

TÉCNICAS ÁGEIS DE PROGRAMAÇÃO

Daniel Wildt, Guilherme Lacerda e Michael da Costa Móra

“Economizar tempo e esforço de um usuário vai me trazer muito mais benefícios do que tentar economizar em processamentos ou armazenamento.”

Luana Muller

Conheça o livro da disciplina

CONHEÇA SEUS PROFESSORES

3

Conheça os professores da disciplina.

EMENTA DA DISCIPLINA

4

Veja a descrição da ementa da disciplina.

BIBLIOGRAFIA DA DISCIPLINA

5

Veja as referências principais de leitura da disciplina.

O QUE COMPÕE O MAPA DA AULA?

6

Confira como funciona o mapa da aula.

MAPA DA AULA

8

Veja as principais ideias e ensinamentos vistos ao longo da aula.

RESUMO DA DISCIPLINA

39

Relembre os principais conceitos da disciplina.

AVALIAÇÃO

40

Veja as informações sobre o teste da disciplina.

Conheça seus professores



DANIEL WILDT

Professor Convidado

Profissional de tecnologia preocupado com desenvolvimento de produtos e serviços com equipes focadas em aprendizado, melhoria contínua e autonomia. Mentora e produz conteúdo em vídeo, áudio e texto sobre: consciência de tempo, experiência de usuário, empreendedorismo e metodologias ágeis. Sócio e mentor na Wildtech, Blogger/Youtuber no danielwildt.com, sócio e diretor na uMov.me.



GUILHERME LACERDA

Professor Convidado

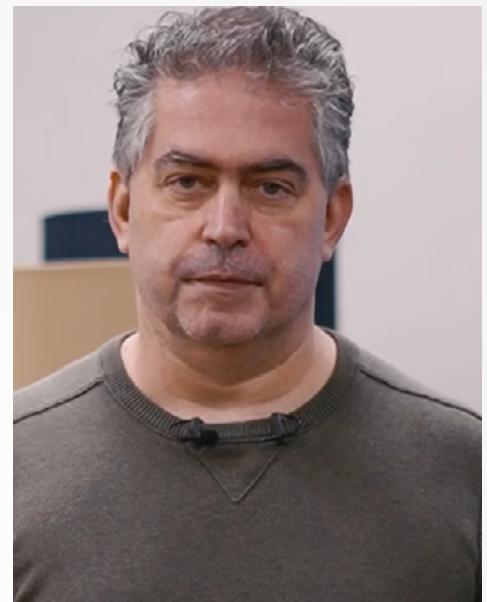
Graduado em Informática pela Universidade da Região da Campanha (2000). Mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (2005). Atualmente, cursa Doutorado em Ciência da Computação na UFRGS, na área de Engenharia de Software. Consultor/Instrutor associado da Wildtech, trabalhando com coaching e mentoring nas áreas de Engenharia de Software, Gerência de Projetos e Produtos e Metodologias Ágeis (eXtreme Programming, SCRUM, Lean). Possui mais de 20 anos de experiência em desenvolvimento de software. Atuou por vários anos como analista/projetista/desenvolvedor de software. Possui as certificações de SCRUM Master (SCM) e SCRUM Professional (CSP) pela SCRUM Alliance. Membro do IASA (International Association of Software Architects). Fundador do Grupo de Usuários de Métodos Ágeis (GUMA), vinculado a SUCESU-RS. É docente de graduação (Ciência da Computação, Análise e Desenvolvimento de Sistemas e Sistemas de Informação, Gestão de TI - Unisinos) e pós-graduação (Engenharia de Software, Desenvolvimento de Aplicações Móveis - Unisinos e Desenvolvimento Full Stack - PUCRS).

Conheça seus professores

MICHAEL DA COSTA MÓRA

Professor PUCRS

Graduado em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS), mestre em Computação e doutor em Ciência da Computação pela mesma universidade. Professor-adjunto do Instituto de Informática. Tem experiência na área de ciência da computação com ênfase em inteligência artificial, atuando principalmente nos seguintes temas: inteligência artificial, aprendizagem de máquina, agentes inteligentes e sistemas multiagentes, engenharia de software e desenvolvimento de sistemas, ensino de programação e de ciência da computação.



Ementa da Disciplina

Fundamentos da agilidade: primórdios, manifesto ágil, princípios da agilidade. Panorama das metodologias ágeis. Extreme programming: características, valores, práticas, as práticas na prática. Test driven development (TDD): origens, codificar – testar – projetar, benefícios e armadilhas, variações, TDD na prática. Behaviour driven design (BDD): origens e princípios, BDD x TDD, benefícios e armadilhas, BDD na prática.

Bibliografia da Disciplina

As publicações destacadas têm acesso gratuito.

Bibliografia básica

BLOODYK, G. Extreme programming: Everything You Need to Know. CreateSpace Independent Publishing Platform; 2017.

FOWLER, M. Refactoring: Improving the Design of Existing Code. Addison-Wesley; 2nd Ed, 2018.

MARTIN, Robert. Desenvolvimento Ágil Limpo: De volta às origens. Alta Books, 2020.

Bibliografia complementar

SMART, J.F. BDD in Action: Behavior-driven development for the whole software lifecycle. Manning Publications, 2014.

BOLBOACA, Adrian . Practical Remote Pair Programming: Best practices, tips, and techniques for collaborating productively with distributed development teams. 1a. Ed. Packt Publishing, 2021.

BECK, K; ANDRES, C. Extreme Programming Explained: Embrace Change. Addison-Wesley; 2nd edition, 2014.

COHN, M. Desenvolvimento de Software com Scrum: Aplicando Métodos Ágeis com Sucesso. Bookman, 2011.

LEMAIRE, M. Refactoring at Scale: Regaining Control of Your Codebase. 1a. Ed. Sebastopol: O'Reilly, 2020.

O que compõe o Mapa da Aula?

MAPA DA AULA

São os capítulos da aula, demarcam momentos importantes da disciplina, servindo como o norte para o seu aprendizado.



EXERCÍCIOS DE FIXAÇÃO

Questões objetivas que buscam reforçar pontos centrais da disciplina, aproximando você do conteúdo de forma prática e exercitando a reflexão sobre os temas discutidos. Na versão online, você pode clicar nas alternativas.



PALAVRAS-CHAVE

Conceituação de termos técnicos, expressões, siglas e palavras específicas do campo da disciplina citados durante a videoaula.



VÍDEOS

Assista novamente aos conteúdos expostos pelos professores em vídeo. Aqui você também poderá encontrar vídeos mencionados em sala de aula.



PERSONALIDADES

Apresentação de figuras públicas e profissionais de referência mencionados pelo(a) professor(a).



LEITURAS INDICADAS

A jornada de aprendizagem não termina ao fim de uma disciplina. Ela segue até onde a sua curiosidade alcança. Aqui você encontra uma lista de indicações de leitura. São artigos e livros sobre temas abordados em aula.



FUNDAMENTOS

Conteúdos essenciais sem os quais você pode ter dificuldade em compreender a matéria. Especialmente importante para alunos de outras áreas, ou que precisam relembrar assuntos e conceitos. Se você estiver por dentro dos conceitos básicos dessa disciplina, pode tranquilamente pular os fundamentos.

CURIOSIDADES

Fatos e informações que dizem respeito a conteúdos da disciplina.

DESTAQUES

Frases dos professores que resumem sua visão sobre um assunto ou situação.

ENTRETENIMENTO

Inserções de conteúdos para tornar a sua experiência mais agradável e significar o conhecimento da aula.

CASE

Neste item, você relembra o case analisado em aula pelo professor.

MOMENTO DINÂMICA

Aqui você encontra a descrição detalhada da dinâmica realizada pelo professor.

Mapa da Aula

Os tempos marcam os principais momentos das videoaulas.

AULA 1 • PARTE 1

PERSONALIDADE

Jim Highsmith



Engenheiro de software norte-americano criador do Adaptive Software Development, descrito em seu livro de 1999 "Adaptive Software Development". É autor de livros na área de metodologia de desenvolvimento de software.

Princípios por trás do manifesto ágil

Entre os princípios envolvidos no desenvolvimento de softwares, Daniel destaca:

- satisfazer o cliente, entregando o software em tempo hábil e continuamente;
- aceitar as mudanças de requisitos, em qualquer fase do projeto;
- entregar software na menor escala de tempo possível;
- equipe de desenvolvimento e cliente são do mesmo time;
- construir projetos com indivíduos motivados e comprometidos com o resultado;
- usar a comunicação efetiva;
- ter o software em funcionamento é a principal medida de progresso;
- atenção contínua à excelência técnica;
- as melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;
- refletir sobre como se tornar mais eficaz, ajustando e adaptando o comportamento da equipe.

02:51



03:16



Qualidade não é opcional, temos que saber tratá-la dentro do nosso desenvolvimento de sistemas.

06:36



Manifesto para desenvolvimento ágil de software

Daniel explica que o Manifesto para desenvolvimento ágil de software foi um movimento em que um conjunto de pessoas se reuniu, em 2001, para buscar maneiras diferentes de entregar projetos com mais qualidade e assertividade. Os signatários desse manifesto, são de várias vertentes, explica o Professor.

08:40



09:12



Toda vez que fazemos uma entrega, temos a oportunidade de aprender com isso e saber como evoluir.

