Aprenda com quem faz

A Terceira Maneira: Aprendizagem e Experimentação

Clara Érica T. Castro

2023

SUMÁRIO

Capítulo 1. Introdução

Introdução

A terceira maneira do *DevOps* é uma das principais para se atingir os objetivos de disseminar a Cultura *DevOps* nos times, criando alta confiança para assumir riscos e possibilitar o aprendizado através de erros. A experimentação é um dos passos primordiais para o crescimento das organizações, principalmente as que pretendem alcançar alto nível de inovação e entrega de valor aos seus clientes de forma exponencial.

Como indicado no Livro Organizações Exponenciais, apesar da experimentação ser um atributo importantíssimo para a aprendizagem nas organizações, é também um dos mais difíceis a ser alcançado, pois, na maioria das vezes, elas estão tão ocupadas na execução das tarefas que não conseguem focar na inovação (ISMAIL. MALONE. GEEST, 2015).

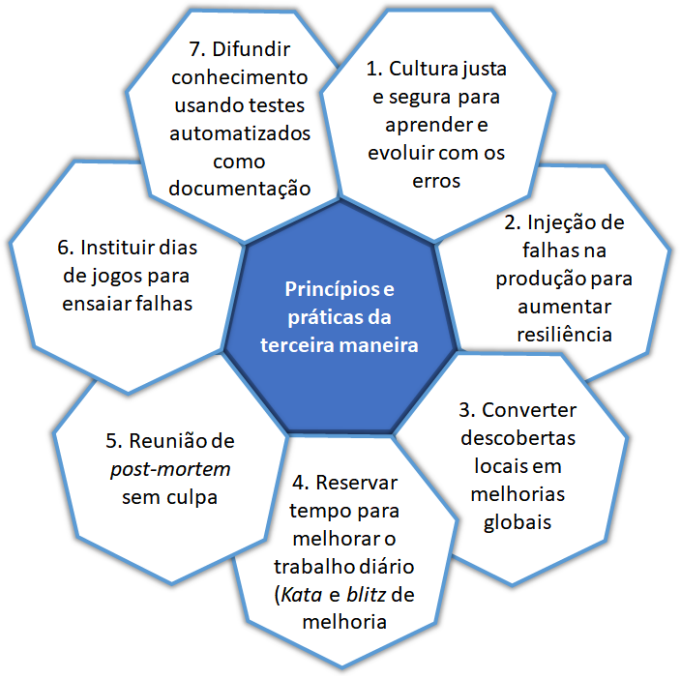
É também muito difícil que uma organização esteja realmente pronta para encarar os fracassos advindos da experimentação, por isso ainda há muito receio das pessoas exporem suas ideias, arriscarem-se a sugerir mudanças ou expor problemas, justamente pelo medo de serem apontadas e lembradas somente pelos fracassos e não pelos acertos durante o processo de aprendizado.

Este tipo de ambiente não é aberto e resulta em falta de confiança e medo, impactando todos os trabalhadores, pois é perpetuada uma cultura de punição, impedindo a criatividade.

O grande desafio recai sobre como motivar e incentivar os times na cultura de automação, colaboração e melhoria contínua, fortalecendo assim os laços de confiança e busca da perfeição e criando equipes de alta performance.

A seguir os princípios e práticas da terceira maneira (MUNIZ et al., 2020 apud KIM et al., 2021):

Figura 1 – Princípios e práticas.



Fonte: Elaborado pela autora.

Motivação

O neurocientista norte-americano Paul Maclean trouxe, por volta dos anos 1960, a teoria do cérebro Trino, que se baseia na divisão do cérebro em três partes:

- Reptiliano: se encontra na base de nosso cérebro e se forma ainda no útero, considerada, assim, a mais primitiva. O termo caiu em desuso de uns tempos para cá, pois é remota a similaridade do ser humano com répteis e alguns neurocientistas se referem a ele simplesmente como o basal. O importante aqui é entender que ele é o instintivo, o que nos faz reagir prontamente ao perigo, como reflexos instintivos de sobrevivência (fuga, velocidade ou força além do normal, ações sem pensar).

- Límbico: é o cérebro emocional, localizado no centro do cérebro, e que se desenvolve principalmente logo após o

nascimento. É o nosso Sistema Nervoso Central, que se forma com base em nossas experiências, ambiente, temperamento e até genética. Primordialmente, nossas emoções.

- Neocórtex: é o cérebro racional, localizado na parte frontal de nosso cérebro. É o que nos leva a pensar racionalmente e objetivamente sobre uma situação para tomada de decisão. Em uma única palavra, nossa razão.

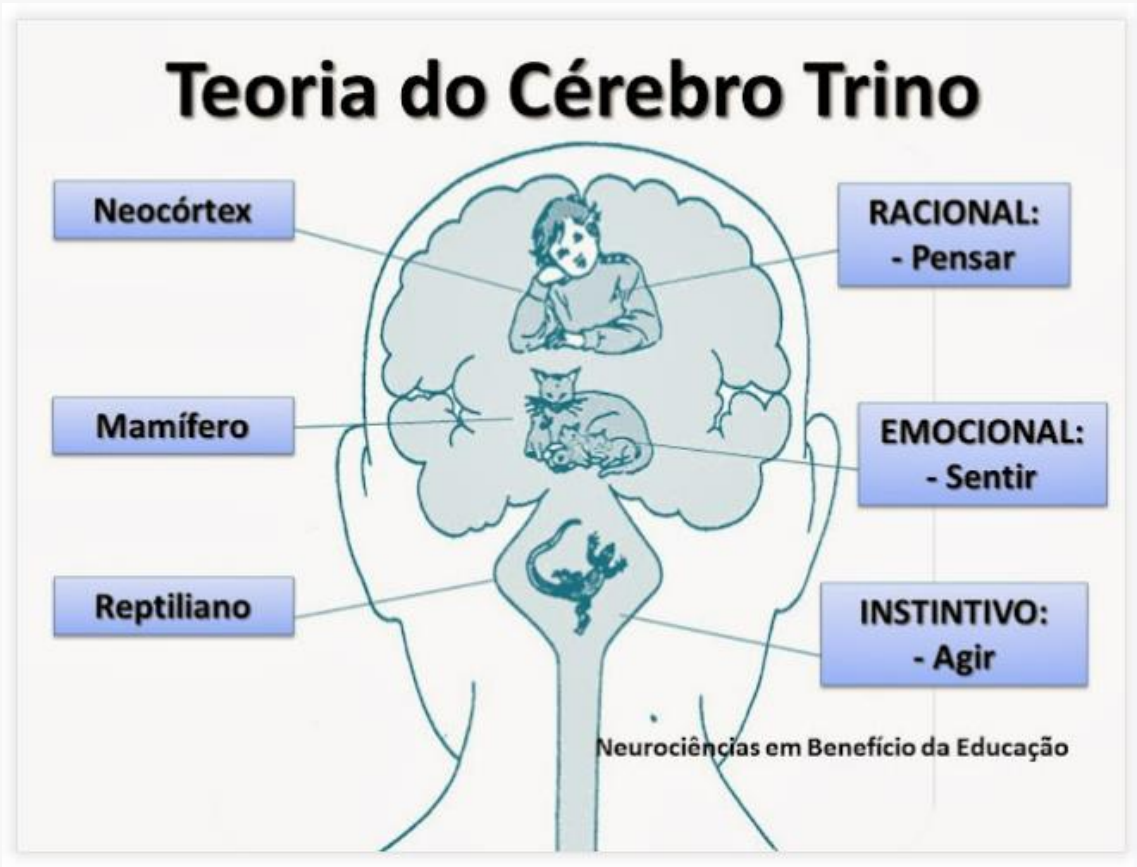
Esta teoria é utilizada em várias áreas para explicar como nos comportamos em determinadas situações. E os estudos que se seguiram apontam que o cérebro reptiliano é o que possui maior influência sobre as demais áreas, pois é algo natural do ser humano.

Um exemplo disto é que a nossa comunicação corporal revela muito mais sobre nós do que a linguagem verbal ou escrita. Percebemos isso quando ocorre um descasamento entre a fala e a expressão corporal de uma pessoa em determinada situação.

Estudos com base nessa teoria demonstram que somos avessos às mudanças e instintivamente agiremos (sem pensar) se formos submetidos a qualquer tipo de pressão, apenas para nos defender. Isso também explica por que o aprendizado gera grande desconforto no começo, pois nos tira dessa zona de conforto que gostamos tanto.

Sem perceber, nosso cérebro lutará para que não haja grandes mudanças de hábitos. E esse é o desafio do aprendizado! Treinar seu cérebro para que pequenos avanços tragam uma porção de felicidade, que o faça parar para pensar, agir e então melhorar constantemente. O autoconhecimento é importante para que possa estar em constante aprendizagem, sem medos.

Figura 2 – Representação do cérebro Trino.



Fonte: <https://neuropsicopedagogianasaladeaula.blogspot.com/2013/12/teoria-do-cerebro-trino.html>.

Leitura adicional em:

- https://www.ibnd.com.br/blog/a-teoria-do-cerebro-trino-o-cerebro-sob-uma-nova-perspectiva.html
- https://www.interaction-design.org/literature/article/the-concept-of-the-triune-brain
- https://neuropsicopedagogianasaladeaula.blogspot.com/2013/12/teoria-do-cerebro-trino.html

Importância

Para que tenhamos equipes motivadas e com alta performance, é necessário que haja um ambiente seguro, só assim os integrantes dos times

se sentirão comprometidos com a aprendizagem contínua, dado que haverá sempre algo novo ou inesperado para se enfrentar e resolver.

Em sistemas complexos, haverá sempre o fator incerteza e interdependência, o que poderá acarretar resultados indesejados. E procurar culpados para os erros não é a solução, pois isto gera um ambiente de medo e insegurança, onde todos possuem medo de revelar os acontecimentos, o que torna uma investigação de causa-raiz praticamente impossível. Um primeiro passo para impedir que isto aconteça é ter um ambiente de segurança psicológica, o que não ativará o nosso lado 'reptiliano' de defesa.

Por isso, as práticas da Terceira Maneira são tão importantes para que se possa manter o círculo virtuoso de aprendizado, com cultura justa, baseada em fatos e dados para tomada de decisão.

Figura 3 – Aprendizado x Segurança Psicológica.

Fonte: <https://www.youtube.com/watch?v=LhoLuui9gX8 Building a psychologically safe workplace | Amy Edmondson | TEDxHGSE>.

Segundo Muniz e Irigoyen, no Livro Jornada Ágil e Digital (2019), os indivíduos podem estar entre as 4 zonas:

- Zona de Conforto: o indivíduo está com alto nível de Segurança Psicológica, porém, com baixa motivação e responsabilidade, tornando-o um trabalhador raso.

- Zona de Apatia: o indivíduo está inserido num ambiente de baixo nível de Segurança Psicológica, está com baixa motivação e responsabilidade, o que o torna abatido, triste, sem ânimo para realizar suas entregas com qualidade e muito menos inovar.

- Zona de Ansiedade: o indivíduo possui alta motivação e se sente responsável pelo que faz, quer inovar e implantar coisas novas, porém, por estar inserido em um ambiente inseguro e negativo, o medo de errar o deixa tenso, ansioso e sem energia para entregar o que realmente deseja.

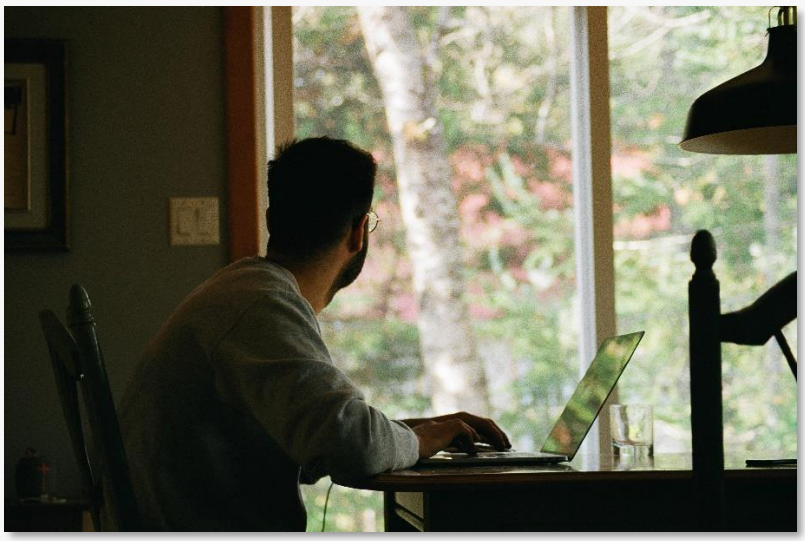
- Zona de Aprendizado: o indivíduo está no melhor cenário possível, pois se sente confiante, seguro, feliz e sabe que sua capacidade de inovar não será criticada, humilhada ou prejudicada no ambiente de trabalho caso aconteça algum problema, ou seja, as possíveis falhas são consideradas como fases do processo de aprendizagem. Desta maneira, é capaz de evoluir sempre e trazer o melhor de si para o time e sua organização.

Ao iniciar as práticas da terceira maneira, recomenda-se cuidado e análise prévia para se assegurar de que a organização possua uma cultura de aprendizado contínuo, que propicie segurança psicológica.

Abrindo um parêntese sobre a Zona de Conforto: existe um termo que se popularizou pelos trabalhadores da geração X através do aplicativo Tik Tok, que preconiza fazer apenas o necessário no trabalho para não ser demitido e aproveitar a vida, o chamado *Quiet Quitting* (MASTERSON,

2022). Ao contrário do que parece, não se trata de demissão voluntária. Apesar do foco desse termo buscar equilíbrio entre vida profissional e pessoal, fazer o mínimo para garantir o emprego pode levar o trabalhador a não buscar novas formas de realizar seu trabalho, resultando em estagnação. As empresas se empenham para sobreviver em meio à competição acirrada de seus concorrentes e desejam que seus produtos ou serviços sejam os preferidos de seus clientes. Poderá ocorrer demissão do profissional como resultado de entregas mínimas.

Figura 4 – What is quiet quitting?



Fonte: <https://www.weforum.org/agenda/2022/09/tiktok-quiet-quitting-explained/?DAG=3&gclid=Cj0KCQiAt66eBhCnARIsAKf3ZNFLT0AO9T4wb1b97viezi5T6Gec8kRHO_H9GJ0pyojw5enhiYMx-OYaAg6JEALw_wcB>.

Na maioria das empresas, a promoção de um profissional ocorre quando ele já desempenha boa parte das funções exigidas no cargo acima do atual, então entregar somente o básico não trará oportunidades de crescimento. Vale a reflexão!

Mudança de Mindset

Além do ambiente em que os indivíduos estão inseridos, é importante ressaltar que cada um possui uma maneira de encarar os fatos e reagir aos acontecimentos subsequentes.

A psicóloga Carol Dweck (2016) desenvolveu um conceito sobre os tipos de mentalidades, o que explica o motivo de pessoas muito inteligentes não conseguirem vencer na vida ou atingir os resultados esperados e pessoas que não eram consideradas tão inteligentes alcançarem muito sucesso.

