Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade

Respostas dos Novos Exercícios

Marco Tulio Valente
https://engsoftmoderna.info

Capítulo 1 - Introdução

Slides com enunciado dos exercícios.

1. Porque um sistema pode ser desenvolvido em um ou dois anos e depois ficar décadas recebendo manutenções, principalmente as evolutivas (novas features). Veja o exemplo dos sistemas bancários.

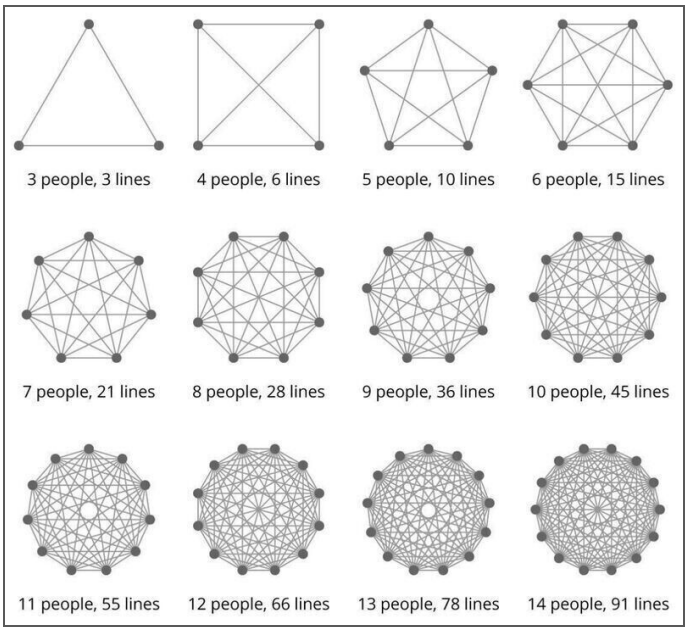
2. (a) Waterfall: projeto da arquitetura, depois projeto da estrutura, depois simulações em um túnel de vento, depois planejamento da construção, depois construção e, então, inauguração da ponte.

(b) Ágil: não tenho certeza de que a ponta é necessária e se ela é a melhor solução; vou construir então uma pinguela; se pinguela for bem aceita, vou construir uma ponte de mão única (e colocar um sinal em ambas as extremidades para controlar o acesso dos carros); por fim, podemos duplicar a ponte.

3. Manter o sistema funcionando do mesmo jeito, com exatamente as mesmas funcionalidades, mesmas saídas, para as mesmas entradas. Inclusive, com os mesmos bugs.

4. Porque para mostrar a ausência de bugs, temos que realizar testes exaustivos, isto é, testar com todas as possíveis entradas. Porém, tais testes são inviáveis, pois mesmo em sistemas pequenos eles levariam dias, talvez meses para rodar.

5. Por dois motivos: (a) mais devs demanda mais tempo para comunicar ou tomar uma decisão (os canais de comunicação crescem combinatorialmente com mais devs em um time, veja figura); (b) devs levam um tempo para entender os sistemas e começar a produzir de fato (ou seja, o processo de onboarding pode levar meses).



6. Esse sistema deveria ser menos ágil e mais waterfall (isto é, com mais planejamento no início, para evitar os altos custos de um bug em produção).

7. Questão dependente de interpretação, com fundo ético e para discussão com os alunos.

8.

F (pois portabilidade não é uma dificuldade essencial)
F (pois suporte ao PIX é um requisito funcional)
F (pois todos os envolvidos no desenvolvimento são stakeholders)
V

Capítulo 2 - Processos

Exercícios sobre XP (slides)

1. O desenvolvimento de software está sujeito a muitas mudanças. Ou seja, é muito difícil antecipar todos os requisitos (isto é, fechar o escopo) de um sistema. E, se isso for feito, a empresa que for contratada para o desenvolvimento pode começar a sacrificar a qualidade das entregas apenas para cumprir os prazos do contrato.

2. Contratos com escopo aberto requerem um maior acompanhamento por parte dos clientes, pois o escopo é redefinido e ajustado periodicamente. Por exemplo, a cada sprint. Exigem também mais confiança na empresa que vai desenvolver o sistema.