10:41



Ainda falamos muito sobre cargo, temos que falar mais sobre papéis.

“ Simplicidade é a arte de maximizar a quantidade de trabalho não realizado. **”**

12:05

“ Nosso desafio, como pessoas desenvolvedoras, é identificar com os clientes que perguntas são essas que podem nos ajudar a se conectar com o problema. **”**

15:39

Falta de comunicação

Daniel comenta que o problema de comunicação é sempre um aspecto muito presente no trabalho com projetos. Ele cita que, às vezes, é preciso tomar cuidado com o pedido do cliente, que já vem com uma “cara de solução” e fazer um passo anterior, ou seja, identificar o problema que ele está apresentando. Daniel também fala sobre o cone da incerteza, pelo qual cada processo de aprendizado no desenvolvimento de software passa.

“ A agilidade me faz entender que a frequência de feedback me permite aprender mais rápido sobre as coisas. **”**

21:22

Adoção de agilidade

Professor Daniel explica que algumas pesquisas têm mostrado como a agilidade pode ser medida e também sobre alguns benefícios da adoção da agilidade, como criação de mais visibilidade, alinhamento com o negócio, geração de valor aos clientes, busca de qualidade, aumento da moral do time. Daniel também destaca alguns desafios nesse contexto e comenta que, quando se fala sobre valores, benefícios e desafios da agilidade, é importante ouvir o cliente e encontrar formas de responder à incerteza.

22:33

“ Quando estivermos trabalhando dentro de um ciclo de entrega e pegarmos uma atividade para trabalhar, iremos passar por todo o ciclo de desenvolvimento dela, do planejamento até a entrega. **”**

23:23

“ Quando eu demoro muito tempo para fazer uma entrega, não consigo celebrar nada, isso prejudica a moral de um time. **”**

25:00

“ O que a agilidade faz é expor os problemas mais rapidamente. **”**

25:31

“ Se você está trabalhando com métodos ágeis, sempre que você causar uma melhoria, seu processo vai ficar mais simples, menos complexo. **”**

EXERCÍCIO DE FIXAÇÃO

Qual aspecto é muito presente no trabalho com projetos de software, de acordo com Daniel Wildt?

Atraso nas entregas.

Falta de comunicação.

Projetos não comprometidos com o resultado.

Equipe desunida.

27:06

“



32:01



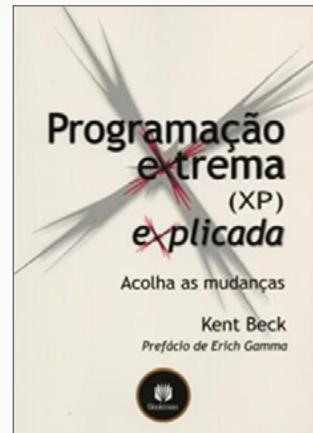
“

Nossa busca tem que ser por controles o mais automatizados possível e gerando evidências o mais transparente possível para que possamos atuar nelas.

”

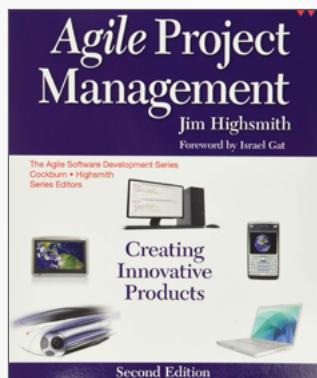
LEITURA INDICADA

Livro: Programação Extrema (Xp) Explicada



LEITURA INDICADA

Livro: Agile Project Management



O livro “Agile Project Management: Creating Innovative Products”, de Jim Highsmith foi publicado em 2009 pela Addison-Wesley Professional.

32:04



32:18



LEITURA INDICADA

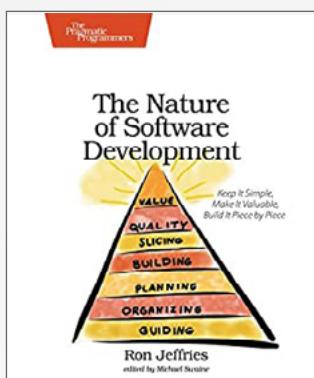
Livro: Implementando o desenvolvimento lean de software



O livro “Implementando o desenvolvimento lean de software: do conceito ao dinheiro” é de Mary Poppendieck e Tom Poppendieck. Foi publicado em 2011 pela Bookman.

LEITURA INDICADA

Livro: **The Nature of software Development**



A obra “*The Nature of Software Development: keep it simple, make it valuable, build it piece by piece*” foi escrita por Ron Jeffries e publicado em 2015 pela Pragmatic Bookshelf.



32:35



32:39

LEITURA INDICADA

Livro: **eXtreme Programming**



O livro “*eXtreme Programming: Práticas para o dia a dia no desenvolvimento ágil de software*” é de autoria de Daniel Wildt, Dionatan Moura, Guilherme Lacerda e Rafael Helm e foi publicado em 2015 pela Casa do Código.

AULA 1 • PARTE 2

Conceito 3C

Professor Daniel explica que o conceito 3C é composto por cartão, conversação e confirmação. O cartão é uma informação limitada, opera como um lembrete. A conversação se refere a quem nos ajuda a ampliar o conhecimento e desenvolver exemplos sobre o que estamos aprendendo. Já na confirmação, conforme conversamos com clientes, conseguimos comprovar os atendimentos, através de validação como mundo real.



01:05



01:14

PERSONALIDADE

Ron Jeffries



Em conjunto com Kent Beck e Ward Cunningham, Ron Jeffries é fundador da metodologia de desenvolvimento de software Extreme Programming por volta de 1996.

“Através das conversas e da nossa prática, vou conseguindo ter confirmações daquilo que foi levantado.”



03:29

EXERCÍCIO DE FIXAÇÃO

Assinale a alternativa que apresenta alguns dos benefícios da adoção da agilidade.

Criação de mais visibilidade e alinhamento com o negócio.

Busca de qualidade e aumento da moral do time.

Geração de valor aos clientes e satisfação dos clientes.

Todas as alternativas estão corretas.

Formação de times

Daniel comenta que, dentro de um time, é preciso se posicionar e isso se faz compreendendo em quais disciplinas se consegue trabalhar. Ele apresenta alguns questionamentos importantes e que contribuem para descobrir a maneira como se deseja se posicionar em um time: quais são as disciplinas em que você se permite atuar dentro de um time? Como a equipe aprende? Como você quer se posicionar? O que você faz quando não sabe o que fazer? O que você faz quando não se tem o que fazer?



05:31 08:06

Spike solution

Daniel explica que o spike solution é uma disciplina encontrada dentro do extreme programming. Com essa ferramenta, é possível aprender antes para depois poder descobrir e habilitar a entrega, ter tempo de investigação, preparar para o fazer e remover risco técnico. Daniel também fala sobre a importância de entender a diferença entre o modo entrega e o modo descoberta.



08:51



10:22



11:00



“

Se eu não souber como a tarefa termina, não estou no modo entrega, estou no modo descoberta ainda.



Ritmo sustentável

O ritmo sustentável fala sobre trabalho de qualidade, sobre qual a quantidade de horas necessárias para entregar o máximo de qualidade e valor. Daniel explica que ritmo sustentável trata de tempo de presença, tempo e foco.



16:17



16:38

“

O normal é você ter trabalho de qualidade, tempo para respirar e para aprender e priorizar o que é mais importante para o seu sistema, para o seu produto.



18:22



A importância da multidisciplina é você saber que tem várias perspectivas participantes de um projeto e saber como você conecta elas.



“ O profissional W está preocupado com os papéis que ele pode desempenhar e com o crescimento de suas habilidades para se compor como uma pessoa que consegue atuar em diferentes perspectivas. ”

19:56

“ A pessoa desenvolvedora tem que entender qual é o papel dela com ela mesma e como se posiciona em relação a um time. ”

25:18

Comando e controle

Daniel ressalta que é preciso tomar cuidado com o comando e controle, como atuar com relação à liberdade de atuação dentro de uma equipe. Ele comenta que, muitas vezes, mesmo tendo uma estrutura altamente hierárquica, a empresa tem a habilidade de papéis e responsabilidades. A hierarquia não significa não ter liberdade, pode só determinar estruturas mais claras de responsabilidades.

Organizando equipes

Daniel fala sobre a forma de organização de equipes com diferentes tipos de pessoas. Ele explica que, com base nas topologias de equipe, podem ter equipes operantes no complicated subsystem, por exemplo. O Professor também fala sobre formas de trabalhar dentro de um time técnico um conceito de experimentação

26:54

33:11

31:06

“ É importantíssimo, dentro da sua estrutura de equipe, saber responder essa pergunta: como eu trago alguém que tem a primeira experiência de tecnologia para meu time? ”

34:38

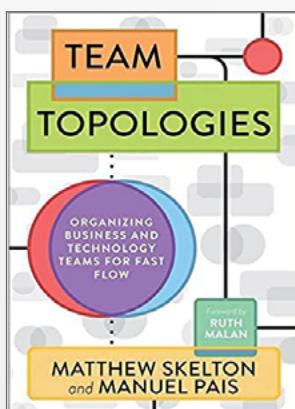
36:54

“ Quando você olhar o seu sistema, é preciso entender como é que você se posiciona, onde está no sistema. ”

“ Ter ciclos de aprendizado, ter ciclos de experimentação, saber que, eventualmente, iremos falhar e se recuperar de falhas, isso é relevante. ”

LEITURA INDICADA

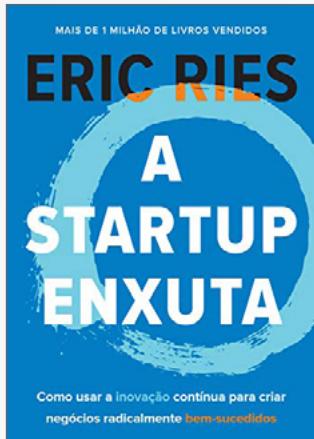
Livro: Team topologies



A obra "Team Topologies: Organizing Business and Technology Teams for Fast Flow" é de autoria de Matthew Skelton e Manuel Pais e foi publicado em 2019 pela It Revolution Press.

