Métodos Ágeis em TI

UNIDADE 03

Extreme Programming

Olá! Será apresentado nesta Unidade o método ágil extreme programming. É um dos métodos precursores a todo o movimento da linha de pensamento ágil, bastante técnico, com diversas práticas de engenharia de software.

Os métodos ágeis surgiram de profissionais que vieram da orientação a objetos, como Kent Beck e Ward Cunningham, duas grandes personalidades do tema métodos ágeis. Eles foram líderes na adoção dessas linguagens orientadas a objetos, ambos vieram principalmente da linguagem *smalltalk*. A partir disso também surgiram outras práticas

da comunidade *smalltalk*, como refatoração, programação em par, mudanças rápidas, *feedback* constante do cliente, reforço do desenvolvimento iterativo, testes automatizados etc.

Kent Beck influenciou muitas pessoas em todas essas práticas e na adoção dos métodos ágeis. Portanto, ele é considerado um grande ícone nessa área. Inclusive, foi ele quem fez a família XUnit (como Junit e NUnit). Mas Kent Beck não estava sozinho, outros profissionais, como Opdyke, criador da refatoração, Cunningham, dos padrões de projetos, fizeram parte.

História

Criado em 1997, o XP possui adeptos e outros que duvidam da sua real utilidade, muitos por falta de conhecimento ou entendimento, achando que no XP, apenas código é o que realmente interessa, descartando o resto, como planejamento, documentação etc.

O XP é um método de desenvolvimento de *software*, leve, não prescritivo e procura fundamentar as suas práticas por um conjunto de valores que serão vistos posteriormente no artigo. O XP, diferentemente do que muitos pensam, também pode ser adotado por desenvolvedores médios, e não apenas por experientes.

O objetivo principal do XP é levar ao extremo um conjunto de práticas ditas como boas na engenharia de *software*. Dentre elas, podemos citar o teste, visto que procurar defeitos é perda de tempo, temos de testar constantemente.

Diferencial

O XP muda um conceito importante em métodos de trabalho, evita o medo da mudança, pois errar é feito com um baixo custo, diferentemente dos métodos tradicionais, em que se sabe que quanto mais tardia a mudança, maiores os custos. Portanto, o XP apoia constantes mudanças, que não devem ser temidas, principalmente quando seguimos os seus valores e as suas práticas.

Para conseguirmos nos adaptar às mudanças, o XP preconiza ciclos curtos, o que proporciona previsibilidade e redução de incertezas/riscos, simplicidade e melhorias constantes de código (*refactoring*), para facilitar a mudança e testes automatizados e integração contínua, a fim de aumentar a confiança.

Valores

Estão apresentadas as descrições dos valores do *extreme programming* a seguir.

Simplicidade

Faremos o que for necessário e solicitado, mas nada mais. Isso maximizará o valor criado para o investimento feito até o momento. Daremos pequenos passos simples para alcançar nosso objetivo e mitigaremos as falhas conforme elas acontecerem. Criaremos algo de que nos orgulhamos e o manteremos a longo prazo por custos razoáveis.

Comunicação

Todos fazem parte da equipe e nos comunicamos pessoalmente diariamente. Trabalharemos juntos em tudo, desde os requisitos até o código. Vamos criar a melhor solução possível para o nosso problema.

Feedback

Levaremos cada compromisso de iteração a sério, entregando um *software* funcional. Demonstramos nosso *software* cedo e frequentemente, então, ouvimos com atenção e fazemos as alterações necessárias. Conversaremos sobre o projeto e adaptaremos nosso processo a ele, não o contrário.

Respeito

Todos dão e sentem o respeito que merecem como um membro valioso da equipe. Todos contribuem com valor, mesmo que seja simplesmente entusiasmo. Os desenvolvedores respeitam a experiência dos clientes e

vice-versa. A administração respeita nosso direito de aceitar responsabilidades e receber autoridade sobre nosso próprio trabalho.

Coragem

Diremos a verdade sobre o progresso e as estimativas. Não documentamos desculpas para o fracasso, pois planejamos sucesso. Não tememos nada porque ninguém trabalha sozinho. Vamos nos adaptar às mudanças sempre que elas acontecerem.