3. Porque XP define de várias práticas específicas de programação, tais como testes automatizados, integração contínua, design incremental, programação em pares, etc.

Exercícios sobre Scrum (slides)

1.

Histórias do topo do backlog: maior prioridade, mais claras e bem definidas, menores.
Histórias do fundo do backlog: o contrário

2.

(a) Escrever cada um dos capítulos
(b) Capítulos que faltam escrever
(c) Seções de um capítulo que está sendo escrito (em um sprint)
(d) Sim, por exemplo, mostrar os capítulos para alguns leitores, para eles revisarem
(e) Um pouco mais difícil, mas talvez um editor. Outra possibilidade: se o escritor for um ghost-writer, o PO pode ser a pessoa que contratou a escrita do livro.

3. Não, pois story points é uma medida totalmente interna de cada time. Por exemplo, uma determinada história pode ter 6 story points para um time e 4 story points para outro time (exatamente a mesma história).

4.

(a) Como dev, você pode argumentar com o PO que a ordem natural é primeiro "abrir arquivo" e depois "editar arquivo". Porém, a última palavra é sempre do PO.

(b) Sim, como respondido acima, a decisão final é do PO.

(c) Você implementaria a edição em um arquivo fixo, com um conteúdo pré-definido, que seria sempre carregado logo no início do editor. Ou seja, a edição sempre ocorreria nesse determinado arquivo.

Exercícios sobre Kanban (slides)

1. O limite WIP de implementação é 5, mas temos mais do que 5 cards nesta coluna do quadro.

2. Veja a resposta no FAQ do cap. 2: link

3. Definindo um limite WIP para cada coluna do quadro.

4. Criando uma coluna específica para revisão e validação das implementações.

Exercícios Finais (slides)

1. Big Design Upfront (BDUF), Testes manuais, Comando e controle.

2. Segue uma possível resposta:

(a)
1 PO: definir priorizar, explicar e tirar dúvidas sobre histórias de usuários
1 SM: guardião do processo Scrum e removedor de obstáculos não-técnicos
1 mobile dev: implementar as telas do PIX no aplicativo para celulares do banco
1 frontend dev: implementar as telas do PIX na versão Web do sistema
2 backend devs: implementar a lógica do PIX no backend do sistema

(b)
PO: entender a nova regra e suas implicações; explicar ela para o time
SM: acompanhar o sprint no qual a nova regra será implementado e remover obstáculos que surjam
mobile dev: implementar as telas referentes à nova regra no aplicativo para celulares do banco
frontend dev: implementar as telas referentes à nova regra na versão Web do sistema
backend devs: implementar a lógica da nova regra no backend do sistema

3.

(a) O PO deve ser um funcionário da universidade, por exemplo, um funcionário da seção de ensino que conheçe profundamente os processos e normas de ensino da universidade.

(b) Aqui temos duas possibilidades:

- O PO pode ser um funcionário da universidade, tal como em (a)
- O PO pode ser um funcionário da fábrica de software que deverá interagir e conversar com os especialistas em ensino da universidade

(c) Neste caso, como não temos um cliente definido, o PO deve ser um funcionário da própria empresa que desenvolve e vende o produto de apoio ao ensino. Em vez de PO, costuma-se frequentemente usar o termo PM (Product Manager) para designar este papel.

4.

(a) Ele pode e deve ser resolvido no próprio sprint.

(b) Se ele for um bug maior e mais complexo, ele pode ir para o backlog do produto. Caso contrário, ele pode ser resolvido de forma imediata e rápida, dentro do sprint atual.

(c) Como trata-se de um bug crítico, ele deve ser resolvido imediatamente. Inclusive, ele ganha prioridade sobre qualquer tarefa que esteja em andamento. Ou seja, deve-se parar tudo para resolver esse bug crítico, pois ele pode estar afetando um grande número de clientes e trazendo, portanto, prejuízos para a empresa.

5.

(a) Mesmo caso da letra (a) da questão anterior. A refatoração pode ser feita dentro do próprio sprint.

(b) Mesmo caso da letra (b) da questão anterior. A refatoração pode virar um item do backlog do produto.