LEITURA INDICADA

Livro: A startup enxuta



A obra "A startup enxuta: Como usar a inovação contínua para criar negócios radicalmente bem-sucedidos" é de autoria de Eric Ries e foi publicada em 2019 pela Editora Sextante.

36:57



LEITURA INDICADA

Livro: O estilo startup



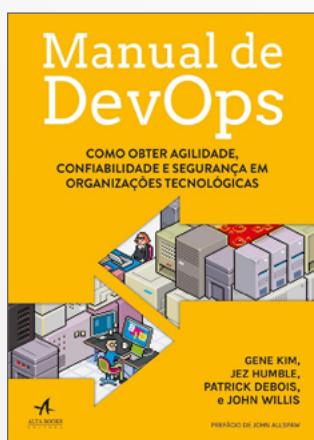
O livro "O estilo startup: Como as empresas modernas usam o empreendedorismo para transformar sua cultura e impulsionar seu crescimento" foi escrito por Eric Ries e publicado em 2019 pela Editora Sextante.

37:03



LEITURA INDICADA

Livro: Manual de DevOps



O livro "Manual de DevOps: como obter agilidade, confiabilidade e segurança em organizações tecnológicas" é de autoria de Gene Kim, Jez Humble, Patrick Debois e John Willis. Foi publicado em 2018 pela Alta Books Editora.

37:27



LEITURA INDICADA

Livro: O projeto fênix



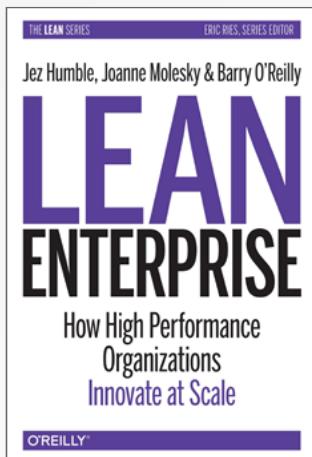
O livro "O projeto fênix: um romance sobre TI, DevOps e sobre ajudar o seu negócio a vencer" é de autoria de Gene Kim, Kevin Behr e George Spafford. Foi publicado pela Alta Books Editora em 2017.

40:47



LEITURA INDICADA

Livro: Lean Enterprise



De autoria de Jez Humble, Barry O'Reilly e Joanne Molesky, o livro "Lean Enterprise: How High Performance Organizations Innovate at Scale" foi publicado pela O'Reilly Media em 2015.

40:49

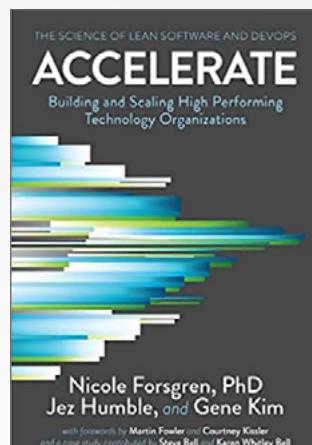


40:49



LEITURA INDICADA

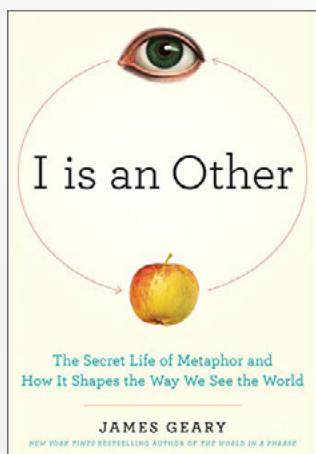
Livro: Accelerate



De autoria de Gene Kim, Jez Humble e Nicole Forsgren, o livro "ACCELERATE: The Science Of Lean Software And Devops Building And Scaling High Performing Technology Organizations" foi publicado em 2018 pela It Revolution.

LEITURA INDICADA

Livro: I Is an Other



O livro "I Is an Other: The Secret Life of Metaphor and How It Shapes the Way We See the World" é de autoria de James Geary e foi publicado em 2011 pela Harper.

AULA 1 • PARTE 3

00:47



01:36



Metáfora

Professor Guilherme Lacerda explica que a metáfora não é só uma figura de linguagem, ela está presente de forma intensa, porém imperceptível em tudo o que os humanos fazem. Guilherme fala sobre as metáforas utilizadas no desenvolvimento de software, como de arquitetura, programação, padrões, etc. As metáforas têm um grande valor, ampliam o poder de comunicação, permitem uma compreensão compartilhada e colaborativa e podem ser usadas em alto e baixo nível e em diferentes níveis de abstração.

“ O trabalho de uma equipe está muito mais relacionado com o produto dessas interações entre as pessoas do que com a soma dos esforços individuais. **”**

O trabalho da pessoa desenvolvedora

O Professor Guilherme fala sobre as principais atividades da pessoa desenvolvedora, que são: design/programação/testes/manutenção/evolução; gestão de configuração e do trabalho; colaboração com outros profissionais. Como desafios, ele destaca: dominar tecnologia(s), manter-se atualizada(o), ter a visão do todo, ser especialista/generalista, interagir com clientes/usuários, preocupar-se constantemente com a qualidade e se adaptar.

“ Precisamos considerar a qualidade como algo presente e essencial no nosso trabalho. **”**

“ Existem estatísticas que mostram que os desenvolvedores levam de 40 a 60% do tempo para fazer uma modificação porque não conhecem o padrão ou porque não existe um padrão. **”**

05:13



As metáforas trazem um grande valor, ampliam o poder de comunicação com o time.



07:02



07:21

10:16



É importante nos atualizarmos do ponto de vista tecnológico, mas mais importante é entender os fundamentos que estão por trás, é isso que vai me dar flexibilidade de aprendizado e de avanço em relação à minha carreira.



15:37

16:16



Padrões e convenções de código



16:55

Guilherme explica que toda e qualquer linguagem tem padrões e convenções de códigos e questiona o motivo de os desenvolvedores não utilizarem essa linguagem. Ele comenta que o padrão/convenção é uma prática do extreme programming, mas que ela também acaba incentivando outras práticas, como a posse coletiva. Os componentes da padronização/convenção podem ser nomenclatura, estrutura do código, terminologia, formatação, boas práticas e exemplos.

Pair e Mob programming

A adoção de padrões e convenções permite que outras pessoas também mexam no código e que o utilizem como um mecanismo de comunicação, comenta Guilherme. Dentro do extreme programming, existe a prática do pair programming, em que se pode exercitar aprendizado, mentoria, discutir problemas, entre outros, e auxiliar pessoas no processo de onboarding. O mob programming é uma variação do pair programming e é utilizada para envolver mais pessoas, inclusive as que não são da área do TI. Guilherme explica que o pair programming também é uma forma de fazer o modern code review, só que de uma forma antecipada.

18:23



O padrão, além de ser uma prática do extreme programming, ele incentiva outras práticas como a posse coletiva.



19:41

20:45



A programação em par incentiva muito o processo de troca e de colaboração.



22:17



Uma reunião de planejamento de time nada mais é do que uma definição tática de como um time vai trabalhar em cima de um escopo definido.



27:51



O pair programming é uma prática superimportante e relevante para as equipes hoje.



Pirâmide de revisão de código

Guilherme apresenta um modelo desenvolvido por Gunnar Morling chamado de pirâmide de revisão de código. O modelo apresenta questões relacionadas à apresentação, à semântica, à implementação, à documentação, ao estilo de codificação, entre outros. O Professor comenta que esse modelo é útil para organizar o processo, não só para entendimento do código, mas também para o compartilhamento de cenários e de situações para melhora.

32:28



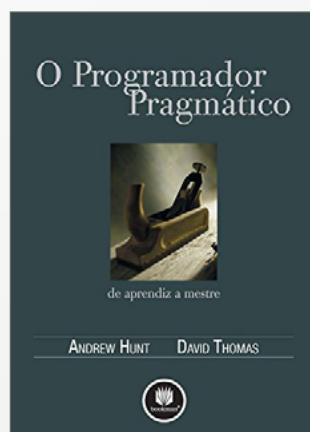
32:28

34:24



LEITURA INDICADA

Livro: **O programador pragmático**



O Programador
Pragmático



de aprendiz a mestre

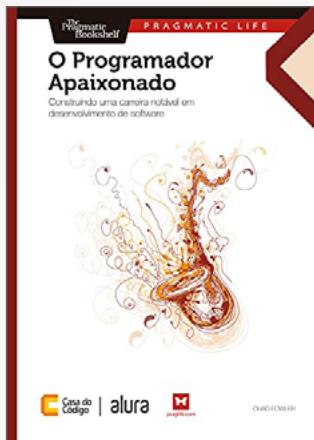
ANDREW HUNT DAVID THOMAS



O livro “O programador pragmático: de aprendiz a mestre” é de autoria de Andrew Hunt e David Thomas e foi publicado em 2010 pela Bookman.

