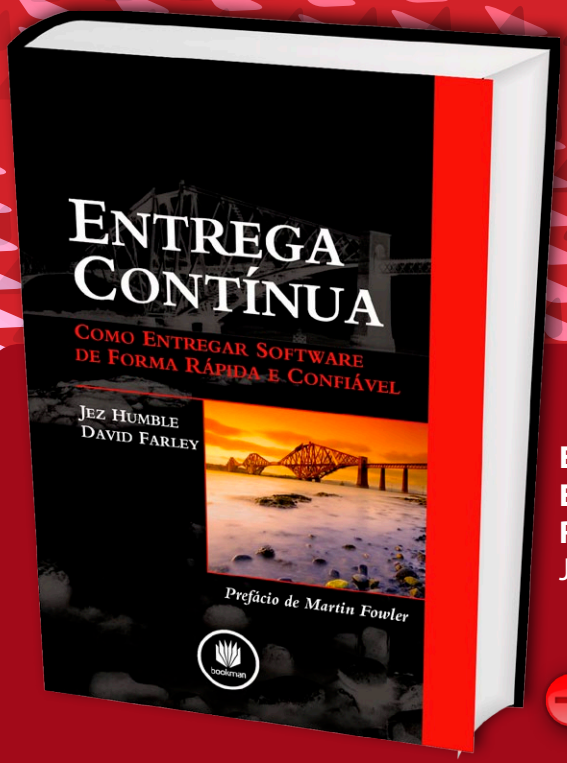
Entrega Contínua

Como Entregar Software de Forma Rápida e Confiável

Jez Humble
David Farley

Prefácio de Martin Fowler

bookman



Entrega Contínua - Como Entregar Software de Forma Rápida e Confiável
Jez Humble, David Farley

SAIBA MAIS

Formato: 17,5X25
Peso: 0,86 kg
Páginas: 496
ISBN: 9788582601037
Ano: 2014

Entregar uma nova versão de software para usuários costuma ser um processo cansativo, arriscado e demorado. Mas por meio da automação dos processos de compilação, implantação e teste, e da colaboração otimizada entre desenvolvedores, testadores e a equipe de operações, times de entrega podem lançar mudanças em questão de horas – algumas vezes, em minutos. Neste livro, Jez Humble e David Farley apresentam os princípios, as práticas e as técnicas de ponta que tornam possível uma entrega rápida e de alta qualidade, independentemente do tamanho do projeto ou da complexidade de seu código.

Parte I

Fundamentos

Capítulo 1

O Problema de Entregar Software

Introdução

O principal problema enfrentado pelos profissionais da área de desenvolvimento de software é: como transformar uma boa ideia em um sistema e entregá-lo aos usuários o quanto antes? Este livro mostra como resolver esse problema.

Vamos nos concentrar em como construir, implantar, testar e entregar software, tema que tem sido pouco abordado. Não estamos afirmando que outras abordagens de desenvolvimento não são importantes, mas que, sem um enfoque em outros aspectos do ciclo de vida de um software – que normalmente são vistos como secundários –, é impossível conseguir entregar versões de software de forma confiável, rápida e com poucos riscos, a fim de apresentar os resultados de nosso trabalho aos usuários da maneira mais eficiente.

Há muitas metodologias de desenvolvimento, mas elas se concentram sobretudo na gerência de requisitos e no impacto desses requisitos no esforço de desenvolvimento em si. Há muitos livros excelentes que tratam detalhadamente de diferentes abordagens para o projeto, desenvolvimento e teste de software, mas eles também só abrangem uma parte de toda a *cadeia de valor* que entrega valor às pessoas e às organizações que pagam pelo nosso esforço.

O que acontece depois da identificação de requisitos, projeto, desenvolvimento e teste das soluções? Como unir e coordenar todas essas atividades para tornar o processo tão eficiente e confiável quanto possível durante sua execução? Como fazer desenvolvedores, testadores e pessoal de operação trabalharem juntos de maneira eficiente?

Este livro descreve um padrão eficaz desde o desenvolvimento do software até sua entrega final. Descrevemos técnicas e boas práticas que podem ajudar a implementar esse padrão e mostramos como essa abordagem se encaixa nos outros aspectos da entrega de software.

O padrão central deste livro é chamado de *pipeline de implantação*. O pipeline de implantação é, em essência, uma implementação automatizada do

processo de compilar todas as partes de uma aplicação, implantá-la em um ambiente qualquer – seja de homologação ou produção – testá-la e efetuar sua entrega final. Cada organização tem implementações diferentes de seu pipeline de implantação, dependendo de sua cadeia de valor para entregar software, mas os princípios que regem o pipeline são os mesmos.

Um exemplo de um pipeline de implantação é mostrado na Figura 1.1.

Figura 1.1 *O pipeline de implantação.*

Resumidamente, o modo como o pipeline de implantação funciona é o seguinte. Cada mudança feita na configuração, no código-fonte, no ambiente ou em dados da aplicação cria uma nova instância do pipeline. Um dos primeiros passos no pipeline é a criação de binários e instaladores. O restante do pipeline executa uma série de testes nos binários para provar que é possível gerar uma entrega de versão. Cada teste no qual os binários em questão – também conhecidos como *release candidate* ou versão candidata – passam aumenta a confiança de que essa combinação de código binário, informações de configurações, ambiente e dados funcionará. Se a versão candidata passa em todos os testes, pode ser realizada a entrega de versão.

O pipeline de implantação baseia-se no processo de *integração contínua,* e é essencialmente o princípio de integração contínua levado à sua conclusão lógica.

O pipeline de implantação tem três objetivos. Em primeiro lugar, ela torna cada parte do processo de compilação, implantação, teste e entrega de versão visível a todos os envolvidos, promovendo a colaboração. Em segundo, melhora o feedback do processo, de modo que os problemas são identificados e resolvidos o mais cedo possível. Finalmente, permite que equipes entreguem e implantem qualquer versão de seu software para qualquer ambiente a qualquer momento por meio de um processo completamente automatizado.

Alguns antipadrões comuns de entrega de versão

O dia do lançamento de uma nova versão de um software tende a ser bem tenso. Por que isso acontece? Na maioria dos projetos, isso ocorre em função do risco associado ao processo – que transforma cada entrega em algo assustador.

Em muitos projetos, a entrega de uma versão é um processo intensamente manual. Os ambientes que hospedam a versão do software em geral são criados de maneira individual por um time de operações. Instala-se software de terceiros do qual a aplicação depende; os artefatos da aplicação em si são copiados para os ambientes de produção. A informação de configuração é copiada ou criada por meio de painéis de operação dos servidores Web, servidores de aplicação ou outros componentes do sistema criados por terceiros. Dados de referência são copiados e, finalmente, a aplicação é iniciada em partes, como uma aplicação distribuída ou orientada a serviços.

As razões para o nervosismo são claras: muitas coisas podem dar errado no processo. Se cada passo não for executado perfeitamente, a aplicação não funcionará da maneira correta. Nesse momento, talvez não esteja claro onde está o erro ou o que deu errado.

O restante deste livro discute como evitar esses riscos – como reduzir o estresse no dia da entrega de uma versão e como garantir que cada entrega de versão seja confiável. Antes disso, porém, precisamos esclarecer quais tipos de falhas no processo estamos tentando evitar. A seguir estão alguns antipadrões comuns que impedem um processo confiável de entrega de versão, mas que mesmo assim são muito comuns em nossa área.

Antipadrão: implantar software manualmente

A maioria das aplicações modernas de qualquer tamanho é complexa e envolve muitas partes móveis. Muitas organizações entregam versões de software de forma manual. Entrega manual é o processo de implantação em que cada passo é visto como separado e atômico, executado individualmente ou por uma equipe. É necessário fazer julgamentos a cada passo do processo, o que o torna mais sujeitos a erro humano. Mesmo que esse não seja o caso, diferenças na ordenação e no tempo de execução dos passos podem levar a resultados diferentes. Essas diferenças raramente são boas.

Os sinais desse antipadrão são:

- Produção de documentação extensa e detalhada que descreve os passos a serem executados e como eles podem falhar.
- Dependência de testes manuais para confirmar que a aplicação está funcionando.
- Chamadas frequentes aos desenvolvedores para que esses expliquem algo que está dando errado no dia da entrega de uma versão.
- Correções frequentes no próprio processo de entrega de uma versão no dia em que ele está sendo executado.
- Ambientes em um cluster que têm configurações diferentes, por exemplo, dois ou mais servidores de aplicação com configurações diferentes de bancos de dados, sistemas de arquivos com interface diferente e assim por diante.

- Entregas de versão que levam mais do que alguns minutos para executar.
- Entregas de versão imprevisíveis, que muitas vezes precisam ser canceladas, e com ambientes que precisam ser restaurados aos seus padrões anteriores em função de problemas desconhecidos.
- Noites em claro antes do dia da entrega de uma versão, tentando entender como fazê-la funcionar.

Em vez disso...