6. Qualidade. Isto é, para cumprir o escopo do contrato, dentro do tempo e preço previstos, a empresa que foi contratada para desenvolver o sistema pode tender a sacrificar a qualidade da implementação. Ou seja, o sistema pode ser entregue com problemas de manutenibilidade do código, problemas de arquitetura e design, e mesmo com alguns bugs.

7. Acrescentar um novo "check" nos critérios para conclusão de histórias (*done criteria*):

> Atualizar documentação, sempre que a interface pública de um módulo ou API mudar.

8.

- PO que não tem tempo para tirar dúvidas do time
- PO que não conhece do domínio e requisitos do sistema
- PO que muda sempre de ideia e quer repriorizar as histórias de um sprint, durante o seu andamento.

Capítulo 3 - Requisitos

Exercícios sobre Histórias de Usuários (link)

1.

- Como cliente do banco, eu gostaria de transferir um valor usando o PIX
- Como cliente do banco, eu gostaria de pagar uma conta usando o PIX
- Como cliente do banco, eu gostaria de criar uma chave PIX
- Como cliente do banco, eu gostaria de ver uma lista com meus últimos PIX
- Como cliente do banco, eu gostaria de repetir um PIX

2. Segue um exemplo:

(a) História épica: Como professor, gostaria de poder aplicar provas online.

(b)

- Como professor, eu gostaria de criar um banco de questões para minhas provas
- Como professor, eu gostaria de criar e configurar uma prova online
- Como aluno, eu gostaria de fazer uma prova online
- Como professor, eu gostaria que uma prova fosse corrigida automaticamente
- Como aluno, eu gostaria de ver o resultado da correção de uma prova que fiz
- Como professor, eu gostaria de baixar um arquivo com todas as notas de uma prova

3. Respondida no próprio slide

4. Respondida no próprio slide

5. Respondida no próprio slide

6. A ideia aqui é que inverter as características INVEST:

Independentes => Dependentes
Abertas para **N**egociação => Fechadas para negociação
Agregar **V**alor => Não agregar ou agregar pouco valor
Estimáveis => de difícil estimativa
Sucintas => Grandes
Testáveis => Difícil escrever testes de aceitação para elas

Exercícios sobre MVPs (link)

1. Em uma pesquisa de mercado, você apresenta sua ideia para o cliente, na forma ainda de um questionário com várias perguntas, para avaliar se ela é viável.

Em um MVP, você constrói um produto (mesmo que mínimo) e o disponibiliza para uso, por clientes reais, para checar se ele é viável.

2.

(a) Por exemplo, você pode criar um grupo do whatsapp, para anunciar e combinar as caronas, sem implementar qualquer sistema ainda, somente para validar sua ideia. Ou seja, todas as tratativas serão feitas via WhastApp.

(b) Mudar o perfil de usuários: por exemplo, de alunos da UFMG para alunos de uma outra universidade. Outro possível pivô: mudar a tecnologia, por exemplo, usar um grupo do Telegram.

3. Alguns exemplos:

- um aplicativo financeiro (conta digital, cartão de crédito, seguros, etc), pois são áreas regulamentadas e fiscalizadas por órgãos do governo, como o Banco Central.
- um aplicativo da área de saúde, onde qualquer falha pode levar à perda de vidas humanas.
- um marketplace, que normalmente precisa de escala para se viabilizar.

4. Não, pois é apenas um modelo, ou protótipo da cozinha do futuro McDonalds. Por outro lado, um MVP pressupõe criar um produto que será usado por usuários reais, os quais muitas vezes nem saberão que se trata ainda de um experimento.

Capítulo 4 - Modelos

Exercícios sobre Diagramas de Classes (link)

1. Basicamente, deve-se fazer uma interpretação classe a classe do Diagrama.

2. e 3. Exercícios são do livro (versão original). A resposta deles pode ser encontrada aqui.

Capítulo 5 - Princípios de Projeto

Exercícios sobre Propriedades de Projeto (link)

1. Pior cenário de coesão possível: todas as 100 KLOC implementadas em um único método main.

Por outro lado, esse caso representa um acoplamento mínimo. Como temos apenas um método, ele não está acoplado a mais nenhum outro método do sistema. Ou seja, nem existe um segundo método para ele se acoplar.

2.