LEITURA INDICADA

Livro: O programador apaixonado



O livro "O programador apaixonado: construindo uma carreira notável em desenvolvimento de software" foi escrito por Fowler Chad e publicado em 2014 pela Casa do Código.

36:24



36:56

LEITURA INDICADA

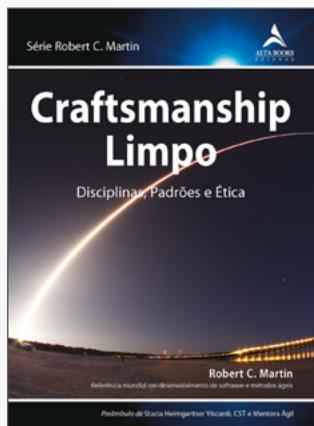
Livro: O codificador limpo



A obra "O codificador limpo: Um código de conduta para programadores profissionais" foi escrita por Bob Martin e publicado em 2012 pela Alta Books.

LEITURA INDICADA

Livro: Craftsmanship limpo



O livro "Craftsmanship limpo: disciplinas, padrões e ética" é de autoria de Robert C. Martin, foi publicado em 2022 pela Alta books.

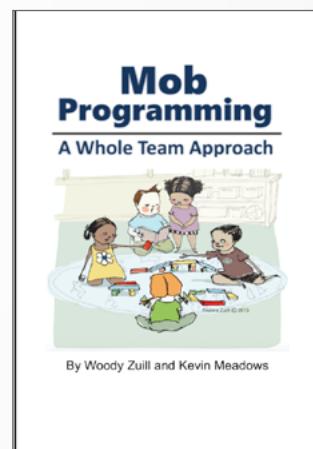
37:22



37:59

LEITURA INDICADA

Livro: Mob programming



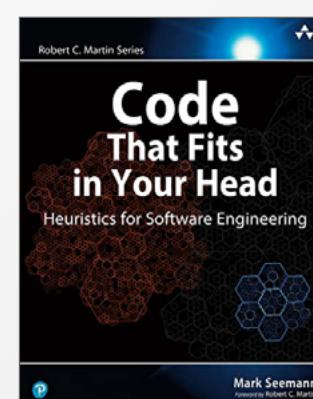
O livro "Mob programming: a Whole team approach" é de autoria de Woody Zuill e Kevin Meadows.

38:59



LEITURA INDICADA

Livro: Code that fits in your head



O livro é escrito por Mark Seemann e foi publicado em 2021 pela Addison-Wesley Professional.

AULA 1 • PARTE 4

Bad smells

Professor Guilherme explica que um código ruim se revela de diferentes formas e envolve questões como legibilidade (nomenclatura) e complexidade das estruturas. Esse fenômeno recebe o nome de bad smells e eles estão presentes nos principais estudos dos últimos 20 anos. O termo começou a ser usado no final dos anos 1990, no livro de Martin Fowler “Refatoração”. Guilherme explica o estudo que ele desenvolveu, que investigou o que conectava os code smells e a refatoração.



00:28

04:36



O smell pode ser um sintoma, um indicativo de um potencial problema de design; um bug não, um bug se revela quando executamos o software.



16:26



Tanto smells quanto refatoração é como se fossem faces de uma mesma moeda, onde o que liga esses dois temas são aspectos de qualidade.



“ Se não soubermos o tipo de refatoração que estamos fazendo, podemos estar criando outros problemas no nosso código. ”



19:12

22:54



Engenharia de software

A engenharia de software é uma disciplina madura, mas emergente, explica Guilherme. O software é um produto diferente de outros produtos de engenharia, por isso é importante entender a sua natureza. Com isso, é possível entender como operar no dia a dia, com os nossos times e práticas. Guilherme comenta que o software precisa atender ao requisito de negócio.

EXERCÍCIO DE FIXAÇÃO

De acordo com Guilherme Lacerda, com o que mais está relacionado o trabalho em equipe?



Com a soma dos esforços individuais.

Com o produto das interações entre as pessoas.

Com o poder de comunicação com o time.

Com a compreensão compartilhada e colaborativa.

26:58



Software é um produto virtual ausente de propriedades físicas.



29:05



Se não definirmos a forma como o time irá operar junto, como serão esses acordos de trabalho, iremos perder produtividade.



Complexidade

Com base na obra de Frederick Brooks, o Professor Guilherme explica que a complexidade que enfrentamos nos projetos de software pode ser dividida em complexidade essencial e em complexidade accidental. A complexidade essencial é aquela da qual não se pode fugir, que faz parte dos negócios, com aquela que precisamos lidar; já a complexidade accidental é aquela que inserimos, são as ferramentas e práticas escolhidas. Guilherme também apresenta as leis de evolução de Lehman que permeiam o desenvolvimento de software, como a lei de mudança contínua e o aumento da complexidade que, segundo o Professor, é o dilema do desenvolvedor.



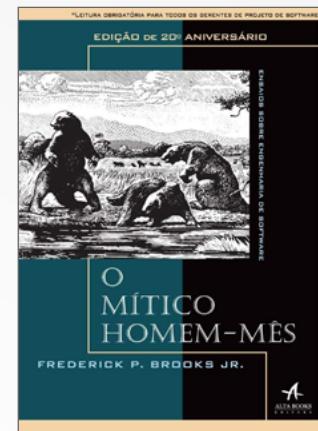
32:35



32:59

LEITURA INDICADA

Livro: O mítico homem-mês



O livro “O mítico homem-mês: ensaios sobre engenharia de software” é de autoria de Frederick Brooks e foi publicado em 2018 pela Alta Books.



Software que não é modificado perde a importância, vai morrer.



36:35



37:40

“Na medida em que modificamos o software, estamos aumentando sua complexidade.”



38:40



40:38

“As manutenções de caráter proativo são aquelas que deveríamos ter consciência de trabalhar.”



42:59

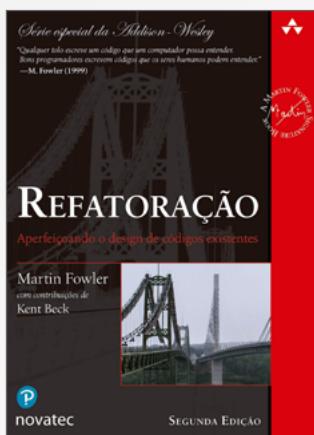
“Com o surgimento do desenvolvimento ágil, não existe mais essa separação entre desenvolvimento e manutenção.”

Manutenção

Guilherme apresenta um recorte do SWEBoK de 2014 — documento produzido pela comunidade de engenharia de software no início dos anos 2000, guia que descreve o corpo de conhecimento da engenharia de software — e explica que as categorias de manutenção podem ser corretivas ou de aprimoramento, e serem realizadas de forma reativa ou proativa. As manutenções de caráter corretivo reativo ou de aperfeiçoamento reativo são fáceis de perceber, pois são aquelas que o time acaba dando mais atenção. No entanto, as manutenções de caráter proativo são aquelas que deveríamos ter consciência de trabalhar. As ações de aprimoramento e proativas é onde se enquadra a refatoração.

LEITURA INDICADA

Livro: Refatoração



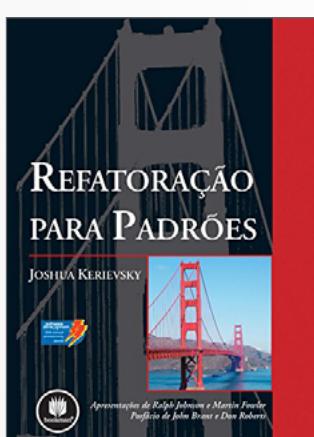
De autoria de Martin Fowler, o livro “Refatoração: Aperfeiçoando o Design de Códigos Existentes” foi publicado em 2020 pela Novatec Editora.



55:06

LEITURA INDICADA

Livro: Refatoração para padrões



De autoria de Joshua Kerievsky, o livro foi publicado em 2008 pela Bookman.



55:15

LEITURA INDICADA

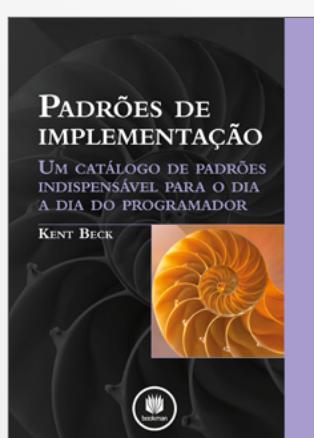
Livro: Código limpo



A obra “Código limpo: Habilidades práticas do Agile Software” é de autoria de Robert Martin e foi publicado em 2009 pela Alta Books.