Com o tempo, o processo de implantação deve tender à automação completa. Somente duas tarefas devem ser executadas por humanos quando uma nova versão for implantada em um ambiente de teste, desenvolvimento ou produção: escolher que versão e ambiente, e apertar um botão para que o processo de implantação seja executado. Entregar software que é empacotado com instaladores deve envolver um único processo automatizado que cria o instalador.

Discutimos bastante a automação ao longo deste livro, e sabemos que algumas pessoas não aceitam completamente essa ideia. Para explicar por que consideramos isso um objetivo indispensável, mostramos agora alguns argumentos:

- Quando o processo não é automático, ocorrerão erros toda vez que ele for executado. A questão é se os erros são ou não significativos. Mesmo com ótimos testes de implantação, é difícil rastrear os defeitos.
- Quando o processo não é automatizado, ele não pode ser repetido com segurança, e perde-se tempo rastreando erros de implantação.
- Um processo manual precisa ser documentado. Manter a documentação é uma tarefa complexa, que consome tempo significativo e envolve a colaboração entre diversas pessoas, de modo que ela geralmente está incompleta ou ultrapassada. Um conjunto de scripts de automação serve como documentação e sempre estará completo e atualizado, ou a implantação não funcionará.
- Automação incentiva a colaboração, porque tudo está expresso em script de código. Documentação depende de suposições sobre o nível de conhecimento do leitor, e na verdade geralmente é escrita como um lembrete para a pessoa que executou a implantação naquele momento, tornando o conhecimento obscuro para outras pessoas.
- Um corolário da afirmação anterior: processos manuais de implantação dependem de um especialista. Se ele está de férias ou pediu demissão, você está em apuro.
- Executar qualquer processo de implantação manualmente é tedioso e repetitivo – e, ironicamente, exige um alto grau de conhecimento. Pedir para que pessoas com esse conhecimento façam algo tedioso e repetitivo – e ao mesmo tempo tecnicamente exigente – é a melhor maneira de garantir que ocorrerão erros humanos (exceto, talvez, por privação de sono

ou embriaguez). Automação permite que sua equipe altamente treinada e cara trabalhe em tarefas que de fato agregam valor ao negócio.

- A única maneira de testar um processo manual de implantação é executá-lo. Isso consome tanto tempo e esforço quanto o próprio processo. Processos de implantação automatizados são fáceis e baratos de testar.
- Já ouvimos o argumento de que processos manuais de implantação são mais auditáveis do que processos automatizados. O fato de alguém acreditar nisso nos deixa perplexos. Em um processo manual, não há qualquer garantia de que a documentação foi seguida. Apenas um processo automatizado é completamente auditável. O que pode ser mais auditável do que um script de implantação?

Se há um processo automatizado, ele deve ser o único modo pelo qual software é implantado. Essa disciplina é a única garantia de que ele funcionará quando necessário. Um dos princípios que descrevemos nesse livro é que o mesmo script de implantação deve ser usado em todos os ambientes. Se você usar o mesmo script, então o caminho da produção à implantação terá sido testado centenas ou mesmo milhares de vezes antes de ser necessário no dia de gerar uma entrega de versão. Problemas identificados na entrega de versão com certeza serão resultado de problemas específicos de configuração do ambiente e não de seus scripts.

Temos certeza de que, às vezes, os processos manuais de entrega de versão também funcionam bem. Podemos até ter tido o azar de ver mais falhas do que o normal em nossa carreira. Entretanto, se esse processo não é visto como sujeito a falhas, por que envolver tantas pessoas e tanta cerimônia nele? Por que tanta documentação? Por que as pessoas precisam fazer isso no fim de semana? Por que há pessoas de plantão, caso as coisas não funcionem conforme o esperado?

Antipadrão: implantar em um ambiente similar ao de produção somente quando o desenvolvimento estiver completo

Nesse antipadrão, a primeira vez em que um software é implantado em um ambiente similar ao de produção (por exemplo, um ambiente de homologação) é quando todo o desenvolvimento está concluído – ou pelo menos o que foi definido como concluído pelo time de desenvolvimento.

Esse antipadrão se parece com o seguinte:

- Se foram envolvidos testadores no processo até o momento, eles testaram somente em máquinas de desenvolvimento.
- A geração de uma versão para um ambiente de homologação é a primeira vez em que o pessoal de operação se envolve com uma versão do software. Em algumas organizações, um time de operação separado é usado para implantar o software em ambientes de homologação e produção. Nesse caso, a primeira vez em que uma pessoa de operação vê o software é o dia em que ele vai para a produção.

- Ou um ambiente similar ao da produção é muito caro e seu acesso é estritamente controlado, ou não existe naquele momento, ou ninguém acha que ele é necessário.
- A equipe de desenvolvimento cria os instaladores corretos, os arquivos de configuração, as migrações de bancos de dados e a documentação necessária para passar às pessoas que executam o processo de implantação – tudo isso ainda não foi testado em um ambiente de produção ou homologação.
- Há pouca ou nenhuma colaboração entre o time de desenvolvimento e as pessoas que realmente fazem a implantação.

Quando ocorre a implantação em homologação, é criada uma equipe para isso. Às vezes essa equipe tem todas as habilidades necessárias, mas em grandes organizações a responsabilidade geralmente é dividida entre vários grupos. DBAs (*database administrators*), equipes de middleware, equipes de servidores Web participam em etapas específicas no processo de instalar a última versão de um software. Como os vários passos nunca foram testados em homologação, ocorrem erros. A documentação não tem todos os passos necessários; a documentação e os scripts fazem suposições erradas sobre versões ou configurações dos ambientes, e o processo falha. A equipe de implantação tem então de adivinhar as intenções da equipe de desenvolvimento.

Geralmente, essa falta de colaboração causa tantos problemas no processo de passagem de desenvolvimento para homologação que todo ele se baseia em chamadas telefônicas, e-mails de última hora e consertos feitos às pressas. Um time disciplinado irá incorporar toda essa comunicação no plano de implantação – mas raramente esse tipo de esforço é eficaz. À medida que a pressão aumenta, o processo de colaboração entre a equipe de desenvolvimento e a equipe de implantação é transformado para que a implantação possa ser feita dentro do prazo dado à equipe responsável.

Durante o processo de implantação, não é incomum descobrir que pressuposições incorretas sobre o ambiente foram incorporadas ao próprio projeto do sistema. Por exemplo, uma aplicação que ajudamos a implantar usava o sistema de arquivos para fazer cache de dados. Isso funcionava muito bem nas máquinas de desenvolvimento, mas isso não acontecia nos ambientes em cluster de produção. Resolver problemas como esse pode levar muito tempo, e realizar uma implantação pode ser impossível sem que eles sejam resolvidos.

Uma vez que a aplicação esteja em homologação, é comum que sejam encontrados defeitos. Infelizmente, muitas vezes não é possível resolver todos eles, pois a data de implantação está se aproximando rapidamente e, nesse estágio do projeto, adiá-la seria inaceitável. Sendo assim, os defeitos mais críticos são resolvidos às pressas, e uma lista de defeitos conhecidos é guardada pelo gerente de projeto, e será priorizada quando o trabalho recomeçar, na próxima versão.

Às vezes a situação pode ser ainda pior. Há algumas coisas que agravam os problemas associados a uma entrega de versão.

- Ao trabalhar em uma nova aplicação, a primeira implantação em homologação é geralmente a mais complicada.
- Quanto maior o ciclo de desenvolvimento de uma versão, mais tempo a equipe de desenvolvimento tem para fazer suposições incorretas antes da implantação, e mais tempo será necessário para corrigi-las.
- Em grandes organizações em que o processo de implantação é dividido entre vários grupos, como desenvolvimento, DBAs, operações, testadores, etc., o custo de coordenação entre esses silos pode ser enorme, e algumas vezes o processo todo fica preso em um inferno de chamadas para o suporte. Nesse cenário, desenvolvedores, testadores e pessoal de operação estão continuamente abrindo chamadas de suporte (ou enviando e-mails) uns para os outros para conseguir ir adiante em qualquer processo de implantação – ou para resolver problemas que surgiram durante o processo.
- Quanto maior a diferença entre os ambientes de produção e de desenvolvimento, menos realistas serão as suposições feitas durante o desenvolvimento. Isso pode ser difícil de quantificar, mas é fácil imaginar que, se você está desenvolvendo em uma máquina Windows e implantando em um cluster Solaris, é provável que haja surpresas.
- Se a aplicação ou seus componentes são instalados por usuários, é provável que você não tenha muito controle sobre o ambiente, especialmente fora de ambientes corporativos. Nesse caso, testes extras serão necessários.

Em vez disso...

A solução é integrar todas as atividades de teste, implantação e entrega de versão ao processo de desenvolvimento. Transforme-as em uma etapa normal e contínua do desenvolvimento, de modo que haja pouco risco, já que você ensaiou antes em muitas ocasiões diferentes, em ambientes cada vez mais próximos da produção. Garanta que todos os envolvidos no processo de entrega de software, de desenvolvedores a testadores, passando por times de implantação e operação, trabalhem juntos desde o começo do projeto.

Nós adoramos testes, e o uso de integração contínua e entrega contínua, como um modo de testar tanto o software quanto o processo de implantação, é a base da abordagem que descrevemos.

