

# Engenharia de Software

**Prof. Vinicius Cardoso Garcia**

[vcg@cin.ufpe.br](mailto:vcg@cin.ufpe.br) :: [@vinicius3w](https://twitter.com/vinicius3w) :: [viniciusgarcia.me](http://viniciusgarcia.me)

[IF977] Engenharia de Software

<http://bit.ly/vcg-es>

# Licença do material

Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-  
Compartilhual 3.0 Não Adaptada



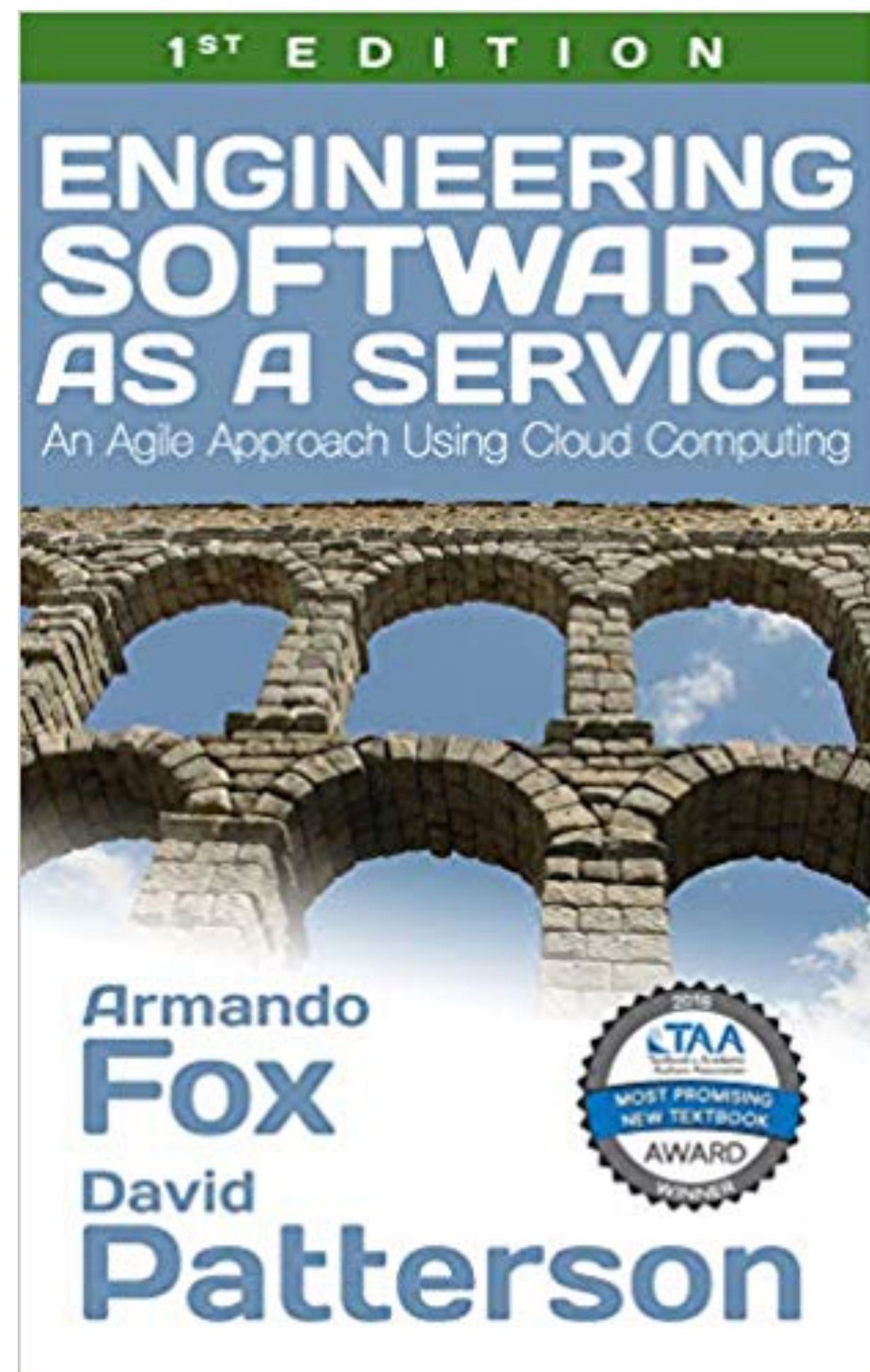
Mais informações visite

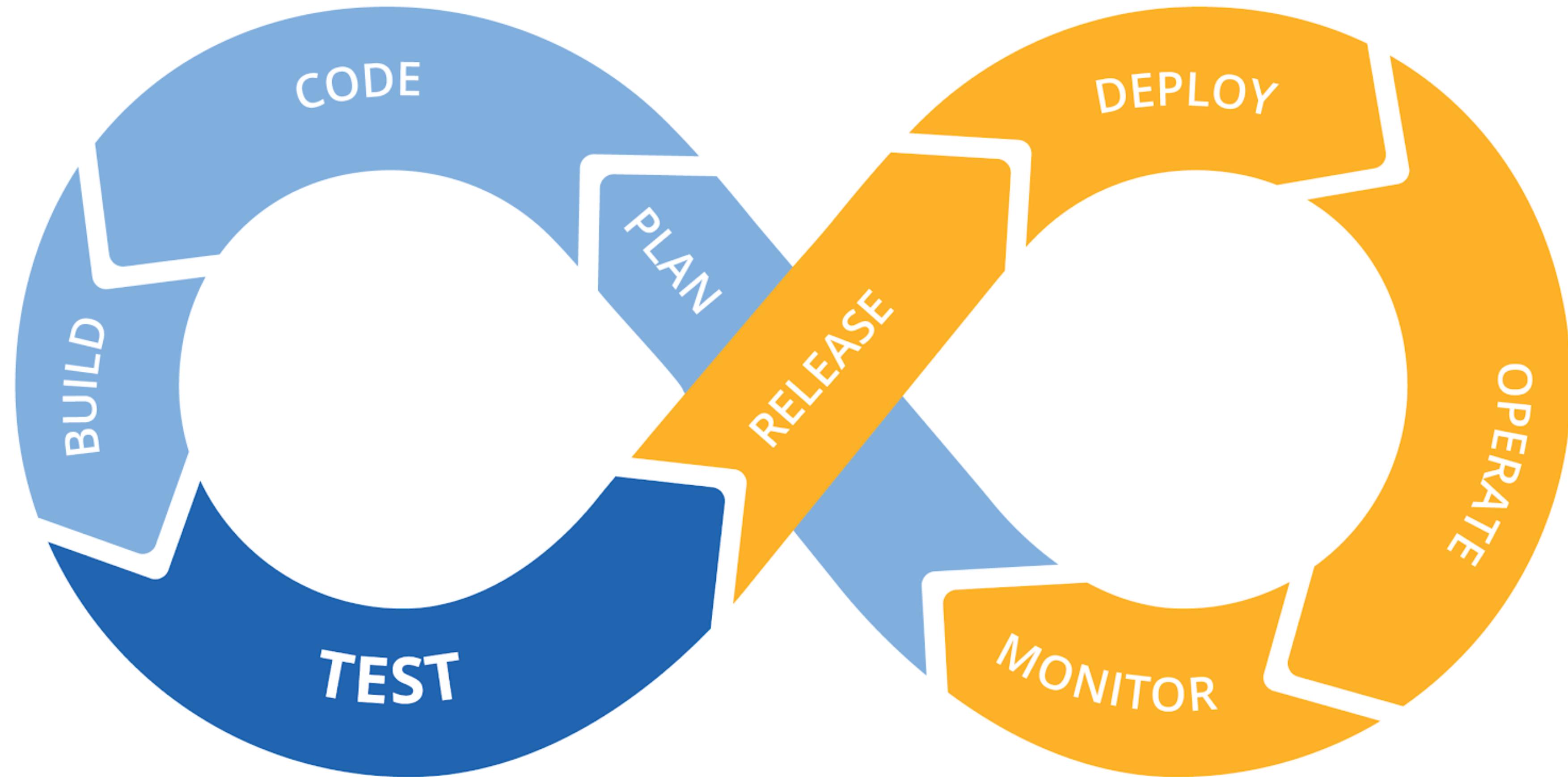
[http://creativecommons.org/licenses/by-nc-sa/  
3.0/deed.pt](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt)



# Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
  - <http://bit.ly/31WYK5f>
- SWEBOK: Guide to the Software Engineering Body of Knowledge (SWEBOK)
  - <http://www.computer.org/web/swebok>
- Engineering Software as a Service: An Agile Approach Using Cloud Computing
  - <http://www.saasbook.info/>
- Marco Tulio Valente. Engenharia de Software Moderna
  - <https://engsoftmoderna.info/>



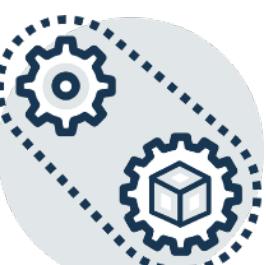


# Integração Contínua & Implantação Contínua

Imagen: <https://www.supportpro.com/blog/ci-cd/>

# DevOps?

- "Por que eu deveria me importar com o DevOps e que impacto isso tem em mim?"
- A resposta curta é que, se você estiver envolvido **na construção de sistemas de software** e sua organização estiver interessada em **reduzir o tempo de lançamento de novos recursos no mercado**, então você deveria se preocupar
  - As práticas de DevOps influenciarão a maneira como você **organiza times**, **constrói sistemas** e até a **estrutura dos sistemas** que você cria



# Definindo DevOps

- O DevOps foi classificado como "em ascensão" em relação ao ciclo Hype do Gartner para desenvolvimento de aplicativos em 2013

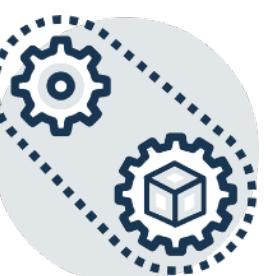