(a) Duas propriedades de projeto são violadas:

- Acoplamento, pois temos um acoplamento ruim, dos dois métodos, com a variável global saldo.
- Information Hiding, pois saldo deveria estar encapsulado em uma estrutura de dados (como uma classe)

(b) Criar uma classe Conta, com um atributo privado (saldo) e dois métodos públicos (depositar e getSaldo).

3.

(a) Acoplamento, pois a estratégia descrita no exercício contribui para diminuir o acoplamento entre diretórios. Ou seja, antes a classe A estava acoplada à classe B (A tem uma referência para B e elas estavam em diretórios diferentes). Agora, movendo a classe B para o mesmo diretório da classe A, esse acoplamento foi eliminado. Na verdade, para melhor entender o exercício, pense em um grafo cujos nodos são diretórios e as arestas são relações de acoplamento.

(b) Coesão, pois agora o diretório da classe A está crescendo, recebendo novas classes com funcionalidades diversas, ganhando responsabilidades diferentes. Logo, perdendo-se coesão.

4. Projeto (A), pois as arestas representam dependências (acoplamento) e, quanto menor o acoplamento, melhor.

5. Projeto (A), pois o ideal é que módulos sejam profundos, como um iceberg, com uma pequena ponta publicamente visível (interface) e uma grande parte submersa (privada). Assim, o módulo será mais útil, pois ele oferecerá uma interface simples (fácil de entender e chamar) para um problema complexo (cuja implementação requer muitas linhas de código).

6. Modularização II, pois ela possui módulos que ocultam mais a informação e o serviço que eles oferecem. Logo, módulos que atendem à propriedade de Ocultamento de Informação. Já na Modularização I, todos os módulos estão acoplados a uma estrutura de dados global (Memory, na figura).

Ou seja, trata-se de um acoplamento ruim, via uma memória global que é acessada para leitura e gravação.

Exercícios sobre Princípios de Projeto (link)

1.

(a) Demeter, pois temos uma longa cadeia de chamada de métodos.

(b) Incluiria na classe Biblioteca um método getNumExemplares. Logo, a chamada ficaria assim:

```
bib.getNumExemplares("ES", "ESM");
```

3. Substituição de Liskov, pois CalculadoraRápida pode ser usada no lugar de Calculadora (por herança). Ou seja, CalculadoraRápida pode substituir Calculadora.

Porém, apesar de ser mais rápida, essa substituição pode não funcionar. O motivo é que a CalculadoraRápida trabalha com uma faixa de valores mais estreita do que a sua classe pai. Isso ocorre, especificamente, no caso da operação que verifica se um número é primo ou não.

Por exemplo, a classe pai (Calculadora) consegue determinar se um número entre 0 e 999 é primo, o que não é possível com a CalculadoraRápida.

4. Demeter, pois para realizar o serviço necessário (obter o certificado) tem-se que fazer diversas chamadas de métodos e entrar na "arquitetura" interna da UFMG.

5. Segue abaixo uma possível resposta, baseada no exemplo de Inversão de Dependência dos slides:

Nesse exemplo, a suposição é que um controle remoto precisa controlar TVs de diversas marcas. Porém, pode ser que isso não seja necessário. Ou seja, basta que o controle remoto trabalhe com TVs da Samsung. E não há necessidade de que ele seja compatível com outras TVs. Logo, DIP neste contexto seria desnecessário ("um canhão, para matar uma formiga").

Capítulo 6 - Padrões de Projeto

Exercícios sobre Primeiro Grupo de Padrões de Projeto (link)

1. Essencialmente, o problema é que um Singleton pode ser usado para "camuflar" a criação de variáveis globais. Ou seja, os dados de uma classe Singleton podem ser usados como estruturas de dados globais, que são lidas e alteradas em diversos pontos de um programa.

2. A implementação do Singleton mostrada nos slides funciona apenas com sistemas single-thread. Para sistemas concorrentes temos que sincronizar a execução de getInstance(). Caso contrário, duas threads podem descobrir ao mesmo tempo que ainda não existe nenhuma instância do singleton e ambas vão então criar suas instâncias. Ou seja, vamos ter duas instâncias do singleton.