Antipadrão: gerência de configuração manual dos ambientes de produção

Muitas organizações gerenciam a configuração dos seus ambientes de produção por meio de uma equipe de operação. Se uma mudança é necessária – como, por exemplo, a mudança de uma configuração do banco de dados ou o aumento do número de threads em um *thread pool* no servidor de aplicação –, ela é feita manualmente nos servidores de produção. Se for feito algum registro

de tal mudança, é provável que seja um registro em um banco de dados de mudanças de configuração.

Sinais desse antipadrão são:

- Depois de o software ter sido implantado com sucesso várias vezes em um ambiente de homologação, a implantação em produção falha.
- Membros diferentes de um cluster de máquinas se comportam de maneiras diferentes – por exemplo, um dos nodos suporta menos carga ou demora mais para processar requisições do que outro.
- A equipe de operações demora muito para preparar um ambiente para a entrega de uma versão.
- Você não pode voltar para uma configuração anterior de um servidor, incluindo sistema operacional, servidor de aplicações, servidor Web, banco de dados ou qualquer outra configuração de infraestrutura.
- Servidores em clusters possuem, de forma não intencional, versões diferentes de sistema operacionais, infraestrutura de terceiros, bibliotecas ou patches.
- A configuração de um sistema é feita por meio da modificação direta da configuração nos sistemas de produção.

Em vez disso...

Todos os aspectos dos seus ambientes de testes, homologação e produção – e em especial a configuração de elementos de terceiros existentes em seu sistema – devem ser executados a partir de controle de versão por um processo automatizado.

Uma das práticas fundamentais que descrevemos neste livro é a gerência de configuração; em parte isso significa ser capaz de recriar repetidamente qualquer parte da infraestrutura usada por sua aplicação de forma automática. Isso significa poder gerenciar sistemas operacionais, seus patches, todas as bibliotecas e configuração requeridas pela aplicação e assim por diante. Você precisa ser capaz de recriar o seu ambiente de produção de forma exata, de preferência de maneira automatizada. A virtualização pode ajudar bastante no início.

Você deve saber exatamente o que está em produção. Isso significa que cada mudança feita para a produção deve ser registrada e auditável. Muitas vezes, implantações falham porque alguém aplicou alguns patches ao ambiente de produção na última implantação e isso não foi registrado em lugar algum. De fato, não deveria ser possível fazer mudanças manuais nos ambientes de teste, homologação e produção. A única maneira de realizar modificações nesses ambientes seria por meio de um processo automatizado.

Aplicações muitas vezes dependem de outras aplicações. Deveríamos poder logo ver de qual é a versão exata de entrega de cada parte de software.

O processo de entrega de um software pode ser estimulante, mas também pode ser exaustivo e deprimente. Quase toda entrega envolve mudanças de última hora, como resolver problemas de conexão ao banco de dados, atualizar a URL de um serviço externo, e assim por diante. Deve haver um modo de introduzir tais mudanças de modo que sejam registradas e testadas. Mais uma vez, a automação é essencial. As mudanças devem ser feitas em um sistema de versionamento e propagadas para produção por meio de processos automatizados.

Da mesma forma, deve ser possível usar o mesmo processo para reverter uma versão em produção se a implantação falhar.

Podemos fazer melhor?

Com certeza, e o objetivo deste livro é mostrar como. Os princípios, as práticas e as técnicas que descrevemos tornam as implantações tediosas, mesmo em ambientes "corporativos" complexos. Entregas de versões de software podem – e devem – ser um processo de baixo risco, frequente, barato, rápido e previsível. Essas práticas foram desenvolvidas ao longo de muitos anos e fizeram uma grande diferença em muitos projetos. Todas as práticas descritas neste livro foram testadas em grandes projetos corporativos com times distribuídos e com times pequenos. Sabemos que elas funcionam, e que são escaláveis para projetos maiores.

O poder de implantações automatizadas

Um dos nossos clientes costumava ter um time enorme de pessoas dedicadas a cada entrega de uma versão de software. O time trabalhava junto por sete dias, incluindo um fim de semana inteiro, para conseguir colocar a aplicação em produção. A taxa de sucessos era baixa, e eram introduzidos erros a cada entrega de versão, com um alto nível de intervenções no dia da entrega da versão, além de várias correções e mudanças nos dias subsequentes para corrigir outros erros introduzidos durante a implantação ou causados por erro humano na configuração do software.

Nós ajudamos esse cliente a implementar um sistema sofisticado de implantação, desenvolvimento, teste e entrega de versão automatizado com todas as práticas e técnicas de desenvolvimento necessárias para suportá-lo. A última entrega de versão que acompanhamos foi colocada em produção em sete segundos. Ninguém percebeu coisa alguma, exceto, é claro, o comportamento novo introduzido nas funcionalidades daquela implantação. Se a implantação tivesse falhado por algum motivo, todas as mudanças poderiam ser revertidas nos mesmos sete segundos.

Nosso objetivo é descrever como o uso de pipelines de implantação, combinado com um alto grau de automação tanto de testes como de entrega, e um uso adequado de gerência de configuração possibilita entregas realizadas apenas apertando um botão – em qualquer ambiente necessário (teste, desenvolvimento ou produção).

Ao longo do texto, descreveremos esse padrão e as técnicas necessárias para adotá-lo e para que ele funcione. Daremos informações sobre abordagens para resolver os problemas que você enfrentará. Descobrimos que as vantagens dessa abordagem claramente superam os custos de alcançá-la.

Nada disso está fora do alcance de qualquer equipe de projeto. Não é necessário um projeto rígido, documentação significativa ou muitas pessoas. No final deste capítulo, esperamos que você entenda os princípios por trás dessa abordagem.

Como alcançar nosso objetivo?

Como mencionamos, nosso objetivo como profissionais de desenvolvimento é entregar software útil e funcional aos usuários o mais rápido o possível.

Velocidade é essencial porque há um custo de oportunidade associado à não entrega do software. Você só começa a obter um retorno sobre o investimento quando uma versão do software for entregue. Assim, um dos principais objetivos deste livro é encontrar formas de reduzir o *tempo de ciclo*, ou seja, o tempo entre a decisão de fazer uma mudança em um software – seja uma correção ou a adição de uma nova funcionalidade – e o momento em que ela está disponível para os usuários.

A entrega rápida também é importante porque permite que você descubra quais correções e funcionalidades implementadas de fato são úteis. O responsável pelas decisões por trás da criação de uma aplicação, que chamaremos de cliente, cria hipóteses sobre qual funcionalidade ou correção é mais útil para os usuários. Entretanto, até que estejam disponíveis para os usuários que escolhem como irão usar o software, elas continuam sendo hipóteses. Portanto, é vital minimizar o tempo de ciclo para que seja estabelecido um ciclo eficaz de feedback.

Uma parte importante da utilidade é a qualidade. Nosso software precisa se adequar a seus objetivos. Qualidade não é o mesmo que perfeição – como Voltaire disse: "O perfeito é o inimigo do bom" – mas o objetivo deve ser entregar software com qualidade suficiente para gerar valor aos seus usuários. Embora seja importante entregar o mais rápido possível, é essencial manter um nível apropriado de qualidade.

Assim, para refinar ligeiramente nosso objetivo, queremos encontrar formas de entregar aos usuários software de alto valor e alta qualidade de maneira eficiente, confiável e rápida.

Descobrimos que, para alcançar esse objetivo – tempo de ciclo curto e alta qualidade – precisamos entregar versões frequentes e automatizadas de nosso software. Por quê?

- **Automatizadas**: Se o processo completo de entrega não é automatizado, ele não é passível de repetição. Todas as suas realizações serão diferentes, em função de mudanças na aplicação, na configuração do sistema, no ambiente ou no próprio processo de entrega. Como os passos são manuais,

são passíveis de erro, e não existe uma forma de rever exatamente o que foi feito. Isso significa que não há como ter controle sobre o processo de entrega de versão e, consequentemente, de garantir que a qualidade seja alta. Entregar software algumas vezes é uma arte, mas deveria ser uma disciplina de engenharia.

- **Frequentes**: Se as entregas de versão são frequentes, a variação entre elas será menor. Isso reduz significativamente o risco associado a elas e torna o processo de reverter mudanças muito mais fácil. Entregas frequentes também conduzem a feedback mais rápido – na verdade, elas exigem isso. Grande parte deste livro se concentra em obter feedback sobre as mudanças feitas em uma aplicação e sua configuração associada (incluindo seu ambiente, processo de implantação e dados) o mais rápido possível.

Feedback é essencial para entregas frequentes e automatizadas. Há dois critérios para que ele seja útil:

- Cada mudança, seja qual for, deve disparar o processo de feedback.
- O feedback deve ser obtido assim que possível.
- A equipe responsável pela entrega deve receber o feedback e agir sobre ele.

Vamos examinar esses três critérios em detalhes e considerar como podemos alcançá-los.

