1. Conceitos básicos do *DevOps*

Antonio Muniz

Este capítulo apresenta as origens do *DevOps* e seus conceitos introdutórios.

1.1. Introdução ao *DevOps* e sua ligação com *Lean* e métodos ágeis

A palavra *DevOps* é a contração de dois termos em inglês que identificam as equipes envolvidas nas atividades de construção e implantação de software:

- ✓ ***Development*** **(Desenvolvimento):** equipe responsável pela identificação dos requisitos com o cliente, a análise, o projeto, a codificação e os testes.
- ✓ ***Operations*** **(Operações):** equipe responsável pela implantação em produção, pelo monitoramento e pela solução de incidentes e problemas.

A criação do termo *DevOps* foi motivada pela apresentação de John Allspaw e Paul Hammond em 2009 na Conferência *Velocity* da O'Reilly intitulada "Mais de 10 Implantações por dia: Cooperação ***Dev*** e ***Ops*** no Flickr". No prefácio do livro "Effective DevOps" (2016), John Allspaw ressalta que muitos lembram somente da parte "10 implantações por dia" quando deveriam se concentrar mais na parte "**Cooperação entre *Dev* e *Ops***". O alerta de um dos criadores do movimento *DevOps* reforça que os benefícios dessa cultura vão muito além da habitual implantação de Jenkins com uma equipe dedicada.

Considere que *DevOps* é uma cultura fortemente colaborativa entre as equipes de Desenvolvimento e Operações para entregar o software funcionando em produção de forma ágil, segura e estável. Mais do que um conceito, é importante destacar que *DevOps* é uma jornada de aproximação entre as pessoas com ações práticas de automação para acelerar as implantações com qualidade, considerando o ponto de vista de todos os envolvidos, a tão falada empatia.

Para enriquecer esse conceito inicial, destaco a seguir algumas visões complementares do mercado sobre *DevOps*, visto sua abrangência e falta de consenso geral:

1. **Amazon**: "*DevOps* é a combinação de filosofias culturais, práticas e ferramentas que aumentam a capacidade de uma empresa de distribuir aplicativos e serviços em alta velocidade: otimizando e aperfeiçoando produtos em um ritmo mais rápido do que o das empresas que usam processos tradicionais de desenvolvimento de software e gerenciamento de infraestrutura".
2. **Gartner**: "o *DevOps* enfatiza as pessoas (e a cultura) e procura melhorar a colaboração entre as operações e as equipes de desenvolvimento. As implementações de *DevOps* utilizam tecnologia – especialmente ferramentas de automação que podem alavancar uma infra cada vez mais programável e dinâmica de uma perspectiva de ciclo de vida".
3. **Atlassian**: "o *DevOps* é um conjunto de práticas que automatizam os processos entre equipes de desenvolvimento de software e de TI para que possam criar, testar e liberar softwares de maneira mais rápida e confiável".

O acrônimo CALMS é muito conhecido para representar a cultura *DevOps* e foi criado em 2010 como CAMS por John Willis e Damon Edwards. Posteriormente, esse termo foi aperfeiçoado por Jez Humble com a inclusão do L para destacar a importância do *Lean* para a melhoria contínua e os processos enxutos. Observe na Figura 1.1 a seguir um resumo do CALMS.

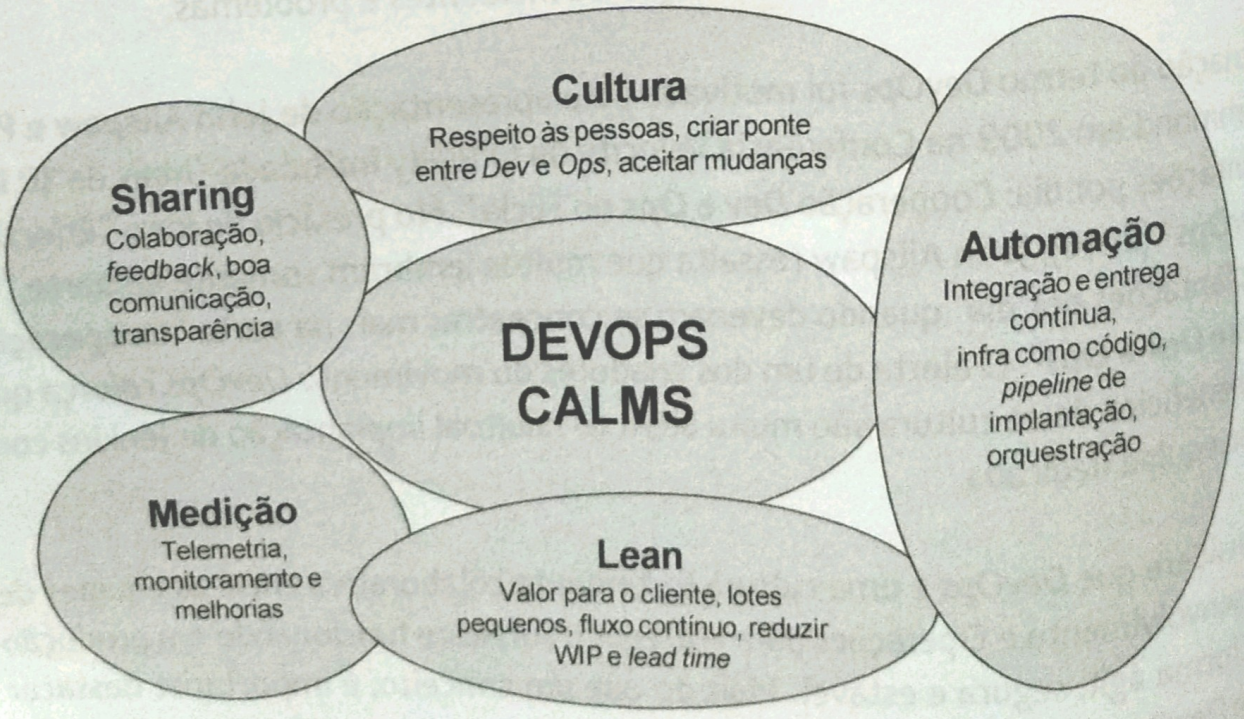

Figura 1.1. *DevOps* CALMS.
Fonte: MUNIZ; ADAPTNOW, videoaula oficial Exin, 2018.

Concordo com a abordagem de Davis e Daniels, no livro "Effective DevOps" (2016), ao recomendar que as organizações invistam em ações concretas para disseminar a cultura *DevOps* e promovam os quatro pilares para o *DevOps* efetivo, destacados a seguir:

1. **Colaboração:** significa construir um resultado específico com interações de pessoas com diferentes experiências e um propósito comum. Um princípio orientador que iniciou o movimento *DevOps* foi a cooperação genuína das equipes de desenvolvimento de software e operações na Flickr em 2009. Antes que um time trabalhe com sucesso interagindo com outro time em um enfoque diferente, os indivíduos precisam construir uma relação de confiança.
2. **Afinidade:** construir relações interdependentes fortes entre os times interfuncionais para que todos vejam sentido ao navegar por objetivos organizacionais complementares e sintam-se interessados naturalmente pelo sentimento de empatia e aprendizagem contínua.
3. **Ferramentas:** funcionam como um acelerador, impulsionando a mudança com base na cultura atual. Se ferramentas, ou a falta delas, atrapalharem as pessoas que já trabalham bem juntas, suas iniciativas não serão bem-sucedidas. Por outro lado, se o custo da colaboração for alto, não investir em ferramentas (ou pior, investir em ferramentas ruins) aumenta esse custo.
4. **Escala:** o dimensionamento leva em conta como os outros três pilares podem ser aplicados à medida que as organizações crescem, amadurecem e até encolhem, considerando questões técnicas e culturais.

O grande motivador para o movimento de integração em *DevOps* é que os departamentos tradicionais de TI são divididos, normalmente, entre duas grandes áreas que costumam ficar isoladas em silos e são orientadas por objetivos conflitantes.

Observe a seguir como esse tradicional conflito de objetivos e interesses cria um muro ou uma espiral descendente, que dificulta fortemente a colaboração entre essas equipes resultando em um tempo cada vez maior para a comercialização de novos produtos e recursos, maiores interrupções por falhas no software (piora da qualidade) e o pior de tudo: aumento da dívida técnica (Figura 1.2).

Entre essas duas grandes áreas de TI geralmente existem alguns departamentos que executam ações de controle ao final do processo para minimizar riscos e impactos negativos nas implantações. Como exemplos desses departamentos temos: segurança da informação, qualidade, governança, gestão de mudanças, *compliance*, auditoria, etc.