LEITURA INDICADA

Livro: Padrões de implementação



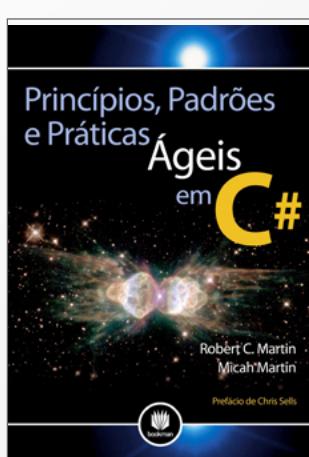
De autoria de Kent Beck, o livro foi publicado em 2013 pela Bookman.



56:11

LEITURA INDICADA

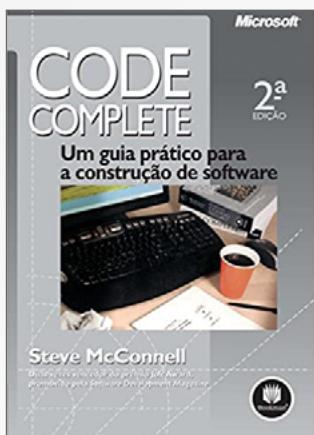
Livro: Princípios, padrões e práticas ágeis em C#



Publicado em 2011 pela Bookman, o livro é de autoria de Robert Martin e Micah Martin.

LEITURA INDICADA

Livro: **Code Complete**



De autoria de Steve McConnell, o livro "Code Complete: um guia prático para a construção de software" foi publicado em 2005 pela Bookman.



57:11 57:33

LEITURA INDICADA

Livro: **Java Efetivo**



O livro "Java Efetivo: as melhores práticas para a plataforma Java" é escrito por Joshua Bloch, foi publicado em 2001 pela Alta Books.

AULA 2 • PARTE 1



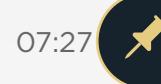
01:07 01:28 05:33

Testes de software

Professor Daniel explica que existem muitas perspectivas, formatos e tipos de testes que podem ser realizados. Os testes precisam bater com diferentes partes do sistema, em qualquer linguagem que se trabalhar vai existir uma ferramenta de cobertura de teste que nos habilita a entender por onde o código está passando e também onde não está. Isso ajuda a entender quais cenários devem ser criados, além dos que já existirem, para garantir uma cobertura de testes ideal.

“ Um dos nossos desafios enquanto pessoas desenvolvedoras de software é poder desenvolver e entender quais são essas técnicas que queremos trabalhar. ”

“ Cem por cento de cobertura de teste não significa assertividade do código. ”



07:27

Hipótese do Guilherme

Daniel apresenta a teoria do Professor Guilherme Lacerda de que, se eu tiver um primeiro requisito de software a ser construído, se eu fizer o código e fizer o teste, quando eu fizer um segundo requisito de software, não posso só fazer o teste do segundo requisito, é preciso fazer também o teste do primeiro e do segundo.

Testes ou design?

Daniel comenta que a discussão ao construir um teste é sobre design de código e que os testes automatizados representam design. Ele também fala sobre a importância das perguntas e explica que, para saber se a tarefa foi concluída, pode-se utilizar as perguntas de apoio: como sabemos que terminamos a nossa tarefa? Como sabemos que estamos preparados para iniciar? Daniel destaca a importância dos critérios de aceite e comenta que eles ajudam a cadenciar o trabalho e a fazer somente o necessário, além de que os cenários demonstram exemplos que definem os limites do nosso trabalho.



09:00

**EXERCÍCIO DE FIXAÇÃO**

De acordo com o Professor Guilherme, quais são as duas leis de Lehman consideradas o dilema do desenvolvedor?

Conservação da familiaridade e estabilidade organizacional.

Evolução de programa de grande porte e crescimento contínuo.

Aumento da complexidade e mudança contínua.

Declínio de qualidade e sistema de feedback.

“ Os critérios de aceite vão ajudar a cadenciar o trabalho e ajudar a gente a fazer somente o necessário. **”**



14:13

“ Quando estamos coletando histórias e entender um determinado cenário do cliente, temos que buscar fazer perguntas abertas. **”**



14:55

15:35



17:56

Automação e integração contínua

Daniel fala sobre o tempo que o código se mantém na máquina, sobre a entrada do código na base de produção, sobre a geração de valor a partir da entrega e também sobre o funcionamento dos builds da equipe.

“ Que estruturas eu tenho de apoio para garantir que o que estou fazendo está indo com a melhor qualidade possível para o controle de versão e podendo chegar mais próximo dos meus clientes? **”**

19:00



Pipeline de entrega

A partir do código fonte, que está rodando localmente, conseguir fazer atualizações e fazer o código chegar para quem ele tem que chegar, esse é o trabalho desejado, explica Daniel. Para conseguir fazer isso, é preciso automatizar a infraestrutura e o processo de liberação de código, tomar conta do desenvolvimento e liberação de produto, entre outras ações. Uma entrega, segundo Daniel, vai acabar sendo um potencial para novas ideias e novas entregas. Com relação ao processo de automação do building, o Professor explica que ele pode ser uma construção de forma interativa.

20:29



Só sabemos se realmente está pronto o que fizemos quando estiver em uso, em produção, e alguém estiver tendo valor a partir do trabalho realizado.

23:42



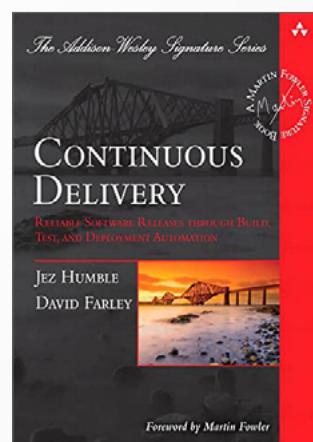
Todo produto moderno vai ter algum tipo de estratégia para validar e poder constituir a liberação de funcionalidade até que ela seja liberada para todo mundo.

25:56



LEITURA INDICADA

Livro: *Continuous delivery*



O livro “*Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*” é de autoria de Jez Humble e David Farley e foi publicado em 2011 pela Pearson.

AULA 2 • PARTE 2

Escrita de histórias

Professor Daniel explica que a escrita de histórias se refere a fazer perguntas para poder descobrir o que é de interesse dos clientes, quem queremos impactar, qual é o objetivo dessa pessoa, entre outros. Para descobrir essas informações, o Professor Daniel sugere o uso da ferramenta do 5W2H: quem queremos impactar? Que atividades queremos impactar? Que valor vamos gerar? O que precisa ser feito? Como sabemos se fizemos o que precisa ser feito? Para descobrir quem atendemos, devemos nos questionar qual é o objetivo de uma persona, qual é o detalhamento e também proporcionar abertura para conversas.



00:44

02:25



Quando eu modelo uma história de usuário, devo modelá-la de um jeito que se torne testável.



EXERCÍCIO DE FIXAÇÃO

Quais podem ser os componentes da padronização/convenção de código?



Estrutura do código, terminologia, formatação, motivação e mecânica.



04:54

Resumo, nomenclatura, boas práticas, motivação e exemplos.



Nomenclatura, estrutura do código, terminologia, formatação, boas práticas e exemplos.

Estrutura de código, resumo, nomenclatura, mecânica e motivação.

VÍDEO

Where Does Growth Come From?



Neste vídeo, Clayton Christensen apresenta um novo conteúdo sobre diferentes maneiras de pensar sobre o crescimento e compartilha um pouco de sua perspectiva única sobre “medir sua vida”.

“ A história de usuário é um conceito em que eu consigo pegar um cenário que o meu sistema tem, de alguma coisa que alguém quer realizar. ”

06:07



LEITURA INDICADA

Livro: Value Proposition Design



O livro “Value Proposition Design: Como construir propostas de valor inovadoras” é de autoria de Alex Osterwalder, Greg Bernarda, Yves Pigneur e Alan Smith e foi publicado em 2019 pela Alta Books.

Cenários e testes

Os testes ajudam a validar o nosso progresso e os cenários devem representar exemplos reais de clientes. Para isso, Daniel recomenda que se usem perguntas para guiar a construção, como:

- Dado que? Em que estado está o sistema, para iniciarmos o trabalho?
Qual a pré-condição?
- Quando? Qual ação está acontecendo no sistema?
- Então? Como ficou o sistema depois da ação realizada? O que é esperado?
Qual foi a reação do sistema?

10:56



12:35



“ A preocupação, quando montamos cenários de testes, é descobrir o que significa esse “então”. ”

15:28



“ É nossa tarefa como pessoas desenvolvedoras de software garantir a independência entre os testes. ”

16:52



“ É importante que eu entenda qual o benefício eu deveria gerar para saber se aquele código que eu fiz é capaz de fazer isso ou não. ”

17:41



Ritmo do TDD

19:57



“ Quando eu quero aprender uma linguagem de programação nova, normalmente o que eu faço é pegar problemas de coding DOJO que eu já resolvi em outras linguagens e eu tento resolver esses problemas na linguagem nova. ”

22:42



“ Essa diferenciação entre instalar software e liberar software é altamente importante. ”

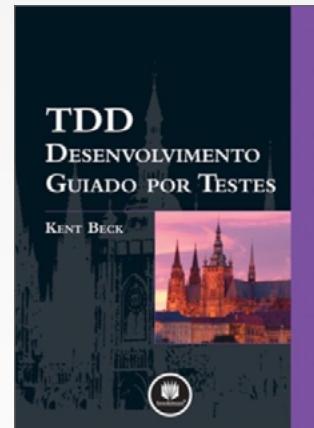
Nosso progresso de trabalho acontece de cenário de teste em cenário de teste a ser construído, explica o Professor Daniel. Quando um teste não passa, você tem oportunidade de modificar comportamento; quando seus testes passam, você tem oportunidade de modificar a estrutura. O professor também fala sobre validação no mundo real e explica que, para isso, é preciso constituir a diferença do que é um deploy e do que é release. Para validar funcionalidades sem impactar todos os clientes, é necessário lançar mão de estratégias de liberação e de feature toggles.