São eles:

- *Mindset* Fixo: são pessoas que acreditam que tudo é resultado de algo nato, inerente à pessoa desde seu nascimento e que não é possível alterar. Este pensamento limita suas ações e se escondem ao simples fato de que 'não nasceram para isso' ou 'que nunca aprenderão algo novo'. Elas não acreditam que podem evoluir e seus esforços são em vão, não enfrentam problemas acreditando que podem realmente resolvê-los.

- *Mindset* de Crescimento: próprio de pessoas que atingiram resultados surpreendentes, por focarem mais no processo do que no resultado e considerando as falhas ou fracassos como forma do processo de aprendizado. Assim, elas estão constantemente pensando em como fazer melhor da próxima vez, sem se fixar somente nos erros e se penalizando por isso. É isto que melhora o desempenho de pessoas com este pensamento, elas sabem que suas habilidades e conhecimentos podem sempre melhorar.

O simples fato de tomar consciência de que existem 2 tipos de *mindsets* (mentalidades) aumenta seu autoconhecimento e permite que possa se preparar para melhorar seu desempenho.

Figura 5 – Mentalidade Fixa e de Crescimento.

Fonte: <https://www.aprimoresuamente.com/carol-kim-mindset/>.

Para testar qual seu *mindset*, veja alguns sites gratuitos:

- https://www.oficinadepsicologia.com/test/mindset-de-crescimento/
- https://www.davidsilva.com.br/teste-seu-mindset-cp/

É por este motivo que as organizações exponenciais necessitam de líderes que tenham o tipo de *mindset* de crescimento: para garantir que suas equipes possuam um ambiente de segurança psicológica que permita o aprendizado contínuo e abertura para inovação. É necessário também ter os perfis adequados para os profissionais que desejam ser Engenheiros *DevOps*, SRE ou Engenheiro de Plataforma (este último está em alta na comunidade e a tendência é que o termo substitua as nominações anteriores, haja vista a busca por profissionais com amplo conhecimento).

Leitura adicional em:

- Qual é o seu MINDSET? - YouTube

- Change your mindset, change the game | Dr. Alia Crum | TEDxTraverseCity - YouTube
- https://psidosucesso.com.br/mindset-e-objetivos/
- https://www.aprimoresuamente.com/carol-dweck-mindset/

Perfis Profissionais

A terceira maneira do *DevOps* impulsiona as organizações a terem ambientes seguros e institui a cultura justa, onde o aprendizado é muito valorizado. Isto também propicia um ambiente saudável que favorece o surgimento de profissionais *T-Shaped*.

Figura 6 – Funcionários Especialistas vs. Generalistas vs. "E-Shaped".

"*I-shaped*" (Especialistas)	"*T-shaped*" (Generalistas)	"*E-shaped*"
Profunda competência em uma área	Profunda competência em uma área	Profunda competência em algumas áreas
Poucas habilidades ou experiências em outras áreas	Amplas habilidades em muitas áreas	Experiência em muitas áreas Habilidades de execução comprovadas Sempre inovando
Cria gargalo rapidamente	Pode intensificar a remoção de gargalos	Potencial quase ilimitado
Insensível aos resíduos e impactos do *downstream*	Sensível aos resíduos e impactos do *downstream*	–
Impede a flexibilidade de planejamento ou a absorção de variabilidade	Ajuda a tornar o planejamento flexível e absorve a variabilidade	–

Fonte adaptada de: Tabela 7.1 Funcionários Especialistas vs. Generalistas vs. "E-SHAPED" (experiência, competência, exploração e execução): *The DevOps Handbook* (2ª. Ed. 2021, p. 194) KIM et al.

Segundo KIM et al. (2021), é necessário encorajar que cada membro da equipe se torne um generalista (*T-shaped*), pois a velocidade de tecnologias emergentes é tanta, que não é sustentável ter pessoas que só possam contribuir em apenas uma única área de conhecimento. E quando temos departamentos superespecializados, estimulamos a existência de silos, pois somente essas equipes possuem o conhecimento para atuar nos

temas. Silos aumentam o número de *handoffs* (passagem de 'bastão' entre equipes, é quando uma pessoa entrega algo para a outra) e isso aumenta o *leadtime* (tempo total entre o início de uma atividade até sua entrega final) de entrega de *software*, pois a cada passagem de atividade entre uma equipe e outra, existe um tempo de pausa no atendimento, pois cada equipe está atuando com um número de atividades em sequência.

É necessário oferecer oportunidade aos engenheiros de adquirirem os conhecimentos necessários para realizarem seu trabalho, bem como utilizar o famoso *job rotation* (troca de papeis dentro da mesma equipe).

E, claro, os *E-shaped* são mais raros, pois "possuem profundo conhecimento em algumas áreas, experiências em muitas áreas, comprovada execução das habilidades, estão sempre inovando e possuem potencial quase ilimitado" (KIM et al., 2021, p.195).

Quando valorizamos os indivíduos somente pelos seus conhecimentos e experiências atuais, recaímos no tema de Carol Dweck sobre o *mindset* fixo, onde existe a crença de que a inteligência e suas habilidades são natas e não podem ser mudadas ou melhoradas.

Levando estes pontos em consideração, a liderança deve buscar pessoas que tenham conhecimento generalista, incentivando e promovendo ações de busca pelo conhecimento (treinamentos) para que seu time evolua e tenha um plano de desenvolvimento profissional. Assim como é responsabilidade de cada profissional buscar esta mentalidade para se capacitar constantemente em busca de sua evolução.

> Capítulo 2

Capítulo 2. Resiliência

Objetivos

Quando se fala em ambiente organizacional, pessoas e times com foco em aprendizado, deve-se pensar também em como aplicar isto nas infraestruturas existentes, tornando-as resilientes e à prova de falhas.

Este é o foco de todas as empresas para que seus serviços estejam sempre disponíveis aos clientes, sem interrupções. Hoje sabemos que mesmo um pequeno período de indisponibilidade em um banco ou site de varejo pode gerar prejuízos enormes, pois ocorre uma fuga dos clientes, que acabam por buscar e utilizar outros sites/aplicativos que estejam no ar e estáveis durante esse período, para atender suas necessidades.

Figura 7 – Indisponibilidades e instabilidades.

Fonte: Elaborado pela autora.

E quando temos empresas com sistemas complexos, se torna difícil e praticamente impossível prever todos os resultados, o que pode ocasionar acidentes bem catastróficos, mesmo com uma gama de ferramentas sendo utilizadas para evitar isso (KIM et al., 2021).

O ideal seria antever os problemas e poder agir de forma automática para o restabelecimento ou solução do problema, ou seja, é necessário poder ter a:

- Deteção (prevenção).
- Ação ou autorremediação no momento do problema (*self-healing)*.
- Melhoria contínua (sistema de aprendizado evolutivo).

Uma sugestão é utilizar o processo de resolução de problemas PDCA – *Plan Do Check Act*, criado pelo cientista americano Walter A. Shewhart na década de 20 e popularizado pelo estatístico W.E. Deming, para gerenciar e retroalimentar a evolução dos processos, focando na melhoria contínua.

Caso Netflix

Em 2011, houve uma grande indisponibilidade na Amazon, afetando a zona toda no leste dos Estados Unidos (AWS US-East) e impactando todos os seus clientes. A Netflix utilizava os serviços da Amazon, mas aparentemente conseguiu lidar com essa falha, sem impactar seus clientes. Esse caso se tornou famoso, fomentando a discussão e curiosidade de todos sobre como a organização conseguiu isso. E foi assim que o serviço que criaram ficou conhecido como *Chaos Monkey* (Macaco do Caos).

Figura 8 – Chaos Monkey.

Fonte: <https://netflix.github.io/chaosmonkey/>.

Como eles conseguiram isso?

Em 2008, a Netflix ainda tinha sua aplicação monolítica; a partir de 2009 passou por uma reestruturação de sua arquitetura, que chamaram de *cloud native*, com base em baixo acoplamento, o que possibilitou maior independência e distribuição dos serviços de uma forma que os tornasse resilientes mesmo em caso de uma grande falha, como a que ocorreu em 2011.

Eles criaram vários tipos de simulações para atacar e testar sua infraestrutura, automatizando a detecção e solução dos problemas, com perfeição. E isto não ocorreu da noite para o dia, foram testando e evoluindo com suas próprias falhas de forma constante. Estas práticas evidenciam como o processo de aprendizado é importante para a evolução das organizações.

Dito isto, também é importante destacar a importância da cultura do aprendizado, de forma justa, pois falhas ocorrerão e não se deve procurar culpados, mas focar na solução do problema.

Fazer as perguntas certas:

- Onde falhamos?
- Por que isso aconteceu?
- O que podemos fazer para que isso não se repita?

Um fato recorrente nas organizações é a busca por culpados e isso gera medo nas pessoas. A punição gera um efeito negativo sobre todos, inibindo a possibilidade de crescimento através de ideias.

Dr. Dekker explica isso com a “Teoria da Maçã Podre” (uma maçã podre pode contaminar as demais maçãs e estragar a caixa inteira), na qual se acredita que eliminar a pessoa que causou o erro fará com que ele não se repita (KIM et al., 2021, p. 464).

Quando não há punição, cria-se o efeito contrário: os profissionais passam a desejar o compartilhamento desses aprendizados para a organização toda, para evitar que os mesmos erros se repitam com outras equipes.

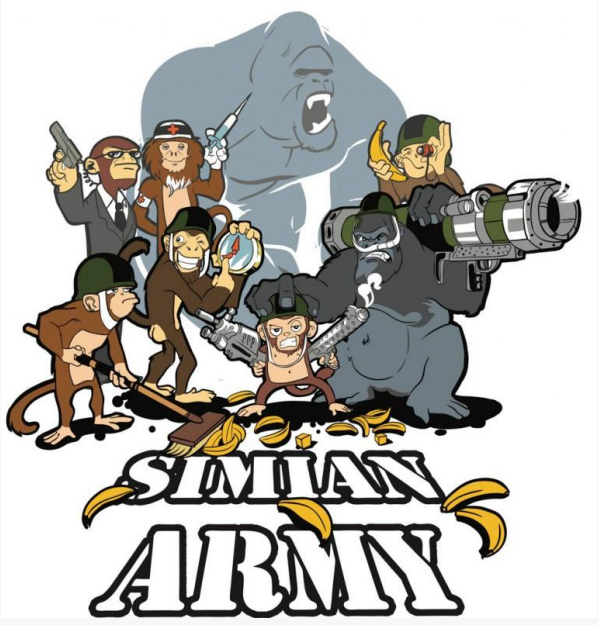
Leitura adicional em:

- https://learn.microsoft.com/pt-pt/azure/architecture/framework/resiliency/chaos-engineering

Tipos de Macacos

A Netflix criou várias maneiras de testar as falhas em seus sistemas, o que chamou de "*Simian Army*", o Exército Simiano (KIM et al., 2021, p.610-611):

Figura 9 – Netflix Simian Army.



Fonte: < https://commons.wikimedia.org/wiki/File:Netflix_simianarmy-768x797.jpg>.

- *Chaos Gorilla*: simula a falha de uma zona de disponibilidade inteira da AWS.

- *Chaos Kong*: simula a falha em regiões inteiras da AWS, como América do Norte ou Europa.

Outros membros do Exército Simiano agora incluem:

- *Latency Monkey* (Macaco da Latência): induz atrasos artificiais ou *downtime* (tempo de inatividade) em sua camada de comunicação *RESTful cliente server* para simular degradação do serviço e assegurar que serviços dependentes respondam apropriadamente.

- *Conformity Monkey* (Macaco da Conformidade): descobre e desliga instâncias AWS que não estão aderentes às boas práticas (por exemplo, quando instâncias não pertencem a um grupo de *autoscaling* ou quando não há endereço do engenheiro de escalonamento listado no catálogo de serviço).

- *Doctor Monkey* (Macaco Doutor): entra em verificações de saúde que são feitas em cada instância e encontra instâncias insalubres e as fecha proativamente se os proprietários não consertarem a causa-raiz a tempo.

- *Janitor Monkey* (Macaco Zelador): garante que seu ambiente de nuvem esteja livre de desordem e desperdício. procura recursos não utilizados e os descarta.

- *Security Monkey* (Macaco de Segurança): uma extensão do Macaco da Conformidade, encontra e finaliza instâncias com violações ou vulnerabilidades, como grupos de segurança AWS configuradas inadequadamente.

Dias de Jogos

Muniz et al. (2020), no Livro Jornada *DevOps*, descreve:

“Dias de Jogo - ou *game days* - é um conceito que vem da engenharia da resiliência, cujo objetivo é criar exercícios programados para aumentar a resiliência através da injeção de falhas de grande escala em sistemas críticos. Durante a injeção de falhas estamos expondo os defeitos latentes em nosso sistema (que são problemas que só aparecem quando falhas são injetadas nele)”.

O objetivo principal destes testes regulares é construir “memória muscular” sobre como os times respondem às falhas em sistemas e processos, devendo cobrir áreas como operações, segurança, confiabilidade, performance e custo (“*AWS Well-Architected Framework - Game Days*”, 2020).

Estes ensaios chamados de dias de jogos foram popularizados por Jesse Robbins, que durante seu trabalho na Amazon passou a ser conhecido como “Mestre do Desastre”. Ele sustenta que “um serviço não está realmente testado até o estragarmos em produção”.

Um dos exemplos citados no livro *The DevOps Handbook* sobre dias de jogos foi a simulação de um terremoto no Vale do Silício, liderada por Kripa Krishnan, Diretor Técnico do Programa na Google, chamado *Disaster Recovery Program (DiRT).* Nesta simulação, houve desconexão do campus *Mountain View* com a Google, falta de luz total nos grandes *data centers*, bem como ataque às cidades onde os engenheiros residiam.

Dentre os aprendizados coletados com essa experiência, destaca-se um bem interessante em que nenhum engenheiro conhecia o processo de compra emergencial de diesel, quando os *data centers* ficaram sem o combustível em seus geradores de *backup*, resultando numa compra com o cartão particular de um dos engenheiros no valor de cinquenta mil dólares (KIM et al., 2021, p. 479).

Como organizar

Quem deve participar da organização dos Dias de Jogos?

Isso dependerá de qual carga de trabalho será testada, pensando em todos os aspectos do seu negócio, desde o desenvolvimento até a produção. Uma vez definido o tipo de falha, chamar todos os envolvidos nesse processo para trazerem ideias do que deverá ser testado.

Para o cenário selecionado, avaliar os problemas que já ocorreram é um bom caminho para testar se agora os procedimentos estão completos, bem como fatos bem atípicos, mas não impossíveis, como inundações em um *data center*, explosão das torres de transmissão de energia em mais de um *data center*, picos de vendas em datas festivas (como Dia das Mães, Natal, *Black Friday)*, ataques cibernéticos, entre outros.