Cada mudança deve disparar o processo de feedback

Uma aplicação de software funcional pode ser dividida em quatro componentes: código executável, configuração, ambiente de hospedagem e dados. Se qualquer um deles muda, o comportamento da aplicação pode mudar. Portanto, é necessário manter esses quatro componentes sob controle e garantir que cada mudança feita em qualquer um deles seja verificada.

Quando uma mudança é feita no código-fonte, o código executável muda. Toda vez que se faz uma mudança no código-fonte, o binário resultante deve ser compilado e testado. Para ter controle sobre esse processo, a compilação e os testes do binário devem ser automatizados. A prática de compilar e testar a aplicação a cada mudança é conhecida como integração contínua e é mais bem descrita no Capítulo 3.

Esse código executável deve ser o mesmo código executável implantado em todos os ambientes, seja de teste ou produção. Se o sistema usa uma linguagem compilada, você precisa garantir que o resultado binário de seu processo de compilação seja reusado sempre que for necessário e nunca recompilado.

Qualquer mudança entre ambientes deve ser capturada como informação de configuração. Qualquer mudança na configuração da aplicação, em qualquer ambiente, deve ser testada. Se o software vai ser instalado diretamente para os usuários, as possíveis configurações devem ser testadas em uma faixa

representativa de sistemas de exemplo. A gerência de configuração é discutida no Capítulo 2.

Se algum dos ambientes em que a aplicação será implantada mudar, o sistema completo deve ser testado levando em conta essas mudanças. Isso inclui as mudanças de sistema operacional, a lista de softwares que suportam a aplicação, as configurações de rede e qualquer infraestrutura interna ou externa. O Capítulo 11 discute a gerência de infraestrutura e ambientes, incluindo a automação da criação e manutenção de ambientes de teste e produção.

Finalmente, se a estrutura dos dados muda, essa mudança também deve ser testada. Discutimos gerência de dados no Capítulo 12.

O que é o processo de feedback? Ele envolve testar cada mudança de forma tão automatizada quanto possível. Os testes variam de acordo com o sistema, mas devem incluir no mínimo as seguintes verificações:

- O processo de criação do código executável deve funcionar. Isso verifica se a sintaxe do código é válida.
- Devem ser feitos testes unitários. Isso verifica se o código se comporta como esperado.
- O software deve atender a certos critérios de qualidade, como cobertura de testes e outras métricas dependentes da tecnologia.
- Devem ser feitos testes de aceitação funcionais. Isso verifica se a aplicação está de acordo com os critérios de aceitação do negócio – isto é, ela entrega o valor de negócio esperado.
- Devem ser feitos testes não funcionais. Isso verifica se a aplicação funciona suficientemente bem em termos de capacidade, disponibilidade e segurança, e se atende às necessidades dos usuários.
- O software deve passar por alguns testes exploratórios e uma demonstração para o cliente e um grupo de usuários. Isso geralmente é feito em um ambiente de testes manuais. Nessa parte do processo, o responsável pelo projeto pode decidir que funcionalidade está faltando ou encontrar defeitos que precisam ser corrigidos e testes automatizados que precisam ser criados para prevenir regressões.

O ambiente em que esses testes são executados deve ser o mais próximo possível do ambiente de produção, para verificar que mudanças no ambiente não influenciaram a capacidade de execução da aplicação.

O feedback deve ser obtido o mais rápido possível

A chave para feedback rápido é a automação. Com processos completamente automatizados, seu único limite é a quantidade de hardware que se pode usar. Se você tem processos manuais, depende de pessoas para fazer o trabalho. Pessoas demoram mais, introduzem problemas e não são auditáveis. Além disso, executar essas tarefas manualmente é tedioso e repetitivo – com certeza não a

melhor maneira de aproveitar essas pessoas. Elas são caras e valiosas, e devem se concentrar na produção de software que agrade aos seus usuários e na entrega deste o mais rápido possível – e não em tarefas tediosas e passíveis de erro como testes de regressão, provisionamento de servidores virtuais, implantação, e assim por diante, que são executadas com maior eficiência por máquinas.

Entretanto, a implementação do pipeline de implantação é algo que consome recursos, especialmente quando você tem uma boa cobertura em seus testes automatizados. Um dos principais objetivos é otimizar o uso de recursos humanos: queremos que as pessoas se concentrem em trabalhos interessantes e deixem o trabalho repetitivo para máquinas.

Podemos caracterizar os testes no estágio de desenvolvimento do pipeline (Figura 1.1) da seguinte forma:

- Eles executam rapidamente.
- Eles são tão completos quanto possível, ou seja, cobrem mais do que 75% da base de código, de modo que quando são executados com sucesso, temos um bom grau de confiança de que a aplicação funciona.
- Se qualquer um deles falhar, isso significa que a aplicação tem uma falha crítica e que, de maneira alguma, a versão deve ser gerada e entregue. Isso significa que testes para verificar a cor de um elemento da interface não devem estar incluídos nesse conjunto de testes.
- Os testes são o mais independentes possível do ambiente, ou seja, o ambiente não precisa ser uma réplica perfeita do ambiente de produção – isso é mais barato e mais simples.

Por outro lado, os testes dos próximos estágios têm as seguintes características:

- Executam mais lentamente e por isso são candidatos à paralelização.
- Alguns deles podem falhar, e ainda podemos gerar uma versão da aplicação em algumas circunstâncias (talvez haja uma correção crítica da versão candidata que faz o desempenho cair abaixo de algum parâmetro definido, mas podemos decidir entregar uma versão mesmo assim).
- Eles rodam em um ambiente bem próximo ao do de produção, de modo que, além dos testes de funcionalidade, também cobrem aspectos do processo de implantação e mudanças no ambiente de produção.

Essa organização do processo de testes significa que temos um alto grau de confiança do software depois do primeiro conjunto de testes que são efetuados em máquinas mais baratas e de forma mais rápida. Se esses testes falham, a versão candidata não progride para os próximos estágios. Isso garante um uso otimizado dos recursos. Você encontrará mais informações sobre isso no Capítulo 5, "Anatomia de um Pipeline de Implantação", e nos Capítulos 7, 8 e 9, que descrevem o estágio de testes iniciais, testes de aceitação automatizados e testes de requisitos não funcionais.

Um dos elementos fundamentais de nossa abordagem é a necessidade de feedback rápido. Garantir esse feedback em mudanças exige atenção ao processo de desenvolvimento do software – em especial ao uso de controle de versão e à organização do código. Os desenvolvedores devem realizar commits frequentes para o sistema de versionamento e separar o código em componentes para gerenciar equipes maiores ou distribuídas. Criar novos branches deve ser evitado na maioria das vezes. Discutimos entrega incremental e o uso de componentes no Capítulo 13, "Como Gerenciar Componentes e Dependências", e branching e *merging* no Capítulo 14, "Controle de Versão Avançado".

A equipe responsável pela entrega deve receber o feedback e aproveitá-lo

É essencial que todos os envolvidos no processo de entrega de software estejam igualmente envolvidos no processo de feedback. Isso inclui desenvolvedores, testadores, equipe de operação, administradores de banco de dados, especialistas de infraestrutura e gerentes. Se as pessoas com essas funções não trabalharem juntas no dia a dia (embora seja recomendado que as equipes sejam multifuncionais), é essencial que se encontrem com frequência e trabalhem para melhorar o processo de entrega. Um processo baseado em melhorias contínuas é essencial para a entrega rápida de software de qualidade. Processos iterativos ajudam a estabelecer um ritmo para esse tipo de atividade – pelo menos uma reunião de retrospectiva por iteração em que todos discutem como melhorar o processo para a próxima iteração.

Ser capaz de reagir ao feedback também significa espalhar a informação. O uso de painéis de informação grandes e visíveis (que não precisam ser eletrônicos) e outros mecanismos de notificação é essencial para garantir que o feedback seja de fato realimentado e seja bem compreendido por todos. Esses painéis devem estar sempre presentes, e deve haver pelo menos um deles no local onde o time de desenvolvimento está.

Finalmente, nenhum feedback é útil se for ignorado. Isso requer disciplina e planejamento. Quando alguma coisa precisa ser feita, é responsabilidade da equipe como um todo parar o que está fazendo e decidir a melhor atitude a ser tomada. Depois que isso for feito, o time pode continuar seu trabalho.

Esse processo é escalável?

Uma objeção comum a isso é que o processo descrito aqui é idealístico. Os opositores dizem que pode funcionar para equipes pequenas, mas é impossível que funcione para projetos grandes e distribuídos!

Nós trabalhamos em muitos projetos de grande porte ao longo dos anos em várias indústrias diferentes. Também tivemos a sorte de trabalhar com colegas com muitas experiências. Todos os processos e técnicas que descrevemos

neste livro foram experimentados e comprovados em projetos reais em todos os tipos de organizações, tanto grandes como pequenas, em diversas situações. Encontrar os mesmos problemas toda vez nesses projetos é o que nos levou a escrever este livro.