<https://www.gartner.com/en/documents/2560015>

- "O DevOps é um conjunto de práticas destinadas a reduzir o tempo entre a confirmação de uma alteração em um sistema e a alteração em produção, garantindo alta qualidade".
  - Trecho de: Bass, Len. "DevOps: A Software Architect's Perspective"



# Implicações

- A **qualidade** da **mudança implantada** em um sistema (geralmente na forma de **código**) é importante ~> \*ilities
- A **confiabilidade** e a **repetibilidade** do **mecanismo de entrega** devem ser altas
- Dois períodos são importantes
  - Quando um desenvolvedor **confirma (commit)** código recém-desenvolvido
  - Implantação desse código na **produção**
- Nossa definição é orientada a objetivos
  - As práticas de **DevOps** não são apenas **testes** e **implantação**
  - é importante incluir uma **perspectiva de operações** na coleção de requisitos
  - o **objetivo** é garantir **alta qualidade** do sistema implantado **ao longo de seu ciclo de vida**

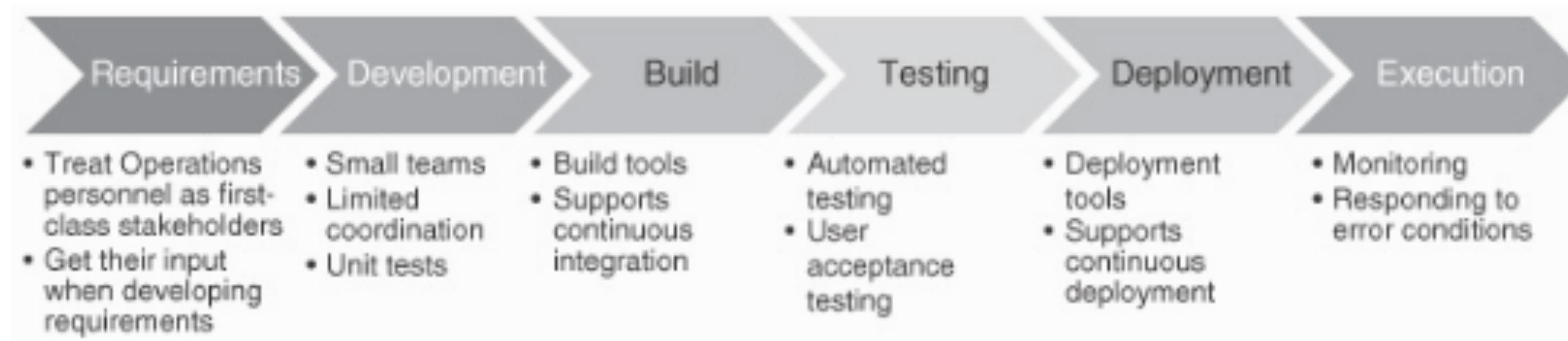


# Práticas DevOps

1. Tratar **Ops** como **cidadãos de primeira classe** do ponto de vista dos **requisitos**
  - As operações têm um conjunto de requisitos que pertencem ao logging e monitoramento
2. Tornar o **Dev** mais **responsável** pelo **tratamento relevante de incidentes**
3. **Impor** o processo de implantação **usado por todos**, incluindo o pessoal de Dev e Ops
  - Garanta uma alta qualidade, evite erros e configurações incorretas
4. Usar implantação **contínua**
  - Reduza o **tempo** entre um desenvolvedor que **confirma o código** em um repositório e o código que está sendo **implantado**
5. Desenvolver **código de infraestrutura**, como scripts de implantação, com o mesmo conjunto de práticas que o código da aplicação