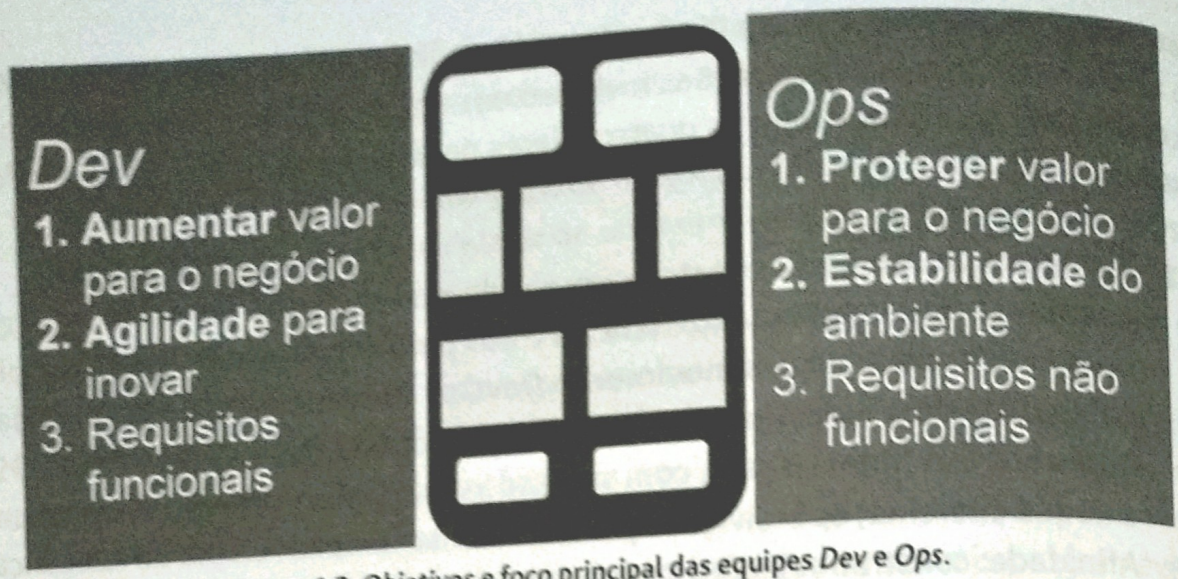

Figura 1.2. Objetivos e foco principal das equipes *Dev* e *Ops*.
Fonte: MUNIZ; ADAPTNOW, videoaula oficial Exin, 2018.

O problema que tem se agravado atualmente é a dinâmica altamente veloz e competitiva do mercado, que não permite mais que uma equipe seja a única responsável pelo controle da qualidade no final do desenvolvimento tradicional, que costuma durar meses ou até anos. A filosofia *DevOps* preconiza que todos os envolvidos se sintam genuinamente proprietários do resultado final que o cliente recebe como experiência na prática, considerando os requisitos funcionais e não funcionais desde o início dos projetos.

Quando não há uma cultura *DevOps* e ocorre um incidente crítico em produção após uma implantação, o cenário a seguir é bastante comum:

- ✓ Equipe de Desenvolvimento afirma que o código da aplicação está funcionando perfeitamente e diz que o problema está relacionado com algum componente de infraestrutura. A frase clássica é "na minha máquina funciona".
- ✓ Equipe de Operação garante que o problema está relacionado com a falta de qualidade do desenvolvimento. A frase clássica é "eles nunca codificam ou testam direito".
- ✓ Dependendo de como está estruturada a organização, a culpa pode recair na equipe de segurança da informação, por ter feito alguma implantação no dia anterior para remover determinada vulnerabilidade em produção.
- ✓ As acusações podem durar horas ou dias e geram o sentimento de injustiça e desconfiança entre as pessoas que prejudicará futuras atividades com as mesmas equipes.

A Figura 1.3 resume essas visões conflitantes e o impacto para o cliente.

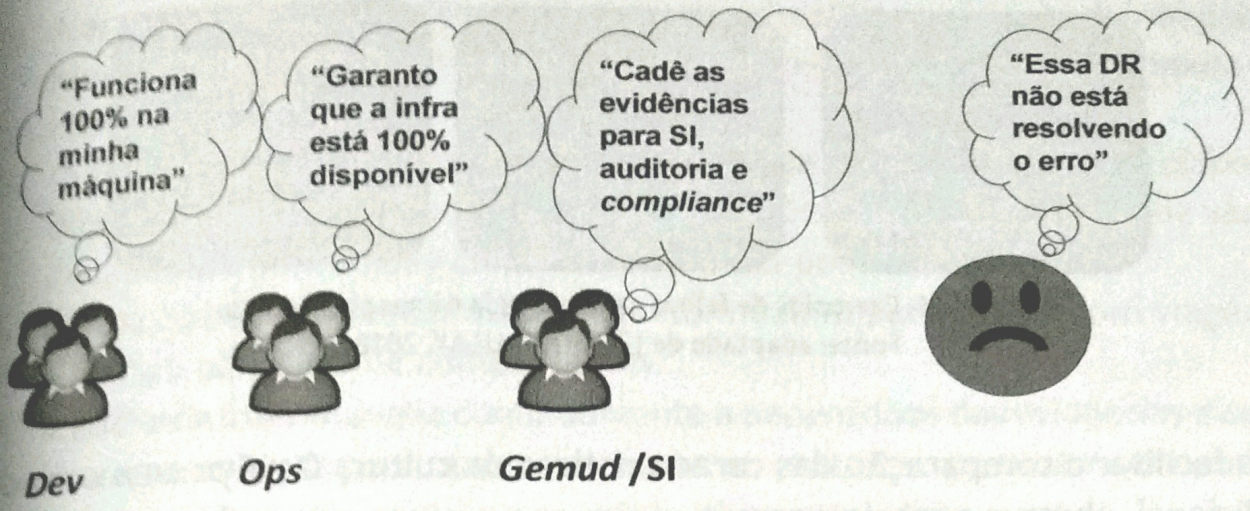

Figura 1.3. Problema clássico após implantação em produção sem cultura *DevOps*.
Fonte: MUNIZ, palestra na *Campus Party*, 2019.

Existem literaturas, incluindo o Gartner, que usam o termo *DevSecOps* para enfatizar a segurança da informação na jornada *DevOps*. O presente livro usará sempre o termo *DevOps*, pois é dessa forma que a prova oficial de certificação Exin aborda o assunto. Além disso, vários autores que iniciaram esse movimento consideram que o termo *DevOps* original já contempla as ações colaborativas como um todo, incluindo segurança da informação, equipe de qualidade, gestão de mudanças, etc.

Não há dúvida de que muitas falhas de software e incidentes de segurança, divulgados com uma frequência cada vez maior, poderiam ser evitados com o *DevOps* no sentido mais amplo, considerando segurança da informação e qualidade.

Por isso, algumas organizações, para resolver falhas de software decorrentes de ambientes de testes e de integração não confiáveis, começaram a se preocupar em integrar especialistas de operação e especialistas de desenvolvimento. Em 2013, segundo o livro "The DevOps Handbook", na Big Fish Games, Paul Farrall definiu dois tipos de ligação *Ops*: o gerente de relacionamento comercial – que faz o papel de impulsionador do negócio – e o engenheiro de *release* dedicado – que é um especialista em desenvolvimento e qualidade do produto.

Ainda assim, na maioria das organizações a situação era diferente. Segundo estudo global realizado em 2017 pelo Instituto de Desenvolvimento e Gestão (IDG), em parceria com a consultoria PwC, a perda financeira anual das empresas brasileiras com incidentes de segurança varia de US$ 10 mil a US$ 20 milhões, dependendo do porte e do ramo de atuação. Além disso, uma falha de software impacta também a imagem das organizações. Observe alguns exemplos reais a seguir (Figura 1.4).

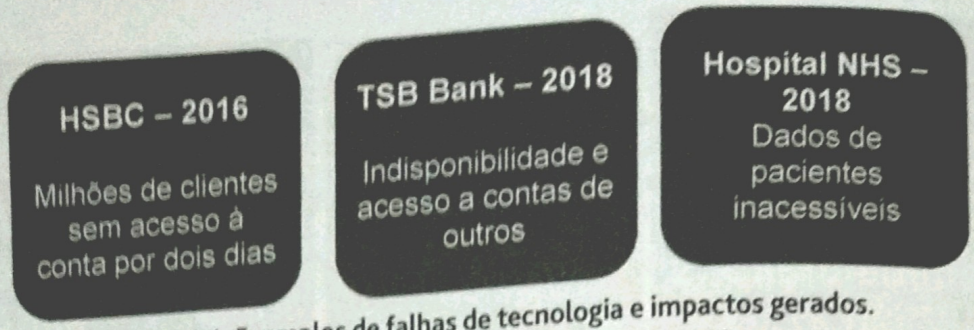

Figura 1.4. Exemplos de falhas de tecnologia e impactos gerados.
Fonte: adaptado de JEE; MACAULAY, 2018.

Para facilitar a comparação das características da cultura *DevOps* com o modelo tradicional, observe a tabela a seguir.

Modelo tradicional	Cultura *DevOps*
Cultura do medo e do grito	Confiança e experimentação
Causa da falha foi o Fulano	Resolveremos a falha sistêmica
Competição entre departamentos	Colaboração multidisciplinar
Cada um no seu quadrado	Visão mais horizontal
Teste manual no final	Teste automatizado na origem
Merge complexo e demorado	Integração em lotes pequenos
Tudo funciona na minha máquina	Ambiente similar de produção
Implantação manual	Implantação automatizada
Implantação demorada	Implantação contínua
Big bang	Teste A/B
Acho que o problema foi xxx	Fatos e dados com telemetria

Vale destacar que a empatia e o senso colaborativo dos times são fundamentais na jornada *DevOps*, e esse *mindset* pode ser expandido para todas as áreas da organização. Reflita como incentivar e adaptar na sua empresa essa cultura nos exemplos a seguir:

1. A área de vendas verifica com a operação o que precisa ser combinado antes de fechar um grande negócio que exigirá a fabricação de milhares de produtos nas próximas semanas.
2. Antes de dizer não, a área de operações combina com todos os envolvidos o que pode ser feito para priorizar uma entrega que é importante para todos na empresa.
3. O marketing avalia previamente com os times as ações necessárias para uma campanha em várias mídias que aumentará muito a demanda nos acessos de vários sites e sistemas.