Os leitores irão notar que este livro é inspirado pela filosofia e pelas ideias do movimento *lean*. Os objetivos da produção *lean* são garantir a entrega rápida de produtos de alta qualidade e focar a redução de desperdícios e de custos. O uso dessas técnicas e ideias resulta em grande economia de recursos e redução dos custos, produtos de maior qualidade e menor tempo de entrada no mercado em várias indústrias. Essa filosofia está começando a se tornar comum também no campo de desenvolvimento e é subjacente a muito do que discutimos neste livro. A aplicação das técnicas *lean* não se limita a sistemas pequenos. Foram criadas e aplicadas a grandes organizações e até mesmo a economias inteiras.

Tanto a teoria quanto a prática são tão relevantes para equipes grandes como para equipes pequenas, e nossa experiência é de que funcionam. Entretanto, não pedimos que você simplesmente acredite no que dizemos. Experimente e descubra por conta própria. Mantenha o que funciona e elimine o que não funciona, e escreva sobre suas experiências para que outros também se beneficiem.

Quais são os benefícios?

O principal benefício da abordagem que descrevemos na seção anterior é que ela cria um processo de entrega confiável, previsível e passível de repetição, que, por sua vez, gera grandes reduções no tempo de ciclo e entrega novas funcionalidades e correções aos usuários rapidamente. A redução de custos em si já é suficiente para cobrir todo o tempo investido no estabelecimento e na manutenção do próprio processo.

Além disso, há muitos outros benefícios; alguns deles já haviam sido previstos anteriormente, enquanto outros surgiram como surpresas agradáveis quando observados.

Dar autonomia às equipes

Um dos princípios fundamentais do pipeline de implantação é que ele é um sistema de autosserviço: permite que testadores, pessoal de operações e suporte obtenham qualquer versão da aplicação que quiserem em qualquer ambiente que escolherem. Em nossa experiência, uma contribuição importante para o tempo de ciclo são as pessoas envolvidas no processo de entrega que estão esperando para receber uma "versão boa" da aplicação. Muitas vezes, receber uma versão boa requer um processo interminável de envio de e-mails, chama-

das de suporte e outras formas ineficientes de comunicação. Quando a equipe de entrega é distribuída, isso se torna uma fonte enorme de ineficiência. Com a implementação de um pipeline de implantação, esse problema é eliminado – todos podem ver quais versões estão disponíveis para os ambientes nos quais têm interesse, e têm a habilidade de fazer a implantação dessas versões com um simples clique em um botão.

O que geralmente vemos como resultado disso é a existência de várias versões em diferentes ambientes, enquanto diversos membros da equipe fazem seu trabalho. A habilidade de implantar facilmente qualquer versão em qualquer ambiente tem muitas vantagens.

- Testadores conseguem selecionar versões anteriores da aplicação para verificar mudanças no comportamento das novas versões.
- A equipe de suporte pode implantar versões anteriores da aplicação que os usuários estão usando para reproduzir um defeito.
- A equipe de operação pode selecionar uma versão que eles sabem que é boa para implantar em produção como parte de um exercício de recuperação de desastres.
- Implantações podem ser feitas com um único clique.

A flexibilidade que nossas ferramentas de implantação oferecem muda a forma como elas funcionam – para melhor. De maneira geral, membros de uma equipe têm mais controle sobre seu trabalho, o que aumenta sua qualidade e faz com que a qualidade da aplicação também aumente. A equipe colabora de forma mais eficiente e menos reativa, e pode trabalhar com maior eficiência porque não perde tanto tempo esperando por versões da aplicação.

Reduzir erros

Erros podem ser introduzidos em software a partir de vários locais. As pessoas que pediram o software podem ter pedido as coisas erradas, os analistas que capturaram os requisitos podem tê-los entendido mal, e os desenvolvedores podem ter criado código com defeitos. Os erros dos quais estamos falando aqui, entretanto, são especificamente aqueles introduzidos em produção em função da má *gerência de configuração*. Iremos explicar o que queremos dizer com gerência de configuração em mais detalhes no Capítulo 2. Por hora, pense em todas as coisas que precisam estar corretas para que uma aplicação funcione – a versão correta do código, é claro, mas também a versão correta do esquema do banco de dados, a configuração correta dos *load-balancers*, a URL correta de *serviço Web* usados em produção e assim por diante. Quando falamos em gerência de configuração, referimo-nos a todos os processos e mecanismos que permitem que você identifique e controle esse conjunto de informação.

A diferença que um byte faz

Há alguns anos, Dave estava trabalhando em um ponto de vendas de grande escala para um atacadista conhecido. Isso aconteceu quando a ideia sobre o que era automação ainda não estava tão desenvolvida, de modo que alguns processos eram bem automatizados, mas outros não. Nessa época, um defeito vergonhoso apareceu em produção na aplicação. De repente, estávamos vendo uma explosão de erros nos logs em circunstâncias desconhecidas e difíceis de determinar. Não conseguíamos reproduzir o problema em nossos ambientes de teste. Tentamos de tudo: testes de carga no ambiente de desempenho, simulações do que parecia um problema patológico de produção, mas não conseguíamos reproduzir o que estava acontecendo. Finalmente, depois de muitas investigações, decidimos auditar tudo o que pensávamos que poderia estar diferente entre os dois sistemas. Enfim, encontramos uma única biblioteca binária, da qual nosso sistema dependia, que pertencia a um software que estávamos usando e que era diferente entre o ambiente de produção e de testes. Mudamos a versão em produção e o problema desapareceu.

A moral dessa história não é que não éramos esforçados ou cuidadosos, ou que éramos espertos porque pensamos em auditar o sistema. A moral é que software pode ser muito frágil. Esse era um sistema consideravelmente grande, com dezenas de milhares de classes, milhares de bibliotecas e muitos pontos de integração com sistemas externos. Ainda assim, um erro sério foi introduzido em produção devido a alguns poucos bytes diferentes entre versões de uma biblioteca de terceiros.

Nos muitos gigabytes de informação que compõem um sistema moderno de software, nenhum ser humano – ou equipe de seres humanos – será capaz de identificar uma mudança na escala do exemplo descrito acima sem auxílio de máquinas. Em vez de esperar para que o problema ocorra, por que não usar máquinas para eliminar essa possibilidade?

O gerenciamento ativo de tudo que pode mudar com o uso de um sistema de versionamento – arquivos de configuração, scripts de criação de banco de dados, scripts de compilação, frameworks de testes e versões de ambiente e configurações de sistema operacionais – permite que computadores trabalhem no que são bons: garantir que tudo está onde esperamos que esteja, pelo menos até o momento em que o código começa a ser executado.

O custo da gerência manual de configuração

Outro projeto em que trabalhamos tinha um grande número de ambientes de testes dedicados. Cada um rodava um servidor de aplicações EJB conhecido. Essa aplicação era desenvolvida como um projeto ágil e tinha boa cobertura de testes automatizados. A compilação local era bem gerenciada, e era relativamente fácil para um desenvolvedor executar o código localmente de forma a continuar o desenvolvimento. Entretanto, isso foi antes de começarmos a ter mais cuidado com

a automação da implantação. Cada ambiente de testes era configurado de forma manual, usando ferramentas de terminal oferecidas pelo servidor de aplicação. Embora uma cópia dos arquivos de configuração usada pelos desenvolvedores para configurar a aplicação localmente fosse guardada sob controle de versão, as configurações dos ambientes de testes não o eram. Cada uma era diferente: cada uma tinha propriedades em ordens diferentes, em algumas faltavam propriedades e outras tinham valores ou nomes de propriedades diferentes. Não existiam dois ambientes similares e todos eram diferentes dos ambientes de produção. Era incrivelmente difícil determinar quais propriedades eram essenciais, quais eram redundantes, quais deveriam ser comuns a todos os ambientes e quais deveriam ser únicas. O resultado era que o projeto empregava uma equipe de cinco pessoas responsáveis por gerenciar as configurações desses diferentes ambientes.

Nossa experiência mostra que essa dependência da configuração manual é comum. Em muitas das organizações com as quais trabalhamos, isso vale tanto para ambientes de produção como de testes. Às vezes, não importa se o servidor A tem um conjunto de conexões limitado a 120 enquanto o servidor B está limitado a 100. Em outras, importa bastante.

Quais diferenças nas configurações importam e quais não importam é algo que você quer descobrir por acaso durante o período mais agitado do dia. Esse tipo de informação de configuração define o ambiente em que o código executa, e frequentemente especifica caminhos novos ao longo desse código. Mudanças nessas configurações devem ser levadas em consideração, e o ambiente em que o código executa deve ser tão bem definido e controlado quanto o comportamento do próprio código. Se tivermos acesso à configuração de seu banco de dados, servidor de aplicações ou servidor Web, garantimos que podemos fazer sua aplicação falhar mais rápido do que tendo acesso ao compilador e ao código-fonte.

Quando tais parâmetros de configuração são definidos e geridos manualmente, eles sofrem com o fato de que humanos tendem a cometer erros em tarefas repetitivas. Um simples erro de grafia pode impedir o funcionamento correto de uma aplicação. Pior do que isso, linguagens de programação em geral possuem verificadores de sintaxe e testes unitários para garantir que não há erros de grafia – esses testes, porém, quase nunca são aplicados a informação de configuração, sobretudo se esta é digitada diretamente em um terminal.