# Ciclo de vida DevOps



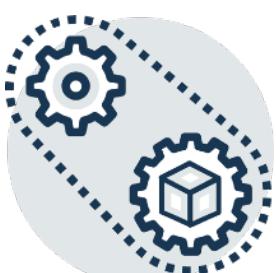
# Por que DevOps!?

- Em 1º de agosto de 2012, a Knight Capital teve uma **falha** de atualização que acabou custando **US \$ 440 milhões**.
- Em 20 de agosto de 2013, o Goldman Sachs teve uma **falha na atualização** que, potencialmente, poderia custar **milhões de dólares**.
- Trecho de: Bass, Len. "DevOps: A Software Architect's Perspective"



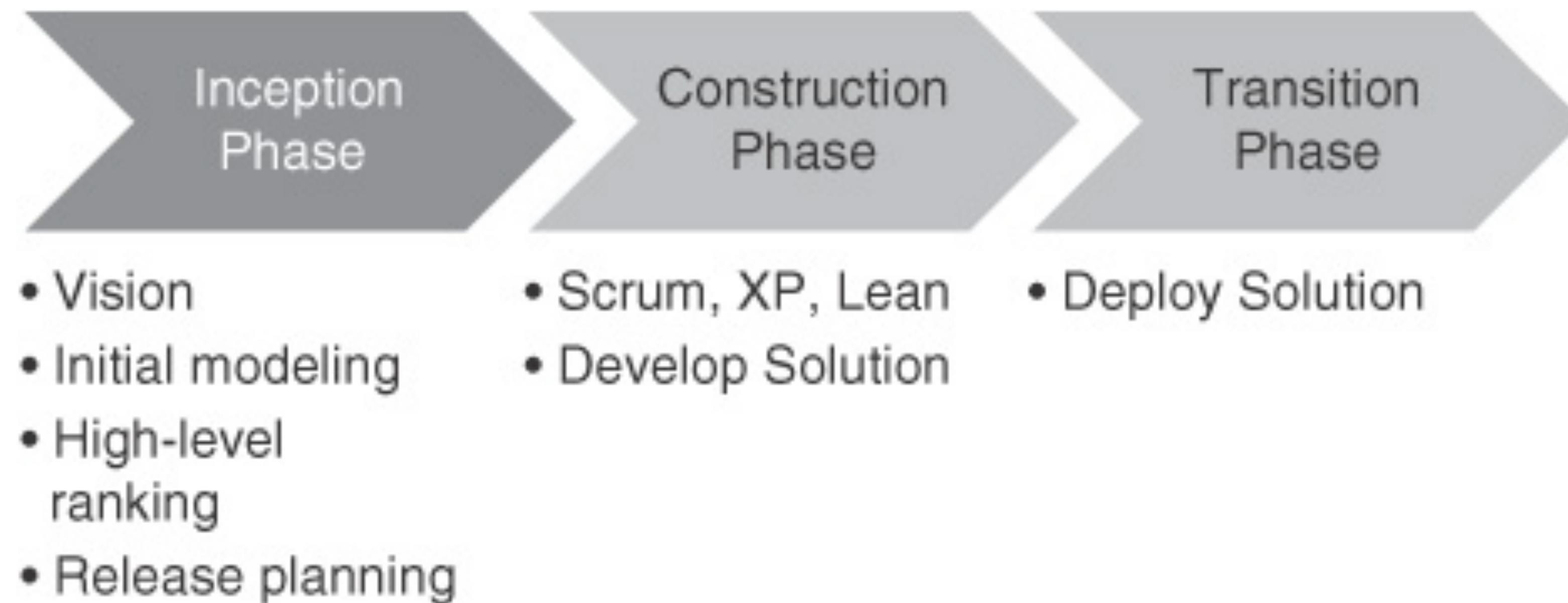
# Perspectiva DevOps

- Automação
  - As etapas de compilação e teste até a execução podem ser automatizadas, até certo ponto
    - Chef cookbooks ou Amazon CloudFormation
    - "Infraestrutura como código"
  - Reduza a incidência de erros e reduza o tempo de implantação
- Responsabilidades da equipe de desenvolvimento
  - Aceita as responsabilidades do DevOps: fornece, suporta e mantém o serviço



# DevOps e Agilidade

- A entrega ágil disciplinada tem três fases
  - criação, construção e transição



# Implicações

- Quais são as **implicações estruturais** das práticas de DevOps?
  - Tanto a **estrutura geral do sistema** e as **técnicas que devem ser usadas** nos elementos do sistema
- O DevOps alcança seus **objetivos** parcialmente, **substituindo** a coordenação **explícita** pela **implícita** e muitas vezes menos coordenada
  - a arquitetura do sistema que está sendo desenvolvido atua como o mecanismo **implícito** de coordenação



# As práticas do DevOps exigem mudanças na arquitetura?

- Se você deve **re-arquitetar** seus sistemas para tirar proveito do DevOps, uma pergunta legítima é "Vale a pena?"
- Algumas práticas de DevOps são **independentes da arquitetura**,
- considerando que, para obter todos os benefícios de terceiros, pode ser necessária uma **refatoração arquitetônica**



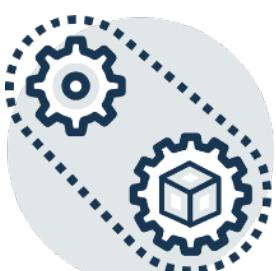
# Estrutura geral da arquitetura

- Warm up
  - um **módulo** é uma unidade de código com **funcionalidade coerente**
  - um **componente** é uma unidade **executável**
- As equipes de desenvolvimento que usam os processos de DevOps geralmente são **pequenas** e devem ter uma **coordenação entre equipes limitada**
- testes de integração e aceitação são **obrigatórios**



# Estrutura geral da arquitetura

- Uma organização pode introduzir a implantação contínua **sem grandes modificações** arquiteturais
  - A implantação **sem a necessidade de coordenação explícita com outras equipes** reduz o tempo necessário para colocar um componente em produção
  - Permitir que **diferentes versões do mesmo serviço** estejam simultaneamente em produção leva à implantação de diferentes membros da equipe sem coordenação com outros membros da equipe.
  - A **reversão de uma implantação** em caso de erros permite várias formas de teste ao vivo
- Arquitetura de [microsserviço](#) é um estilo arquitetônico que **atende** a esses requisitos



# Building and Testing

Testing leads to failure, and failure leads to understanding.