4. A área jurídica reavalia seus processos para simplificar as exigências de documentação e eliminar aprovações para compras com baixa complexidade ou custos baixos.
5. O departamento de RH estimula uma relação de confiança com seus colaboradores e elimina a obrigatoriedade de comprovantes (cópia todo mês de vários documentos para auxílio creche/escola, bater ponto, etc.).
6. A área administrativa combina um teto máximo para gastos com viagem e elimina a obrigação de comprovantes.
7. A auditoria interna avalia continuamente a necessidade das evidências e combina com as áreas para manter somente os controles que realmente podem mitigar os riscos do negócio e as exigências regulatórias.
8. A área de produtos convida uma equipe multidisciplinar para esclarecer as ideias na origem e entender a opinião das outras áreas da organização.
9. Antes de dizer não, a segurança da informação estabelece uma relação de confiança sobre a importância de reduzir vulnerabilidades e passa a ser vista como uma área que colabora para que todos consigam implantar essas práticas em suas atividades diárias.
10. Os times se sentem proprietários do resultado final entregue ao cliente que recebe o produto ou serviço e não transferem essa responsabilidade para quem atua no SAC.

A Figura 1.5 resume um pouco o impacto positivo da cultura *DevOps*.

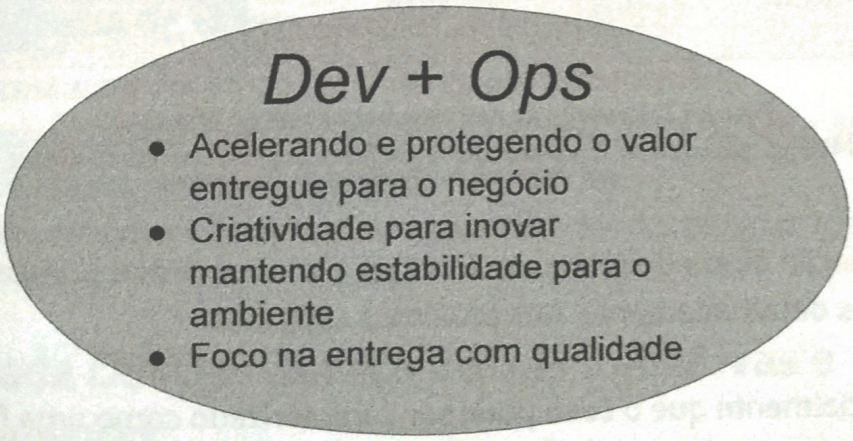

Figura 1.5. Redução dos conflitos entre as equipes com *DevOps*.
Fonte: elaborado pelo autor.

As ferramentas para automação têm uma grande importância para o sucesso do *DevOps*, porém o principal desafio é criar uma cultura em que todos se tornem comprometidos com a qualidade desde o início do ciclo de vida do software até sua entrega em produção e monitoramento da experiência real do usuário.

A Figura 1.6 destaca alguns benefícios que várias empresas alcançaram com a jornada *DevOps* em vários países e segmentos de mercado.

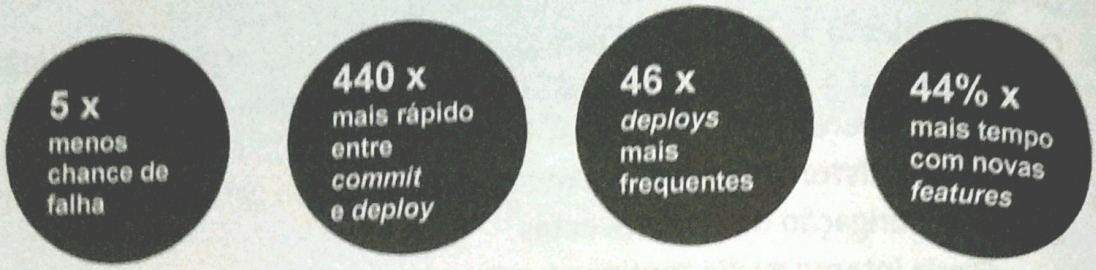

Figura 1.6. Benefícios com a adoção de *DevOps*.
Fonte: adaptado de PUPPET, 2017.

Um exemplo interessante sobre *deploy* frequente e entrega de novas *features* é o WhatsApp, que apresento na Figura 1.7.

Figura 1.7. Exemplo de entregas frequentes do WhatsApp.
Fonte: MUNIZ, palestra no *The Developer's Conference* (TDC) em Porto Alegre, 2018.

Existe uma relação direta do *Lean* com *DevOps*, e muitas dessas práticas serão apresentadas, mais detalhadamente, nos próximos capítulos.

Considere inicialmente que o *Lean* pode ser caracterizado como uma filosofia colaborativa para redução de desperdícios e entrega de valor que o cliente está disposto a pagar.

Destaco a seguir alguns conceitos iniciais do *Lean* aplicados ao *DevOps*:

- ✓ **Fluxo de valor:** é o processo que concretiza uma necessidade de negócio em um produto ou serviço para entrega de valor ao cliente.

- ✓ **Mapeamento do fluxo de valor:** visa entender como o processo funciona com foco na entrega de valor ao cliente e identifica gargalos ou desperdícios.
- ✓ ***Gemba*:** é o local onde as coisas acontecem. Todos deveriam ir ao *gemba* com frequência para conhecer o "chão de fábrica" e evitar suposições sem dados e fatos.
- ✓ ***Obeya*:** também conhecida nas organizações como "sala de guerra", o objetivo é facilitar a gestão visual e a coordenação para solução de problemas sem os entraves das estruturas clássicas das organizações.
- ✓ ***Kanban*:** quadro que permite visualizar onde o trabalho está fluindo bem e onde está na fila ou interrompido. Veja um exemplo desse quadro no decorrer deste capítulo, quando falarmos sobre WIP.
- ✓ **Corda de *Andon*:** dispositivo que existe nas fábricas da Toyota para interromper a linha de produção quando é encontrado algum defeito nos produtos. O objetivo da aglomeração (*swarming*) é conter os problemas antes que estes tenham a chance de se espalhar. As equipes e líderes que podem ajudar a resolver o problema na origem são aglomerados imediatamente, podendo mobilizar os executivos, executivos de tecnologia e até a alta administração.
- ✓ **Desperdício:** toda atividade que consome recursos sem adicionar valor na visão do cliente. Existem oito tipos de desperdícios identificados no *Lean*:
  - **Defeito e retrabalho:** desfazer, refazer algo.
  - **Movimentação:** caminhadas, deslocamentos, viagens.
  - **Espera:** pessoas aguardando informações, materiais ou outras equipes.
  - **Transporte:** transferências desnecessárias de materiais ou informações.
  - **Estoques:** informações ou materiais sem uso.
  - **Processamento:** etapa redundante ou desnecessária.
  - **Desconexão ou superprodução:** fluxo deficiente ou falta de sincronismo entre etapas (antes ou depois do necessário).
  - **Conhecimento:** não aproveitar as habilidades das pessoas adequadamente.

1.2. Jornada *DevOps* abraça outras iniciativas e *frameworks*

Conforme ilustrado a seguir, tenha em mente que *DevOps* é uma jornada que contempla a conjunção de várias filosofias, métodos, práticas e ferramentas que unem as equipes para a entrega ágil de software confiável (Figura 1.8).

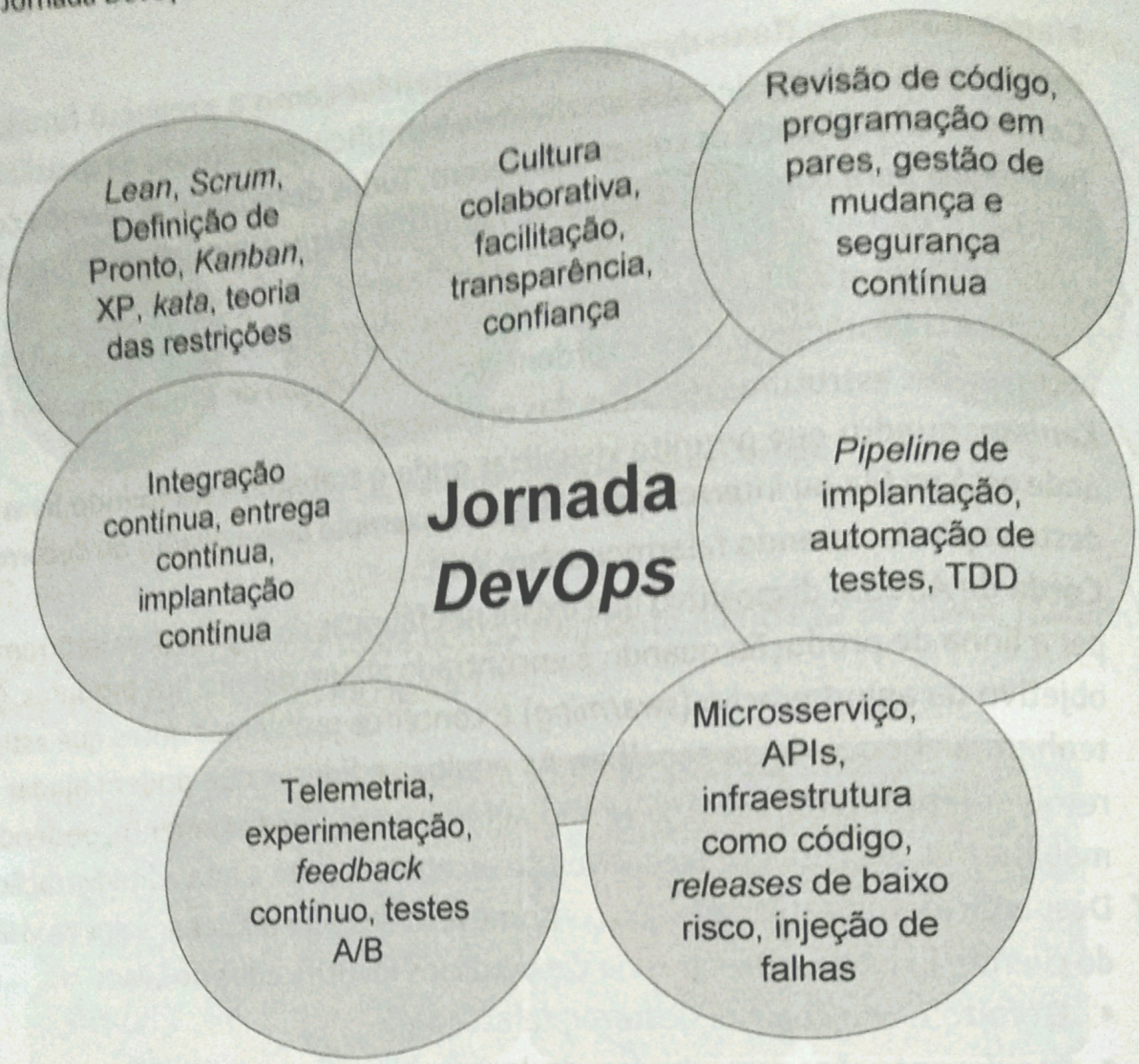

Figura 1.8. Resumo da jornada *DevOps*.
Fonte: MUNIZ, palestra no *DevOps Days* BH e TDC Porto Alegre, 2018.