O simples ato de adicionar as informações de configuração em um sistema de controle de versão é uma evolução enorme. No mínimo, o sistema alertará sobre o fato de que você mudou a configuração de forma inadvertida. Isso elimina pelo menos uma fonte muito comum de erros.

Quando toda a informação de configuração estiver armazenada em um sistema de controle de versão, o próximo passo óbvio é eliminar o intermediário e deixar que o computador aplique a configuração em vez de digitá-la novamente. Isso é mais fácil em algumas tecnologias do que em outras e, às vezes, você e os próprios fornecedores de infraestrutura ficarão surpresos com até que ponto você pode chegar se começar a pensar com atenção sobre configuração – mesmo dos sistemas de terceiros mais complicados. Discutiremos isso em mais detalhes no Capítulo 4 e no Capítulo 11.

Reduzir o estresse

Entre os benefícios óbvios, o mais agradável é a redução do estresse de todos os envolvidos com uma entrega de versão. A maioria das pessoas que já se envolveu em um projeto de software próximo da data de sua entrega de versão sabe o quão estressante o evento pode ser. Nossa experiência mostra que isso também pode se tornar uma fonte de problemas. Já vimos gerentes de projetos conservadores, sensíveis e conscientes sobre qualidade perguntarem para seus desenvolvedores: "Vocês não podem simplesmente modificar o código?", ou administradores de bancos de dados sensatos inserirem dados diretamente no banco de dados para aplicações que não conhecem. Em ambas as ocasiões, e em muitas outras como essas, a mudança foi uma resposta direta à pressão de "simplesmente fazer a coisa funcionar".

Não nos entenda mal – também já estivemos na mesma situação. Não estamos nem sugerindo que essa seja sempre a resposta errada: se você colocou código em produção que está fazendo sua organização perder dinheiro, quase qualquer medida que impeça isso pode ser justificada.

O argumento que estamos abordando é outro. Em ambos os casos acima, as ações rápidas para colocar o sistema em produção não foram guiadas por motivações comerciais, mas por pressão mais sutil de entregar a versão no dia previsto. O problema é que tais entregas em produção são eventos importantes. Enquanto isso acontecer, esses eventos serão cheios de cerimônia e de nervosismo.

Por um momento, imagine se sua próxima entrega pudesse ser feita com um simples clique em um botão. Imagine que ela pudesse ser feita em alguns minutos, ou mesmo em segundos, e que, se acontecesse o pior, você pudesse voltar ao estado anterior no mesmo tempo. Imagine que você estivesse entregando frequentemente, de modo que a variação entre o que está em produção e o que está em desenvolvimento fosse muito pequena. Se isso fosse verdade, o risco de qualquer entrega seria significativamente menor, e toda aquela sensação desagradável de que você está apostando sua carreira no sucesso dessa entrega seria bem reduzida.

Para um conjunto pequeno de projetos, esse ideal pode não ser atingível na prática. Entretanto, para a maioria dos projetos ele é, embora exija certo grau de esforço. A chave para reduzir o estresse é ter o tipo de processo automatizado que descrevemos, exercitá-lo frequentemente e ter uma boa explicação pronta para quando for necessário voltar atrás se acontecer o pior. Na primeira vez em que você exercitar automação em seus projetos, isso será difícil – mas com o tempo ficará mais fácil, e os benefícios para você e para o projeto serão incalculáveis.

Flexibilidade de implantação

Executar a aplicação em um ambiente novo deveria ser uma tarefa simples – idealmente, seria apenas uma questão de ativar novas máquinas físicas ou virtuais e criar algum tipo de informação que descreve as propriedades únicas daquele ambiente. Você conseguiria usar seu processo automatizado para pre-

parar o ambiente para a entrega, e então implantar nele uma versão escolhida do software.

Executando software corporativo em um laptop

Estávamos trabalhando em um projeto recente cujo plano de negócios havia sido subitamente invalidado por uma mudança na legislação. O projeto deveria criar um sistema central para um novo tipo de negócio que seria distribuído internacionalmente, e por isso o software tinha sido projetado para rodar em uma coleção grande e heterogênea de máquinas caras. Naturalmente, todos estavam um pouco desanimados com a notícia de que a razão de o projeto existir acabara de desaparecer.

No entanto, víamos um ponto positivo nisso. A organização para a qual estávamos desenvolvendo o software fizera uma análise de downsizing. Nossos clientes perguntaram qual seria a menor especificação de máquina para rodar o sistema e como os custos de capital poderiam ser limitados. Nossa resposta foi que o software rodaria no laptop em que trabalhávamos. Eles ficaram surpresos, já que esse era um sistema multiusuário sofisticado e, após refletirem sobre isso, perguntaram como sabíamos que funcionaria. Respondemos mostrando o funcionamento de todos os testes e perguntamos que tipo de carga o sistema deveria suportar. Com a resposta em mãos, mudamos uma linha nos parâmetros dos testes de desempenho e os executamos. Mostramos que o laptop era lento, mas nem tanto. Um servidor configurado de forma decente suportaria as necessidades deles e, quando estivesse disponível, levaria apenas alguns minutos para executar a aplicação em seu ambiente.

Esse tipo de flexibilidade não é somente uma função do tipo de técnicas automatizadas que estamos descrevendo neste livro; a aplicação também era bem projetada. Entretanto, nossa habilidade de colocar o software onde fosse necessário, sob demanda, deu ao nosso cliente e a nós uma grande confiança em nossa habilidade de gerenciar a entrega a qualquer momento. Quando as entregas de versão se tornam menos sensíveis, é mais fácil considerar aspectos como o ambiente ágil ideal para a entrega no fim de uma iteração. Mesmo se isso não for apropriado para um tipo específico de projeto, ainda assim significa que geralmente temos nossos fins de semana de volta.

A prática leva à perfeição

Nos projetos em que trabalhamos, tentamos obter um ambiente dedicado para cada desenvolvedor ou par de desenvolvedores. Entretanto, mesmo em projetos que não chegam a esse ponto, qualquer time que usa integração contínua ou técnicas de desenvolvimento incrementais e iterativas precisará implantar a aplicação em diversos ambientes com frequência.

A melhor tática é usar a mesma estratégia de implantação seja qual for o ambiente final. Não devem existir estratégias especiais para QA (*quality assurance*), para testes de aceitação ou para o ambiente de produção. Dessa forma, toda vez que a aplicação é implantada em algum ambiente, confirma-se que o

processo funciona. Em essência, o processo final de entrega em produção está sendo ensaiado continuamente.

Há um caso especial em que se permite alguma variação: o ambiente de desenvolvimento. Faz sentido que os desenvolvedores precisem criar binários em vez de usar binários previamente preparados em algum outro lugar, de modo que essa limitação pode ser removida dessas entregas específicas. Mesmo assim, deve-se tentar utilizar o processo também em ambientes de desenvolvimento.

A versão candidata

O que é uma versão candidata? Uma mudança feita no código pode ou não fazer parte de uma versão. Se você olhar para a mudança e se perguntar se deve ou não implantá-la em uma versão, qualquer resposta será apenas um palpite. Todo o processo de compilação, implantação e testes que aplicamos à mudança valida se ela pode ou não fazer parte de uma versão. O processo nos dá confiança de que a mudança é segura. Analisamos essa pequena mudança – seja nova funcionalidade, correção ou ajuste do sistema para conseguir alguma alteração de desempenho – e verificamos com algum grau de confiança se podemos ou não gerar uma versão do sistema com ela. Para reduzir ainda mais o risco, queremos fazer essa validação no menor tempo possível.

Embora qualquer mudança gere um artefato que teoricamente pode ser entregue aos usuários, elas não começam assim. Qualquer mudança deve ser verificada de acordo com sua aptidão para isso. Se o produto resultante é livre de defeito e segue os critérios de aceitação estabelecidos pelo cliente, então ele pode ser entregue.

A maioria das abordagens para entregar versões de software identifica versões candidatas no fim do processo, o que faz algum sentido quando há trabalho associado à identificação. Quando este livro foi escrito, a entrada da Wikipédia descrevendo os passos de desenvolvimento (Figura 1.2) mostrava "versão candidata" como um passo distinto do processo. Nós temos uma ideia um pouco diferente quanto a isso.

Figura 1.2 *Visão tradicional de versão candidata.*

Abordagens tradicionais para o desenvolvimento de software adiam a escolha de uma versão candidata até que diversos passos custosos foram rea-

lizados para garantir que o software tem qualidade suficiente e em que a funcionalidade requerida. Entretanto, em um ambiente em que compilação e verificação são automatizadas de forma significativa e acompanhadas por testes igualmente completos, não há necessidade de se perder tempo e dinheiro em testes manuais intensivos no final do projeto. Nesse estágio, a qualidade da aplicação é significativamente mais alta, de modo que o teste manual é apenas uma afirmação de que funcionalmente a aplicação está completa.