— Burt Rutan

# Pergunta

- Qual é a **razão mais importante** pela qual os **arquitetos** provavelmente deve se preocupar com as **infraestruturas** de desenvolvimento e implantação?
  - Eles ou os gerentes de projeto são responsáveis por garantir que a infraestrutura de desenvolvimento possa atender aos requisitos

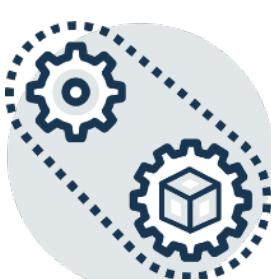
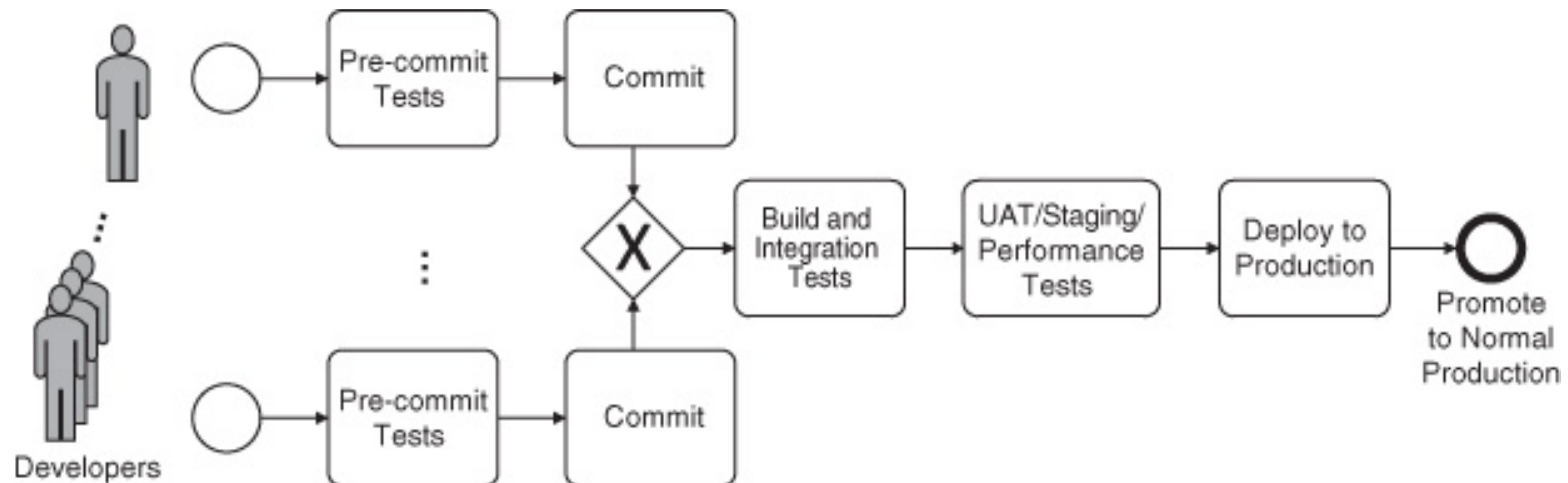


# Requisitos da Infraestrutura

- Os membros do time podem trabalhar em diferentes versões do sistema simultaneamente
- O código desenvolvido por um membro do time não substitui o código desenvolvido por outro membro, por acidente
- O trabalho não se perde se um membro do time sai de repente da equipe
- O código dos membros do time pode ser facilmente [re]testado
- O código dos membros do time pode ser facilmente integrado ao código produzido por outros membros da mesma equipe
- O código produzido por um time pode ser facilmente integrado ao código produzido por outros times
- Uma versão integrada do sistema pode ser facilmente implantada em vários ambientes (por exemplo, teste, preparo e produção)
- Uma versão integrada do sistema pode ser fácil e totalmente testada sem afetar a sua versão de produção
- Uma nova versão do sistema implantada recentemente pode ser supervisionada de perto
- Versões mais antigas do código estão disponíveis caso ocorra um problema depois que o código for colocado em produção
- O código pode ser revertido no caso de um problema