Podemos considerar que *DevOps* é uma continuação natural da jornada ágil, visando entregar valor ao cliente com foco em adaptabilidade e aprendizado contínuo. Um dos benefícios da adoção de *DevOps* com métodos ágeis é a mobilização de todas as equipes que participam do fluxo de valor em uma abordagem ponta a ponta, desde o levantamento de requisitos até a entrega do software no ambiente de produção.

Uma dúvida bastante recorrente na comunidade e nas organizações é onde o *DevOps* se encaixa nos *frameworks* e movimentos existentes, como *Design Thinking*, *Lean*, *Scrum*, XP, *Kanban*, *Lean startup*, etc. Para facilitar esse entendimento, apresento na figura a seguir um resumo com essas ligações.

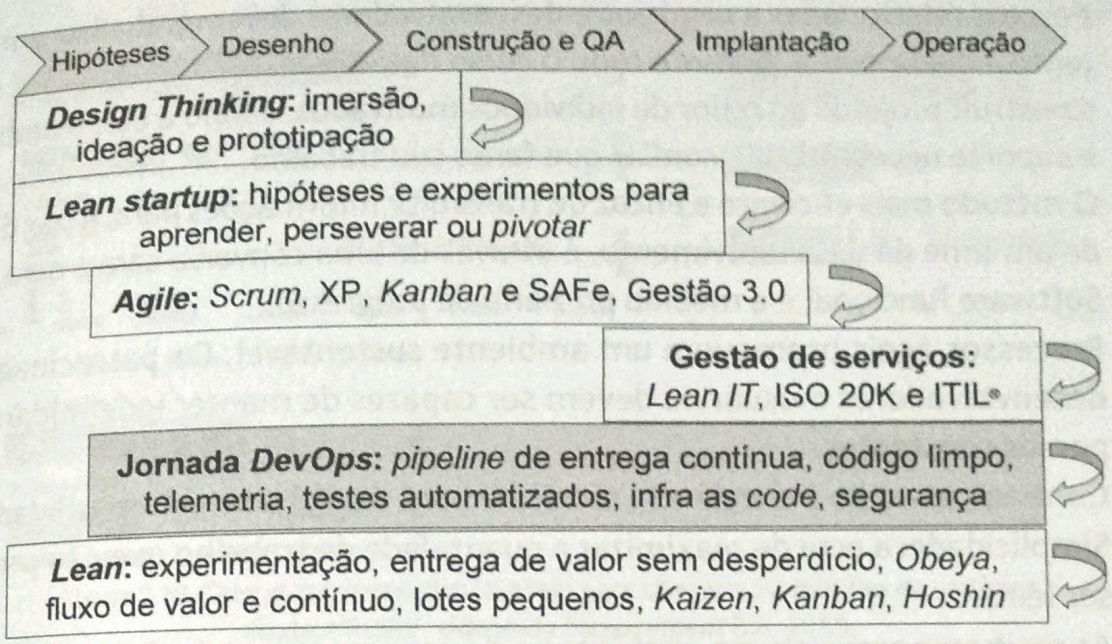

Figura 1.9. Jornada *DevOps* abraça outros movimentos e modelos.
Fonte: elaborado pelo autor.

O Manifesto Ágil foi criado em 2001 por um grupo de especialistas com notável conhecimento em desenvolvimento de software e colabora bastante com a adoção do *DevOps*. Um dos grandes benefícios dos métodos ágeis é a divisão do produto em componentes menores, visando permitir entregas frequentes e adaptação às mudanças cada vez mais velozes que vivemos atualmente. Conforme descrito a seguir, a ideia dessa iniciativa foi "descobrir maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazer o mesmo. Através deste trabalho, passamos a valorizar":

- ✓ Indivíduos e interações mais do que processos e ferramentas.
- ✓ Software em funcionamento mais do que documentação abrangente.
- ✓ Colaboração com o cliente mais do que negociação de contratos.
- ✓ Responder a mudanças mais do que seguir um plano.

Existem 12 princípios que embasam o Manifesto Ágil:

1. A maior prioridade do time é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência para os períodos mais curtos.

4. Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
5. Construir projetos ao redor de indivíduos motivados, dando a eles o ambiente e suporte necessários, e confiar que farão seu trabalho.
6. O método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
7. Software funcional é a medida primária de progresso.
8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter indefinidamente passos constantes.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
11. As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.
12. Em intervalos regulares, o time reflete sobre como ficar mais efetivo, então se ajusta e otimiza seu comportamento de acordo.

1.3. Como o *DevOps* potencializa a transformação digital

Klaus Schwab considera, no livro "A Quarta Revolução Industrial", que estamos vivenciando uma revolução diferente de tudo o que a humanidade já experimentou, pois não há precedente para as mudanças que estão ocorrendo em relação à velocidade, à amplitude e à profundidade. Diferentemente de outras épocas em que havia relativa estabilidade no mercado, temos atualmente um mundo totalmente volátil, incerto, complexo e ambíguo (VUCA).

Observe na Figura 1.10 um exemplo simples da velocidade exponencial na adoção das novas tecnologias.

Como o fundamento do *DevOps* é acelerar o fluxo para a implantação em produção com qualidade, *feedback* rápido e experimentação contínua, torna-se um requisito importante que as organizações iniciem imediatamente essa jornada, visando não perder mercado para seus concorrentes ou tornar-se irrelevantes.

A seguir estão ilustrados os elementos básicos da transformação digital e sua total dependência com as implantações tecnológicas de forma ágil, confiável, adaptativa e focada em experimentação e aprendizado contínuo (Figura 1.11).

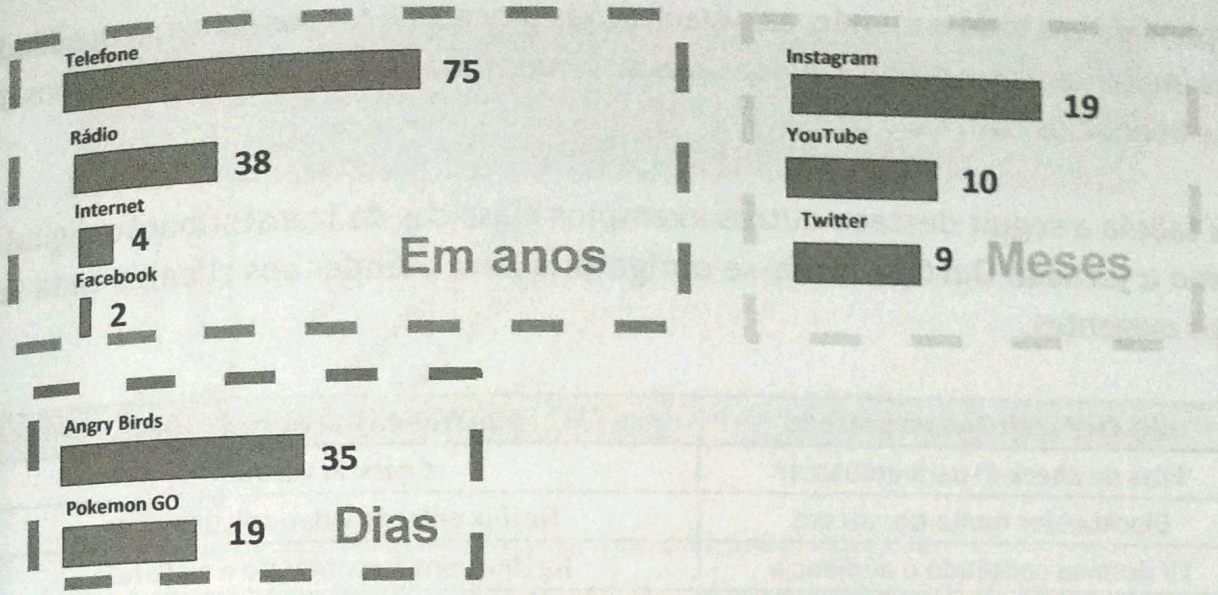

Figura 1.10. Tempo que invenções levaram para chegar a 50 milhões de usuários.
Fonte: MUNIZ, videoaula *Management* 3.0, 2019.