Será necessário definir quem executará a falha, a sequência de ações (de preferência automaticamente), quem fará o monitoramento (uma ou mais pessoas de áreas distintas), preparar o ambiente onde ocorrerá a falha (normalmente um bem parecido com o de produção), comunicar a todos que se trata de uma simulação, em que dia e hora ocorrerá, término previsto, coletar os resultados, observar, analisar, aprender e divulgar os resultados.

Alguns passos para facilitar a organização dos Dias de Jogos:

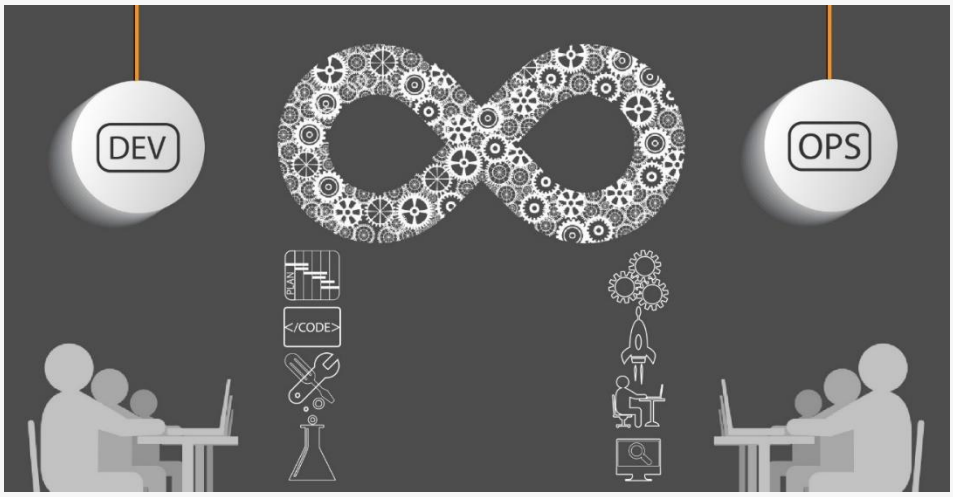
- Planejar a interrupção: qual cenário deseja testar?
- Adotar medidas quando a falha ocorrer: o que deve ser feito?
- Testar medidas: validar se as medidas adotadas foram eficazes.
- Executar interrupção: verificar se os resultados da ação foram satisfatórios.

- Seguir o plano e aprender: verificar as situações que não foram previstas, adicionar as situações para o plano.

Desta maneira, será possível se tornar cada vez mais resiliente, pois a cada modo de falha testado surgirão situações novas que se tornarão aprendizados. Isso inclui problemas nos processos, pessoas, infraestrutura (*on-premise*, *cloud, hybrid*) ferramentas ou qualquer item que não tenha atendido às expectativas.

Conforme descrito em seu site, a AWS Well-Architected Framework - Game Days (2020) orienta analisar o Dia de Jogo, coletando os depoimentos dos participantes e afetados, para poder endereçar as ações, sejam elas de melhoria de ferramentas, treinamentos, manuais mais detalhados etc. Qualquer oportunidade de melhoria em qualquer área deve ser documentada para que seja testada num próximo Dia de Jogo.

Figura 10 – DevOps Game Day.



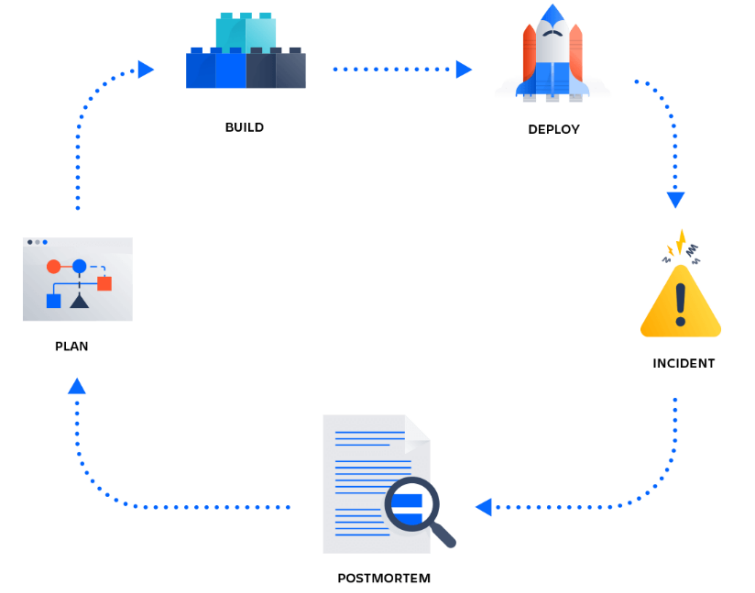
Fonte: < https://wallpaperaccess.com/devops#google_vignette>.

> Capítulo 3

Post-mortem

O objetivo de uma reunião *Post-mortem* ou *Blameless Post-mortem* é a de identificar a causa-raiz de problemas para a devida aplicação da correção de forma definitiva.

Figura 11 – The importance of an incident postmortem process



Fonte: <https://www.atlassian.com/incident-management/postmortem>.

Post-mortem significa após a morte, ou seja, depois do fato ocorrido. É como uma reunião investigativa. Imagine que, após um acidente de carro numa pista de corrida, a Seguradora ou a Organizadora do evento investigará o local do acidente, verificará as marcas no chão dos carros envolvidos, os vídeos com imagens de todos os ângulos, as condições do asfalto e do tempo, a equipe que trabalhou nos carros fazendo os ajustes, a inclinação da pista, questionará as pessoas envolvidas e testemunhas que presenciaram o fato e, desta maneira, poderá analisar melhor a sequência de fatos para entender o que ocorreu. Em alguns casos, o resultado poderá ser a melhoria na qualidade do asfalto, uma correção na inclinação da pista,

mudanças nas medidas de segurança dos pilotos de corrida em suas cabines etc.

A parte crucial para que a reunião seja efetiva é a palavra *Blameless* (sem culpa), pois não se trata de punir alguém pelo seu erro, mas sim de entender melhor o que ocorreu para que possamos corrigir ou estar preparados para prevenir numa próxima ocasião.

Quando não se busca culpados, retiramos o medo das pessoas relatarem exatamente o que ocorreu. Fazendo um comparativo com a Agilidade, seria como uma cerimônia de retrospectiva.

Retirar a culpa é importante, mas isto não quer dizer que se retira a responsabilidade de alguém. É um momento em que se estudam as circunstâncias que levaram ou possibilitaram que a falha ocorresse por alguém: removendo-se a culpa é possível retirar o medo; sem medo, você ganha honestidade (KIM et al., 2021, p.609 apud MACRI).

Fazendo isso de forma recorrente, pode-se ter as práticas de análise e correção muito rapidamente, um ciclo de aprendizado repetitivo com melhoria contínua, independentemente do tamanho do time ou de onde ele ocorra.

No livro Google SRE, o capítulo 15 é dedicado totalmente para o tema e como ele é tratado na empresa, onde se valoriza a colaboração e o compartilhamento de conhecimentos em qualquer etapa do problema. Se o trabalho de um SRE já é bem complexo, imagine se houver discriminação e acusações (apontamento de dedos) para aqueles que cometerem erros?

No livro, é definido como: “um *post-mortem* é um registro escrito de um incidente, seu impacto, as ações tomadas para mitigá-lo ou resolvê-lo, a(s) causa(s) raiz(es), e as ações de acompanhamento para evitar que o incidente se repita” (LUNNEY. LUEDER. O’CONNOR, 2017).

Os principais objetivos da reunião são:

- Identificar o que se fez bem, para poder repetir em situações parecidas.
- Identificar o que não ocorreu tão bem, para poder fazer de maneira diferente e melhor, refinando as técnicas utilizadas no futuro.
- Identificar o que saiu errado e sugerir o que pode ser feito melhor, com abordagens que garantam a qualidade e segurança na próxima ocasião.

Como organizar a reunião

O segredo do sucesso da reunião é que ela seja agendada o mais rápido possível, logo após a resolução do problema, pois os detalhes são esquecidos rapidamente e as condições podem se alterar, prejudicando o entendimento e rastreio dos eventos responsáveis pelo ocorrido.

Conforme recomendado pelo livro "*The DevOps Handbook*" (2021, p. 466-470), as pessoas que devem participar da reunião são:

- Quem pode ter contribuído nas decisões que ocasionaram o erro.
- As que identificaram o problema.
- As que responderam e diagnosticaram o problema.
- As que foram afetadas pelo problema.
- Outros interessados (qualquer pessoa com interesse no tema).

O que deve ser feito:

- Criar uma *timeline* dos eventos de forma detalhada, sob várias perspectivas, sem punir as pessoas que causaram o erro.
- Empoderamento e autonomia para que cada um possa melhorar a segurança, detalhando sua parte na falha.
- Incentivar que os que cometeram os erros se tornem os especialistas no assunto e eduquem os demais da organização para que isso não se repita.
- Aceitar que há espaço para que as pessoas possam aceitar ou não as recomendações.
- Propor contramedidas para que os mesmos erros não se repitam, definindo os responsáveis e a data-alvo para atendimento da solução (neste caso, pode haver várias áreas envolvidas).

Por fim, deve-se publicar o resultado da reunião:

- Divulgar amplamente os resultados em local centralizado e de fácil acesso a todos da organização, pois isto aumenta a confiança dos clientes internos e externos, devido à transparência.
- Incentivar que todos leiam as informações para aumentar o aprendizado organizacional, o que evitará recorrências.
- Se necessário, impedir que os incidentes graves sejam encerrados sem que a reunião tenha acontecido e concluído.
- Estas ações ajudam a tornar aprendizados e melhorias locais em globais, todos da organização podem aprender e praticar com elas.

Ferramentas que ajudam a organizar um *post-mortem:*

- https://www.smartsheet.com/content/project-post-mortem-templates
- https://docs.google.com/spreadsheets/d/1q-MVeH03mWEYRRyOPQickK0rlDOQc2no1JuK9xqIHk0/copy?usp=sharing

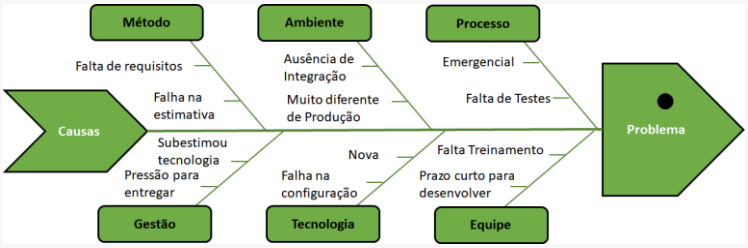
Exemplo de post-mortem:

- https://sre.google/sre-book/example-postmortem/

Diagrama de Ishikawa para identificação de causa-raiz

Também conhecido como Espinha de Peixe, é usado para descobrir e listar todos as possíveis causas de um problema.

Figura 12 – Diagrama de Ishikawa



Fonte: Elaborado pela autora

Leitura adicional em:

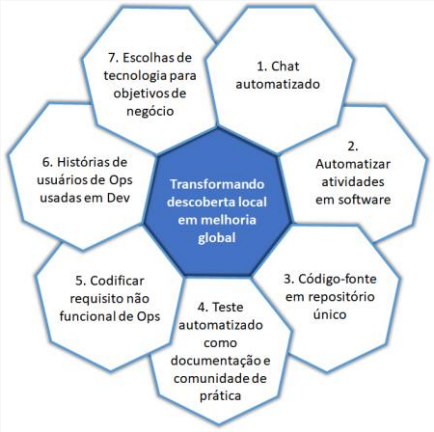
- https://annelisegripp.com.br/retrospectivas-ageis/

Como transformar descobertas Locais em melhorias Globais

Ao se criar ambientes seguros com cultura justa, as pessoas se sentem encorajadas a falar sobre suas falhas em reuniões livres de culpa, pois sabem que não serão penalizadas e tudo será tratado como aprendizado, elas se sentem parte da solução e não dos problemas.

Alguns mecanismos podem ser adotados para melhorar ainda mais este ambiente, multiplicando os resultados positivos.

Figura 13 – Transformando descoberta local em melhoria global



Fonte: adaptado de MUNIZ et al., 2020, p. 169

Salas de Chat ou Chat Bots

Facilitam a comunicação entre times e podem ter gatilhos automatizados, criando transparência e documentação de seu trabalho. Um exemplo disso foi a adoção desta técnica pela GitHub com seus *ChatOps*, onde introduziram uma automação dentro das salas de bate-papos do time de Operações. A aplicação Hubot foi criada para interagir com o time *Ops*, atendendo comandos de *deploy* enviados pelo *chat* ao invés de um *script* via linha de comando, ou seja, ao digitar o comando no chat ele se comunica com a aplicação, que executa a instrução recebida subindo a *branch* do código para algum ambiente, podendo ser de testes ou até mesmo para produção.

Alguns benefícios desta técnica incluem:

- Qualquer pessoa pode ver tudo que está acontecendo.
- Engenheiros podem ver e aprender como é a rotina de trabalho, mesmo em seu primeiro dia.
- As pessoas se sentem mais abertas a perguntar e pedir ajuda, pois veem outros se ajudando mutuamente.
- Aprendizado organizacional é mais rápido e organizado.

O histórico do *chat* pode ser utilizado como documentação e até mesmo reuniões formais e atas podem ser substituídas por essa forma de trabalho, economizando tempo, pois tudo é resolvido e alinhado de forma dinâmica.

Leitura adicional em: https://hubot.github.com/

Automatizar atividades em *software*

Um dos maiores desafios de se manter padrões e processos de *software*, desde a arquitetura, ambientes, gestão de infraestrutura e testes, é manter sua documentação atualizada. Ou ela é inexistente ou muitos não possuem tempo para atualizá-la de maneira padronizada.

Ao invés destes documentos serem gerados e atualizados em documentos *Word*, utilizar um local único como repositório centralizado dessas informações, traz a grande vantagem de disponibilidade à toda organização, sendo possível reutilizá-la também.

Um exemplo deste uso foi o que a *General Electric* criou: um mecanismo automatizado chamado *ArchOps* que possibilitou que seus desenhos fossem documentados através do próprio código, com diagramas que são automatizados e atualizados constantemente e de forma dinâmica, sem a necessidade de alterar documentos estáticos. Isto possibilitou que qualquer um pudesse ver a arquitetura de forma clara e atualizada.

Leitura adicional em:

https://www.bloorresearch.com/technology/archops/?cn-reloaded=1

Código-fonte em repositório único

Ao concentrar num único local todos os códigos, configurações de infra, padrões de testes e segurança, configurações de *pipeline*s, padrões de

codificação, tutoriais e *wikis*, qualquer pessoa na organização terá uma compreensão maior do sistema todo e não somente de uma parte.

Pensando em microsserviços, isso ajuda muito os desenvolvedores no conhecimento do todo e não somente do pedaço de código que ele está atuando.

