *lean*, pois sem o respeito às pessoas, nenhum dos outros princípios faz sentido. Respeitar as pessoas não é apenas tratar bem e ser educado com clientes e colaboradores que trabalham ao seu lado, mas deixar o orgulho e o ego fora do ambiente de trabalho. Independentemente do nível técnico de quem esteja propondo uma solução, essa opinião deve ser levada em conta e, se válida, deve ser implementada. Isso envolve não só respeito, mas também confiança. Se uma pessoa está com alguma responsabilidade em mãos, ela é plenamente capaz de resolver aquilo e é nossa obrigação ajudar e confiar nas decisões dessa pessoa.

Princípio 7: otimizar o todo — a otimização de apenas uma parte do processo pode trazer grandes problemas, como o excesso de responsabilidades ou muito

desgaste das áreas responsáveis por parte do processo que estão tentando "otimizar". O ideal não é olhar apenas para o desenvolvimento (o que muito comumente acontece), mas sim para como aquele pedido está sendo atendido, como ele está sendo detalhado e repassado para entrar em desenvolvimento, etc. Sem a otimização do todo, não é possível funcionar nenhum dos princípios anteriores, por isso, apesar de ser bem fácil o entendimento, é muito importante que esse princípio seja colocado em prática.

Agora que sabemos um pouco mais sobre os princípios da metodologia *lean* de desenvolvimento de *software*, o grande desafio é aplicar uma metodologia de desenvolvimento de veículos automotivos na produção de *software*, produtos completamente distintos, mas que atualmente se complementam.

3 *Lean* no desenvolvimento de *software*

O desenvolvimento de *software* tem tido cada vez mais importância na vida moderna, com aplicativos e *hardwares* embutidos, como podemos observar nos *smartphones*, sendo fundamental para a realização de diversas tarefas, desde assistir a um simples vídeo até pedir comida. Assim, o sistema criado pelo engenheiro japonês Taiichi Ohno, o *lean*, pode contribuir no processo de desenvolvimento de *software* para aumentar a produtividade e a qualidade desse tipo de produto.

O casal Mary Poppendieck e Tom Poppendieck, em seu livro *Lean Software Development: An Agile Toolkit* (2003), identifica sete princípios fundamentais da metodologia *lean* e os adapta para o mundo do desenvolvimento de *software*, com o objetivo de tornar o desenvolvimento melhor, mais barato e rápido.

Todavia, deve ficar claro que a aplicação de uma metodologia ágil de desenvolvimento de *software* na produção de um *software* requer experiência, disciplina e respeito dos profissionais envolvidos no processo, buscando sempre colocar o cliente como coadjuvante do processo. Não existe uma fórmula mágica nem uma receita de bolo de como aplicar a metodologia *lean* no desenvolvimento de *software*. Com base em alguns princípios apresentados por Poppendieck e Poppendieck (2003), vamos apresentar algumas dicas de como preparar a equipe para o uso da metodologia.

- Dica 1: estude sobre a metodologia. O primeiro passo para aplicar a metodologia *lean* de desenvolvimento de *software* foi dado, mas os estudos devem continuar. Sendo assim, busque compreender as suas vantagens e desvantagens.

- Dica 2: análise de empresas nas quais a metodologia *lean* foi aplicada. Esta dica pode parecer óbvia demais, mas pesquisar por empresas nas quais uma determinada metodologia foi aplicada irá auxiliar a entender se a metodologia serve para o produto que será desenvolvido.
- Dica 3: qualifique a sua equipe. No Sistema Toyota de Produção, a habilidade individual e o trabalho em equipe são bem valorizados, e como esse sistema foi o percursor da metodologia *lean*, essa afirmação é válida, indo diretamente ao encontro dos sete princípios levantados por Mary e Tom. O treinamento pode ser por meio de cursos presenciais ou *on-line*. Uma sugestão para uma melhor integração da equipe seria utilizar a técnica de aprendizagem de programação Coding Dojo; ela vai ao encontro dos sete princípios da metodologia *lean* de desenvolvimento de *software*, direta e indiretamente.

Coding Dojo

Um Coding Dojo (Dojo de Codificação) é uma reunião em que vários desenvolvedores se reúnem para resolver um desafio de programação (BACHE, c2013), sendo que os principais objetivos são a diversão e o envolvimento de práticas deliberadas de desenvolvimento, a fim de melhorar as suas habilidades. O Coding Dojo tem as seguintes características:

- ambiente não competitivo, colaborativo e divertido;
- todos os níveis de habilidade são bem-vindos;
- seguro para experimentar novas ideias.

Saiba mais

Dojo é o local especial onde se treinam artes marciais japonesas. Muito mais do que uma simples área de atividades físicas, o Dojo deve ser respeitado como se fosse um templo. Por isso, é comum ver o praticante fazendo uma reverência antes de entrar e ao sair do Dojo, tal como se faz nos templos japoneses (MENEZES, c2015).

As características do Coding Dojo estão de acordo com o sexto princípio da metodologia *lean* de desenvolvimento de *software*, respeitar as pessoas, além

de fazer uma ótima integração na equipe de desenvolvimento. Os requisitos para que um Coding Dojo seja feito são listados a seguir:

- um computador conectado a um projetor;
- uma tela de projeção onde todos possam ver a solução do problema;
- um piloto, um copiloto e um mestre;
- mais pessoas que desejem participar; um bom número seria no mínimo cinco e no máximo nove.