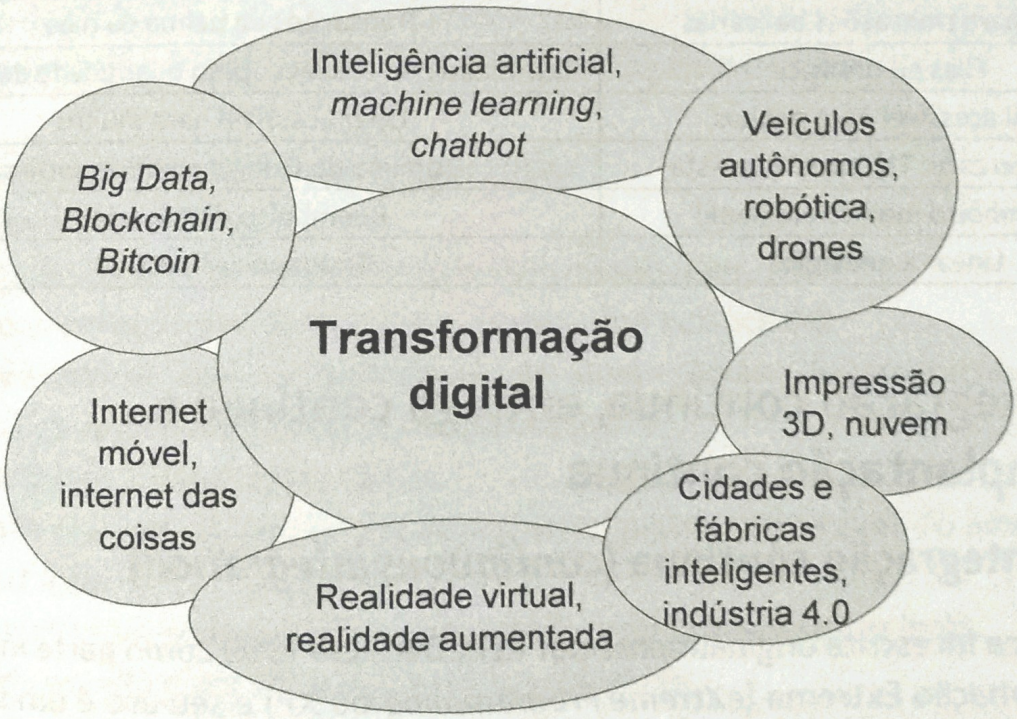

Figura 1.11. Alguns componentes da transformação digital.
Fonte: MUNIZ, palestra no *The Developer's Conference* (TDC) em Porto Alegre, 2018.

Um exemplo da necessidade de *DevOps* para a transformação digital é a criação do supermercado do futuro, chamado de Amazon Go. Diferentemente do modelo clássico, em que precisamos seguir três passos básicos dentro da loja (colocar no carrinho, retirar do carrinho no caixa, colocar no carrinho após passar no caixa), a tecnologia criada elimina os dois últimos passos com uso de aplicativo e câmeras que fazem a leitura das compras realizadas e fecham a conta na saída sem nenhuma atividade manual dos funcionários. Como a tendência é que essa tecnologia seja

expandida em todo o mundo, as organizações precisarão ter confiança na qualidade das implantações e disponibilidade dos sistemas para não gerar mais problemas do que benefícios com essa adesão.

Na tabela a seguir destaco outros exemplos clássicos da transformação digital e como a Jornada *DevOps* torna-se obrigatória para atender aos clientes cada vez mais exigentes.

***Mindset* aceitável no passado**	***Mindset* digital exige *DevOps***
Filas no *check-in* para embarcar	*Check-in* via *app*
Blockbuster multa por atraso	Netflix entrega vídeo sob demanda
TV domina conteúdo e audiência	Netflix domina conteúdo e audiência
Guia Quatro Rodas para viagem	Google compra Waze por US$ 1 bi
Momento Kodak em papel	Facebook compra Instagram por US$ 1 bi
Comunicação por e-mail	Facebook compra WhatsApp por US$ 22 bi
Filas para transações bancárias	Transações na palma da mão
Filas no orelhão	Quantidade de celular supera quantidade de pessoas
Táxi acessível para poucos	Uber acessível para muitos
Anúncio caro: TV, jornal e revista	Domínio do Google e mídias sociais
Reembolso médico em papel	Reembolso digital e ágil
Linear e analógico	Exponencial e digital

1.4. Integração contínua, entrega contínua e implantação contínua

1.4.1. Integração contínua (*continuous integration*)

Essa técnica foi escrita originalmente por Kent Beck em 1999 como parte integrante da Programação Extrema (*eXtreme Programming* ou XP) e seu uso é um requisito fundamental para *DevOps*. Martin Fowler destaca que essa é uma prática de desenvolvimento de software em que cada participante do time integra seu trabalho pelo menos uma vez no dia. Cada integração é verificada por um *build* automatizado (incluindo testes) para detectar erros de imediato e permitir que as atividades de desenvolvimento de software tenham mais qualidade e agilidade.

Enquanto os métodos tradicionais adiam a integração até o final do desenvolvimento, a integração contínua estabelece que o código seja compilado para cada mudança e execute testes automatizados minimamente confiáveis (Figura 1.12).

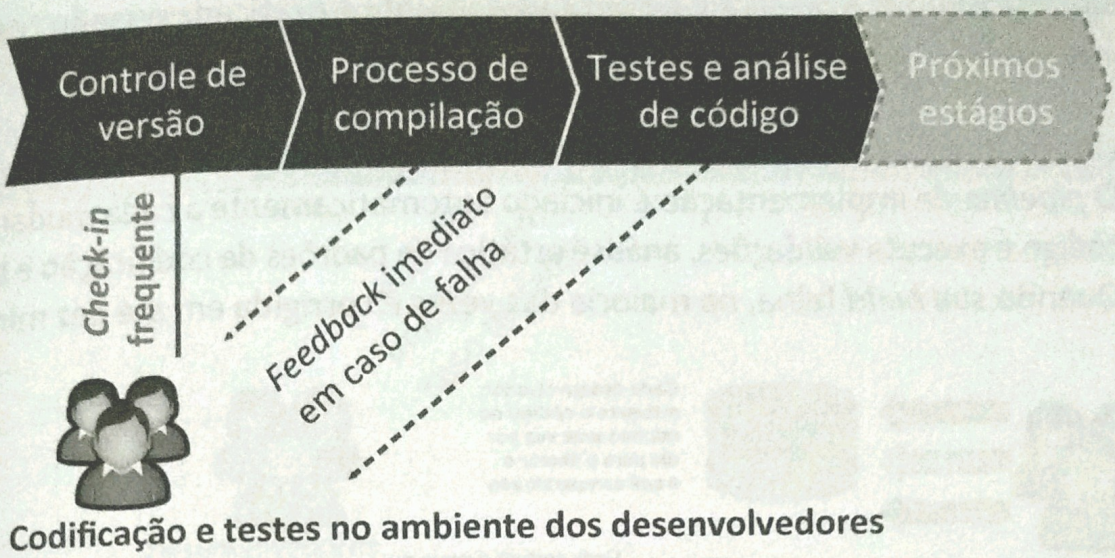

Figura 1.12. Visão resumida da integração contínua.
Fonte: MUNIZ; ADAPTNOW, videoaula oficial Exin, 2018.

Em artigo de 2006, Martin Fowler destaca que a integração contínua pode ser mais efetiva com determinados princípios, conforme resumo a seguir:

1. Manter um repositório de origem único.
2. Automatize o *build*.
3. Faça seu autoteste de construção (testes unitários, *TDD*).
4. Todos devem fazer *commit* pelo menos uma vez por dia.
5. Todo *commit* deve ser centralizado em uma máquina de integração.
6. Corrigir quebra de código imediatamente.
7. Mantenha a construção rápida.
8. Teste em um ambiente que seja o mais próximo possível do ambiente de produção.
9. Garanta que qualquer um possa obter o executável mais recente.
10. Todo mundo pode ver o que está acontecendo.
11. Automatizar a implantação.

Embora os testes automatizados tenham uma grande importância nas iniciativas *DevOps*, é importante que as equipes deixem claro que não há como garantir que nunca ocorrerá erro após sua execução. Um bom exemplo foi a descoberta, em fevereiro de 2019, de uma falha no iPhone que faz o aplicativo travar quando você fala cinco vezes a palavra "hífen" ao usar a funcionalidade de captura de voz. O erro parece inofensivo, será que isso foi uma brincadeira do desenvolvedor? Seria possível prever esse comportamento nos cenários de testes a um custo viável?

Segundo Jez Humble, a integração contínua somente é praticada quando todas as ações a seguir são realizadas (Figura 1.13):

1. O time integra o código pelo menos uma vez por dia em um único *trunk/master*.
2. O *pipeline* de implementação é iniciado automaticamente a cada mudança de código e executa validações, análise estática de padrões de codificação e testes.
3. Quando sua *build* falha, na maioria das vezes é corrigida em até dez minutos.