3. Esta resposta consta do livro (página 293, conforme copiado a seguir):

- Comunicação com um cliente remoto, isto é, pode-se usar um proxy para encapsular protocolos e detalhes de comunicação. Esses proxies são chamados de stubs.
- Alocação de memória por demanda para objetos que consomem muita memória. Por exemplo, uma classe pode manipular uma imagem em alta resolução. Então, podemos usar um proxy para evitar que a imagem fique carregada o tempo todo na memória principal. Ela somente será carregada, possivelmente do disco, antes da execução de alguns métodos.
- Controlar o acesso de diversos clientes a um objeto base. Por exemplo, os clientes devem estar autenticados e ter permissão para executar certas operações do objeto base. Com isso, a classe do objeto base pode se concentrar na implementação de requisitos funcionais.

4.

a. Coesão, pois as responsabilidades por certos requisitos, normalmente requisitos não-funcionais, saem da classe do objeto base e vão para o proxy.

b. Responsabilidade Única, pois tipicamente, após a criação do Proxy, a classe do objeto base fica responsável pela implementação apenas de requisitos funcionais (lógica de negócio).

c. Acoplamento, pois os clientes se acoplam apenas à fachada e não a uma diversidade de métodos do sistema que está por trás da fachada.

d. Não é sempre, mas normalmente o uso de Adaptadores ocorre em conjunto com Inversão de Dependência. Para ilustrar, no exemplo dos slides temos uma interface padrão que é usada pelo sistema (Projetor) e depois podemos ter adaptadores que vão adaptar tal interface para as interfaces de projetores específicos, tais como Samsung, LG, etc. Ou seja, no sistema cliente, estamos preferindo usar uma interface em vez de classes concretas de cada projetor.

e. Segregação de Interfaces, pois se projetadas de modo incorreto Fachadas podem ter diversos métodos, sendo que a maioria dos clientes vai usar apenas uma pequena parcela deles.

5.

a. Herança (ou implementação de interface) e composição, respectivamente

b. Projetor, AdaptadorProjetorSamsung e ProjetorSamsung

Capítulo 7 - Arquitetura

Slides com enunciado dos exercícios.

1.

(a) Microsoft Excel (versão para desktop)

MVC, pois é um sistema com interface gráfica, que roda localmente e de forma stand-alone (sem comunicação constante com um backend), e que precisa de um modelo (no caso, os dados das planilhas). Em resumo, o Excel é semelhante às aplicações Smalltalk-80, para as quais o modelo MVC puro foi originalmente proposto.

(b) Nubank App (mobile)

Também MVC (ou alguma variação desse modelo, tal como MVVM), pois além de uma interface gráfica, existe a necessidade armazenar dados locais, para tornar a aplicação mais responsiva e ágil. Na verdade, o Nubank usa Flutter, que é um framework que favorece o desenvolvimento de aplicações em uma arquitetura próxima de MVC.

(c) Twitter Web (front-end)

Single Page Application (SPA), pois o frontend do Twitter é uma aplicação responsiva, que não segue o modelo tradicional de aplicações Web, de troca constante de páginas a cada requisição. Na verdade, inicialmente o Twitter usava Ruby on Rails, ou seja, uma solução full-stack, baseada em MVC. No entanto, já faz um tempo que eles migraram para outras linguagens (Java e Scala, principalmente). Hoje, o frontend Web é implementado em React.

(d) Google Slides (front-end)

Single Page Application (SPA), pois também é uma aplicação Web bastante responsiva e com bastante autonomia do servidor.

(e) Twitter (backend)

Microsserviços, provavelmente também com algumas soluções Pub/Sub, pois é uma aplicação grande e complexa, desenvolvida por um número grande de times (justificativa para adoção de microsserviços) e que precisa escalar e ter alta disponibilidade (justificativa para adoção de Pub/Sub).

(f) Moodle

O Moodle é implementado em PHP. Logo, sua arquitetura é baseada (ou inspirada) no padrão MVC. Como as páginas, em geral, são menos responsivas e precisam ser carregadas ou salvas a cada transação, o seu frontend não é uma Single Page Application (SPA).

2.