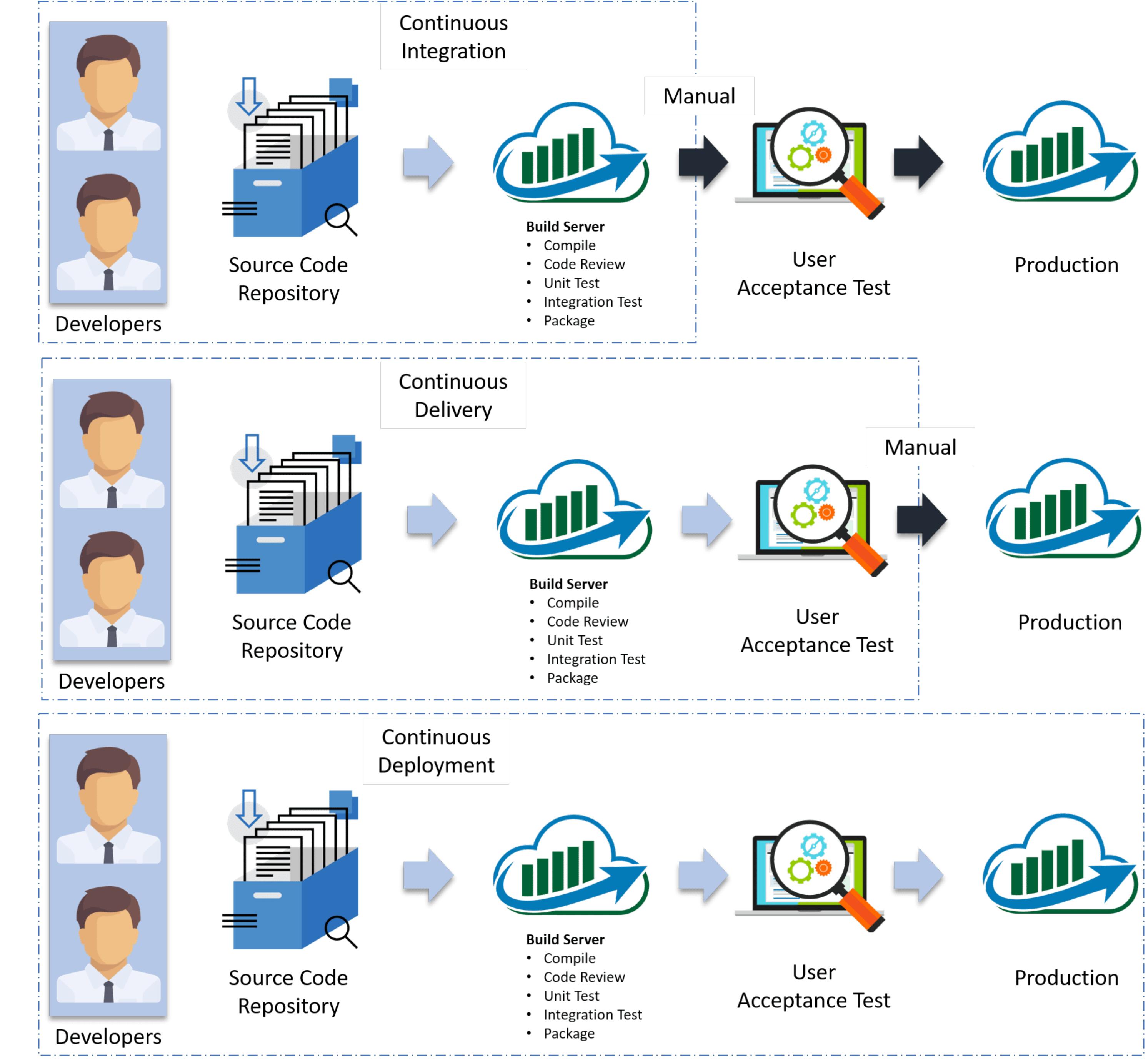
# Pipeline de Implantação



# Integração, Entrega e Implantação Contínua

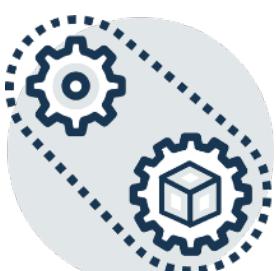
- Uma maneira de definir a **integração contínua** é ter acionadores **automáticos** entre uma fase e a seguinte, até os **testes de integração**
  - Ou seja, se a construção for bem-sucedida, os testes de integração serão acionados. Caso contrário, o desenvolvedor responsável pela falha é notificado
- A **entrega contínua** é definida como tendo **gatilhos automatizados** até o sistema de preparação
  - Esta é a caixa denominada UAT (teste de aceitação do usuário) / testes de preparação / desempenho
- **Implantação contínua** significa que a penúltima etapa (ou seja, implantação no sistema de produção) também é **automatizada**
  - Depois que um serviço é implantado na produção, ele é monitorado de perto por um período e, em seguida, é promovido à produção normal.
  - Nesta fase final, o monitoramento e o teste ainda existem, mas o serviço não é diferente de outros serviços nesse sentido.





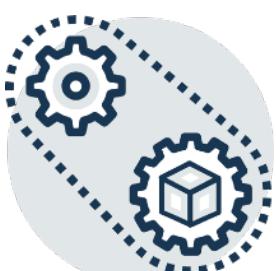
# Então....

- Usamos o pipeline de implantação como um tema organizador para o ciclo de transformação de um MVP em produto
- Em seguida, discutiremos as preocupações transversais das diferentes etapas, seguidas das seções no estágio de pré-confirmação, teste de compilação e integração, UAT / testes de preparação / desempenho, produção e pós-produção
- Antes de avançar para essa discussão, no entanto, discutimos o movimento de um sistema através do pipeline



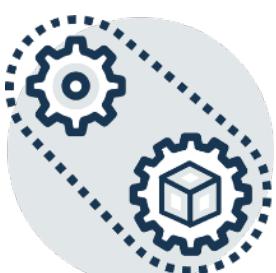
# Movendo um sistema pelo pipeline de implantação

- O código **confirmado** percorre as etapas mostradas na Figura, mas o código não se move por vontade própria
- Em vez disso, é movido por **ferramentas controladas** por seus programas (scripts) ou por comandos do desenvolvedor/operador. Dois aspectos desse movimento são de interesse:
  - Rastreabilidade
  - O ambiente associado a cada etapa do pipeline