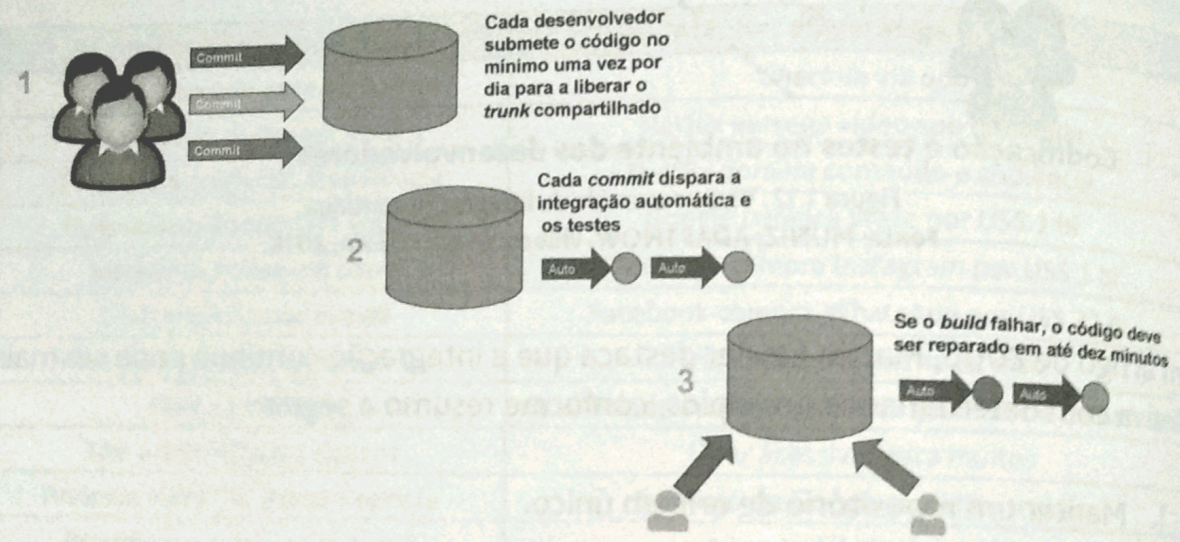

Figura 1.13. Integração contínua e certificação de testes.
Fonte: adaptado de FOWLER, 2017.

1.4.2. Entrega contínua (*continuous delivery*)

Podemos considerar que a entrega contínua é uma evolução natural quando existe o interesse de expandir os benefícios da automação dos testes e *feedback* imediato para os próximos estágios que não são cobertos pela integração contínua.

De acordo com Jez Humble, entrega contínua é a capacidade de disponibilizar mudanças de forma segura e rápida garantindo que o código esteja sempre pronto para implantação, mesmo diante de milhares de desenvolvedores fazendo alterações diariamente.

Observe a seguir que a integração contínua é a base para a entrega contínua e permanece a ideia de *feedback* imediato para as equipes em caso de falha (Figura 1.14).

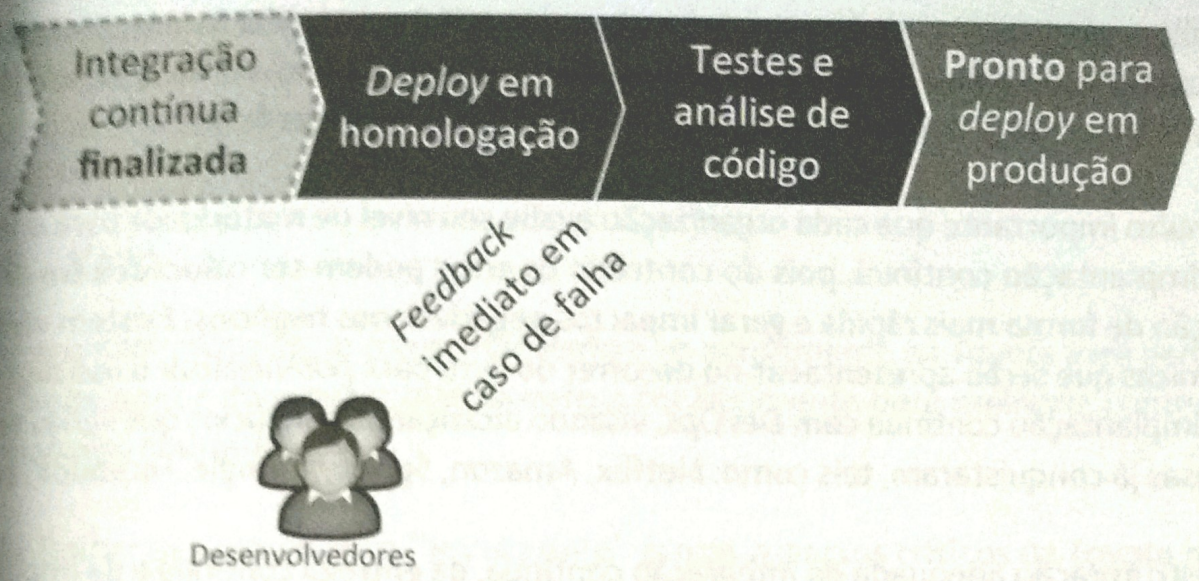

Figura 1.14. Visão resumida da entrega contínua.
Fonte: MUNIZ; ADAPTNOW, videoaula oficial Exin, 2018.

1.4.3. Implantação contínua (*continuous deployment*)

É a evolução natural da entrega contínua e consiste no *deploy* automático em produção após a execução com sucesso dos testes automatizados e das validações previstas, conforme ilustrado a seguir (Figura 1.15).

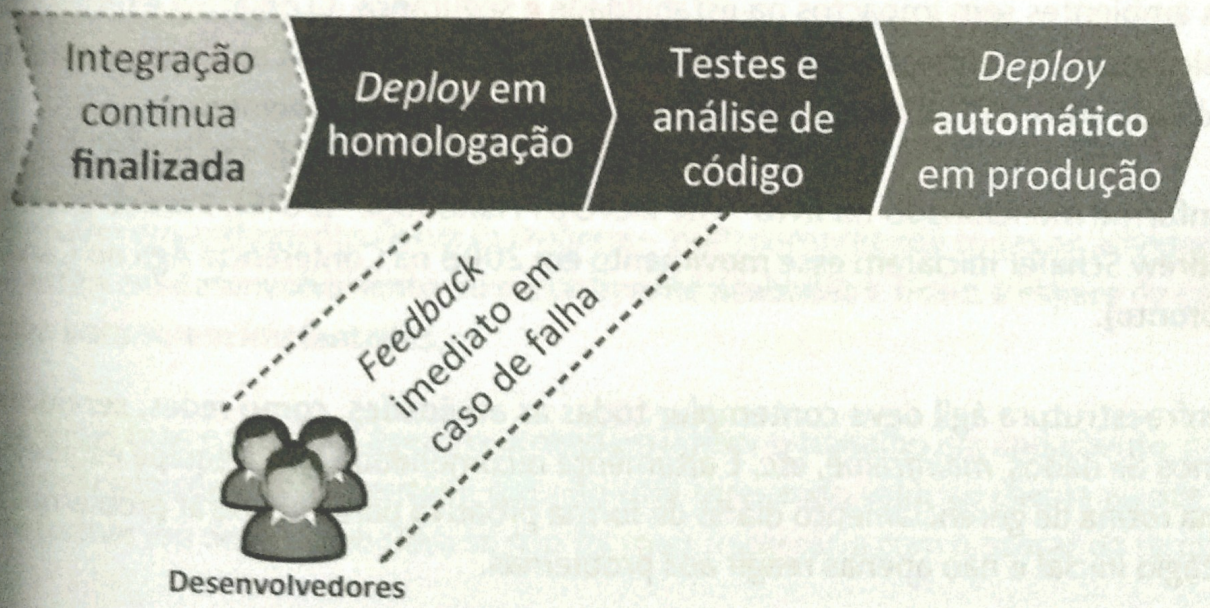

Figura 1.15. Visão resumida da implantação contínua.
Fonte: MUNIZ; ADAPTNOW, videoaula oficial Exin, 2018.

A implantação contínua é usada principalmente para aplicações *web* e aplicativos, mas pode ser experimentada em qualquer tipo de tecnologia. A ideia é disponibilizar pequenas mudanças em produção com a maior frequência possível. É também possível

replicar os ambientes de forma confiável, assim os desenvolvedores podem ensaiar a implantação do código em produção em um ambiente onde a infraestrutura já é muito parecida, e o mais próximo possível, do ambiente de produção.

É muito importante que cada organização avalie seu nível de maturidade para o uso da implantação contínua, pois do contrário os erros podem ser colocados em produção de forma mais rápida e gerar impactos negativos nos negócios. Existem várias técnicas que serão apresentadas no decorrer do livro para potencializar o uso seguro da implantação contínua com *DevOps*, visando alcançar os benefícios que várias empresas já conquistaram, tais como: Netflix, Amazon, Spotify, Google, Facebook, etc.

A estruturação adequada da integração contínua, da entrega contínua e da implantação contínua permite a criação do *pipeline* de implementação, e seu uso eficiente depende diretamente que os ambientes sejam o mais próximo possível do ambiente de produção.

1.5. Infraestrutura ágil e infraestrutura como código

A infraestrutura ágil pode ser considerada uma resposta ao Manifesto Ágil e estabelece um conjunto de automações que permite à área de operações realizar alterações nos ambientes sem impactos na estabilidade e segurança. O objetivo é provisionar rapidamente a infraestrutura, e toda a atividade manual das equipes passa a ser realizada automaticamente com testes para garantia dos ambientes.

Conforme mencionado no livro "The DevOps Handbook" (2016), Patrick Debois e Andrew Schafer iniciaram esse movimento em 2008 na Conferência Ágil no Canadá (Toronto).

A infraestrutura ágil deve contemplar todas as atividades, como redes, servidores, banco de dados, *mainframe*, etc. É altamente recomendado que a equipe estabeleça uma rotina de gerenciamento diário de forma proativa para encontrar problemas no estágio inicial e não apenas reagir aos problemas.

A utilização de um repositório único e ferramentas comuns potencializa o termo radiadores de informação criado por Alistair Cockburn (um dos autores do Manifesto Ágil). Os radiadores de informação são elementos visuais que tornam o ambiente informativo o suficiente para que todos tenham a visibilidade clara e objetiva da situação dos itens que fazem parte do fluxo de valor.