Uma pequena observação aqui é que, dependendo do tipo de informação disponível no código e do tipo de organização, alguns repositórios não podem estar abertos a todos, vale a regra de se criar grupos de acessos bem definidos para que se tenha a proteção adequada para que somente os profissionais devidos tenham acesso para essa porção de código. Normalmente, códigos relativos a criptografia, antifraude e segurança merecem esse tipo de atenção.

A Google utiliza esse tipo de mecanismo e em 2015 já dizia que possuía um único repositório de código-fonte, com mais de 1 bilhão de arquivos e mais de 2 bilhões de linhas de códigos, compartilhados com seus mais de 25 mil engenheiros, espalhados em todas as suas frentes, como *Google Maps, Google Search, Google Docs, Google Calendar, Gmail e YouTube*.

Além disso, qualquer artefato que possa ser compartilhado reside nesse repositório, incluindo:

- Padrões de configuração de suas bibliotecas, infraestrutura e ambientes.
- Ferramentas de *deploy* (implantação).
- Ferramentas e padrões de testes, incluindo segurança.
- Ferramentas de *deploy* de *pipelines*.
- Ferramentas de monitoramento e análises.

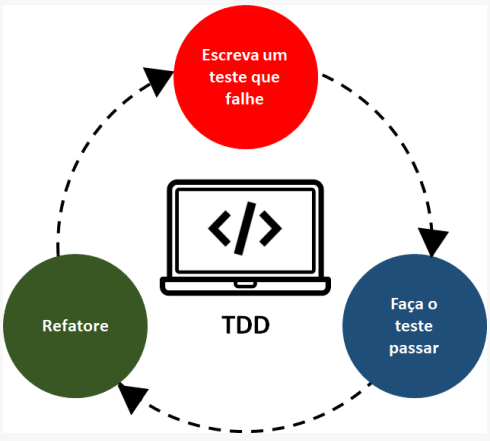
- Tutoriais e padrões.
- Componentes.
- Código-fonte.

Testes automatizados como documentação e comunidade de prática

Ao criar testes automatizados, geramos também exemplos de como a aplicação funciona. Quando essas informações estão em bibliotecas compartilhadas na organização, tem-se uma rápida propagação de informações atualizadas (atual e suas melhorias) praticamente em tempo real, assegurando assim que se tenha uma autodocumentação de como funcionam a aplicações, uma documentação viva.

Um exemplo disso é quando se utiliza práticas de TDD *(Test Driven Development)*, onde os testes automatizados são escritos antes mesmo da codificação deles. Desta maneira, tanto as entradas como possíveis saídas de cada pedaço de comunicação do *software* estarão documentadas.

Figura 14 – TDD *(Test Driven Development).*



Fonte: elaborado pela autora.

As comunidades de práticas são encontros onde as pessoas discutem o comportamento do sistema, seja por meio de salas de bate-papo, como *Teams*, *Slack Hangouts* ou qualquer outro meio que possibilite

encontros virtuais, para facilitar a rápida propagação de conhecimento; um espaço em que qualquer um que tenha dúvidas possa obter suas respostas prontamente dos usuários (são até mais rápidos em responder do que os desenvolvedores, pois conhecem a fundo o comportamento e utilização dos sistemas).

Blitz de melhoria (*Improvement Blitz* ou *Kaizen Blitz*)

É uma maneira de melhorar o trabalho diário: deve-se reservar tempo para isso, o que ajudará a reduzir a dívida técnica *(technical debt).*

Como organizar: reunir um grupo para se concentrar em processos com problemas, para que trabalhem intensamente para eliminá-los. A *blitz* pode durar alguns dias ou uma semana para atingir seu objetivo, mas é importante que todas as áreas, desde o negócio até sistemas e operações, compreendam que é um tempo investido para melhoria contínua. O ideal é que toda organização se mobilize para a *blitz*, pois reforça a valorização de resolver problemas e a cultura de solução de dívida técnica.

Figura 15 – Trabalho de Equipe

Fonte: <https://giphy.com/gifs/office-teamwork-coworkers-dSetNZo2AJfptAk9hp>.

Também são conhecidos como *hack days* ou *hackatons* e 20% do tempo é investido em inovação. Deve-se reservar tempo de times *Dev* e *Ops* para que trabalhem na melhoria em conjunto.

Ao final do período, todos devem apresentar o problema que queriam resolver e como trabalharam para eliminá-lo.

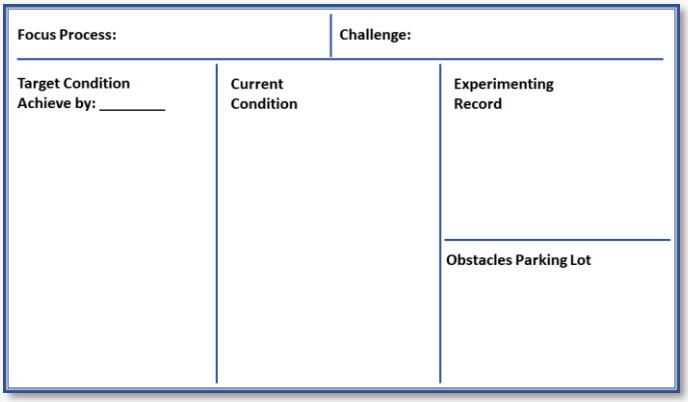
Melhoria *Kata* (*Kata Improvement*)

A melhoria *Kata* é a maneira de tornar o trabalho de melhoria numa prática habitual, porque somente com a prática se chega à excelência. Seus pré-requisitos são: prática e repetição.

A maior dificuldade para se chegar ao hábito é a constância da prática, por isso é importante ser recorrente. E o trabalho de melhoria necessita persistência:

- Descrever o seu desafio final (aonde quer chegar em 6 meses ou 3 anos?).
- Definir a situação atual (listar os fatos e dados atuais).
- Experimentar soluções (talvez não consiga sucesso nas primeiras tentativas).
- Não desanime e continue a buscar a solução, com vários pequenos experimentos.

Figura 16 – Learner's Storyboard.

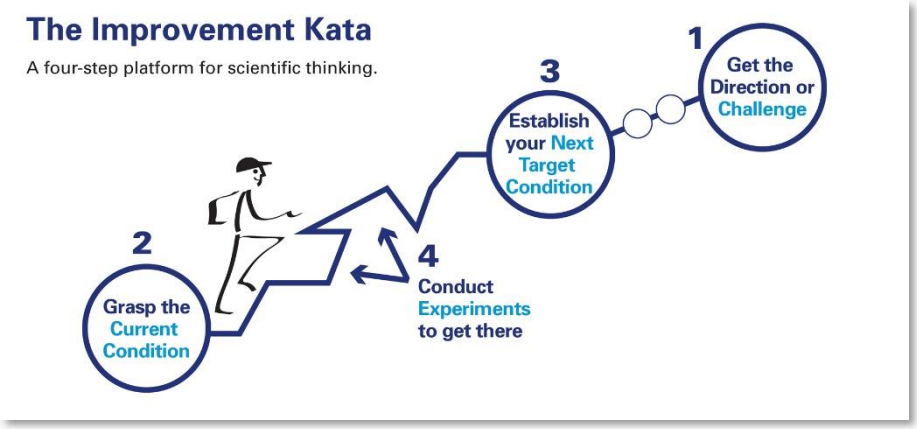


Fonte: adaptado de <https://www.oreilly.com/library/view/the-toyota-kata/9781259861031/appendix.xhtml>.

Leitura adicional em:

- https://www.lean.org/the-lean-post/articles/a-video-primer-on-improvement-and-coaching-kata/

Figura 17 – The Improvement Kata



Fonte: <https://www.lean.org/the-lean-post/articles/a-video-primer-on-improvement-and-coaching-kata/>.

> Capítulo 4

Capítulo 4. Projetando a Operação

Requisitos não funcionais (*NFR – Non Functional Requirements*)

Requisitos não funcionais são aqueles relacionados à utilização da aplicação, como disponibilidade, desempenho, usabilidade, confiabilidade, segurança, quais tecnologias estão envolvidas, e sua manutenibilidade.

Quando os times de desenvolvimento participam na identificação e solução de incidentes, principalmente os pós-implantação, o software e as aplicações tornam-se cada vez mais confiáveis e mais bem desenhadas para as Operações.

Exemplos de requisitos não funcionais incluem garantir que haja:

- Telemetria suficiente no ambiente de produção em suas aplicações e ambientes.
- Capacidade de rastrear precisamente as dependências.
- Serviços que são resilientes e que degradam de forma controlada.
- Compatibilidade entre versões anteriores e posteriores.
- A capacidade de arquivar dados para gerenciar o tamanho dos conjuntos de dados de produção.
- A capacidade de entender e pesquisar facilmente as mensagens de logs entres os serviços.
- A capacidade de rastrear as solicitações dos usuários através de vários serviços.
- Configuração simples e centralizada em tempo de execução usando *feature flags* etc.

Desta maneira, torna-se mais fácil para todos os novos serviços e existentes alavancar o conhecimento coletivo e a experiência da organização.

Histórias de usuário de Operações Reutilizáveis

Construir histórias de usuário de Operações que sejam reutilizadas dentro de desenvolvimento é algo importante para garantir que tenhamos o mínimo de esforço na próxima execução.

E quando um trabalho de Operações não pode ser totalmente automatizado ou mudado para um autoatendimento, nosso objetivo é torná-lo o mais repetitivo e determinístico possível, padronizando o serviço necessário, automatizando tudo o que for possível e documentando o trabalho.

Deve-se também reduzir as transferências entre equipes (*hand offs*), para redução de tempos de espera e possíveis erros, o que permitirá melhor planejamento de etapas futuras.

Alguns exemplos de histórias para reuso seriam:

- Criação de ambientes (sejam eles *on-premise* ou *cloud)*.
- *Scripts* de implantação.
- Validação de capacidade.
- Validação de segurança.

Isso garantirá reuso de atividades em muitos ou quase todos os projetos.

Assegurar escolhas de tecnologias que ajudem a alcançar os objetivos organizacionais

Um dos principais objetivos do *DevOps* é a produtividade do desenvolvedor. E quando se tem arquiteturas orientadas a serviços, pequenos times podem construir e executar em qualquer linguagem ou framework para melhor servir às necessidades da organização.

Porém, deve-se ter o cuidado de escolher as tecnologias e plataformas adequadas para que os times de Operação possam suportá-las. Como colaboração também é um dos pontos básicos do *DevOps*, os times de Desenvolvimento e Operações devem poder influenciar nessas decisões ou serem isentos de responsabilidade, caso sejam usadas plataformas não suportadas por eles.

Quando ambas as equipes participam das decisões sobre infraestrutura e serviços de produção, bem como listam todas as dependências que são suportadas atualmente, é possível descobrir qual delas podem estar criando um volume razoável de falhas, retrabalho ou trabalho não planejado em suas rotinas diárias.

O objetivo é identificar as infraestruturas ou plataformas que potencialmente atrapalham os trabalhos para assim, possibilitar que Operações possam focar naquilo que realmente ajuda a organização a atingir seus objetivos.

É primordial identificar as tecnologias que:

- Impedem ou reduzem a vazão do trabalho.
- Desproporcionalmente criam altos níveis de trabalho não planejado.
- Desproporcionalmente criam alto volume de pedidos de suporte.

- São os mais inconsistentes com nossos resultados de arquitetura desejados, como por exemplo produtividade, estabilidade, segurança, confiabilidade, continuidade do negócio).

Todas as técnicas descritas até aqui demonstram como é possível chegar no 'estado da arte' em compartilhamento, não somente de tarefas de *Dev* e *Ops*, mas de toda organização. Importante que todos tenham este pensamento, de sempre incorporar o conhecimento adquirido em pequenas situações no conhecimento coletivo, gerando assim, uma base de experiências cumulativas acessível a todos.

Uma dica é ter uma *Wiki* organizada por áreas de conhecimento, onde todos podem gerar manuais e dicas, lições aprendidas, padrões etc.

> Capítulo 5

Capítulo 5. Segurança

Segurança da Informação

Há três princípios básicos quando falamos de segurança da informação:

- Integridade: relacionada à acuracidade e completude da informação.
- Confidencialidade: relacionada a garantir que a informação não seja disponibilizada ou compartilhada para entidades, processos ou indivíduos não autorizados.
- Disponibilidade: relacionada ao fato de estar acessível e utilizável quando necessário, desde que essa solicitação seja oriunda de uma entidade autorizada.

DevOps engloba as melhores práticas entre áreas e a segurança tem papel fundamental para o sucesso das entregas, seja do lado da construção de aplicações através do software, quanto do lado da infraestrutura, que possibilitará que os códigos funcionem adequadamente.

Segundo o relatório *State of API Security* (2022), "94% dos que participaram da pesquisa disseram que experimentaram problemas de segurança em APIs em produção". O tema é crucial para que as organizações mantenham sua imagem íntegra no mercado.

A ISO (*International Organization for Standardization)* é uma organização independente e não governamental que visa definir padrões internacionais sobre a gestão da segurança da informação, ajudando as organizações a manterem seus ativos seguros.

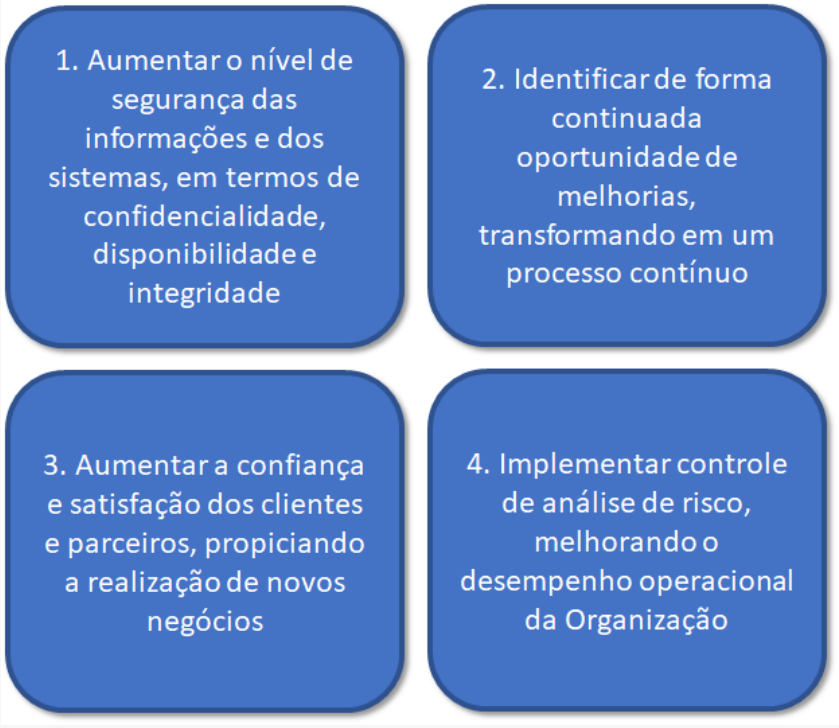
Figura 18 – Selo ISO 27001.

Fonte: <https://defensec.com.br/iso-iec-27001-do-gap-analysis-a-certificacao-overview-pt1/>.