Com os requisitos organizados, o mestre oferece um desafio ao grupo. O piloto, um dos participantes sentado ao computador, é a única pessoa que pode utilizá-lo para concluir o desafio, enquanto o copiloto permanece ao seu lado, mas somente para observá-lo em ação e oferecer indicações. Todos os demais observam e podem discutir entre si e com o copiloto e o piloto. Qualquer um pode ainda fazer perguntas ao mestre, mas este só pode responder com outra pergunta.

Um dos objetivos do Coding Dojo é a interação, por isso ao final de um tempo preestabelecido pelos participantes, em média entre 5 e 7 minutos, é feita uma troca entre um dos participantes da seguinte maneira: o piloto volta para a plateia, o copiloto se torna o próximo piloto e alguém da plateia passa a ser o copiloto, o que vai ao encontro do quarto princípio da metodologia *lean*, adiar comprometimentos.

Ao final do Coding Dojo é feita uma retrospectiva, na qual podem ser utilizadas diversas técnicas, com o objetivo de responder a três perguntas básicas.

1. O que aprendemos com o Coding Dojo de hoje?
2. O que podemos melhorar para a realização dos próximos Coding Dojos?
3. O que devemos continuar fazendo nos próximos Coding Dojos?

Esses são um dos pontos mais altos do Dojo, pois é o momento em que os conhecimentos são apresentados a todos, indo ao encontro do terceiro princípio da metodologia *lean*, criar conhecimento.

Saiba mais

No artigo "Coding Dojos como aprendizado interno", de Vitor Ruiz Leonel (2013), podemos aprender como funciona o uso da técnica de Coding Dojo como ferramenta de aprendizado e de trocas de experiências em uma empresa.

No artigo "Coding Dojo 101", Humberto Rocha (2016) demonstra um vídeo de um Coding Dojo feito na empresa Globo.com.

No vídeo "Coding Dojo", Daniel Cukier (2010) explica os conceitos básicos do Coding Dojo na empresa.

Além do ambiente altamente divertido e do espírito de equipe que os Coding Dojos proporcionam, também são ensinadas técnicas de programação que vão ao encontro dos princípios um, cinco e sete da metodologia *lean*, eliminar desperdícios, entregar rápido e otimizar o todo respectivamente, que são trazidos de outra metodologia ágil, a *Extreme programming* (TELES, 2014), como as seguintes.

- *Pair programming*: programação em par.
- Passos de bebê: cada etapa para a solução deve ser pequena o suficiente para que todos possam compreender e replicar mais tarde.
- *Refactoring*: refatoração de código.
- *Test driven development* — TDD: antes de ser escrita uma implementação de código é escrito um teste.

Como podemos observar, no desenvolvimento de *software*, é imprescindível a adoção de uma metodologia de desenvolvimento ágil, com o objetivo de buscar competitividade no mercado, e o interessante é que todas as metodologias valorizam o capital intelectual das pessoas, recurso mais precioso em todo o processo.

Referências

BACHE, E. *The Coding Dojo handbook*: a practical guide to creating a space where good programmers can become great programmers. [*S. l.*]: The Author, c2013.

BERNARDO, K. Kanban: do início ao fim. *Cultura Ágil*, 8 dez. 2014. Disponível em: https://www.culturaagil.com.br/kanban-do-inicio-ao-fim/#. Acesso em: 1 set. 2020.

BROOKSHEAR, J. G. *Ciência da computação*: uma visão abrangente. 11. ed. Porto Alegre: Bookman, 2013.

KEMP, A. *Taiichi Ohno*: hero of the Toyota production system, 2018. Disponível em: https://www.qad.com/blog/2018/03/taiichi-ohno-toyota-production-system. Acesso em: 1 set. 2020.

KOSCIANSKI, A.; SOARES, M. S. *Qualidade de software*: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2. ed. São Paulo: Novatec, 2007.

MARTINS, P. G.; LAUGENI, F. P. *Administração da produção*. São Paulo: Saraiva, 2015.

MENEZES, I. R. (adapt.). *O Dojo*: o lugar de iluminação. c2015. Disponível em: https://www.segmentodojo.com.br/o-dojo/o-que-e-dojo/. Acesso em: 1 set. 2020.

OHNO, T. *Toyota production system*: beyond large-scale production. Portland, OR: Productivity Press, 1988.

POPPENDIECK, M.; POPPENDIECK, T. *Implementando o desenvolvimento Lean de software*: do conceito ao dinheiro. Porto Alegre: Bookman, 2010.

POPPENDIECK, M.; POPPENDIECK, T. *Lean software development*: an agile toolkit. Boston, MA: Addison-Wesley Professional, 2003.

SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Prentice Hall, 2011.

TELES, V. M. *Extreme programming*: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. 2. ed. São Paulo: Novatec, 2014.

Leituras recomendadas

CUKIER, D. *Coding Dojo*: introdução. [*S. l.: s. n.*], 2010. 1 vídeo (11 min). Publicado pelo canal TecNoz. Disponível em: https://www.youtube.com/watch?v=E-jFKkaAc7k. Acesso em: 1 set. 2020.

LEONEL, V. R. *Coding Dojos como aprendizado interno*. 2013. Disponível em: https://partiu.loggi.com/coding-dojos-como-aprendizado-interno-203eb5ef8e92. Acesso em: 1 set. 2020.

ROCHA, H. *Coding Dojo 101*, 2016. Disponível em: https://humberto.io/pt-br/blog/coding-dojo-101/. Acesso em: 1 set. 2020.

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