A evolução da infraestrutura ágil pode ser denominada infraestrutura como código (IaC – *Infrastructure as a Code*). Existem diversos softwares que podem automatizar a criação de ambientes, como Chef, Puppet, Ansible, etc.

1.6. *Kata*

O termo *Kata* refere-se ao comportamento e ao pensamento na Toyota para gerenciar as pessoas e disseminar diariamente o conhecimento para melhoria contínua do trabalho realizado.

Mike Rother destaca no livro "Toyota Kata" que os aspectos críticos da Toyota não são visíveis e aborda em detalhes as duas questões a seguir:

1. Quais são as rotinas de gestão e pensamento ocultos que estão por trás do sucesso da Toyota com a melhoria contínua e a adaptação?
2. Como outras empresas podem desenvolver rotinas e pensamentos similares em suas organizações?

Considerando que *DevOps* envolve a integração e a sinergia de atividades, torna-se importante refletir sobre como o *Kata* pode ser aplicado para disseminar o conhecimento tácito entre as equipes.

1.7. Trabalho em andamento (WIP)

O trabalho em andamento (*Work In Progress* – WIP) compreende todas as atividades que estão em desenvolvimento ou parcialmente acabadas e ficam à espera de conclusão para serem implantadas.

As práticas *Lean* e métodos ágeis recomendam limitar o trabalho em andamento, pois é considerado como um inventário que não está agregando valor ao cliente e pode se transformar em desperdício puro se não for mais necessário com o passar do tempo.

Desta forma, o sistema empurrado passa para um sistema puxado, onde não será mais possível iniciar alguma atividade antes de finalizar outra que esteja em andamento.

Um exemplo simples de como a limitação do WIP pode evitar o desperdício é que podemos trocar 10 atividades em andamento com 50% de conclusão, mas nenhuma efetivamente concluída, por 5 atividades completas e 5 atividades a iniciar.

Observe no exemplo da figura a seguir que na perspectiva do cliente o que vale é a entrega final e de nada adianta a ocupação total das equipes em muitas atividades. Dessa forma, é mais eficaz o foco de todos para entregar com qualidade e de forma incremental o famoso lote pequeno preconizado pelo *Lean*. Essa prática colabora também para evitar a disseminação da clássica e irritante frase "está pronto, falta apenas testar" (Figura 1.16).

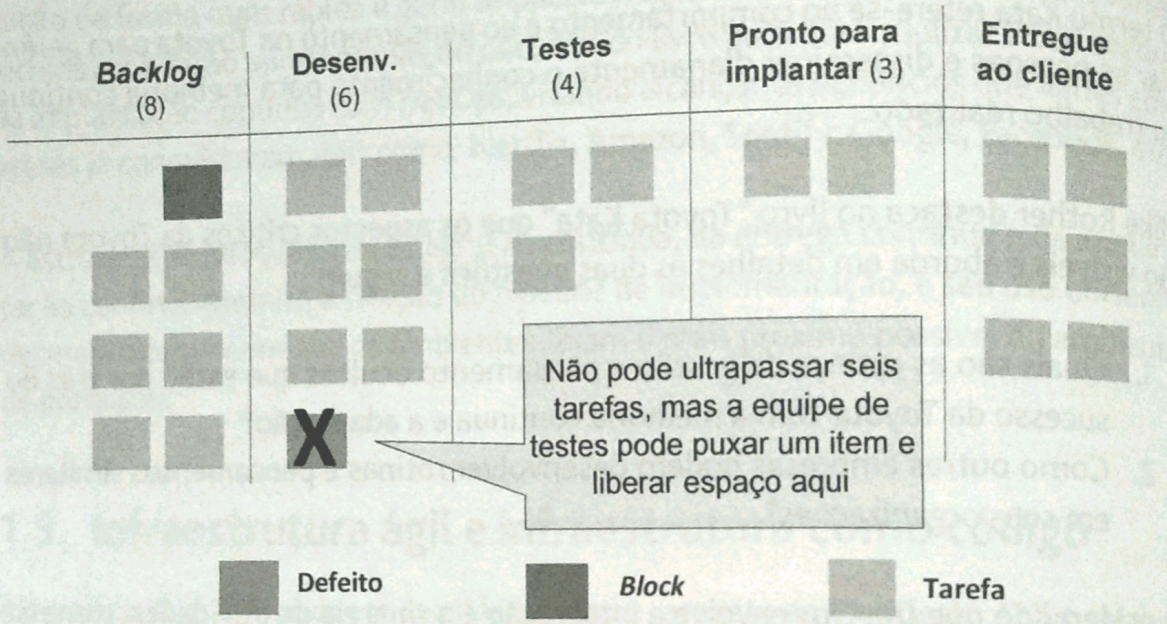

Figura 1.16. Visão resumida da implantação contínua.
Fonte: MUNIZ; ADAPTNOW, videoaula oficial Exin, 2018.

1.8. Débito técnico

Também conhecido como Dívida Técnica, este conceito envolve a degradação dos sistemas ao longo do tempo quando não existem ações para garantir a qualidade do código. Um débito técnico pode ser um código mal estruturado, uma vulnerabilidade de segurança, um ponto no código não performático ou qualquer outra falha que muitas vezes não impacta em um defeito no software, mas evidencia uma certa ausência de qualidade.

Quando não resolvemos essas situações, criam-se problemas cada vez mais difíceis de corrigir com o passar do tempo, pois os objetivos conflitantes das áreas *Dev* e *Ops* contribuem para o aumento do débito técnico.

DevOps preconiza a realização de *blitz* de melhoria, e essa ação consiste em agrupar frequentemente equipes multidisciplinares para resolver a dívida técnica. Um débito técnico que não seja corrigido depois de um determinado tempo pode se tornar um

problema grande e ocasionar quebra da aplicação e até mesmo instabilidades quando a carga de dados trafegada exceder um determinado limite.

1.9. Tempo de espera (*lead time*)

Apresentarei os conceitos complementares do *Lean* nesta seção para facilitar o entendimento do fluxo de valor.

- ✓ ***Lead time* (tempo de execução, ciclo ou espera):** significa o tempo que medimos do início da criação de uma solicitação até sua entrega, ou seja, é o tempo gasto para transformar os recursos usados nos resultados entregues para o cliente. Taichi Ono, considerado o grande responsável pela criação do Sistema Toyota de Produção, que é a base do *Lean*, resume bem a importância de medir o *lead time*: "tudo o que estamos tentando fazer é reduzir a linha do tempo".
- ✓ **Tempo de processamento:** representa o tempo realmente usado para executar a etapa do processo e inicia somente quando começa a trabalhar até sua conclusão, ou seja, omite o tempo em que o trabalho fica na fila.
- ✓ **% conclusão e precisão (completo e correto):** indica o percentual que uma atividade foi concluída com sucesso na perspectiva do cliente sobre a qualidade entregue.

1.10. *DevOps* depende de código limpo

Embora as práticas *DevOps* descritas neste primeiro capítulo ajudem com implantações contínuas, um código mal escrito poderá desencadear uma série de problemas que não deixam o software de pé, seja no curto ou longo prazo.

Quando não existe uma preocupação genuína com a qualidade na origem, haverá grande desperdício para manter o código funcionando. Na pior das hipóteses, a casa pode cair e gerar um impacto negativo exponencial.

Uma das obras mais brilhantes sobre software com qualidade foi organizada pelo experiente Robert C. Martin com o título "Código Limpo: habilidades práticas do agile software" (2011). Caso você tenha interesse em ser um embaixador do movimento *DevOps*, recomendo fortemente que incentive todos ao seu redor a conhecer e usar essas práticas, mesmo que não trabalhem diretamente com codificação. A figura a seguir apresenta um breve resumo dessas técnicas (Figura 1.17).

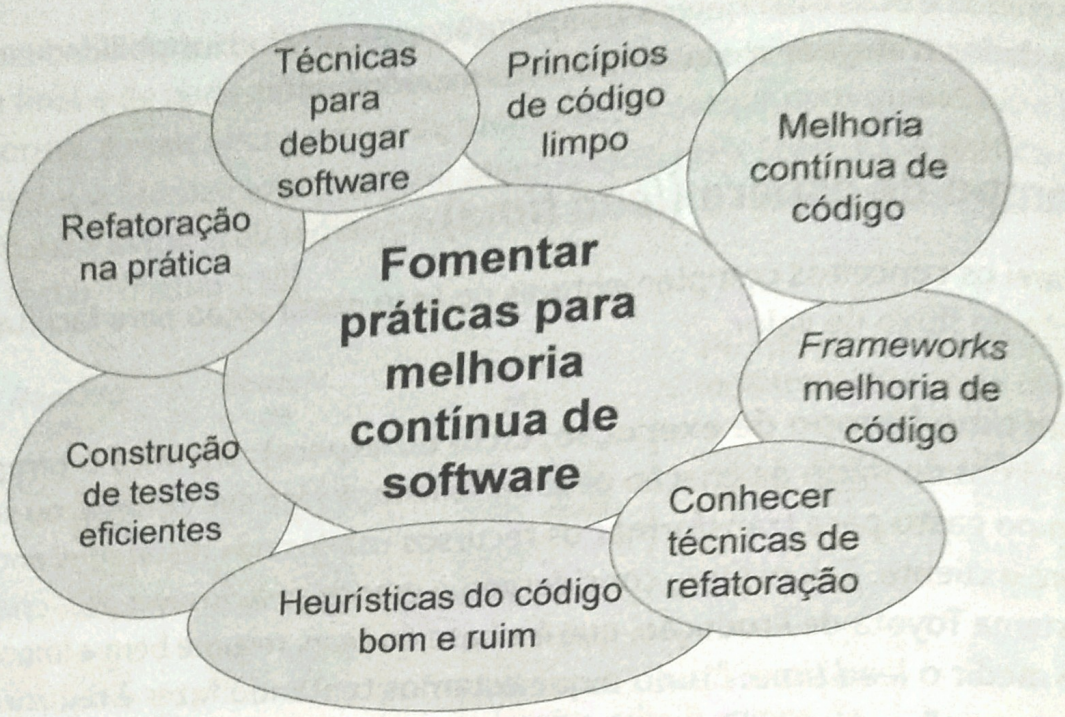

Figura 1.17. Práticas do código limpo e refatoração.
Fonte: adaptado de MARTIN, 2011.