Práticas

Extreme programming é dinâmica e flexível, porém, é necessária muita disciplina para usá-la em um projeto.

Para demonstrar isso, está apresentado a seguir um conjunto sugerido de boas práticas em projetos usando XP.

- ***The customer is always available*** (o cliente sempre disponível)

Constante disponibilidade do cliente para colaborar em dúvidas, alterações, e prioridades em um escopo, ou seja, proporcionando um dinamismo ativo ao projeto.

- ***Metaphor*** (uso de metáforas no projeto)

Visando a facilitar a comunicação da equipe, caso seja possível, é estabelecido o uso de metáforas em pontos-chave ao projeto, como a definição de um nome que seja comum à equipe e simbolize algo de fácil assimilação.

- ***Release and iteration planning*** (planejamento de iterações e *releases*)

Entre o cliente e os técnicos são estimuladas reuniões usando quadros brancos, com o objetivo de captar e definir as histórias de usuário e para poder estimar o tempo ideal das iterações, o projeto, elaborar estratégias e tentar prever as contingências para projeto.

- ***Small releases*** (pequenas versões)

Conforme as iterações são concluídas, o cliente recebe pequenas versões (*releases*) do sistema, visando a com que seja colocado em prática e validado aquilo que está sendo implementado. Isso também permite que mais cedo possam ser detectadas necessidades de alterações de requisitos no *software*.

- ***Acceptance tests*** (testes de aceitação)

São definidos pelo usuário na fase inicial do projeto e são os critérios de aceitação do *software* conforme a estratégia de entrega. Representa exatamente a métrica de aderência do *software* desenvolvido/implantado ao universo do cliente.

- ***Test driven development*** (desenvolvimento orientado aos testes)

Aplicados a partir de testes unitários do código produzido, além de serem preparados utilizando os critérios de aceitação definidos previamente pelo cliente. Garante também a redução de erros de programação e aumenta a fidelidade do código produzido ao padrão estabelecido para o projeto. Por meio dessa prática, definimos antes da codificação, os testes unitários dos métodos críticos do *software* e os testes funcionais de aceitação para as funcionalidades.

- ***Continuous integration*** (integração contínua)

Os diversos módulos do *software* estão integrados diversas vezes por dia e todos os testes são executados. O código não passa até obter sucesso em 100% dos testes, facilitando, dessa forma, o trabalho de implantação da solução.

- ***Simple design*** (simplicidade de projeto)

O código está, a qualquer momento, na forma mais simples e clara, conforme os padrões definidos pela equipe de desenvolvimento, facilitando a compreensão e possível continuidade por qualquer um de seus membros.

- ***Refactoring*** (refatoração – melhoria constante do código)

A cada nova funcionalidade adicionada, é trabalhado o *design* do código até ficar na sua forma mais simples, mesmo que isso implique em "mexer" em um código que esteja em funcionamento.

- ***Pair programming*** (programação em pares)

Todo código de produção é desenvolvido por duas pessoas trabalhando com o mesmo teclado, o mesmo *mouse* e o mesmo monitor, somando forças para a implementação do código. À primeira vista pode parecer loucura, pois se imagina estar gastando dois

recursos humanos ao mesmo tempo, para fazer a mesma tarefa e sem possibilidade de avanço substancial no projeto. Mas, na verdade, essa prática tem pontos positivos, como:

- Compartilhamento de conhecimento sobre as regras de negócio do projeto por todos da equipe de desenvolvimento.
- Fortalecimento da prática de propriedade coletiva do código.
- Nivelação de conhecimento técnico dos programadores.
- Elevação dos níveis de atenção ao código produzido, pois um supervisiona e orienta o trabalho do outro. Dessa forma, minimiza-se a possibilidade de erros no código, erros de lógica e produção de um código fora dos padrões estabelecidos pela equipe.

- ***Move people around*** (rodízio de pessoas)

As duplas de programação são revezadas periodicamente, com o objetivo de uniformizar os códigos produzidos, deixar todos os módulos do sistema com mesmo padrão de código/pensamento e compartilhar o código com todos da equipe.