A certificação ISO 27001 “trata da qualidade da gestão da segurança da informação” e no Brasil “é regulamentada pela Associação Brasileira de Normas Técnicas (ABNT – NBR)” (“ISO/IEC 27001 - Do GAP Analysis à Certificação (Overview) - pt1”, 2020).

A seguir, os principais objetivos da certificação (CORDEIRO, 2020).

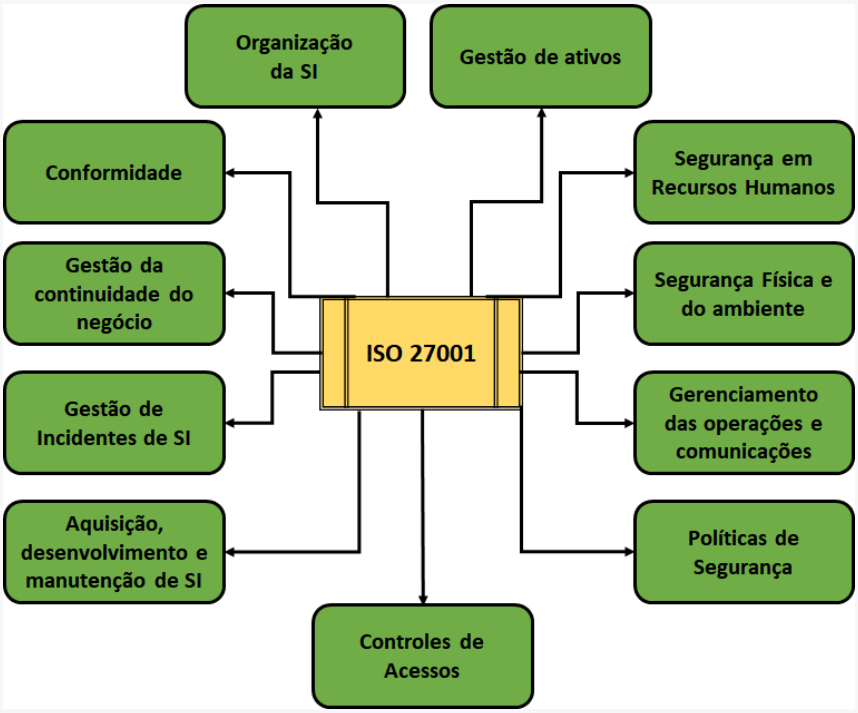
Figura 19 – Principais objetivos da certificação para uma organização com relação à segurança



Fonte: adaptado de CORDEIRO, 2020.

Para se entender melhor os benefícios desta certificação, há que se entender onde esse ecossistema tem sua abrangência. Ela permeia todos os aspectos necessários para garantir que uma organização esteja blindada contra possíveis problemas em todos os âmbitos (internos e externos).

Figura 20 – Ecossistema da ISO 27001.



Fonte: adaptado de CORDEIRO, 2020.

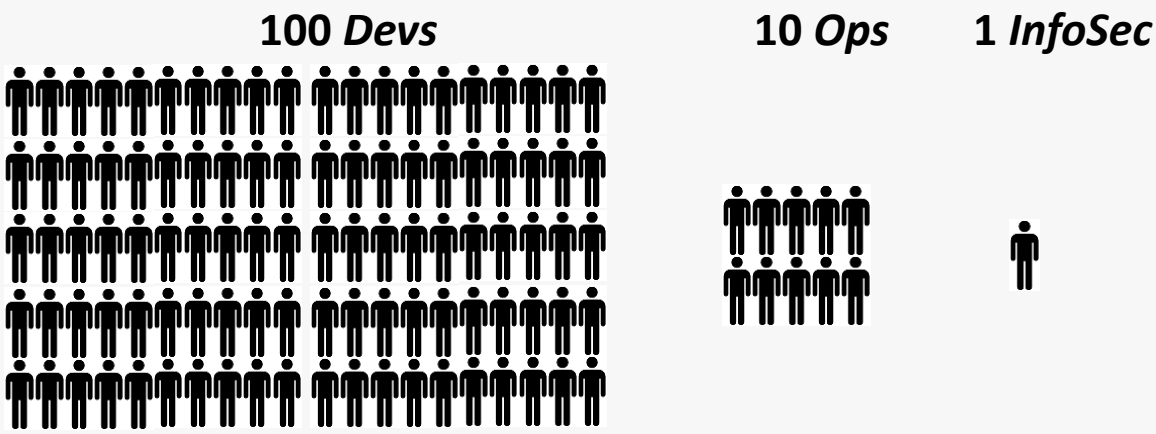
Segurança como gargalo no final do ciclo

Quando falamos de desenvolvimento seguro, deve-se levar em consideração esses três princípios básicos (integridade, confidencialidade e disponibilidade) em qualquer etapa do ciclo de vida de desenvolvimento do *software*, mais conhecido pelo acrônimo SDLC, do inglês *Software Development Life Cycle*.

É muito comum que empresas tenham times de segurança somente no final desse ciclo, pois a quantidade de pessoas é limitada e isso dificulta o fluxo de entrega de valor ao cliente. Sendo segurança um tema tão importante, esse time tem o poder de não autorizar subidas de códigos no

ambiente de produção que não estejam aderentes aos padrões definidos de segurança.

Figura 21 – Distribuição típica entre equipes *Dev, Ops* e *Infosec* nas organizações.



Fonte: KIM et al., 2021, p.526.

Por esta razão, esse time deve participar ativamente desde o início do ciclo, para garantir a escrita segura do código e que a automação de todos os controles necessários também estejam presentes nesse momento e não somente no final.

E isso só será possível se houver automação integrada com a segurança da informação no trabalho diário, senão times *Infosec* só conseguirão validar a conformidade, que é o oposto do que se espera de um time de engenharia de segurança.

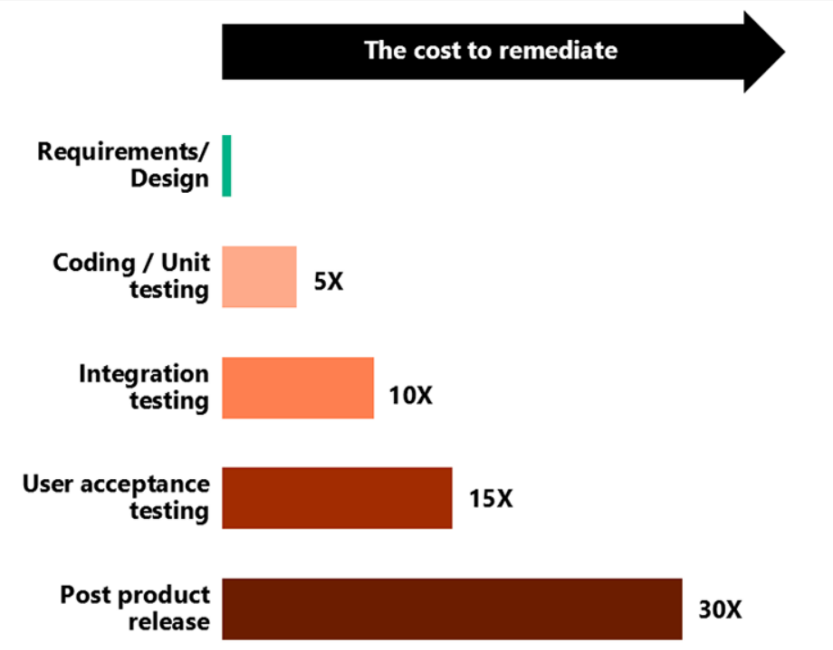
Por isso, é necessário que os times de desenvolvimento estejam envolvidos com a *Infosec* o mais cedo possível no SDLC e não somente no final da entrega do projeto. Isso é praticar *shift left*, incluindo testes de segurança desde o início.

E quando se fala em *shift left*, vêm à tona os motivos que nos fazem movimentar os testes e feedbacks para o início do SDLC: porque queremos

segurança e qualidade, e com qualidade podemos entregar produtos comercializáveis, reduzir custos e nos manter além dos competidores.

Uma leitura muito boa está na plataforma de ensino da *Microsoft Learning*, na qual se fala da mudança de mentalidade de "garantia da qualidade" para "Qualidade Contínua": *Explain DevOps Continuous Delivery and Continuous Quality - Training | Microsoft Learn*.

Figura 22 – O custo para remediar.



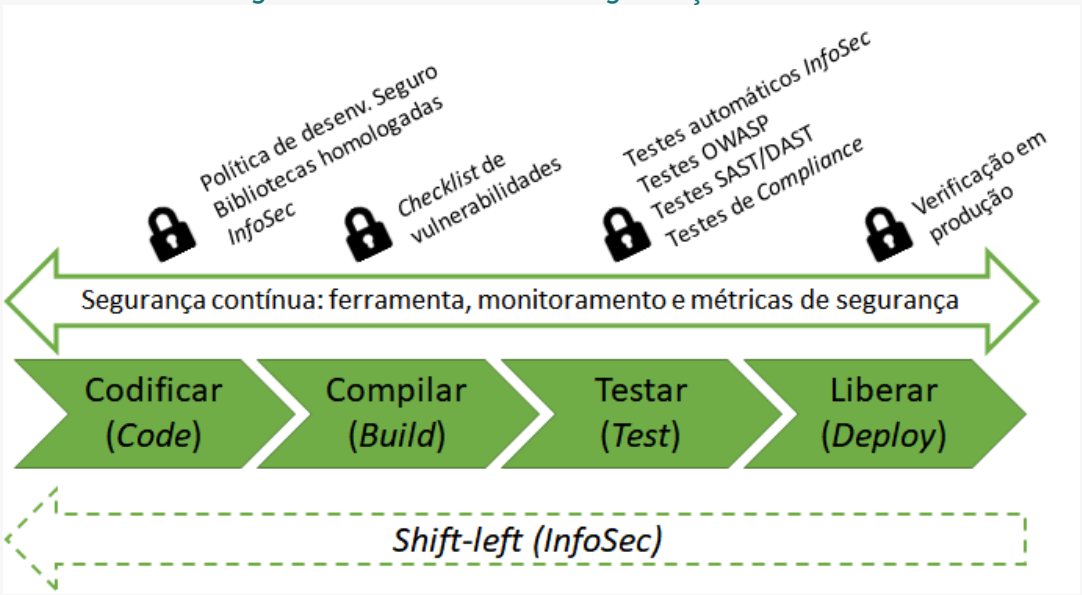
Fonte: Microsft Learning < https://learn.microsoft.com/en-us/training/modules/explain-devops-continous-delivery-quality/3-explore-continuous-quality>.

No site, é demonstrado que o custo de se consertar um erro no ambiente de Produção é 30 vezes maior do que quando é encontrado no início do projeto, durante a escrita dos requisitos e *design*. Se forem envolvidos outros custos e dependendo do produto, esta conta pode chegar a números ainda maiores.

Segurança Preventiva

E como aplicar a segurança preventiva no ciclo *DevOps*? Segundo MUNIZ et al. (2020), é possível implantar através do conceito da segurança contínua (*Continuous Security – CS*), utilizando ferramentas automatizadas, monitorando continuamente a segurança e com métricas muito bem definidas.

Figura 23 –Shift Left e a segurança contínua.



Fonte: adaptado de MUNIZ et al. 2020 apud SANTOS. MUNIZ. ADAPTNOW, palestra DevSecOps Infnet, 2018.

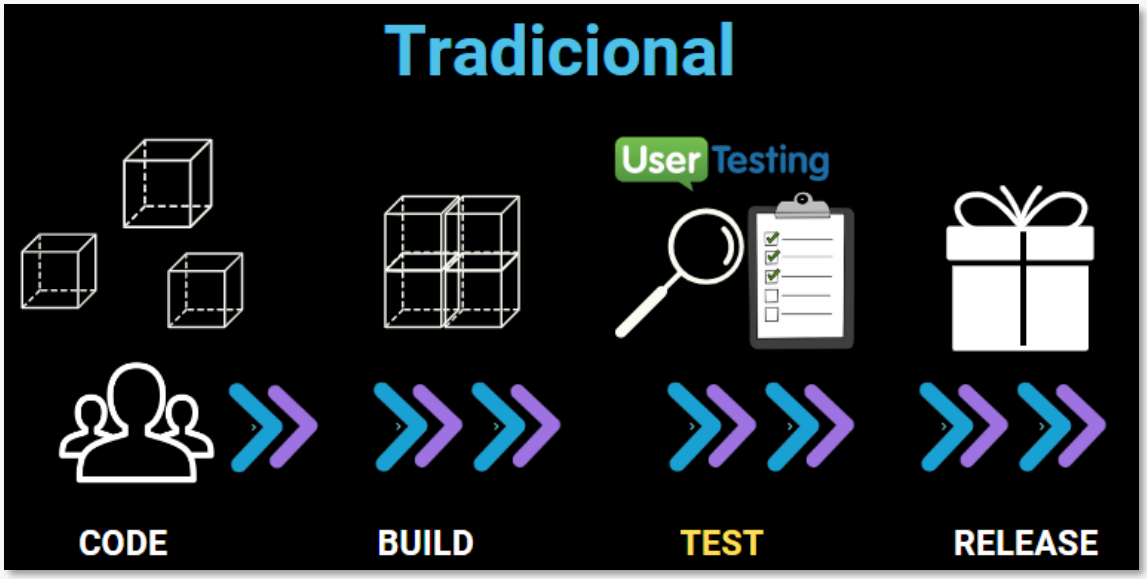
Shift Left

Este termo significa “jogar para esquerda”; é focar na prevenção de erros no início do SDLC através de inúmeros testes, mais próximo do desenvolvedor, por ser mais simples, rápido e mais barato para resolver os problemas que forem encontrados.

Basicamente é testar o mais cedo possível, testar muito e é mais prevenção do que detecção de erros.

No modelo tradicional de projetos cascata, a fase de testes era apenas uma e a responsabilidade dos testes era de uma equipe ou pessoa.

Figura 24 – Modelo tradicional SDLC – cascata.