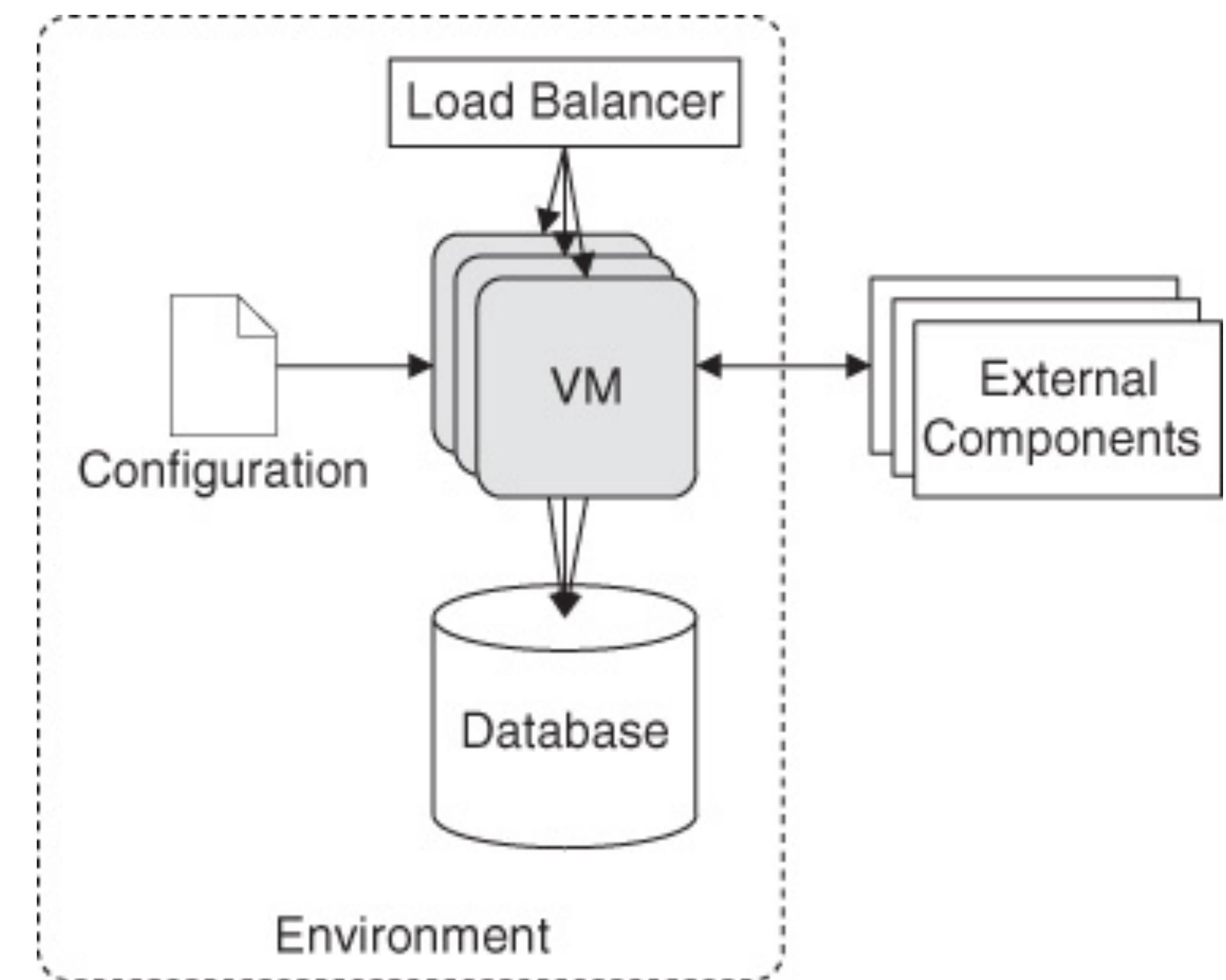
# Rastreabilidade

- Rastreabilidade significa que, para qualquer sistema em produção, é possível determinar exatamente como ele foi para produção
  - Isso significa acompanhar não apenas o código fonte, mas também todos os comandos de todas as ferramentas que atuaram nos elementos do sistema
- Comandos individuais são difíceis de rastrear. Por esse motivo, controlar ferramentas por scripts é muito melhor do que controlar ferramentas por comandos
  - Infraestrutura como código



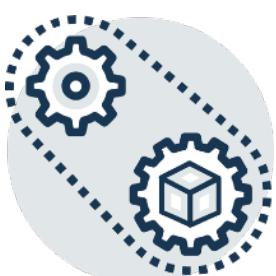
# O ambiente

- Um sistema em execução pode ser visto como uma **coleção de código em execução, um ambiente, configuração, sistemas fora do ambiente** com o qual o sistema primário interage e **dados**
- À medida que o sistema se move **pelo pipeline de implantação**, esses itens trabalham juntos para gerar o comportamento ou as informações desejadas



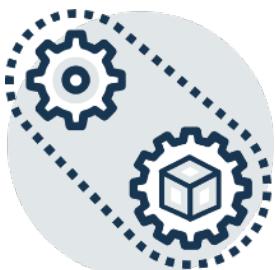
# Pre-commit

- O código é o módulo do sistema no qual o desenvolvedor está trabalhando
- A pré-confirmação requer coordenação dentro de uma equipe
- O ambiente geralmente é um laptop ou uma área de trabalho, os sistemas externos são eliminados ou “mockados” e apenas dados limitados são usados para teste
- Sistemas externos somente leitura, por exemplo, um feed RSS, podem ser acessados durante o estágio de pré-confirmação
- Os parâmetros de configuração devem refletir o ambiente e também controlar o nível de depuração



# Building e testes de integração

- O ambiente geralmente é um **servidor de integração contínua**
- O código é compilado e o componente é criado e inserido em uma imagem de VM ou em um contêiner
- Esta imagem da VM não muda nas etapas subsequentes do pipeline
- Durante o teste de integração, um conjunto de dados de teste forma um banco de dados de teste
- Os parâmetros de configuração conectam o sistema construído a um ambiente de teste de integração



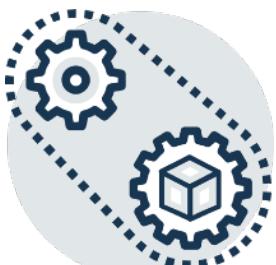
# UAT/staging/performance testing

- O ambiente é o mais próximo possível da produção
- Testes de aceitação automatizados são executados e o teste de estresse é realizado através do uso de cargas de trabalho geradas artificialmente
- O banco de dados deve ter algum subconjunto de dados de produção reais nele
- Os parâmetros de configuração conectam o sistema testado ao ambiente de teste maior
- O acesso ao banco de dados de produção não deve ser permitido no ambiente de preparação



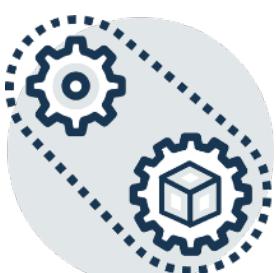
# Produção

- O ambiente de produção deve acessar o banco de dados ativo e ter recursos suficientes para lidar adequadamente com sua carga de trabalho
- Parâmetros de configuração conectam o sistema ao ambiente de produção



# Uma lista mais longa de ambientes (Wikipedia)

- Local: laptop/desktop/estação de trabalho do desenvolvedor
- Desenvolvimento: servidor de desenvolvimento, também conhecido como sandbox
- Integração: destino de criação de integração contínua (CI) ou para teste de efeitos colaterais pelo desenvolvedor
- Teste/controle de qualidade: para testes funcionais, de desempenho, garantia de qualidade etc.
- UAT: teste de aceitação do usuário
- Stage/Pré-produção: Espelho do ambiente de produção
- Produção/Live: atende usuários finais/ clientes