De fato, adiar os testes para o final do processo de desenvolvimento, de acordo com nossa experiência, é uma maneira infalível de *reduzir* a qualidade da aplicação. É melhor descobrir e resolver defeitos no momento em que são introduzidos. Se forem descobertos depois, é sempre mais caro resolvê-los. Os desenvolvedores já esqueceram o que estavam fazendo no momento em que criaram o defeito, e a funcionalidade pode ter mudado nesse período. Deixar os testes para o fim geralmente significa que não há tempo para resolver de fato os problemas, e apenas uma pequena parte é tratada. Queremos, portanto, encontrar e resolver esses problemas o mais cedo o possível, de preferência *antes* que sejam introduzidos no código final.

Todo check-in é uma versão em potencial

Toda mudança que um desenvolvedor realiza no código pretende agregar valor ao software de alguma maneira. Toda mudança registrada no sistema de controle de versão supostamente melhora o sistema que está sendo criado. Como podemos saber se isso é mesmo verdade? A única maneira viável é exercitar o software de modo a verificar que ele possui o valor que esperamos. Muitos projetos adiam essa parte do processo até que a maioria ou toda a funcionalidade esteja presente. Se algo não funciona nesse momento, normalmente a correção exige uma quantidade razoável de trabalho. Essa frase é descrita como integração e é a parte mais imprevisível e menos gerenciável do processo de desenvolvimento. Em função de ser uma etapa difícil, muitos times a adiam, integrando-a com pouca frequência, o que a torna ainda pior.

Em software, quando algo é difícil, a maneira de reduzir o sofrimento é fazer isso com maior frequência, e não com menor. Em vez de integrar ocasionalmente, devemos integrar como resultado de qualquer mudança no sistema. A prática de integração contínua leva isso ao extremo, criando uma mudança de paradigma no processo de desenvolvimento. A integração contínua detecta qualquer mudança que introduza problemas no sistema ou que não atenda aos critérios de aceitação especificados pelo cliente no momento em que foi introduzida. Equipes podem, então, corrigir o problema assim que ele ocorre (essa é a primeira regra de integração contínua). Quando essa prática é seguida, o software está *sempre* em um estado funcional. Se seus testes cobrem suficientemente a aplicação e você os está executando em um ambiente suficientemente similar ao de produção, então a versão do software está *sempre pronta para a entrega*.

Cada mudança é, essencialmente, uma versão candidata. Toda vez que uma mudança é introduzida no sistema de versão, a expectativa é que todos os testes resultem em sucesso, produzindo código funcional que pode ser implantado em produção. Essa é a pressuposição inicial. O trabalho do sistema de integração contínua é negar essa pressuposição, mostrando que uma versão candidata em especial não atende aos critérios necessários para produção.

Princípios da entrega de software

As ideias subjacentes a este livro vieram do trabalho realizado pelos autores em diversos projetos ao longo dos anos. Quando começamos a sintetizar nossos pensamentos e capturá-los nessas páginas, notamos que os mesmos princípios apareciam repetidamente. Iremos enumerá-los aqui. Parte do que dizemos está sujeito a interpretações e a ressalvas; os princípios abaixo não estão. Eles não podem ser deixados de lado se queremos que o processo de entrega seja eficaz.

Criar um processo de confiabilidade e repetitividade de entrega de versão

Esse princípio é uma declaração do nosso propósito ao escrever este livro: entregar uma versão de software deve ser fácil, porque você testou cada parte do processo centenas de vezes antes. Deve ser tão simples quanto apertar um botão. A repetitividade e confiabilidade derivam desses dois princípios: automatizar quase tudo e manter tudo o que você precisa para compilar, configurar, implantar e testar a aplicação dentro de um sistema de controle de versão.

Implantar um software em última instância envolve três coisas:

- Fornecer e gerenciar o ambiente em que a aplicação executará (configuração de hardware, software, infraestrutura e sistemas externos).
- Instalar a versão correta da aplicação nesse ambiente.
- Configurar a aplicação, incluindo qualquer dado ou estado que a aplicação exija.

Deve-se fazer a implantação da aplicação em produção usando um processo completamente automatizado a partir do sistema de controle de versão. A configuração da aplicação também deve ser um processo automatizado, e os scripts e dados necessários, mantidos em um sistema de controle de versão ou bancos de dados. Obviamente, hardware não pode ser versionado; mas, com a chegada de tecnologia de virtualização com custo acessível e ferramentas como Puppet, o provisionamento também pode ser automatizado.

O resto deste livro é essencialmente uma descrição de técnicas para atingir esse objetivo.

Automatize quase tudo

Há algumas coisas impossíveis de se automatizar. Testes exploratórios dependem de profissionais experientes; demonstração para representantes da comunidade de usuários não podem ser feitas por computadores; aprovações para propósitos de conformação à regulamentação ou a padrões depende de intervenção humana. Entretanto, a lista de processos que não podem ser automatizados é menor do que se pensa. Em geral, todo o processo de compilação deve ser automatizado até o ponto em que sejam necessárias direção ou decisões humanas. Isso também vale para o processo de implantação e, de fato, para todo o processo de entrega. Testes de aceitação podem ser automatizados, upgrades ou downgrades do banco de dados também podem e mesmo configurações de rede e firewall. Você deve automatizar tudo o que puder.

Os autores deste livro podem dizer honestamente que ainda não encontraram um processo de compilação ou de implantação que não possa ser automatizado com trabalho e imaginação suficientes.

Muitos dos times de desenvolvimento não automatizam seu processo de entrega porque isso parece uma tarefa assustadora. É mais fácil apenas fazer tudo de forma manual. Talvez isso se aplique na primeira vez em que você executa algum passo do processo, mas certamente não vale para a décima vez em que você o realiza, e provavelmente também não para a terceira ou quarta vez.

Automação é um pré-requisito para o pipeline de implantação, porque é somente por meio dela que podemos garantir que as pessoas terão o que precisam ao apertar de um botão. Entretanto, você não precisa automatizar tudo de uma vez só; deve começar procurando pelas partes do processo que são gargalos. Você pode, e deve, automatizar de forma gradual, com o tempo.

Mantenha tudo sob controle de versão

Tudo que é necessário para compilar, configurar, implantar, testar e entregar uma versão de sua aplicação deve ser mantido em algum tipo de sistema de versionamento. Isso inclui documentação, scripts de testes, casos de teste automatizados, scripts de configuração e rede, scripts de implantação, criação de banco de dados, atualizações e inicialização, configuração da plataforma tecnológica, bibliotecas, ferramental, documentação técnicas e assim por diante. Tudo isso deve estar sob controle de versão, e as versões relevantes devem ser identificáveis para cada compilação. Isso significa que o *conjunto* deve ter algum identificador único, como o número da compilação ou o número do item de mudança no controle de versão.

Qualquer membro novo do time deve poder sentar-se em sua nova estação de trabalho, realizar um check-out da versão atual do software no sistema de versionamento e executar um comando único para compilar e implantar a aplicação em um ambiente acessível, incluindo a estação de trabalho local.

Também deve ser possível ver qual versão da aplicação está instalada em cada ambiente, e de quais versões do código e da configuração ela veio.

Se é difícil, faça com mais frequência e amenize o sofrimento

Esse é o princípio mais geral em nossa lista, e é provável que fosse mais bem descrito como uma heurística. Talvez seja a heurística mais útil que conhecemos no contexto de entrega de software e está presente em tudo que dizemos. Integração normalmente é um processo sofrido. Se isso acontece em seu projeto, integre toda vez que alguém submete uma nova versão do código, e faça isso desde o começo do projeto. Se testar é um processo difícil que ocorre logo antes da entrega da versão, não deixe para o fim. Em vez disso, teste continuamente desde o início do projeto.

Se entregar é algo penoso, tenha como objetivo fazer isso sempre que uma mudança passar pelos testes automatizados. Se você não pode entregar para os usuários reais toda vez, use um ambiente similar ao de produção depois de cada nova versão. Se criar a documentação da aplicação é difícil, faça isso a cada nova funcionalidade em vez de deixar para o fim. Crie documentação para cada parte da funcionalidade que satisfaz os princípios de entrega, e automatize o que puder.

Dependendo do seu nível de prática, o esforço para atingir esse ponto pode ser considerável, e ao mesmo tempo você precisa entregar o software. Tenha objetivos intermediários, como uma versão interna em intervalos de algumas semanas ou, se já está fazendo isso, toda semana. Progressivamente aproxime-se do ideal – mesmo passos pequenos têm grandes benefícios.

XP, ou *Extreme Programming* (Programação Extrema), é essencialmente a aplicação dessa heurística ao processo de desenvolvimento. Muitos dos conselhos neste livro vêm da experiência de aplicar o mesmo princípio ao processo de entrega de versão de software.

A qualidade deve estar presente desde o início

Esse princípio e o último que mencionamos nesta seção – melhoria contínua – são copiados sem vergonha alguma do movimento *lean*. "Construa com qualidade desde o início" era o lema de W. Edwards Deming que, entre outras distinções, foi um dos pioneiros do movimento *lean*. Quanto mais cedo você conseguir identificar defeitos, mais barato é para corrigi-los. E é ainda mais barato corrigi-los se eles sequer entrarem no controle de versão.