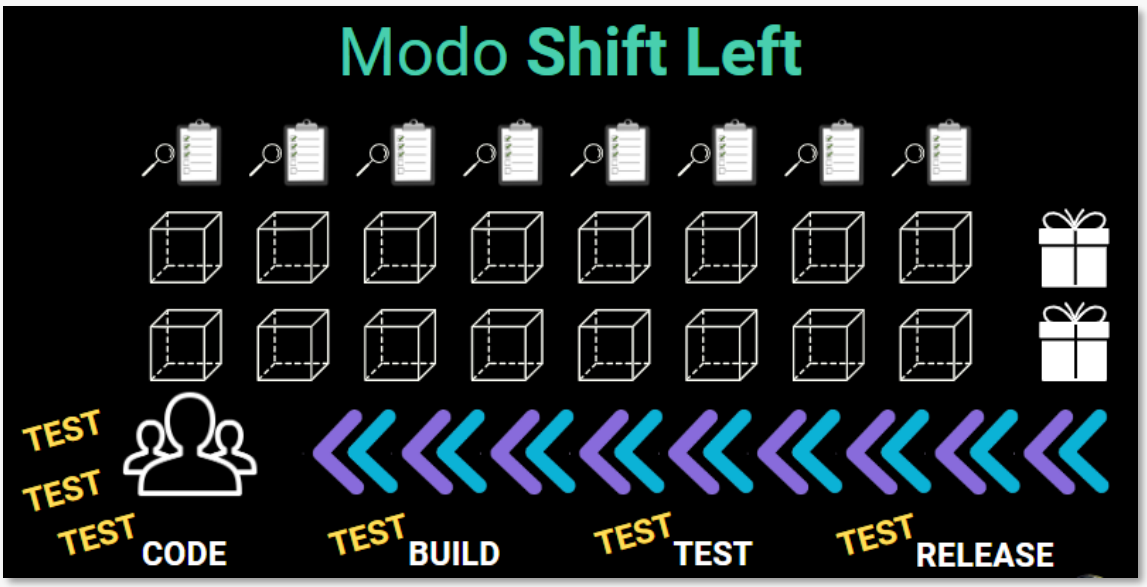
Fonte: elaborado pela autora.

No modelo *Shift Left* os testes estão principalmente no início do SDLC e em vários outros pontos, passando a ser responsabilidade de todos.

Figura 25 – *Shift Left.*

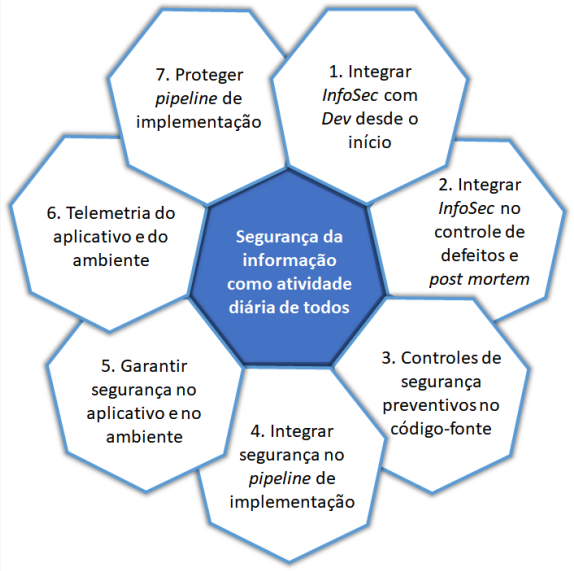
Fonte: elaborado pela autora.

Figura 26 – Modo *Shift Left.*



Fonte: elaborado pela autora.

Figura 27 – Sete maneiras para integrar melhor o time *InfoSec* na colaboração *DevOps*.



Fonte: adaptado de MUNIZ et al., 2020

Integrar *InfoSec* com *Dev* desde o início

Conhecida como conformidade por demonstração, é a maneira mais fácil de se prevenir que a segurança da informação seja um bloqueador justamente na entrega do projeto. Ela consiste em convidar os times de

segurança da informação para demonstrações do produto ao final de cada intervalo de desenvolvimento.

Desta maneira, fica claro para todos quais são as atividades da equipe em relação às metas da organização. É possível também avaliar as implementações durante o período e processo de criação, facilitando o entendimento por parte dos times de segurança, e com isso, podem oferecer soluções e informações para todos atingirem os objetivos de segurança e conformidade, em tempo hábil de se fazer qualquer tipo de correção.

Integrar *InfoSec* no controle de defeitos e *post-mortem*

As ferramentas utilizadas por *Dev* e *Ops* devem ser as mesmas para se registrar os problemas de segurança. Assim, o trabalho de segurança é compartilhado e permite a visibilidade de todos, o que ajuda na priorização de temas relativos ao fluxo de desenvolvimento.

As reuniões *blameless post-mortem* (pós-morte sem culpa) também devem ser aplicadas para os problemas de segurança, pois quando os problemas são compartilhados, aumentam a colaboração e a empatia entre os times.

Controles de segurança preventivos no código-fonte

Os repositórios compartilhados se tornam uma fonte rica de conhecimento coletivo de toda organização, bem como permitem reuso. Podem ser utilizados não só para código, mas para *pipelines* de implementação, padrões, *scripts* e segurança.

Um bom exemplo de bom uso dessa recomendação é ter bibliotecas de código e de recomendação de configurações nesse local, como ferramentas de criptografia ou autenticação de dois fatores. Seguindo sempre o cuidado de se ter bem definidas as responsabilidades de acesso e uso.

Com essas informações de segurança compartilhadas, tudo que é criado estará disponível, fácil de se encontrar e reutilizável, fazendo parte do trabalho diário de *Dev* e *Ops*. O mesmo ocorre para controle de versão: mantém todos a par das alterações que estão sendo realizadas.

Integrar segurança no *pipeline* de implementação

Há algum tempo, todos os problemas de segurança encontrados numa aplicação eram colocados num relatório com centenas de páginas (um arquivo PDF, por exemplo) e levados ao conhecimento de *Dev* e *Ops*, que poderia ser totalmente ignorado pela pressão de prazos ou problemas sendo encontrados muito tarde no SDLC para correção.

Ao implementar testes de segurança no *pipeline* de implementação se garante o *feedback rápido ao Dev* e ao *Ops*, o que permite atuação imediata no caso de vulnerabilidade.

Ferramentas que agilizam essas verificações: SonarQube, Veracode, Fortify, Gauntlet. Mais adiante serão detalhados o SonarQube e Fortify.

Garantir segurança no aplicativo e no ambiente

Muitas vezes, os testes de desenvolvimento focam na exatidão da funcionalidade, ou seja, somente naquilo que deve funcionar como pretendido, o “caminho feliz”. De outro lado, uma avaliação precisa e efetiva de times de qualidade, segurança e antifraude frequentemente focam nos “caminhos tristes”.

Por exemplo, se temos um site que venha a validar os dados de um cartão de crédito, é necessário incluir números inválidos para avaliar que eles sejam realmente rejeitados.

Idealmente, estes testes devem ser gerados de forma automática, seja como testes unitários ou funcionais, para minimizar riscos de erros em testes manuais.

Será apresentado com mais detalhes à frente sobre análise estática e dinâmica como mecanismos de segurança.

Garantir que os ambientes estejam seguros é outro fator muito importante para uma organização. Ferramentas podem ser utilizadas para garantir um estado de baixo risco sempre, verificando se há boas configurações e controles com monitoramento constante. Testes automatizados também podem ser criados para garantir que toda configuração esteja bem aplicada, como a segurança dos dados, tamanhos de chaves e assim por diante.

Telemetria

De nada adianta planejar e configurar ferramentas se não houver acompanhamento de como elas funcionam e analisar os resultados.

Os controles devem ser revistos e acompanhados diariamente para que as brechas de segurança sejam detectadas e sanadas rapidamente. Isso evita que a organização fique sabendo que seu aplicativo está com problemas somente quando recebe algum tipo de reclamação do cliente, ou seja, tardiamente.

Alguns controles são: monitoramento, existência de *logs* e alertas.

Telemetria de Segurança em aplicações

Segundo KIM et al. (2021), alguns exemplos de como criar telemetria para aplicações devem incluir:

- Sucesso e insucesso de *logins* de usuário.
- *Resets* de senha de usuário.

- *Resets* de endereço de *email* de usuário.
- Mudanças do cartão de crédito do usuário.

Esses indicadores podem detectar problemas no comportamento do usuário, evitando acessos indevidos e fraudulentos.

Figura 28 – *Login.*

Fonte: <https://www.freepik.com/free-photos-vectors/login>.

Telemetria de Segurança em ambientes

Deve-se identificar tentativas de acesso não autorizado nos ambientes, principalmente aqueles sobre os quais não temos controle, como os hospedados na *cloud*.

O Manual do *DevOps* (2021) indica alguns itens que devem ser monitorados e que tenham potencial alerta de:

- Mudanças de sistema operacional (ex.: em produção, em nossa infraestrutura).
- Mudanças de grupo de segurança.

- Mudanças em todas as configurações de produção (ex.: OSSEC, Puppet, Chef, Tripwire, Kubernetes, infraestrutura de rede, *middleware)*.
- Mudanças na infraestrutura de *cloud* (ex.: VPC, grupos de segurança, usuários e privilégios).
- Tentativas XSS (ex.: ataques *cross-site scripting*).
- Tentativas SQLi (ex.: ataques de SQL *injection*).
- Erros de servidor *web* (ex.: erros 4XX e 5XX).

Figura 29 – *Security.*

Fonte: <https://www.freepik.com/free-vector/cyber-security-concept_4520117.htm#query=security&position=43&from_view=search&track=sph> Image by rawpixel.com on Freepik.

Proteger *pipeline* de implantação

A infraestrutura que suporta todo o processo de CI/CD também serve de brecha para ataques. Caso alguém consiga penetrar nos servidores em que os *pipelines* de implantação rodam, podem acessar as credenciais dos controles de versão, permitindo acesso ao código-fonte, possibilitando o roubo ou a inclusão de código malicioso para atacar as aplicações e serviços.

Deve-se pensar também em proteger a integridade dos *pipelines* de implantação de ataques internos, pois os desenvolvedores podem inserir código malicioso que permita acessos indevidos. Por isso, são necessários testes de penetração para mitigar estes riscos, bem como testes de código, *code reviews*, entre outros.

Algumas estratégias incluem (KIM et al., 2021):

- Proteger os servidores que executam o processo de CI/CD e garantir que seja possível reproduzi-los de forma automatizada, prevenindo que eles sejam comprometidos.
- Revisar todas as mudanças introduzidas no controle de versões, tanto através de *pair programming* em tempo de commit ou processo de *code review* entre o *commit* e *merge* dentro da *branch* principal, para prevenir que os servidores de integração contínua rodem descontroladamente o código (por exemplo, testes unitários podem conter código malicioso que permite acesso não autorizado).
- Instrumentar o repositório para detectar quando teste de código possui chamadas suspeitas de API.
- Assegurar que todo processo de CI execute em seu container ou VM isolada e que ela seja recriada de uma conhecida, boa e verificada base de imagem no início de cada build.
- Assegurar que as credenciais de controle de versão usadas pelo sistema de CI são apenas leitura.

Mais sobre time *InfoSec*

Papel fundamental em qualquer organização, esse time assume a segurança através da gestão de identidades com seus controles de acessos, camadas de *firewalls*, *backups*, contingência de infraestrutura,

disponibilidade, estabilidade etc. Todos esses pontos garantem a produtividade e uma melhor experiência aos clientes.

A preocupação da segurança reside em ambos os lados da organização: fora e dentro. Ser impenetrável, resistente a ataques e ao mesmo tempo, impedir que haja vazamentos de dentro para fora, como informações ou códigos sigilosos e dados de clientes.

O termo *DevSecOps* é muito usado ultimamente, mas vale alertar que o termo original *DevOps* já engloba a segurança, suas iniciativas e práticas.

Segurança é responsabilidade de todos e deve-se automatizar o que for possível de maneira que passe a fazer parte da rotina diária, idealmente em *pipeline*s. Como visto nos módulos anteriores, ao se implementar ferramentas que deem *feedbacks*, os aprendizados são maiores, pois é possível corrigir os problemas nos estágios iniciais do desenvolvimento.

Leitura adicional sobre Segurança Contínua no link:

- https://learn.microsoft.com/pt-br/training/modules/explore-devops-continuous-security-operations/2-explore-continuous-security

Em resumo, é muito importante proteger a segurança:

- Do ambiente.
- Do código.
- Do *pipeline*.
- Do aplicativo.

Mecanismos de Segurança SAST, DAST, IAST, SCA, RASP

Há algum tempo, as organizações focavam em testar e consertar os problemas de segurança no final do SDLC, o que trazia custos e riscos muito elevados. Foi assim que elas passaram a implementar *shift-left* para alavancar seus negócios, introduzindo ferramentas nas fases iniciais do desenvolvimento, o que possibilita feedback e correção mais rápidos, ou quase instantâneos.

SAST (*Static Application Security Testing*)

A Análise Estática de Teste de Segurança ocorre no código-fonte, antes que ele seja compilado *(build)*, para descobrir os problemas durante o início do desenvolvimento. Ele também é conhecido como "teste caixa branca", onde não é necessário ter a aplicação executando em algum ambiente para testá-lo.

E o que significa compilar? No site Stack Overflow (2015), há uma boa explicação:

> Compilação é o ato / processo de traduzir um programa feito em uma linguagem de alto nível para uma linguagem de máquina, para que suas instruções sejam executadas pelo processador, ou seja, cria o executável de um programa escrito em uma linguagem de alto nível.

Como leitura adicional e continuação dessa explicação, consulte o site:

- https://pt.stackoverflow.com/questions/63193/o-que-significa-compilar.

As vulnerabilidades encontradas são de fácil localização, pois a ferramenta apresentará a localização exata do problema com o número da linha, nível de severidade e tipo de problema que ela pode causar se for enviada dessa maneira ao ambiente de produção.

E quando a ferramenta é implementada de forma automática dentro do processo de CI (*continuous integration)*, os ganhos são ainda

maiores porque as vulnerabilidades são rapidamente identificadas e corrigidas, de forma simples.

As principais vantagens do SAST, segundo o time de *AppSec* da *Zup Innovation* (2022), são:

- Auxilia na redução de riscos de segurança.
- Ganho de eficiência em *code review* de segurança.
- Maior assertividade na identificação de vulnerabilidades invisíveis para uma análise feita manualmente (por exemplo: *Buffer Overflow*).
- Auxilia no processo de desenvolvimento com informações detalhadas e recomendações.

E as principais limitações:

- Implementação em escala pode ser um desafio, devido à complexidade e particularidades de cada projeto.
- Incapacidade de identificar vulnerabilidades em outras camadas da aplicação, como nas bibliotecas de terceiros e integração com outros sistemas, uma vez que o *scan* SAST se restringe apenas ao código-fonte.
- É muito comum que ferramentas desse tipo apontem falsos positivos.

Antes de implementar uma estratégia de SAST nos *pipelines* de *build*, é importante que a equipe de *InfoSec* faça uma análise prévia para identificar possíveis falsos positivos. Isto evitará que os times de desenvolvimento a vejam como uma ferramenta que apresenta erros que não são exatamente vulnerabilidades. Cada organização, projeto e

tecnologia possui um contexto específico, por isso, devem ser analisados por essa perspectiva para não causar desgaste e desperdício de tempo dos desenvolvedores.

DAST (*Dynamic Application Security Testing*)

A Análise Dinâmica de segurança permite a validação da aplicação em tempo de execução e para isso, ela deve estar implantada em algum ambiente. Ela é complementar ao SAST e normalmente são feitos testes de penetração (*Pentest*), onde são simulados ataques e invasões. Também conhecida como “testes caixa preta”, a aplicação é testada do ponto de vista de um atacante (invasor).