- ***Collective code ownership*** (propriedade coletiva – o código é de todos da equipe)

Uma vez aplicados a programação em pares e o rodízio de pessoas, a equipe é responsável por cada arquivo de código. Não é preciso pedir autorização para alterar qualquer arquivo, mantendo um padrão prático de comunicação da equipe.

- ***Coding standards*** (padronização do código)

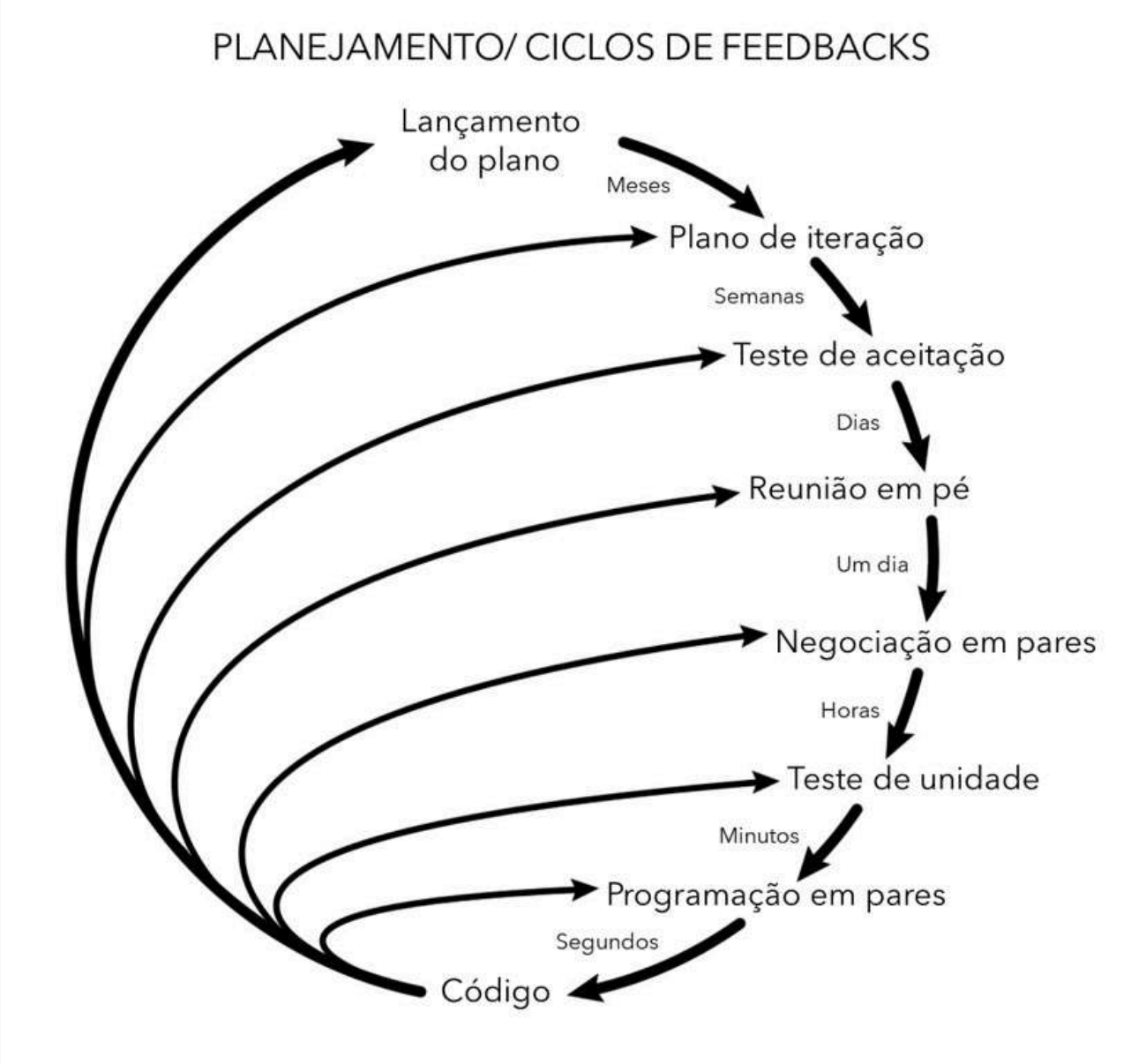
Todo código é desenvolvido seguindo um padrão, mas toda equipe deve seguir o mesmo padrão. Dessa forma, todos terão a mesma visão do código.