As técnicas que descrevemos neste livro, como integração contínua, cobertura extensiva de testes automatizados e implantação automatizada, têm como objetivo identificar defeitos o mais cedo possível no processo de entrega (uma aplicação do princípio de amenizar o sofrimento o mais rápido possível). O próximo passo é corrigi-los. Um alarme de incêndio torna-se inútil se todo mundo o ignorar. Times de entrega devem ser disciplinados o suficiente para corrigir defeitos assim que forem encontrados.

Há dois outros corolários deste princípio. Primeiro, teste não é uma fase, e certamente não deve vir depois da fase de desenvolvimento. Se os testes forem deixados para o fim, será tarde demais e não haverá tempo para corrigir os defeitos. Segundo, os testes também não são responsabilidade, pura ou mesmo principal, dos testadores. Todos no time de entrega são responsáveis pela qualidade em todos os momentos.

Pronto quer dizer versão entregue

Quantas vezes você já ouvir um desenvolvedor dizer que uma história ou funcionalidade está "pronta"? Ou já ouviu um gerente de projeto perguntar a um desenvolvedor se algo está "pronto"? O que "pronto" realmente quer dizer? De fato, uma funcionalidade só está pronta quando está entregando valor aos usuários. Isso é parte da motivação da prática de entrega contínua (veja o Capítulo 10, "Implantação e Entrega de Versões de Aplicações").

Para alguns times de entrega ágeis, "pronto" significa entrega de uma versão em produção. Essa é a situação ideal para um projeto de desenvolvimento de software. Entretanto, nem sempre é prático usar isso como uma medida de "pronto". Pode demorar um pouco até que a entrega inicial de um sistema esteja em um estado em que usuários externos reais possam se beneficiar dela. Então, vamos voltar à segunda melhor opção e dizer que uma funcionalidade está "pronta" quando foi demonstrada com sucesso e experimentada por representantes da comunidade de usuários, em um ambiente similar à produção.

Não existe "80% pronto": algo está pronto ou não. É possível estimar o trabalho necessário para que algo esteja pronto – mas serão sempre estimativas. Usar estimativas para determinar o trabalho que ainda precisa ser feito resulta em recriminações e apontar de dedos quando se descobre que esse cálculo está, como invariavelmente acontece, errado.

Esse princípio tem um corolário interessante: não é possível que uma única pessoa faça tudo que falta para que algo esteja pronto. É necessário um número maior de pessoas na equipe trabalhando em conjunto para que a tarefa seja concluída. Por isso é importante que todos – testadores, pessoal de operação, pessoal de suporte e desenvolvedores – trabalhem juntos para entregar – um princípio tão importante que possui uma seção dedicada a ele.

Todos são responsáveis pelo processo de entrega

Em um contexto ideal, todos dentro da organização estão alinhados com seus objetivos e trabalham em conjunto para atingi-los. Em última instância, uma equipe falha ou tem sucesso como um time e não como indivíduos. Entretanto, em muitos projetos, a realidade é que os desenvolvedores entregam seu trabalho por baixo da porta para os testadores. Os testadores, por sua vez, fazem a mesma coisa com os times de operações no momento da entrega de versão. Quando algo dá errado, as pessoas perdem mais tempo culpando umas às outras do que corrigindo os defeitos que inevitavelmente surgem em tal abordagem.

Se você está trabalhando em uma organização pequena ou um departamento relativamente independente, é provável que tenha controle completo sobre os recursos necessários para entregar uma aplicação. Se esse é o caso, fantástico. Caso contrário, conseguir colocar esse princípio em prática pode exigir trabalho duro durante um bom tempo para quebrar as barreiras entre os silos que isolam as pessoas em papéis diferentes.

Comece juntando todos os envolvidos no início do processo e garanta que eles têm oportunidade de se comunicar com frequência. Quando as barreiras começarem a cair, a comunicação deve ser contínua, mas você deve buscar esse objetivo de forma incremental. Inicie um sistema em que todos podem ver rapidamente o estado da aplicação, suas condições, as várias compilações, que testes passaram e o estado dos ambientes que foram implantados. Esse sistema deve garantir que as pessoas possam fazer seu trabalho, tal como a implantação de ambientes sob seu controle.

Esse é um dos princípios centrais do movimento DevOps. O movimento DevOps está centrado no mesmo objetivo que estabelecemos para este livro: encorajar uma colaboração maior entre todos os envolvidos no processo de entrega de software de valia com maior confiança e rapidez [aNgvoV].

Melhoria contínua

Vale a pena enfatizar que a primeira entrega de versão de uma aplicação é o primeiro estágio em seu ciclo de vida. Todas as aplicações evoluem, e haverá mais entregas. É importante que o processo de entrega também evolua com elas.

O time completo deve se encontrar rapidamente e fazer uma retrospectiva sobre o processo de desenvolvimento. Isso significa que ele deve refletir sobre o que funcionou bem, o que não funcionou, e discutir ideias sobre como melhorar as coisas. Alguém deve ser nomeado como responsável pela ideia e garantir que ela seja realizada. Então, na próxima vez em que o time se encontrar, essas pessoas devem reportar sobre o que foi feito. Isso é conhecido como o *ciclo de Deming*: planejar, executar, verificar e agir.

É essencial que todos na organização estejam envolvidos com esse processo. Permitir que haja feedback somente dentro dos silos e não entre todos é uma receita para o desastre: leva a otimizações localizadas em vez de a otimização geral – e, em última instância, dedos apontados para os outros.

Resumo

Tradicionalmente, a entrega de versão de software sempre foi um momento estressante para os envolvidos. Ao mesmo tempo, comparada com as disciplinas associadas à criação e ao gerenciamento de código, é tratada com um processo manual, sem verificação, que se baseia em técnicas de gerência de configuração *ad hoc* para aspectos cruciais do sistema. Em nossa visão, o estresse associado com essas entregas e sua natureza manual estão relacionados.

Ao adotar as técnicas de automação deste livro, colhemos seus benefícios. Ganhamos visibilidade para verificar mudanças, tornar o processo passível de repetição em diversos ambientes e eliminar em grande escala as chances de que ocorram erros em produção. Ganhamos a habilidade de entregar mudanças logo e colher os benefícios de negócio mais rapidamente, porque o processo em si não é mais um impedimento. A implementação de um sistema automatizado encoraja a implementação de outras partes, como desenvolvimento guiado por comportamento (*behavior-driven development* – BDD) e gerência extensiva de configuração.

Também ganhamos fins de semana com nossas famílias e amigos, bem como vidas menos estressadas, e somos ao mesmo tempo mais produtivos. Como não gostar disso? A vida é muito curta para passarmos os nossos fins de semana em salas de servidores implantando aplicações.

A automação do desenvolvimento, testes e entrega tem um impacto profundo na velocidade, qualidade e no custo do software. Um dos autores trabalhou em um complexo sistema distribuído. Implantar esse sistema em produção, incluindo migração de dados em grandes bancos de dados, demora de 5 a 20 minutos, dependendo das migrações associadas àquela entrega específica. Mover dados demanda um bom tempo. Um sistema similar que conhecemos gasta cerca de trinta dias na mesma parte do processo.

O restante do livro é centrado em conselhos mais concretos e recomendações, mas queremos que este capítulo ofereça uma visão ideal, mas realística, de seu escopo. Os projetos aos quais nos referimos aqui são projetos reais e, embora tenhamos disfarçado um pouco para proteger os culpados, tentamos não exagerar nos detalhes técnicos ou no valor das técnicas.

Jez Humble | David Farley

Entrega Contínua

Como Entregar Software de Forma Rápida e Confiável

Entregar uma nova versão de software para usuários costuma ser um processo cansativo, arriscado e demorado. Mas por meio da automação dos processos de compilação, implantação e teste, e da colaboração otimizada entre desenvolvedores, testadores e a equipe de operações, times de entrega podem lançar mudanças em questão de horas – algumas vezes, em minutos –, independentemente do tamanho do projeto ou da complexidade de seu código. Neste livro, Jez Humble e David Farley apresentam os princípios, as práticas e as técnicas de ponta que tornam possível uma entrega rápida e de alta qualidade.

Descubra como:

- Automatizar todas as etapas dos processos de desenvolvimento, integração, teste e implantação de software
- Implementar pipeline de implantação nos níveis organizacional e do time
- Otimizar a colaboração entre desenvolvedores, testadores e a equipe de operações
- Desenvolver características de forma incremental em times grandes e distribuídos
- Implementar uma estratégia eficaz de gestão de configuração
- Automatizar testes de aceitação, da análise à implementação
- Testar capacidade e outros requisitos não funcionais
- Pôr em prática a implementação contínua
- Gerenciar infraestrutura, dados, componentes e dependências
- Lidar com gestão de riscos, observância e auditoria

Entrega Contínua é um recurso indispensável para desenvolvedores, programadores, administradores de sistemas, testadores, gestores e equipes que desejam criar uma cultura de entrega rápida, frequente e confiável de novas versões de software.

ENGENHARIA DE SOFTWARE

ISBN 978-85-8260-103-7

www.grupoa.com.br