Como os testes simulam esses ataques, é necessário ter um ambiente exclusivo, pois haverá alteração em bancos de dados e isso poderá afetar outros projetos que estejam sendo testados, caso o ambiente seja compartilhado. Esses ambientes normalmente são os de homologação, pré-produção e às vezes, até na produção.

A escolha de quais aplicações devem sofrer este tipo de teste dinâmico também é responsabilidade do time *InfoSec* juntamente com os times de sistemas, negócios, infraestrutura e operação.

Conforme a CIBERSECURITY (2020), as principais características das ferramentas DAST são:

- Trabalham apenas no aplicativo compilado.
- São completamente independentes do idioma usado para criar o aplicativo.
- Descobrem problemas relacionados a dados e configuração.
- Relatam menos falsos positivos do que as ferramentas SAST.

- Não é possível identificar a origem exata do problema (ou seja, a linha de código).

As vulnerabilidades identificadas pelos mecanismos DAST normalmente incluem (SENGUPTA, 2021):

- Vulnerabilidades de validação de dados/insumos.
- Ataques de injeção.
- Gestão de sessões quebradas.
- APIs inseguras.

IAST (*Interactive Application Security Testing*)

Normalmente, as ferramentas oferecidas no mercado oferecem os dois tipos de análises SAST e DAST e, se combinadas, podem trazer vantagens para aplicações *web* e microsserviços, foi assim que surgiu o IAST.

Os testes funcionais manuais ou automatizados são transformados em testes de segurança, criando inúmeras variantes de testes de segurança, que permitem detectar as vulnerabilidades. Ou seja, os agentes conseguem testar as aplicações em execução para análise do código e do comportamento delas, fazendo uma correlação com o fluxo de dados e a execução do código ao mesmo tempo, e assim fornecem a localização precisa das vulnerabilidades aos desenvolvedores. Isso traz mais exatidão e menos falsos positivos e coleta informações sobre performance, funcionalidade e bugs.

- Vantagem: é possível sua integração no *pipeline* CI/CD.
- Desvantagem: os agentes podem afetar o desempenho do aplicativo.

Leitura adicional em:

- https://www.nova8.com.br/2020/01/o-que-e-iast/

SCA (*Software Composition Analysis*)

A análise da composição do *software* consiste em validar as bibliotecas que compõem o projeto em busca de vulnerabilidades. No mundo atual, é muito comum que projetos se utilizem de bibliotecas de terceiros para economizar em códigos, ter mais agilidade de desenvolvimento e segurança. Em contrapartida, isto pode oferecer brechas de violação, podendo comprometer os projetos.

É aí que estas ferramentas podem ser úteis, pois também facilitam a detecção de versões de bibliotecas mais estáveis, evitando horas de refatoração de código.

RASP (*Runtime Application Self Protection*)

Trata-se de uma avaliação de aplicações *web* quanto ao seu comportamento e contexto, que consegue identificar entradas ou ameaças indevidas em tempo real. E, para isso, utiliza a própria aplicação para monitorar, detectar e mitigar os ataques de forma autônoma.

- Vantagem: pode ser integrado em todas as fases do SDLC.

Ao ser instalada no servidor, a plataforma RASP interceptará a comunicação entre o usuário e o aplicativo. E seu agente executará as validações necessárias, tanto da autenticação do usuário quanto de seus dados, dentro da aplicação.

Leitura adicional em:

- SAST, DAST, IAST, and RASP Differences (crashtest-security.com)

Reflexão sobre as ferramentas de segurança

Todas as ferramentas relatadas são complementares, nenhuma delas sozinha garante a segurança total de seus códigos e aplicações. Existem muitos outros fatores a serem considerados para uma estratégia de segurança ser bem-sucedida, como ter uma arquitetura muito bem definida, cultura de desenvolvimento seguro, profissionais preparados, cultura *Shift Left* e *Shift Right*, padrões estabelecidos e o elo mais fraco: as pessoas.

O próprio Manual do *DevOps* não traz nada específico sobre as ferramentas de segurança, mas formas de trabalhar, dicas e *cases* de como ampliar a segurança contínua.

A seguir, duas ferramentas como sugestão de implantar segurança e qualidade para o código.

SonarQube

Esta é uma ferramenta SAST de Qualidade de Código muito utilizada mundialmente. Ela é bem robusta e possui versão gratuita (*Community)* e outras com mais funcionalidades agregadas que são pagas, como a *Developer*, *Enterprise e Data center.* Para cada versão mais avançada aumenta-se o número de tecnologias contempladas para análise.

O lema dela é “código limpo” e executa um *scan* em todo o código-fonte em busca de 3 tipos de problemas: *bugs,* vulnerabilidades e *code smells*.

- *Bug*: é um erro que pode causar danos em sua aplicação.
- Vulnerabilidade: é um problema de segurança.
- *Code smell:* é um problema que afeta a manutenção do código, ele deixa o código cada vez mais complexo de entender, dificultando sua manutenção.

Traz também uma visão de quantidade de linhas duplicadas e expõe o resultado de testes unitários.

Sua grande vantagem é trazer os resultados em um *dashboard web*, onde o desenvolvedor conseguirá ver exatamente onde os problemas estão em seu código (linha exata), traz o motivo dele ser um problema e uma sugestão de solução.

Apresenta dados na linha do tempo das execuções (gráfico) e permite filtros variados para garantir diferentes visões da saúde do código.

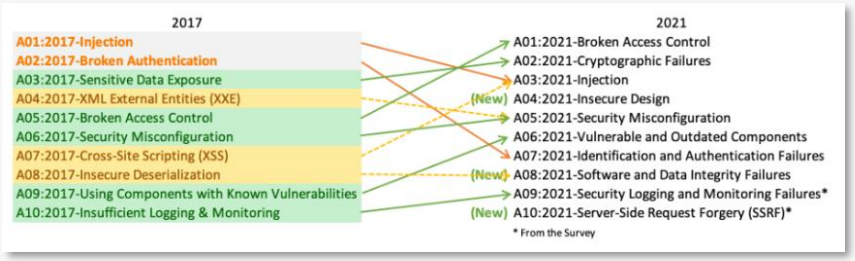
Ao ser integrado num *pipeline* de *build*, fará isso de forma automática, trazendo a segunda maneira do *DevOps* na prática: *feedback*.

É possível também definir regras para bloquear os casos que não atendam às melhores práticas de desenvolvimento, ajudando os desenvolvedores a aprenderem com suas falhas.

Na parte de segurança, contempla a verificação das principais vulnerabilidades, de acordo com a OWASP.

OWASP significa *Open Web Application Security Project* e "é uma organização internacional sem fins lucrativos que trabalha para melhorar a segurança do *software*." (OWASP, [s.d.]). É um projeto aberto de segurança de aplicações *web* e fornece de forma gratuita vários materiais, como artigos, documentação, metodologias e ferramentas.

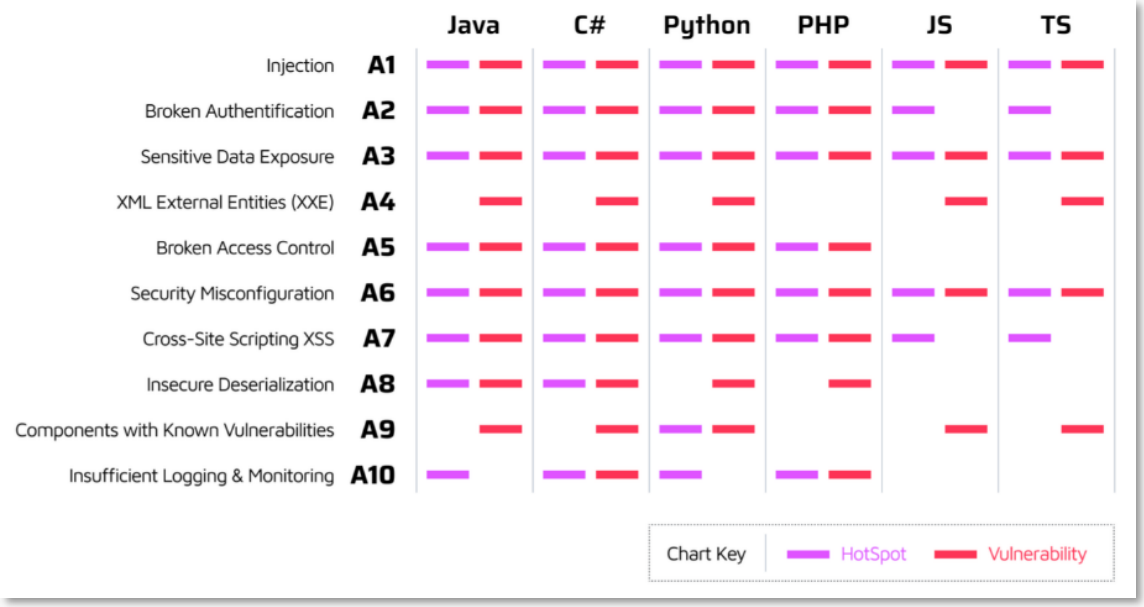
Figura 30 – O que mudou no Top 10 para 2021.



Fonte: OWASP Top 10 2021 < https://owasp.org/Top10/>.

O Sonar disponibiliza em seu site a relação de vulnerabilidades que estão no *Top* 10 da OWASP *versus* as tecnologias que ele efetua inspeção de código. Essa tabela ainda reflete os *Top* 10 de 2017.

Figura 31 - Ao levantar questões relacionadas ao OWASP Top 10 para os desenvolvedores no início do processo, o Sonar ajuda você a proteger seus sistemas, seus dados e seus usuários.



Fonte: Sonar < https://www.sonarsource.com/solutions/security/owasp/>.

Leitura adicional no link:

- O que é OWASP? O que é o OWASP Top 10? | Cloudflare

Fortify

Essa ferramenta oferece soluções SAST, DAST e SCA. Amplamente utilizada pelas organizações, cobre a segurança em APIs, aplicações *web*, aplicações *mobile*, IaC *(Infrastructure As Code)*, *containers* e IoT (*Internet of Things).*

Times *InfoSec* definem onde, como e quando cada solução deve ser aplicada. E envolvem times de Arquitetura, Engenharia, Infraestrutura e Operações para tomada de decisão.

Avalia uma grande variedade de tecnologias e os resultados são expostos aos desenvolvedores por severidade, para que possam focar nos problemas mais impactantes.

- SAST: pode ser integrada aos *pipelines* de *build* de todos os ambientes da organização.
- DAST: deve ser definido onde rodar, normalmente não se integra aos *pipelines* e é executado em ambiente separado dos utilizados pelos demais times para não os impactar, uma vez que simula os ataques e ocorrem alterações em bancos de dados.
- SCA: pode ser integrado aos *pipelines* do *DevOps*.

Leitura adicional no link:

- https://www.youtube.com/c/FortifyUnplugged/videos

> Capítulo 6

Capítulo 6. Processos

Gestão da Configuração

Trata-se de ter processos de controle que facilitam a manutenção do *software.* Como os códigos passam a ser o ativo mais importante de uma organização e eles estão em constante alteração por vários desenvolvedores e equipes, cometer um erro é mais fácil do que se imagina.

Para que isto não ocorra, a Gerência de Configuração do *Software* deve criar mecanismos que o protejam de erros durante seu ciclo de vida, permitindo ganho de produtividade, eficiência, menos retrabalho, rastreabilidade das alterações, histórico, estabilidade do ambiente.

Os problemas mais comuns são perda de versão, sobreposição de versão, falta de integração entre versões de *Devs* distintos.

Uma boa gestão das configurações permitirá que novas funcionalidades sejam desenvolvidas em paralelo, totalmente independente das alterações de outros *Devs*.

A estrutura de como esses códigos e projetos são armazenados também merece atenção, pois padronização também ajudará na consolidação dos dados, como um índice de livro.

Segundo a DEVMEDIA (2023), a perspectiva gerencial é dividida em "cinco funções, que são: identificação da configuração, controle da configuração, contabilização da situação da configuração, auditoria da configuração e gerenciamento de liberação e entrega."

E, continuando, também define na perspectiva do desenvolvimento, que "é dividida em três sistemas principais: controle de modificações, controle de versões e gerenciamento de construção".

Leitura adicional em:

- https://en.wikipedia.org/wiki/Branching_(version_control)
- https://www.atlassian.com/agile/software-development/git

Gestão de Mudanças (GEMUD)

Para que as alterações dos *Devs* possam seguir seu caminho para a implantação do *software*, a maioria das empresas inclui controles e processos de aprovação antes que alguma entrega seja implantada no ambiente de produção para os clientes.

À medida que as entregas geram problemas e incidentes, criam-se cada vez mais processos, aprovações e geração de evidências, no intuito de aumentar a segurança e reduzir os riscos. Isto aumenta o tempo de entrega e gera uma sensação de confiança.

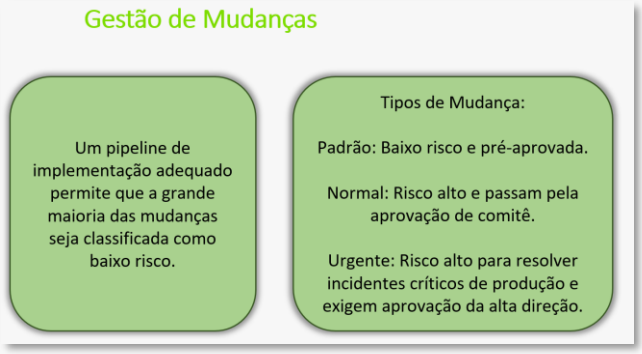
Figura 32 – Resumo do processo tradicional e com DevOps.

Fonte: adaptado de MUNIZ et al., 2020.

Tipos de mudanças comum nas empresas:

- Padrão: baixo risco e pré-aprovada.
- Normal: possuem risco alto e passam pela aprovação de comitê.
- Urgente: risco alto para resolver incidentes críticos em produção e exigem aprovação da alta direção.

Figura 33 – Categorias de mudanças.



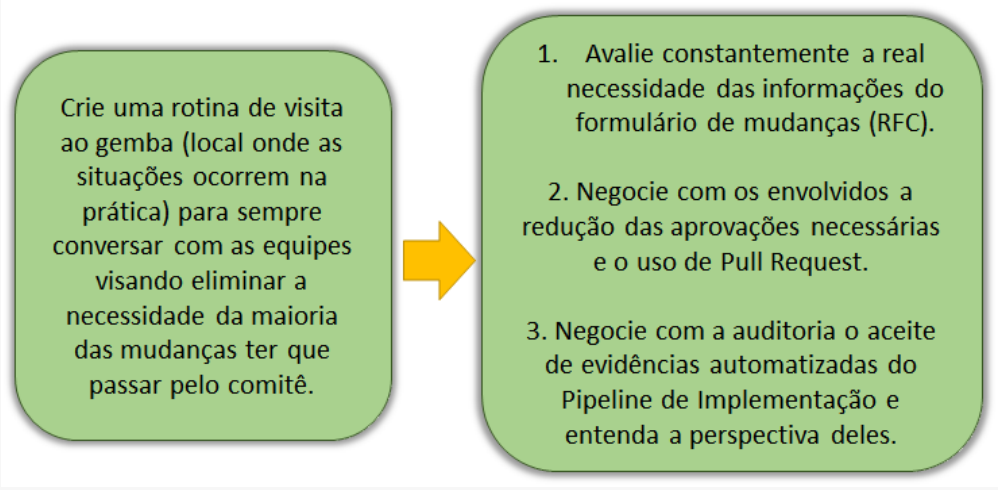
Fonte: adaptado de MUNIZ et al., 2020.