- ***Sustainable pace*** (ritmo sustentável)

Trabalhar por longos períodos é contraproducente. Portanto, sempre que possível, deve-se evitar a sobrecarga de trabalho de todos da equipe, criando condições favoráveis ao uso da carga normal de trabalho. É necessário deixar a equipe livre para relaxar, brincar, ou fazer o que bem entender para equilibrar o trabalho mental e físico.

A Figura 1, apresentada a seguir, demonstra com qual frequência as práticas apresentadas são utilizadas por um time que adota o método ágil *extreme programming*.

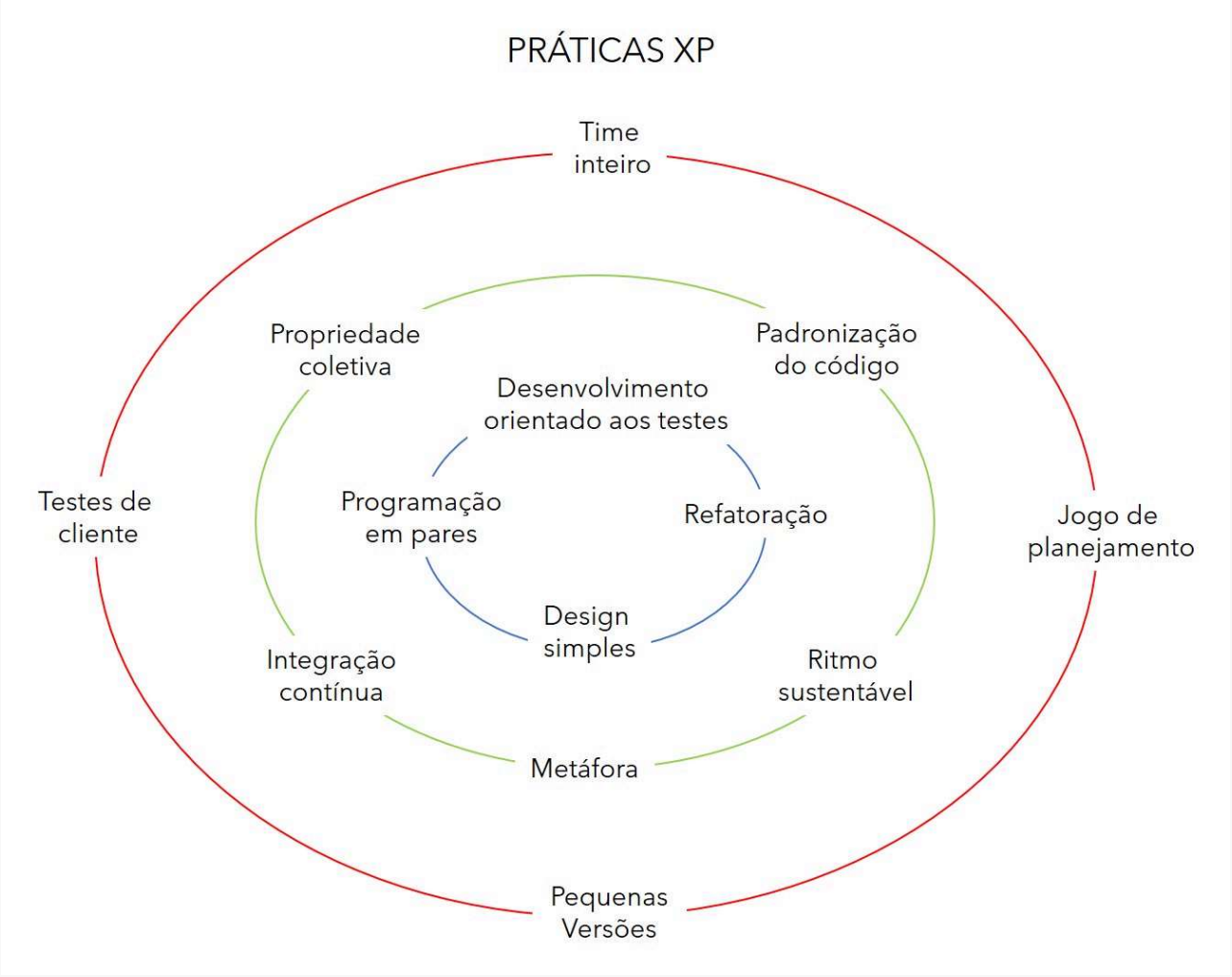
Figura 1: Quando usar as práticas *de extreme programming*