25:18



LEITURA INDICADA

Livro: TDD: desenvolvimento guiado por testes



Exemplo prático

O Professor Daniel apresenta um exemplo prático de uma conversa realizada com um cliente para a disponibilização de uma funcionalidade em um caixa eletrônico.

27:31



37:14

“Quando estamos começando a aprender sobre teste, é importante usar um conceito que é baby steps.”

AULA 2 • PARTE 3

03:46



Dívida técnica

Criada por Ward Cunningham e originária do setor financeiro, a dívida técnica é uma metáfora. A dívida técnica traz a percepção de que, muitas vezes, temos que tomar uma decisão em relação ao nosso projeto, consciente ou inconsciente, mas que, dependendo do momento, de como aquilo aconteceu, logo em seguida, teremos de pagar essa dívida. O fato de precisarmos lidar com a dívida técnica vai, aos poucos, endereçando baixa moral e motivação na equipe, afeta a baixa qualidade do código e endereça uma baixa produtividade. O Professor Guilherme também apresenta os quadrantes para lidar com a dívida técnica, que podem ser: negligente, proposital, imprudente e prudente.

03:31



03:31

“Grande parte da nossa atividade enquanto pessoa desenvolvedora de software é ler e dar manutenção a códigos das outras pessoas.”

08:17



“O fato de termos que lidar com a dívida técnica vai, aos poucos, endereçando baixa moral e motivação na equipe, afeta a baixa qualidade do código e endereça uma baixa produtividade.”

Tipos de dívida técnica

O Professor Guilherme apresenta alguns tipos de dívida técnica, como teste, building manual, arquitetura, design e codificação. Guilherme explica que, em razão do crescimento da área em pesquisa, é preciso identificar algumas das atividades atreladas à dívida técnica, como identificação, medição, priorização, prevenção, monitoramento, pagamento, documentação e comunicação.

12:50



15:03



15:03

“A priorização da dívida técnica é considerada a atividade mais importante.”

“ Não conseguiremos terminar com a dívida técnica, mas precisamos saber como lidar com a dívida técnica. **”**

17:01

“

22:54

“ A dívida técnica precisa ser encarada como um problema de gestão, pensando na evolução dos produtos e serviços que são oferecidos. **”**

EXERCÍCIO DE FIXAÇÃO

De acordo com Frederick Brooks, a complexidade existente nos projetos de software pode ter dois tipos. Quais são eles?

Contínua e accidental.

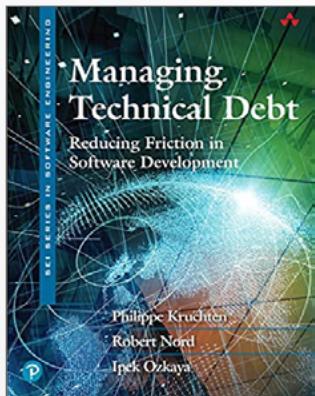
Diversa e essencial.

Acidental e diversa.

Essencial e accidental.

LEITURA INDICADA

Livro: Managing Technical Debt



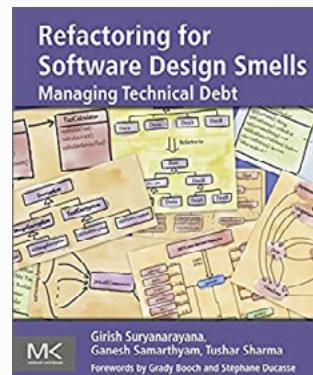
O livro “Managing Technical Debt: Reducing Friction in Software Development (SEI Series in Software Engineering)” é de autoria de Philippe Kruchten, Robert Nord e Ipek Oskaya e foi publicado em 2019 pela Addison-Wesley Professional.

25:13



LEITURA INDICADA

Livro: Refactoring for Software Design Smells



25:48



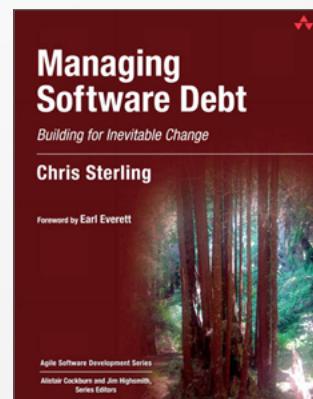
De autoria de Girish Suryanarayana, Ganesh Samarthyam e Tushar Sharma, o livro “Refactoring for Software Design Smells: Managing Technical Debt” foi publicado em 2014 pela Morgan Kaufmann.

26:06



LEITURA INDICADA

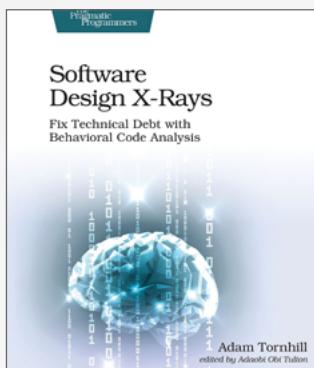
Livro: Managing Software Debt



De autoria de Chris Sterling, o livro “Managing Software Debt: Building for Inevitable Change (Agile Software Development)” foi publicado em 2010 pela Addison-Wesley Professional.

LEITURA INDICADA

Livro: Software Design X-Rays



O livro "Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis" é de autoria de Adam Tornhill e foi publicado em 2018 pela Pragmatic Bookshelf.



26:18



27:44

Refactoring

A refatoração é um processo complexo, explica o Professor Guilherme. O termo tem sido utilizado na indústria, mas, com o tempo, ele foi sendo deturpado. Hoje, qualquer alteração em código é chamada de refatoração, mas não é, explica Guilherme. A refatoração está relacionada a uma mudança de estrutura interna, mas o comportamento observável tem que se manter o mesmo. O Professor apresenta os dados de algumas pesquisas sobre refatoração realizadas.



O grande lance da refatoração é que, ao refatorar, eu vou rodar a switch de testes e ela vai continuar passando.



29:22



32:08



O grande motivador para a realização da refatoração é a questão da qualidade, a melhoria do design.



33:13

Código legado

O Professor Guilherme fala sobre a forma como o autor Michael Feathers lida com o código legado. O autor recomenda que se refatore e que se realizem testes, que se considere o tempo de latência, entre outras ações.



34:00

LEITURA INDICADA

Livro: Trabalho eficaz com código legado



Se estou mexendo em um código legado, precisaria ter uma forma para saber se estou no caminho ou não.



35:43

O livro é de autoria de Michael C. Feather e foi publicado em 2013 pela Bookman.

AULA 2 • PARTE 4

Análise de código

Guilherme comenta que a análise de código é uma disciplina indispensável para pessoas desenvolvedoras de software. Ela auxilia a ampliação da capacidade cognitiva de programação, o conhecimento de outros paradigmas, padrões e linguagens (e problemas também) e a ampliação de habilidades. O Professor explica que, entre as habilidades que precisam ser desenvolvidas, estão: conhecer heurísticas de análise para as estruturas (módulos/pacotes, classes, métodos), compreender aspectos de qualidade de software (atributos externos e internos), aplicar métricas de análise, estratégias de visualização e ferramentas de apoio (compreensão de software, mineração de repositórios) e estratégias (análise estática, análise dinâmica, análise temporal, análise comportamental).

Guilherme fala também sobre as estratégias utilizadas para aumentar a percepção de análise de código, como a análise estática, a dinâmica, a histórica e a comportamental.

“ As pessoas estão muito mais preocupadas em adotar uma nova tecnologia do que entender o paradigma que está por trás. ”

“ A análise de código ela deveria ser feita sempre, de forma antecipada. ”

“ A gente precisa criar mecanismos no nosso fluxo de trabalho para permitir que a gente possa fazer análise de código. ”



00:46



05:19



“ A análise estática é utilizada quando estamos trabalhando no código fonte ou no binário. ”



12:20

Pilares da análise de código

14:37



“ Observem esses elementos que fazem parte dos pilares da análise, mantenham eles na visão de vocês. ”

17:57



16:24

O Kata

20:37



21:23

O Professor Guilherme explica que o Kata é um método de treinamento de comportamento e de automatização de determinadas práticas que pode ser utilizado na análise de código. O primeiro passo, é definir um objetivo para análise; em seguida, deve-se rodar as ferramentas de análise para encontrar o ponto desejado (níveis de granularidade, uso de testes automatizados e padrões adotados). É possível começar de dentro para fora, ou seja, analisando as funções/métodos, as estruturas e o design; considerando os pilares; e subindo a granularidade, quando necessário.

EXERCÍCIO DE FIXAÇÃO

De acordo com Guilherme Lacerda, quais são as categorias de manutenção que toda pessoa desenvolvedora de software deveria ter a consciência de trabalhar?

Manutenções corretivas reativas.

Manutenções de aprimoramento.

Manutenções proativas.