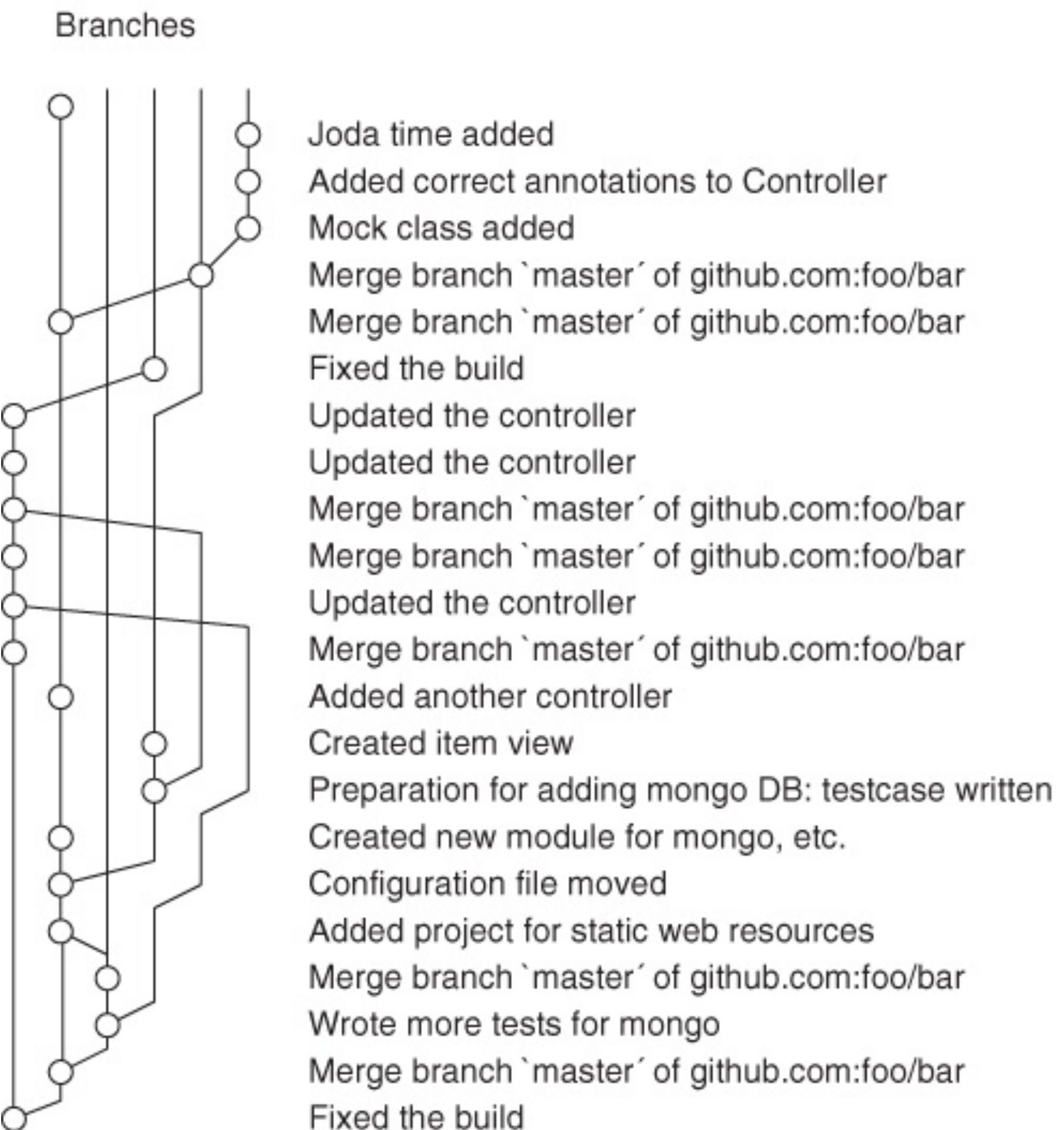


# **Teste em desenvolvimento e pré-confirmiação**



# Controle de Versão e Branching

- Você pode ter muitas branches e perder o controle de qual branch você deve trabalhar para uma tarefa específica.
  - Tarefas de curta duração não devem criar uma nova ramificação
- Mesclar dois ramos pode ser difícil
  - Diferentes ramificações evoluem simultaneamente e, muitas vezes, os desenvolvedores tocam muitas partes diferentes do código



# Feature Toggles

- Uma **feature toggle** (também chamada de feature flag ou feature switch) é uma declaração "if" em torno de código imaturo
- Uma nova feature que não está pronta para teste ou produção é desativada no próprio código-fonte, por exemplo, definindo uma variável booleana global
  - A prática comum coloca os feature switches na configuração
- O switch é alternado na produção (ou seja, a feature está ativada) apenas quando a feature está pronta para ser lançada e passou com êxito em todos os testes necessários
- Quando há muitas features toggles, gerenciá-las se torna complicado



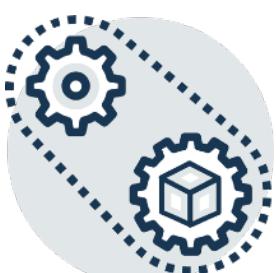
# Parâmetros de Configuração

- Um **parâmetro de configuração** é uma variável configurável externamente que altera o comportamento de um sistema
- O número de parâmetros de configuração deve ser mantido em um nível gerenciável
- Uma decisão a ser tomada sobre os parâmetros de configuração é se os valores devem ser os mesmos nas diferentes etapas do pipeline de implantação
  - Valores são os mesmos em vários ambientes
  - Os valores são diferentes dependendo do ambiente
  - Os valores devem ser mantidos em sigilo

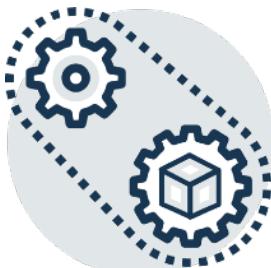


# Testes durante testes de desenvolvimento e pré-confirmação

- Test-driven development
  - Uma virtude dessa prática é que testes de caminho feliz são criados para todo o código
- Testes unitários
  - Uma prática comum é escrever o código de uma maneira que artefatos complicados, mas necessários (como conexões de banco de dados) formem uma entrada para uma classe - os testes de unidade podem fornecer versões simuladas desses artefatos, que exigem menos sobrecarga e são executados mais rapidamente