Com as práticas *DevOps* adequadas, o *pipeline* de implantação permitirá que a maioria das mudanças sejam do tipo baixo risco.

Quando temos *pipelines* automatizados implantados de forma correta, estas aprovações se tornam desnecessárias e o processo flui de maneira fluida e segura, trazendo mais qualidade e confiança nas entregas. Importante também monitorar proativamente o *software* e os serviços em produção para uma reação rápida, se necessário.

É responsabilidade dos gerentes de liberação pensar em como simplificar e automatizar continuamente os processos.

Figura 34 – Simplificando a Gestão de Mudanças



Fonte: adaptado de MUNIZ et al., 2020.

Transparência no fluxo de valor:

- Listar as mudanças dos últimos três meses.
- Divulgar a lista de todos os problemas que ocorreram nessas mudanças.
- Compartilhar os indicadores das mudanças (ex.: tempo médio para reparar a mudança, tempo médio entre as falhas).
- Demonstrar o controle do ambiente: onde ocorrem os testes automatizados e as implantações.
- Demonstrar o controle dos erros e como são corrigidos (processo). Se existirem ferramentas de automação, devem ser demonstradas, bem como os controles de acessos existentes.

Simplificar e automatizar as atividades de solicitação e gestão de mudanças aumenta a eficiência, mas também aumenta a confiança no processo *DevOps* para todos da organização e, principalmente, os controladores de *compliance*, auditores, gestão de mudanças e stakeholders.

> Capítulo 7

Capítulo 7. AIOPs

Aplicação da Inteligência Artificial na Operação

Com o aumento de informações de forma exponencial, as organizações podem facilmente perder a capacidade de controlar e monitorar esse volume manualmente, o que pode ser prejudicial para seu controle e previsibilidade de problemas. Uma solução para isto é automatizar e simplificar qualquer atividade, utilizando ferramentas para controle dos seus ativos.

Segundo a Gartner, "*AIOps* combina *big data* e *machine learning* para automatizar processos de operações de TI, incluindo correlação de eventos, detecção de anomalias e determinação de causalidade" (2023).

Com isso, é possível agilizar a identificação de problemas, ter automação para corrigi-los imediatamente *(self-healing)* e aumentar a capacidade de prevenção de anomalias (identificação antes que ocorram), permitindo que os sistemas continuem ativos sem interrupção, melhorando a jornada do usuário em seus aplicativos.

As soluções *AIOps* facilitam a gestão e tomada de decisão de líderes e profissionais de operações de TI, o que lhes permite mais tempo para focar em atividades mais estratégicas.

Dada a complexidade de se gerenciar as infraestruturas na visão de ameaças, falhas, cargas, integrações, serviços de terceiros, *IoT (Internet of Things)*, espalhados em ambientes físicos, híbridos ou nuvem, é necessário ter ferramentas de IA para ajudar na automação, o que agiliza a análise que seria feita de maneira manual.

Figura 35 – *AIOps*

Fonte: https://www.freepik.com/free-photo/robot-hand-finger-ai-background-technology-graphics_17851034.htm#query=ai&from_query=aiops&position=37&from_view=search&track=sph">Image by rawpixel.com on Freepik.

A Dynatrace descreve, em “A Promessa da IA”, que *AIOps* é a capacidade de “possibilitar operações autônomas, impulsionar a inovação e oferecer novos modos de envolvimento do cliente, automatizando tudo” (DYNATRACE, 2019, p. 3).

É importante ressaltar a complexidade de identificar com precisão a causa de anomalias e problemas no ambiente de produção. Um exemplo dado pela Dynatrace sobre essa complexidade: como detectar a causa-raiz exata de uma aplicação com microsserviços gerando milhares de alertas globalmente? As suas dependências são fatores que dificultam a identificação exata, pois geram uma explosão de alertas e o monitoramento convencional não ajudaria muito.

Quando se tem a inteligência embutida nos sistemas que permitem a rastreabilidade, esse trabalho se torna mais fácil e confiável, tanto que permitirá até uma autorremediação, otimizando o trabalho das equipes de operações.

Elementos

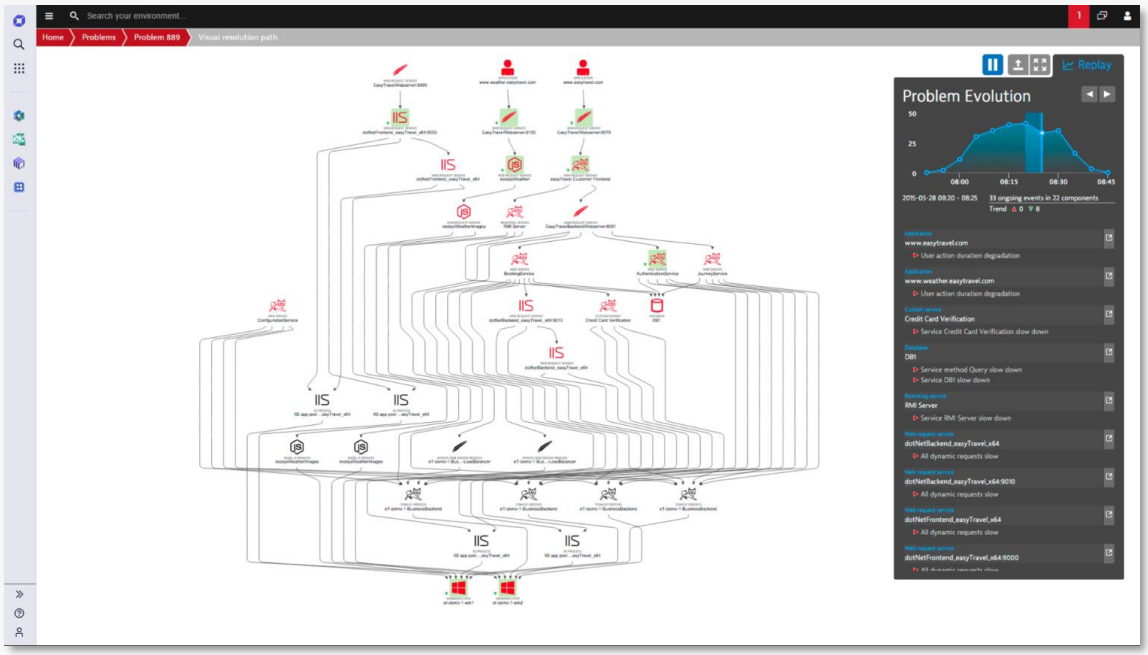
De acordo com MUNIZ et al. (2020) apud OEHRLICH (2019), uma solução de *AIOps* deve possuir determinadas funcionalidades:

- Capacidade de consumir dados de um conjunto de tipos e fontes diferentes.
- Capacidade de analisar dados em tempo real e a qualquer momento depois (dados históricos).
- Capacidade de ativar o armazenamento de dados para acesso a qualquer momento.
- Capacidade de permitir acesso aos dados de forma segura e em qualquer momento, com base no cargo ou função na empresa.
- Capacidade de aproveitar o aprendizado de máquina *(machine learning)* para analisar dados gerados por máquinas e seres humanos e utilizá-lo para aprendizagem e fornecimento de análises.
- Capacidade de aproveitar as análises para automação e ação de proatividade.
- Capacidade de apresentar as análises em contexto para a pessoa ou equipe funcional.

É necessário que as plataformas de *AIOps* sejam construídas para coletar os dados em tempo real, de maneira abrangente, completa e inteligente. Se bem determinadas e construídas, trarão informações precisas sobre a rota das causas, tornando a rastreabilidade mais simples e direta.

A seguir, uma imagem de um relatório da IA da Dynatrace chamada Davis, analisando bilhões de dependências em milissegundos. Essa IA também analisa e descobre mudanças nos ambientes com mapeamento automático.

Figura 36 – Ferramenta Davis da Dynatrace



Fonte: <https://www.dynatrace.com/platform/aiops/>.

Algumas ferramentas utilizadas no mercado para monitoramento de Operações:

- Dynatrace
- Datadog
- Zabbix

Referências

Appendix: Forms and Templates - The Toyota Kata Practice Guide: Developing Scientific Thinking Skills for Superior Results-in 20 Minutes a Day [Book]. Disponível em: <https://www.oreilly.com/library/view/the-toyota-kata/9781259861031/appendix.xhtml>. Acesso em: 22 fev. 2023.

ATLASSIAN. The importance of an incident postmortem process | Atlassian. Disponível em: <https://www.atlassian.com/incident-management/postmortem>.

Awesome Devops Wallpapers - WallpaperAccess. Disponível em: <https://wallpaperaccess.com/devops#google_vignette>. Acesso em: 22 fev. 2023.

AWS Well-Architected Framework - Game Days. Disponível em: <https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.concept.gameday.en.html>. 22 fev. 2023.

BEVAN, P. ArchOps. Disponível em: <https://www.bloorresearch.com/technology/archops/?cn-reloaded=1>. Acesso em: 22 fev. 2023.

CARRINGTON, B.; LEAPER, L. The Improvement Kata - A four step platform for scientific thinking. Disponível em: <https://www.lean.org/the-lean-post/articles/a-video-primer-on-improvement-and-coaching-kata/>. Acesso em: 22 fev. 2023.

CIBERSECURITY, N. (ED.). O que é IAST? Conhecimento Especializado, 17 jan. 2020. Disponível em: <https://www.nova8.com.br/2020/01/o-que-e-iast/>. Acesso em: 22 fev. 2023.

CLOUDFLARE. O que é OWASP? O que é o OWASP Top 10? Disponível em: <https://www.cloudflare.com/pt-br/learning/security/threats/owasp-top-10/>. Acesso em: 22 fev. 2023.

CORDEIRO, D. ISO/IEC 27001 – Do GAP Analysis à Certificação (Overview) – pt1 (Defensec, Ed.) Defensec - O seu Blog de CyberSegurança, 14 dez. 2020. Disponível em: <https://defensec.com.br/iso-iec-27001-do-gap-analysis-a-certificacao-overview-pt1/>. Acesso em: 22 fev. 2023.

DALY, L. Git makes software development, well, easier. Disponível em: <https://www.atlassian.com/agile/software-development/git>. Acesso em: 22 fev. 2023.

DEVMEDIA. Gerência de Configuração de Software. Disponível em: <https://www.devmedia.com.br/gerencia-de-configuracao-de-software/9145>. Acesso em: 22 fev. 2023.

DWECK, Carol. Mindset: A nova psicologia do sucesso. 1a. ed. Rio de Janeiro: SCHWARCZ S.A., 2016. ISBN 978-85-438-0824-6 p.174-p.190.

DYNATRACE. AIOps Done Right - Automating the Next Generation of Enterprise Software. [s.l.] Dynatrace LLC, 2019. Disponível em < https://www.dynatrace.com/resources/ebooks/aiops-done-right/>. Acesso em: 22 fev. 2023.

DYNATRACE. Artificial Intelligence for IT Operations (AIOps). Disponível em: <https://www.dynatrace.com/platform/aiops/>. Acesso em: 22 fev. 2023.

Free Project Post-mortem Templates | Smartsheet. Disponível em: <https://www.smartsheet.com/content/project-post-mortem-templates>. Acesso em: 22 fev. 2023.

Google - Site Reliability Engineering. Disponível em: <https://sre.google/sre-book/example-postmortem/>. Acesso em: 22 fev. 2023.

GRIPP, A. Retrospectiva ... Você sabe fazer uma reunião dessas? Disponível em: <https://annelisegripp.com.br/retrospectivas-ageis/>. Acesso em: 22 fev. 2023.

Home - Chaos Monkey. Disponível em: <https://netflix.github.io/chaosmonkey>. Acesso em: 22 fev. 2023.

HUBOT. Disponível em: <https://hubot.github.com/>. Acesso em: 22 fev. 2023.

INNOVATION, Z. (ED.). Ferramentas SSDLC: SAST, DAST e SCA - Segurança da Informação, 1 dez. 2022. Disponível em: <https://www.zup.com.br/blog/ferramentas-ssdlc>. Acesso em: 22 fev. 2023.

ISMAIL, S.; MALONE, M.; GEEST, Y. Organizações Exponenciais. Tradução: Gerson Yamagami. 1a. ed. Barueri, SP: HSM do Brasil S.A., 2015.

KIM, G. et al. The DevOps Handbook: how to create world-class agility, reliability, and security in technology organizations. 2a. ed. Portland: IT Revolution Press, 2021.

LUNNEY, J.; LUEDER, S.; O'CONNOR, G. Postmortem Culture: Learning from Failure. In: Google SRE. Sebastopol, CA 95472, United States: O'Reilly Media, Inc., 2017.

MASTERSON, V. What is quiet quitting? Disponível em: <https://www.weforum.org/agenda/2022/09/tiktok-quiet-quitting-explained/?DAG=3&gclid=Cj0KCQiAt66eBhCnARIsAKf3ZNFLT0AO9T4wb1b97viezi5T6Gec8kRHO_H9GJ0pyojw5enhiYMx-OYaAg6JEALw_wcB>. Acesso em: 22 fev. 2023.

MUNIZ, A. et al. Jornada DevOps 2ª edição. Rio de Janeiro: BRASPORT Livros e Multimídia Ltda., 2020.

MUNIZ, A.; IRIGOYEN, A. Jornada Ágil e Digital. Rio de Janeiro: BRASPORT Livros e Multimídia Ltda., 2019.

OWASP. OWASP foundation, the open source foundation for application security. Disponível em: <https://owasp.org/>. Acesso em: 22 fev. 2023.

OWASP. OWASP Top 10:2021. Disponível em: <https://owasp.org/Top10/>. Acesso em: 22 fev. 2023.

SENGUPTA, S. SAST, DAST, IAST, RASP Explained in Short. Disponível em: <https://crashtest-security.com/sast-dast-iast-rasp/#static-application-security-testing-sast>. Acesso em: 22 fev. 2023.

SONAR. OWASP top 10 - we've got you covered! Disponível em: <https://www.sonarsource.com/solutions/security/owasp/>. Acesso em: 22 fev. 2023.

Terminologia - O que significa compilar? Disponível em: <https://pt.stackoverflow.com/questions/63193/o-que-significa-compilar>. Acesso em: 22 fev. 2023.

ZENDESK, E. O que é ciclo PDCA? Quais são suas etapas? Como aplicar? O que é ciclo PDCA? Quais são suas etapas? Como aplicar?, 13 dez. 2022. Disponível em: <https://www.zendesk.com.br/blog/ciclo-pdca-o-que-e/>. Acesso em: 22 fev. 2023.