O diagrama a seguir demonstra a frequência de uso das práticas de *extreme programming.* Fonte: Adaptado de Wells (2009).

E a Figura 2, apresentada a seguir, demonstra quem se envolve mais com cada prática. No círculo central, em azul, as práticas mais utilizadas pelos desenvolvedores individualmente. No círculo intermediário, em verde, as práticas que são seguidas pelo time. E no círculo externo, em vermelho, as práticas que envolvem também o cliente e, eventualmente, outras áreas da empresa.

Figura 2: Quem usa as práticas de *extreme programming*

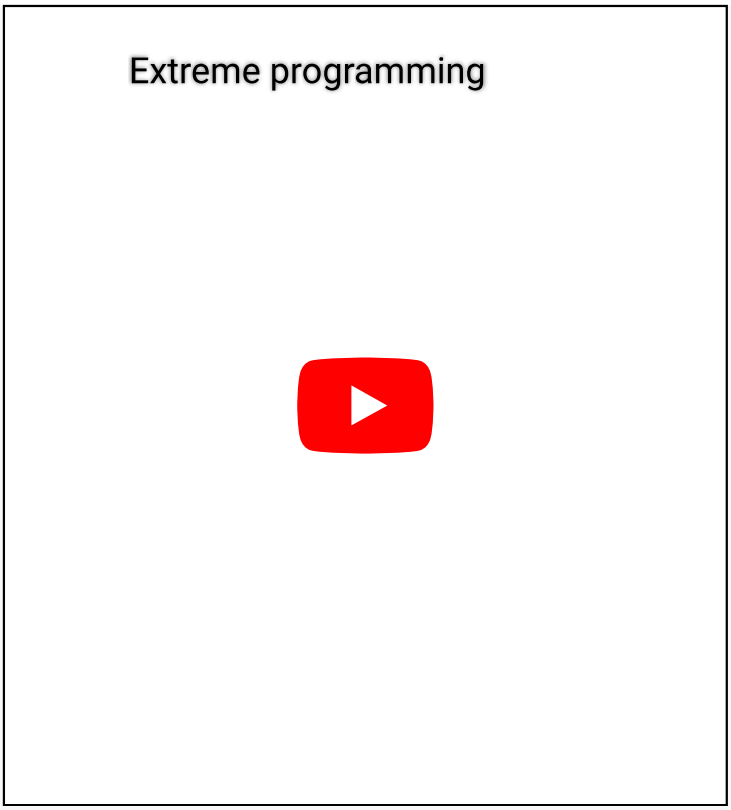


O diagrama a seguir demonstra a frequência de uso das práticas de extreme programming. Fonte: Adaptado de https://ronjeffries.com/xprog/what-is-extreme-programming/

A seguir, será apresentado um *case* sobre o uso do método ágil *extreme programming*, contando como funcionaram as práticas descritas anteriormente. Além disso, haverá um momento de conversa com um profissional de métodos ágeis, o qual discorrerá sobre o uso do método ágil *extreme programming*.

Case de extreme programming

Será apresentado neste vídeo um case de adoção do método ágil *extreme programming*, explicando como as práticas foram utilizadas.