Quando falamos em transformação ágil, é frequente que o foco seja dado na conscientização das equipes sobre o novo *mindset*, dando pouca atenção à capacitação técnica dos times para desenvolver software com qualidade e arquitetura adequada. Também é comum não colocar no jogo as equipes de operação (infraestrutura, segurança da informação, gestão de mudanças, auditoria, QA, etc.).

O movimento *DevOps* chegou para complementar essa lacuna, considerando o envolvimento de todas as equipes multidisciplinares do fluxo de valor.

Destaco a seguir algumas reflexões que serão discutidas com mais detalhes ao longo da nossa Jornada *DevOps*:

1. Uma empresa implantou o *pipeline* de integração contínua e agora é possível colocar código em produção todo dia. Porém, o fluxo de valor completo não foi mapeado e não há testes automatizados. Qual é o impacto na qualidade das entregas e quais conflitos vão surgir entre as áreas quando a quantidade de incidentes em produção aumentar?
2. Seu chefe chega aos gritos dizendo que seu código tem que ser entregue em duas semanas, mesmo que seja necessário cortar todos os testes que estavam previstos. Qual é a chance de o software quebrar em produção? Como essa cultura do medo impacta toda a equipe? Será que seu chefe também é tratado

dessa forma pelo cliente ou área de negócio e desconta na equipe porque não tem habilidade emocional para negociar ou suportar essa pressão?

3. A equipe de operações está super empolgada porque foi aprovado o investimento para uso de ferramentas de infraestrutura como código. As equipes de Operações estão preparadas para trabalhar com programação? Os conflitos de objetivo com a área de desenvolvimento serão solucionados com essa ação?
4. A gestão de mudanças ouviu falar em *DevOps* e está muito preocupada com os boatos sobre a extinção de evidências e do comitê de mudanças, que é um dos pontos cobrados pela auditoria. Como o *DevOps* pode colaborar com essa situação?
5. A empresa em que você trabalha é fortemente auditada por órgãos fiscalizadores para mitigar riscos de segurança e todo o seu processo ocorre no final do ciclo de desenvolvimento. Como fica a segurança das implantações frequentes praticadas com *DevOps*?

1.11. *DevOps* e terceirização (CALMSS)

Uma outra evolução do termo CALMS inclui mais um S (*Sourcing*) para destacar a questão de terceirização. Segundo Eveline Oehrlich, a Forrester criou essa nova definição para representar o entendimento de outras pessoas de que as iniciativas *DevOps* precisam ser suportadas por uma sólida estratégia de terceirização.

O engajamento de equipes terceirizadas é bastante valorizado por Davis e Daniel no livro "Effective DevOps" (2016). Ele apresenta uma visão humanizada de como as empresas acabam discriminando as pessoas que não são funcionários formalizados, mas desempenham tarefas tão importantes quanto os demais.

Considero ser muito importante essa reflexão, que costuma ser deixada de lado nas organizações. Em geral, a liderança e as equipes internas não percebem como pequenas ações podem minar o sentimento de pertencimento do time como um organismo integrado e gerar conflitos desnecessários (ex.: terceirizados trabalham o ano inteiro ao lado de todos, mas não podem participar da festa de final do ano, eventos, reuniões, comemorações, etc.).

Em uma das palestras a que assisti no *DevOps Days* do Rio de Janeiro em 2018, a Analia Irigoyen, minha grande amiga e coautora deste livro, trouxe de forma brilhante esse debate para a comunidade.

O tema da sua palestra "DevOps + Fornecedores = É possível" apresentou dificuldades, fatores de sucesso, lições aprendidas e técnicas gerenciais já vivenciadas na terceirização de desenvolvimento e/ou de testes (automatizados ou não).

Ao contratar novos fornecedores foi discutido que, inicialmente, seja planejada uma atividade para identificar, analisar e mitigar os riscos relacionados a:

- ✓ infraestrutura;
- ✓ conhecimento dos profissionais da contratada relacionado ao uso do processo, suas técnicas e ferramentas;
- ✓ conhecimento do negócio; e
- ✓ comunicação (interna e externa).

Outras lições aprendidas expostas durante a palestra a respeito de contratações de fornecedores e uso do *DevOps* foram:

- ✓ Uma arquitetura de referência flexível e disponível para o fornecedor.
- ✓ A decisão pelas ferramentas que serão utilizadas na automação não é nada fácil, são muitas opções. Nesse sentido, o melhor quando se vai terceirizar é usar as ferramentas mais utilizadas no mercado, aumentando a concorrência e diminuindo o risco de rotatividade da equipe no fornecedor.
- ✓ Decisões arquiteturais que envolvam mudanças na arquitetura de referência devem ser feitas em conjunto (cliente x fornecedor).
- ✓ Realização de pilotos com o objetivo principal de aprendizado do processo e da arquitetura pelo fornecedor e cliente. Nesse momento é importante gerar empatia, objetivar sempre a colaboração e tomar decisões sempre considerando a otimização do fluxo de valor.
- ✓ Desenvolver padrões de interface (telas), de histórias e de modelos de impressão, facilitando a integração de times externos e internos.
- ✓ O fornecedor deve preparar os *mocks* para integração entre sistemas internos e externos.
- ✓ O fornecedor precisa conhecer as recomendações de resolução e *browsers* utilizados de quem está contratando.
- ✓ Definição de massa de testes de forma colaborativa.
- ✓ Definição do processo de entrega do fornecedor, não esquecendo de:
  - Definição da configuração do ambiente com apoio de ferramenta.
  - Frequência de entrega – sempre que possível trabalhar com a mesma frequência dos times internos, levando em consideração a complexidade do processo de entrega (*releases* com alta complexidade que precisam de testes manuais).

- Estratégia de *branches* e acesso a ferramenta de gerência de configuração de forma remota.
- Revisão de código frequente.
- Critérios de "pronto" mais rígidos e claros possível, alinhados aos times internos o máximo possível, respeitando as restrições organizacionais.
- Configuração do sonar padrão (com critérios bem definidos de qualidade, incluindo a cobertura de testes).

✓ Definição da política de segurança de acesso a dados (política de sigilo, *logs*, criptografia, dados sensíveis, segurança nos *desktops* do fornecedor, acesso VPN, HTTP, acordo de confidencialidade com todos os funcionários do fornecedor, segurança quanto a ataques externos e tentativas de invasão, antivírus). O fornecedor precisa conhecer e praticar todas as regras organizacionais de segurança da contratante.

✓ Estabelecer a infra necessária para que o fornecedor:
- Tenha acesso remoto para trabalhar nas ferramentas; o custo de incluir as informações em ferramenta é muito alto e atrapalha a otimização do fluxo.
- Preocupe-se em obter licença de software para os fornecedores, quando aplicável.
- Proveja a infraestrutura necessária para que o fornecedor trabalhe dentro das instalações do contratante quando existir algum *deploy* em produção (salas de guerra).

✓ Estabelecer um canal único de comunicação utilizando ferramenta colaborativa, com o objetivo de compartilhamento de conhecimento e integração entre os times internos e externos.

✓ Participação do fornecedor e do contratante nas cerimônias ágeis como um só time exercendo os papéis necessários para diminuir as barreiras e os ruídos de comunicação, incluindo principalmente o refinamento do *backlog*, garantindo que todos estejam trabalhando com os mesmos objetivos e tenham o mesmo conhecimento do negócio e dos critérios de pronto.

✓ Estabelecer uma gestão visível e transparente com a utilização de métricas com o objetivo principal de melhoria contínua (sem julgamento de valor e procurar culpados): *System Cycle Time*, % de cobertura de testes; tempo de *refactoring*; % de cobertura de testes; taxa de defeitos; e % incidentes na entrega.

✓ Estabelecer uma comunicação colaborativa com o fornecedor: responsabilidades claras, espírito colaborativo (todos são responsáveis pela entrega, inclusive o fornecedor, que deve estar disponível também nas implantações e nos incidentes em produção), contratos ganha-ganha – considerando a confiança nas equipes e em métricas justas.

✓ Usar o "Delegation Poker" e o alinhamento das restrições, presentes no livro "Management 3.0 – Leading Agile Developers, Developing Agile Leaders"

(2011), para evitar as cercas invisíveis e apoiar a definição dos assuntos nos quais a equipe do fornecedor pode tomar decisões ou ainda não está madura o suficiente para decidir. Exigir que o fornecedor desenvolva as competências do seu time para otimizar cada vez mais o fluxo de valor.

✓ Trabalhar por ordem de serviço dividindo o trabalho em pequenos pacotes (*sprints*), alinhando principalmente as restrições e o contexto organizacional. Usar tipos de contratos ágeis sempre que possível: Preço Fixo Ágil e T&M Ágil, *Money for Nothing Change for Free*, Preço fixo por unidade de Trabalho, Progressivo, Objetivo de Custo. A palestra do Rafael Rodrigues (2015) detalha um pouco mais esses tipos de contratos e o artigo de Eduardo Peres (CAROLI. ORG, s.d.) também é uma boa referência sobre o assunto.