# **Build e Testes de Integração**



37



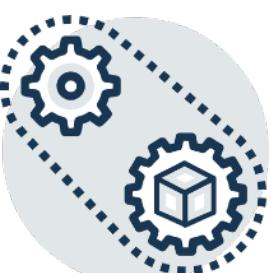
# Scripts de Build

- Os testes de build e integração são executados por um servidor de integração contínua (CI)
- A entrada para este servidor devem ser scripts que podem ser chamados por um único comando ("build")
- Essa prática garante que a compilação seja repetível e rastreável



# Packaging

- Pacotes específicos de tempo de execução
- Pacotes do sistema operacional
- Imagens de VM
- Contêineres leves
- Existem duas estratégias dominantes para aplicar alterações em uma aplicação ao usar imagens de VM ou contêineres leves:
  - Imagens pesadas não podem ser alteradas em tempo de execução: servidores imutáveis
  - Imagens leves são bastante semelhantes às imagens muito pesadas, com a exceção de que certas alterações nas instâncias são permitidas em tempo de execução



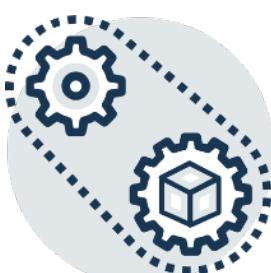
# Integração Contínua e Status de Build

- Depois que o build é configurado como um script que pode ser chamado como um único comando, a integração contínua pode ser feita da seguinte maneira:
  - O servidor de CI é notificado sobre novas confirmações ou verifica periodicamente por elas
  - Quando uma nova confirmação é detectada, o servidor de CI recupera-a
  - O servidor de CI executa os scripts de build
  - Se a compilação for bem-sucedida, o servidor de CI executará os testes automatizados
  - O servidor de CI fornece resultados de suas atividades à equipe de desenvolvimento (por exemplo, por meio de uma página da Web interna, e-mail ou canais como slack ou telegram)



# Testes de Integração

- Teste de integração é a etapa na qual o artefato executável construído é testado
- O ambiente inclui conexões com serviços externos, como um banco de dados substituto
- A inclusão de outros serviços requer mecanismos para distinguir entre solicitações de produção e teste
  - Essa distinção pode ser alcançada fornecendo serviços simulados
- Como em todos os testes que discutimos, os testes de integração são executados por um equipamento de teste e os resultados dos testes são registrados e relatados



# **UAT/Staging/Testes de Desempenho**

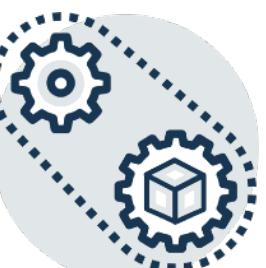


42



# UAT/Staging/Testes de Desempenho

- Staging é a última etapa do pipeline de implantação antes da implantação do sistema na produção
  - Os testes de aceitação do usuário (UATs) são testes em que os usuários em potencial trabalham com uma revisão atual do sistema por meio de sua interface do usuário e o testam, de acordo com um script de teste ou de forma exploratória
  - Testes de aceitação automatizados são a versão automatizada de UATs repetitivos
  - Os testes de fumaça (smoke tests) são um subconjunto dos testes de aceitação automatizados que são usados para analisar rapidamente se um novo commit quebra algumas das principais funções do aplicativo
  - Testes não funcionais testam aspectos como desempenho, segurança, capacidade e disponibilidade



# Produção



44



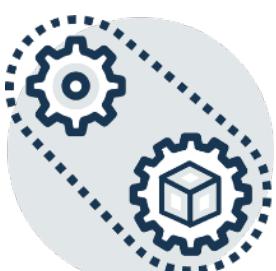
# Teste de Liberação Antecipada

- A abordagem mais tradicional é uma **versão beta**
- O **teste Canário** é um método de implantar a nova versão em alguns servidores primeiro, para ver o desempenho deles
- O **teste A/B** é semelhante ao teste canário, exceto que os testes têm como objetivo determinar **qual versão apresenta melhor desempenho** em termos de determinados **indicadores-chave** de desempenho no nível de **negócios**



# Detecção de erros

- Mesmo os sistemas que passaram em todos os testes ainda podem ter erros
  - As técnicas usadas para determinar erros não funcionais incluem o monitoramento do sistema para indicações de mau comportamento
  - Isso pode consistir em monitorar o tempo de resposta às solicitações do usuário, os comprimentos da fila e assim por diante
- Habilitar o diagnóstico de erros é um dos motivos da ênfase no uso de ferramentas automatizadas que mantêm o histórico de suas atividades
- Depois que o erro é diagnosticado e reparado, a causa do erro pode ser feita como um dos testes de regressão para versões futuras



# Live Testing

- Outra forma de teste depois que o sistema foi colocado em produção é realmente **perturbar** o sistema em execução
- A Netflix tem um conjunto de ferramentas de teste chamado **Simian Arm**
  - Por exemplo, o **Chaos Monkey** mata VMs ativas aleatoriamente
  - O **Latency Monkey** injeta atrasos nas mensagens



# Incidentes



48



# Incidentes

- Não importa quão bem você teste ou organize uma implantação, **existirão erros assim que um sistema entrar em produção**
- Entender as possíveis **causas** de erros pós-implantação ajuda a **diagnosticar** problemas **mais rapidamente**



# Sumário

- Um arquiteto envolvido em um projeto DevOps deve garantir o seguinte:
  - As várias ferramentas e ambientes são configurados para permitir que suas atividades sejam **rastreáveis e repetíveis**
  - Os parâmetros de configuração devem ser organizados com base em se eles mudarão para ambientes **diferentes** e em sua **confidencialidade**
  - Cada **etapa** do pipeline de implantação possui uma **coleção de testes automatizados** com um equipamento de teste **apropriado**
  - As feature toggles são removidas quando o código que alternam é colocado em produção e considerado implantado com sucesso



# Leitura Futura

- Para uma discussão mais detalhada de muitos dos problemas abordados nesta aula, consulte o livro: [Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation](#)
- O Simian Army é definido e discutido em: J. Kojo, V. Asokan, G. Campbell, and A. Tull. “Nicobar: Dynamic Scripting Library for Java,” February 10, 2015, <http://techblog.netflix.com>
- Principal livro para DevOps: J. Humble and D. Farley. **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation**, Addison-Wesley Professional, 2010.