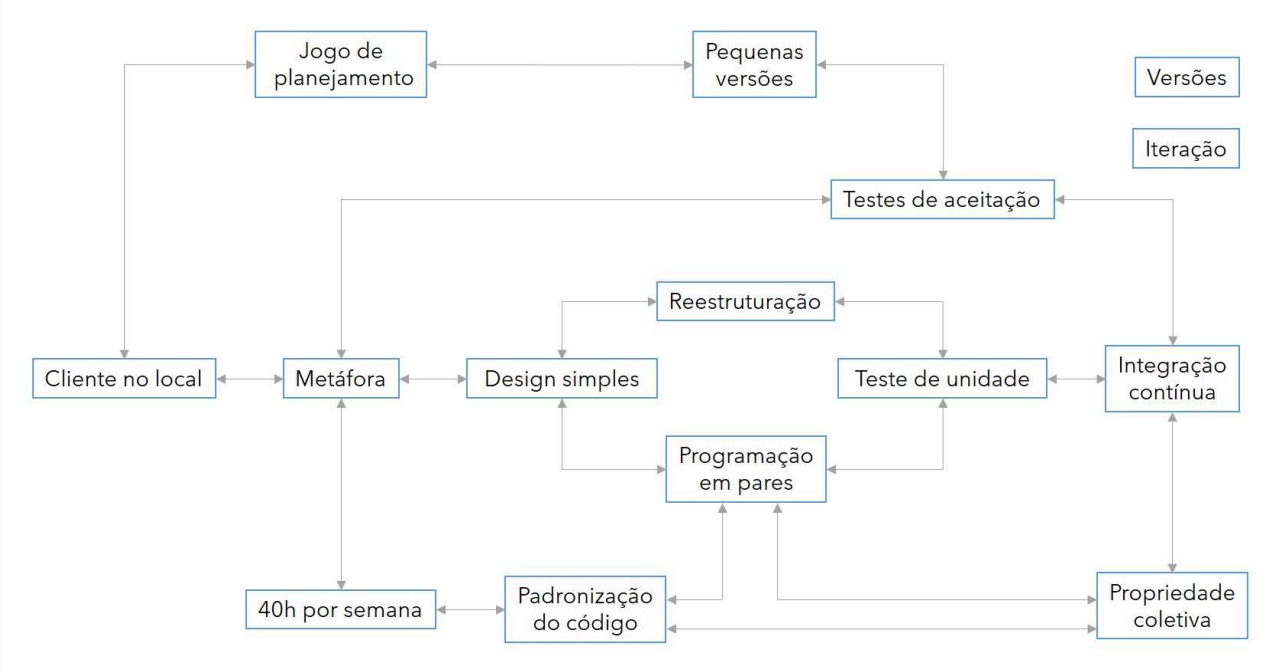
Entrevista sobre *extreme programming*

Neste vídeo, será entrevistado um profissional de métodos ágeis. Discutiremos as principais características do extreme programming, com uma visão atual e prática de mercado.

Conclusão

Foram apresentadas sugestões de boas práticas, pois o XP não é uma metodologia estática, e sim dinâmica, proporcionando liberdade para cada um modelar sua própria forma de trabalho com ele. É comum que não se consiga aplicar todas essas práticas em um projeto, e sim que se faça uma espécie de *mix* de práticas XP, com práticas pessoais mais intuitivas e geradas a partir de experiências anteriores em projetos, ou mesmo oriundas de práticas de outras metodologias como *scrum*. Porém, é necessário entender que todas essas práticas são recomendadas, pois são sinérgicas, e a adoção de uma delas influencia positivamente, permitindo ou favorecendo a adoção de outras, conforme a Figura 3 a seguir.

Figura 3: Práticas de *extreme programming*



O diagrama a seguir demonstra a sinergia entre as práticas de*extreme programming.*Fonte: Adaptado de Singh (2019).

Referências

SINGH, V. Extreme Programming. **ToolsQA**, Agile, 2019. Disponível em: https://www.toolsqa.com/agile/extreme-programming/. Acesso em: 28 abr. 2021.

WELLS, D. Introducing Extreme Programming. **Extreme Programming**, Introduction, 2009. Disponível em: http://www.extremeprogramming.org/introduction.htm. Acesso em: 28 abr. 2021.