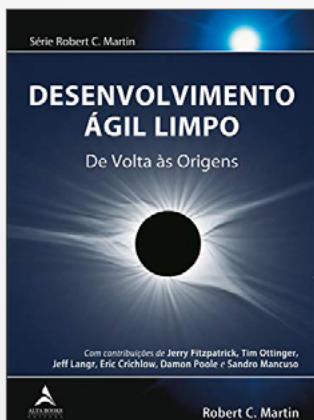
Manutenções de aperfeiçoamento reativo.

Liderança técnica

Guilherme fala sobre a importância de desenvolver habilidades comportamentais, aspectos fundamentais para quem deseja ser um líder técnico. O professor cita também os conhecimentos tácitos e explícitos. Finalizando, ele traz uma reflexão posta no livro de Robert Martin “Desenvolvimento ágil limpo”: os desenvolvedores estão se distanciando da agilidade ou a agilidade está se distanciando dos desenvolvedores?

LEITURA INDICADA

Livro: Desenvolvimento ágil limpo



O livro “Desenvolvimento ágil limpo: de volta às origens” é de autoria de Robert Martin e foi publicado em 2020 pela Alta Books.

22:40



DR-Tools

O Professore Guilherme apresenta o DR-Tools, um software criado por ele para criar uma comunidade de discussão de habilidades técnicas e de desenvolvimento necessárias para desenvolvedores profissionais.

42:10



Algumas dicas

Professor Guilherme dá algumas dicas pensando na evolução do desenvolvimento de software. Ele fala sobre a regra dos 30 segundos, a regra do escoteiro/refatoração oportunista e sobre a metáfora do jornal. Sugere que se olhe código de outros (principalmente projetos open-source) e que, ao pegar um código da Web, ele seja ajustado. O Professor também recomenda que o profissional em desenvolvimento conheça suas ferramentas; use automação em diferentes níveis; defina políticas de qualidade; ao analisar o código, marque pontos para discussão com o time; estude e pratique, monte o plano de metas com o time; crie uma rotina com o time para discutir problemas, práticas e ferramentas; experimente (novas LPs, IDEs, ambientes) através de Dojos; participe das comunidades e eventos; e entenda que é uma jornada a seguir.

46:27



46:47



48:39

“É da natureza da nossa carreira, do nosso papel, evoluir, mas também precisamos desenvolver habilidades comportamentais.”

AULA 3 • PARTE 1

02:02



O extreme programming

Professor Michael inicia sua aula falando sobre o surgimento do eXtreme programming (XP). Em meados de 1990, Kent Beck buscou formas simples e eficientes para desenvolver softwares. Em março de 1996, os projetos com novos conceitos resultaram na metodologia XP. O XP é uma metodologia ágil, leve desenvolvida para equipes pequenas e médias (2 a 12 pessoas) e para requisitos vagos e em constante evolução. É baseada nos seguintes valores: simplicidade, comunicação, coragem e feedback.

04:50



A comunicação entre os membros da equipe é central no XP também.

05:37



O ciclo de melhoria contínua é algo presente no XP.

07:14



Um pouco dessa defesa do XP é a defesa da redução de custos, à medida que eu tenha mudanças ao longo do projeto.

07:29



Boas práticas do XP

O professor Michael explica que o XP se organiza em torno de algumas boas práticas. Entre elas, estão: planning game, small releases, metaphor, simple design, test first design, refactoring, stand-up meeting, continuous integration, pair programming, move people around, collective code ownership, coding standards, 40-hour week, on-site customer e acceptance tests.

08:39



O planning game tem essa finalidade: definir o conjunto de histórias que a release vai entregar.

12:04



Com essa junção de teste unitário, teste automatizado e refatoração, colocamos em evidência a possibilidade de refatorar e isso traz um ganho muito interessante.

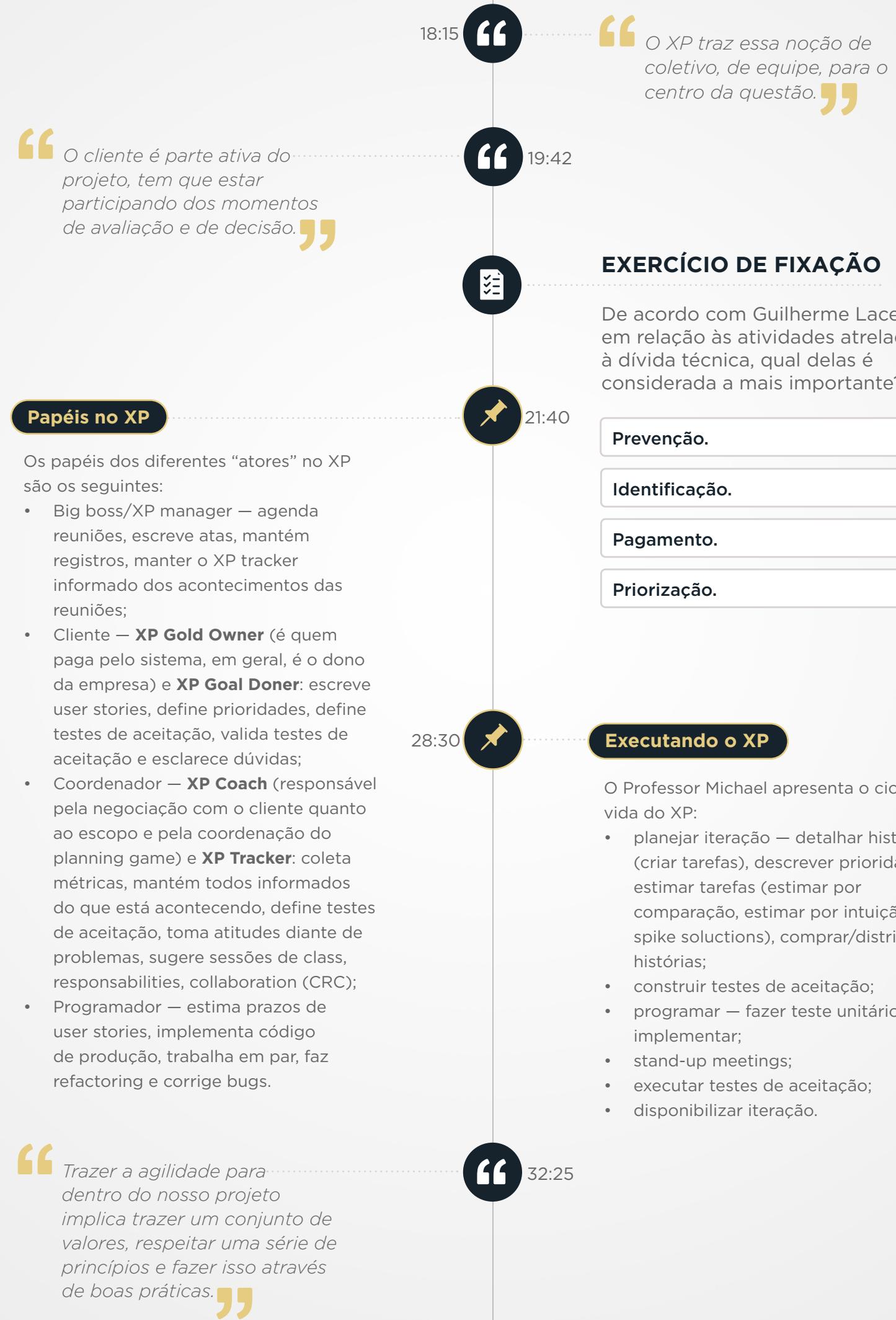
12:43



Essa prática de programação em pares é uma das boas práticas do XP.

15:58

As reuniões de alinhamento, de stand-up meeting e as reuniões diárias têm esse papel forte tanto na manutenção da comunicação constante, como também na questão do feedback.



33:30



Testes unitários

Os testes unitários são fundamentais para reduzir a quantidade de erros no código, reduz a ocorrência de erros simples e a incidências no momento de integração. O Professor Michael explica que a ideia do teste unitário é testar o componente individual. Michael explica que, para construir o teste unitário, é necessária a especificação do módulo antes da sua implementação e para a execução, o código fonte do módulo. Com relação à nomenclatura, as classes drivers são aquelas que contém os casos de teste e que procuram exercitar os métodos da classe “alvo” buscando detectar falhas; já os dublês simulam o comportamento de classes necessárias ao funcionamento da classe “alvo” e que ainda não foram desenvolvidas.

34:13



38:15



AULA 3 • PARTE 2

Classes driver

Michael explica que, para criar uma classe driver é necessário ter uma especificação da classe “alvo”. A partir do projeto de uma classe, pode-se especificar os casos de teste. Deve-se criar um conjunto de casos de teste capaz de cobrir as funcionalidades básicas da classe. Como vantagens do uso das classes drivers, estão a exigência de reflexão sobre as funcionalidades da classe e sua implementação antes de seu desenvolvimento, permissão da identificação rápida de bugs mais simples, garante que a classe cumpra um conjunto de requisitos mínimos (os garantidos pelos testes) e facilitação da detecção de efeitos colaterais no caso de manutenção ou refactoring. Como dificuldades no uso dessas classes, Michael destaca a necessidade de construção do “cenário” em cada método, a necessidade de construção de um programa para execução dos casos de teste, dificuldade em se trabalhar com grandes conjuntos de dados de teste, dificuldade para coletar os resultados e dificuldade para automatizar a execução dos testes.