(a) Porque os microsserviços de um sistema são completamente isolados uns dos outros. Ou seja, eles são processos de sistema operacional independentes. Logo, a chance de uma mudança em uma parte X do sistema causar um bug em uma parte Y (assumindo que X e Y pertencem a microsserviços diferentes, implementados por times diferentes) são mínimas.

(b) Porque, no caso de monolitos, não existe mais a independência e isolamento que comentamos na letra (a). Por exemplo, os módulos de um monolito podem compartilhar memória. Assim, uma mudança em X pode facilmente causar um efeito colateral em Y. Logo, se o time que implementa X tiver liberdade para colocar suas mudanças em produção, as chances de elas terem um efeito indesejado em uma outra parte do sistema aumentam de forma considerável.

(c) Em linhas gerais, esta resposta está no livro (página 246), conforme copiado a seguir:

> O ideal é que M1 e M2 sejam independentes também do ponto de vista de bancos de dados, como mostrado na próxima figura. O principal motivo é que quando se tem um único banco de dados ele também pode se transformar em um gargalo à evolução do sistema.
>
> Por exemplo, equipes e arquiteturas tradicionais de desenvolvimento costumam ter um administrador de dados, a quem cabe cuidar das tabelas do banco de dados. Qualquer mudança no banco de dados — como a criação de uma coluna em uma tabela — precisa da aprovação do administrador de dados. Logo, essa autoridade central tem que conciliar os interesses, muitas vezes conflitantes, das diversas equipes de desenvolvimento. Por isso, suas decisões podem se tornar lentas e burocráticas, atrasando a evolução do sistema.

3. Este exercício foi inspirado no seguinte post da Amazon Prime Video. Nesse post, eles explicam porque migraram um sistema de uma arquitetura baseada em serverless (ou microsserviços, tal como a arquitetura #1 do exercício) para uma arquitetura monolítica (tal como a arquitetura #2 do exercício).

Ou seja, a resposta do exercício é que a arquitetura #2 é mais escalável.

Motivo: ela demanda menos carregamento de vídeos da memória secundária para a memória principal. Como dito no enunciado do exercício, isso é feito apenas uma vez. Com isso, economiza-se em tempo de execução e também em custos pagos à plataforma de computação em nuvem.

Seguem, para ilustrar a resposta, alguns trechos copiados do blog:

> *The move from a distributed microservices architecture to a monolith application helped achieve higher scale, resilience, and reduce costs.*

> Microservices and serverless components are tools that do work at high scale, but whether to use them over monolith has to be made on a case-by-case basis.

> Moving our service to a monolith reduced our infrastructure cost by over 90%.

Capítulo 8 - Testes

Exercícios da Introdução (link)

1.

* Porque eles são mais lentos e mais frágeis (exemplo: mudanças na interface com o usuário podem quebrar tais testes facilmente)

* Porque quando eles falham é mais difícil identificar a parte exata do código responsável pela falha.

2. Será impresso: 11 e 11

Justificativa: os métodos de teste (@Test) são independentes e, para isso, eles rodam sobre uma instância nova da classe de teste. Em outras palavras, antes de chamar cada método de teste, o framework de teste cria uma nova instância da classe de teste. Em seguida, essa instância é usada para executar o método de teste.

Exercícios sobre Cobertura (slides)

1.

Cmd
f(0,0): 25%
f(1,1): 100%
f(0,0) e f(1,1): 100%

Branches
f(0,0): 25%
f(1,1): 50%
f(0,0) e f(1,1): 75%

2.

a. Quando a função é usada com a nota 90.

b. Comandos = branches = 100%

3. Falsa, o exercício anterior é um exemplo (nele temos um bug, apesar de 100% de cobertura)

4.

* Porque certos códigos são menos importantes; são usados com pouca frequência e, por isso, não vale a pena criar testes para eles. Exemplo: um relatório simples, que é usado apenas uma vez por ano por uma certa empresa

* Porque certos códigos são mais difíceis de serem testados (por exemplo, eles envolvem muito código de infraestrutura; um exemplo é o código da servlet que vamos usar a seguir para explicar o conceito de testabilidade)

Exercício sobre Teste de Integração (slides)

1. Como dito no "subject" do mail, ele faz parte de um teste de integração. Porém, em vez de ser executado em um banco de dados apenas de teste, ele foi executado em um banco de dados de produção, contendo endereços de e-mail de usuários reais da HBO.