01:42



05:31



A linguagem de construção da classe driver é a mesma linguagem de construção da classe alvo, sem nenhuma ferramenta adicional.

07:15



Práticas como refatoração, alteração de código e integração contínua são viabilizadas para esse tipo de teste [unitário].

XUnit

O XUnit é um padrão de ferramenta que surgiu no final dos anos 1990 para facilitar a implementação de códigos e testes, principalmente testes unitários e testes de integração (estruturados, eficientes e automatizados). Sua concepção adapta-se facilmente aos IDEs de desenvolvimento. Professor Michael apresenta algumas recomendações para a implementação de testes unitários, como projetar casos de teste independentes uns dos outros, não testar apenas o “positivo”, mas garantir que seu código responda adequadamente em todos os cenários, criar um driver para cada classe, incluir o nome do método em cada teste, depurar os testes quando for o caso e não se esquecer de que os testes também são código. O Professor ainda fala sobre os limites, as anotações, as asserções e os recursos do JUnit.

“ Se queremos, de fato, implementar os testes de forma unitária, temos que tentar não cruzar certos limites. ”

Processo de teste unitário

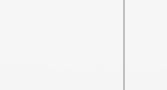
Para o processo de desenvolvimento com teste unitário, é preciso seguir alguns passos:

- definir a interface (esqueleto) da classe alvo;
- definir o conjunto de casos de teste;
- implementar a classe driver;
- completar a codificação da classe alvo;
- executar os testes;
- corrigir os bugs, se houverem;
- complementar os testes.

“ Todo esse cenário de classes driver, classes duplê e classes alvo, é o que vai viabilizar fazer o que o XP preconiza. ”



09:23



EXERCÍCIO DE FIXAÇÃO

Qual das disciplinas é considerada indispensável para pessoas desenvolvedoras de software segundo o Professor Guilherme Lacerda?

Refatoração.

Desenvolvimento de habilidades comportamentais.

Análise de código.

Código legado.

“ É necessário que projetemos casos de testes independentes uns dos outros. ”

“ Não se esqueça de, ao construir um nome de método de teste, deixe claro que é um método de teste. ”

“ É mais difícil introduzir erros nos códigos de testes quando esses testes estão implementados respeitando os limites colocados. ”

Outras possibilidades do JUnit

O JUnit5 oferece uma série de recursos adicionais para facilitar a implementação dos casos de teste que permitem, entre outros, a verificação de exceções, a verificação do tempo de execução e a comparação de coleções. Na verificação de exceções, o teste irá reportar um erro se a exceção não for gerada. Para testes que devem falhar quando uma exceção é gerada, basta deixar a exceção ocorrer. Na verificação do tempo de execução, pode-se verificar se um método não está ultrapassando o tempo de execução previsto para ele.

AULA 3 • PARTE 3

00:25



Técnicas de teste funcional e estrutural

As técnicas de teste funcional usam como entrada a especificação do módulo. Professor Michael explica que, nos casos de especificações baseadas em contratos, é fundamental gerar casos de teste com valores válidos buscando verificar se o módulo se comporta como especificado.

No caso de programação por contratos deve-se gerar casos de teste que busquem verificar se a implementação atende as especificações do contrato.

Michael destaca a importância de testar os valores limites, as classes de equivalência, os diagramas de estado, as condições de erro e os valores inválidos. A técnica de teste estrutural garante a cobertura do teste e usa como entrada o código fonte. Esse tipo de estrutura é usado para refinar os casos de teste, gerar o grafo de programa e procurar garantir a cobertura do grafo de programa

01:42



Devo sempre me lembrar de testar os limites dos dados.

06:47



Refatorar constantemente, frequentemente, é uma prática bem-vinda de melhoria da qualidade e de simplificação do código.

07:10



Refatoração

A refatoração é uma (pequena) modificação no sistema que não altera o seu comportamento funcional, mas que melhora alguma qualidade não-funcional (simplicidade, flexibilidade, clareza, desempenho). Alguns exemplos de refatoração são mudança do nome de variáveis, mudanças nas interfaces dos objetos, pequenas mudanças arquiteturais, encapsular código repetido em um novo método e generalização de métodos. Suas aplicações podem melhorar código antigo e/ou feito em ciclos de desenvolvimento anteriores e atuar no desenvolvimento incremental. O Professor ainda fala sobre os passos da refatoração e sobre quando utilizá-la.

07:27



Refatoração é a ideia de alterar o código sem alterar sua funcionalidade.

09:10



Refatoração é aquela pequena alteração que, aparentemente, naquele momento tem pouco impacto, mas quando começo a acumular várias pequenas alterações, aquilo faz diferença na qualidade do código.

EXERCÍCIO DE FIXAÇÃO

Assinale a alternativa que apresenta algumas das boas práticas nas quais o XP está alicerçado.

Planning game, small releases, metaphor, 40-hour week.

Simple design, test first design, refactoring, move people around.

Stand-up meeting, continuous integration, pair programming, collective code ownership.

Todas as alternativas estão corretas.

Catálogo de refatorações I

Michael explica que, antes de começar a refatoração, é preciso verificar se você tem um conjunto sólido de testes para verificar a funcionalidade do código a ser refatorado. Ele destaca que refatorações podem adicionar erros, mas que os testes vão ajudá-lo a detectar erros se eles forem criados. O Professor apresenta também a estrutura típica para descrever uma refatoração de um catálogo, composta por nome, resumo, motivação, mecânica e exemplos. Ele demonstra alguns exemplos dos catálogos de refatoração, como o Extract Method, o Inline Method, o Replace Temp with Quaery, o Replace Inheritance With Delegation, o Collpase Hierarchy, o Replace Conditional With Polymorphism e o Introduce Null Object.

12:29



20:15



20:25



23:53



28:53



Ter um bom vocabulário de refatorações, um bom conjunto de refatorações possíveis e saber quando aplicá-las criteriosamente e sistematicamente acaba sendo uma grande qualidade.



Refatoração é alterar o código sem mudar a funcionalidade.



Extrair trechos do método e criar métodos para ele ajuda até a documentar aquele código.



Refatoração é um processo de escolha, eu vou escolher o que aplicar, onde aplicar e quando aplicar.

AULA 3 • PARTE 4

Catálogo de refatorações II

O Professor Michael segue demonstrando alguns exemplos dos catálogos de refatoração, o Inline Method, o Replace Temp with Quaery, o Replace Inheritance With Delegation, o Collpase Hierarchy, o Replace Conditional With Polymorphism e o Introduce Null Object.

00:25



05:27



Não existe a refatoração certa e errada, existem catálogos, existem cenários, onde eu vou ter que avaliar se vale a pena utilizar ou não aquela abordagem.

EXERCÍCIO DE FIXAÇÃO

Marque a alternativa que apresenta alguns exemplos de refatoração.

Encapsulamento de código repetido em um novo método e atuação no desenvolvimento incremental.

Melhoria do código antigo e/ou feito em ciclos de desenvolvimento anteriores.

Generalização de métodos.

Mudança do nome de variáveis, mudanças nas interfaces dos objetos, pequenas mudanças arquiteturais.



22:07



Considerações finais

O Professor Michael comenta que refatoração muda o programa em passos pequenos, se você comete um erro, é fácil consertar. Qualquer um pode escrever código que o computador consegue entender, bons programadores escrevem código que pessoas conseguem entender. Michael ainda comenta que, quando você sente que é preciso escrever um comentário para explicar o código melhor, tente refatorar primeiro.

Resumo da disciplina

Veja, nesta página, um resumo dos principais conceitos vistos ao longo da disciplina.

AULA 1

O desafio das pessoas desenvolvedoras é identificar com os clientes as perguntas que podem ajudá-las a se conectar com o problema.



A programação em par incentiva muito o processo de troca e de colaboração.



Na prática das metodologias ágeis, é importante ter ciclos de aprendizado, ter ciclos de experimentação e saber que, eventualmente, haverá falhas, mas que elas serão recuperadas.

AULA 2

É tarefa das pessoas desenvolvedoras de software garantir a independência entre os testes.



Só é possível saber se o que foi feito realmente está pronto quando estiver em uso, em produção, e alguém estiver tendo valor a partir do trabalho realizado.



O grande motivador para a realização da refatoração é a questão da qualidade, a melhoria do design.

AULA 3

O ciclo de melhoria contínua é algo presente no extreme programming.



Trazer a agilidade para dentro do projeto implica trazer um conjunto de valores, respeitar uma série de princípios e fazer isso através de boas práticas.

Ter um bom vocabulário de refatorações, um bom conjunto de refatorações possíveis e saber quando aplicá-las criteriosamente e sistematicamente acaba sendo uma grande qualidade.



Avaliação

Veja as instruções para realizar a avaliação da disciplina.

Já está disponível o teste online da disciplina. O prazo para realização é de **dois meses a partir da data de lançamento das aulas**.

Lembre-se que cada disciplina possui uma avaliação online.
A nota mínima para aprovação é 6.

Fique tranquilo! Caso você perca o prazo do teste online, ficará aberto o teste de recuperação, que pode ser realizado até o final do seu curso. A única diferença é que a nota máxima atribuída na recuperação é 8.