Capítulo 9 - Refactoring

Slides com enunciado dos exercícios.

1. A ideia dessa citação é a seguinte: às vezes, está sendo difícil realizar uma mudança em um sistema software, para que ele implemente uma nova funcionalidade, por exemplo. No entanto, essa dificuldade pode estar sendo causada por um design ruim. Logo, se melhorarmos esse design (realizando algumas refatorações), o que antes estava difícil pode ficar mais fácil.

2. Extract Method e Inline Method
   Pull Up Method e Push Down Method

3.

(a) Quando já existe uma variável chamada "b" no escopo local.

(b) Quando na classe B já existe um método com a mesma assinatura de f.

4.

(a) O método f foi movido da classe C para a classe B.

(b) Não é um refactoring, pois:

Antes (esquerda): a classe B herda o método f de A. Logo, o programa imprime "oi".

Depois (direita): a classe B ganhou o seu próprio método f. Logo, o programa agora imprime "olá". Ou seja, o seu comportamento (resultado) mudou.

5.

(a) A classe B foi movida do pacote1 para o pacote2.

(b) Não é um refactoring.

Antes (esquerda): a chamada de m("abc") feita em A vai executar o método m(string), pois ele está implementado em B, que é uma classe do mesmo pacote de A. Em Java, quando não existe nem private ou public, a visibilidade é dentro do pacote.

Depois (direita): a chamada de m("abc") feita em A vai executar o método m(string), pois agora o método m(string) não é mais visível a partir de classes do pacote1 (como é o caso da classe A). Motivo: classe B está em um pacote diferente (pacote2)

6. Veja uma resposta detalhada aqui.

Capítulo 10 - DevOps

Slides com enunciado dos exercícios.

1.

(a) Por exemplo, o nome da função foi alterado, de destaca_texto para destaca_txt

(b) Por exemplo, a lógica da função foi alterada para em vez de destacar via negrito, destacar via itálico.

2.

CI: integrar o código com frequência (fazer merges com frequência)
Continuous Delivery: após integrar, o código deve estar sempre pronto para entrar em produção
Continuous Deployment: após integrar, o código vai de fato entrar em produção.

3. Continuous Delivery: pois não vale a pena disponibilizar, para os clientes, todos os dias, uma nova versão do driver. Os clientes vão ficar incomodados de serem "convidados" a instalar um novo drive quase que todo dia.

4. A principal diferença é que #ifdefs são processados antes da compilação de um programa C, por uma ferramenta independente, chamada pré-processador. Seja, por exemplo, o seguinte programa:

```
#define DEBUG 1

int main() {
   #ifdef DEBUG
      printf("Executando F");
   #endif
   printf("B");
}
```

Quando desenvolve-se em C, o programa acima será primeiro fornecido como entrada para o pré-processador da linguagem. Esse pré-processador irá rodar e entregar o seguinte programa para o compilador:

```
int main() {
   printf("Executando F");
   printf("B");
}
```

Em outras palavras, todos os #ifdefs são processados antes e não são "vistos" pelo compilador de C.

Por outro lado, feature flags são implementados por meio de comandos if (da própria linguagem de programação). Ou seja, eles são compilados e implicam em um teste em tempo de execução.

5. Por exemplo, suponha um sistema de provas online, com versões pagas e gratuitas. Apenas na versão paga existe a possibilidade de baixar as notas de uma prova no formato csv. Logo, feature flags podem ser usados para "habilitar" essa feature apenas para aqueles clientes que estão pagando para usar o sistema.

6.

Teste A/B: quando temos duas versões de um sistema e queremos decidir qual delas é "melhor": a versão A (atual, versão de controle) ou a versão B (nova, versão de tratamento)? A melhor versão será depois colocada em produção.

Release canário: quando queremos ter certeza de que uma nova versão não tem bugs críticos. Por isso, vamos primeiro lançá-la para um número pequeno de usuários (isto é, fazer um release canário). Depois, ela será então lançada para todos os usuários. Veja que não existe dúvida: a nova versão será colocada em produção. Porém, queremos fazer isso de forma gradativa.