A Terceira Maneira: Os Princípios da Aprendizagem e Experimentação

Bootcamp: Profissional DevOps

Analia Irigoyen

2021

A Terceira Maneira: Os Princípios da Aprendizagem e Experimentação

Bootcamp: Profissional DevOps

Analia Irigoyen

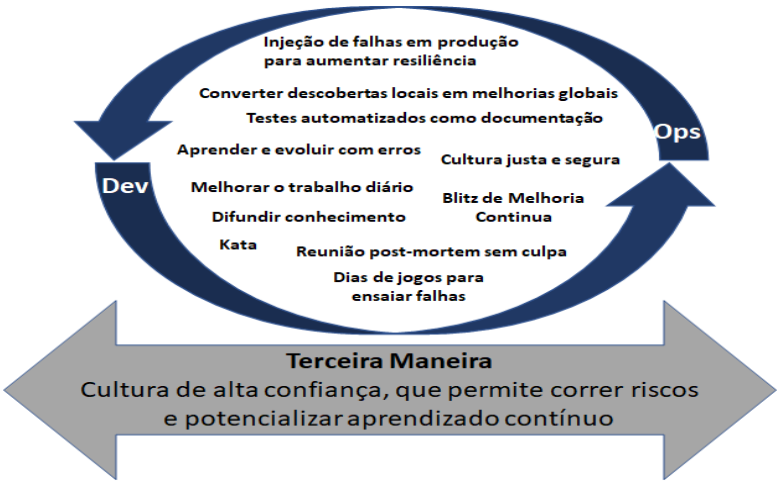
Sumário

Capítulo 1. Introdução a Terceira Maneira do DevOps e a Segurança e Gestão de Mudanças

Introdução

A terceira maneira tem como objetivo principal criar uma cultura de confiança para que seja possível errar e aprender com os erros. É importante correr riscos para potencializar um aprendizado contínuo. A terceira maneira é fundamental para que a cultura DevOps esteja presente e sempre acesa nos times. Sem ela, a cultura de automação e colaboração vai se enfraquecendo ao longo do tempo. Temos que nos habituar e, consequentemente, habituar os times em que trabalhamos a melhorar continuamente até a perfeição, e como a perfeição não existe, a melhoria contínua não terá fim.

Figura 1 – Terceira Maneira.



Fonte: adaptado de (MUNIZ, 2019).

Conforme a figura acima, alguns dos principais princípios e práticas da terceira maneira são:

- **Aprendizado com falhas**: o foco é desenvolver aprendizado contínuo nas equipes e evitar a cultura de culpa e punição. A Netflix (Muniz, 2019) criou um processo de injeção de falhas chamado Chaos Monkey, que aumenta a resiliência.

- **Cultura justa**: existem algumas importantes dicas para promover esta cultura justa. São elas: nunca nomear ou envergonhar quem é o culpado da falha, incentivar quem compartilha problemas do sistema, criar confiança para que a equipe aprenda com problemas e lições aprendidas sem culpa, e injeção controlada de falhas em produção.

- **Reunião post-mortem**: após a solução de incidentes, o objetivo é entender e divulgar as ações que deram certo, que podem ser aprimoradas, assim como erros e medidas para evitar recorrência. É muito importante publicar e divulgar o resultado da reunião.

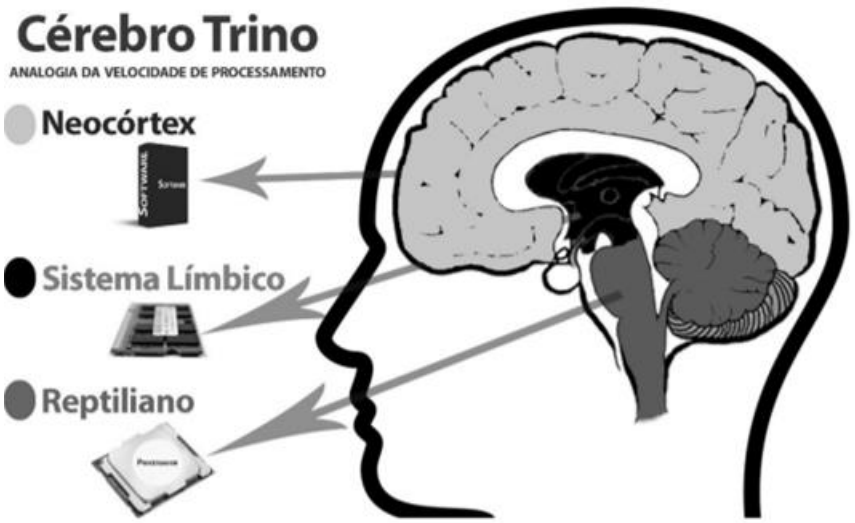
- **Dias de Jogos**: esse conceito vem da engenharia da resiliência, cujo objetivo é criar exercícios programados para aumentar a resiliência através da injeção de falhas de grande escala em sistemas críticos.

- **Divulgar e compartilhar aprendizados (descobertas) entre os times**: empresas de alto desempenho obtém os mesmos resultados (ou melhores) refinando as operações diárias, introduzindo tensão continuamente para aumentar o desempenho e, assim, gerando mais resiliência nos seus sistemas.

Motivação

Segundo o Livro Ágil e Digital (2020), em 1970, Paul McLean (1990), psiquiatra e neurocientista americano, apresentou uma teoria chamada Teoria do Cérebro Trino, que mostra pela primeira vez nosso cérebro como uma estrutura dividida em três partes distintas: Neocórtex, Sistema Límbico e Cérebro Reptiliano (Figura 2).

Figura 2 – Representação do Cérebro Trino.



Fonte: Livro *Jornada Ágil e Digital*, Muniz e Irigoyen (2019).

O sistema Neocórtex tem como principal responsabilidade o raciocínio lógico, é devido a isso que somos capazes de ler, escrever e nos comunicar.

O sistema Límbico, que é conhecido como cérebro emocional, é o responsável por nossas emoções, pelo aprendizado, pela memória e pelo nosso comportamento social.

Como é possível observar pela figura, o nosso sistema reptiliano é o primeiro a agir, é o nosso instinto quando estamos em alguma situação de perigo ou sob pressão; nossa primeira reação é a defensiva: agir sem pensar somente para nos defender.

Isso significa que, pela nossa natureza humana e por primeira reação:

- Somos, por natureza, avesso às mudanças.
- Aprendizado gera desconforto no início, sair da zona de conforto é sempre um desafio.
- O seu cérebro vai criar mil razões para impedi-lo de mudar os hábitos.

- A sensação de prazer ao conseguir melhorar com fatos e dados não tem preço: pare, reflita e mude. Use mais o límbico e o néocortex para “treinar” o cérebro a aprender.

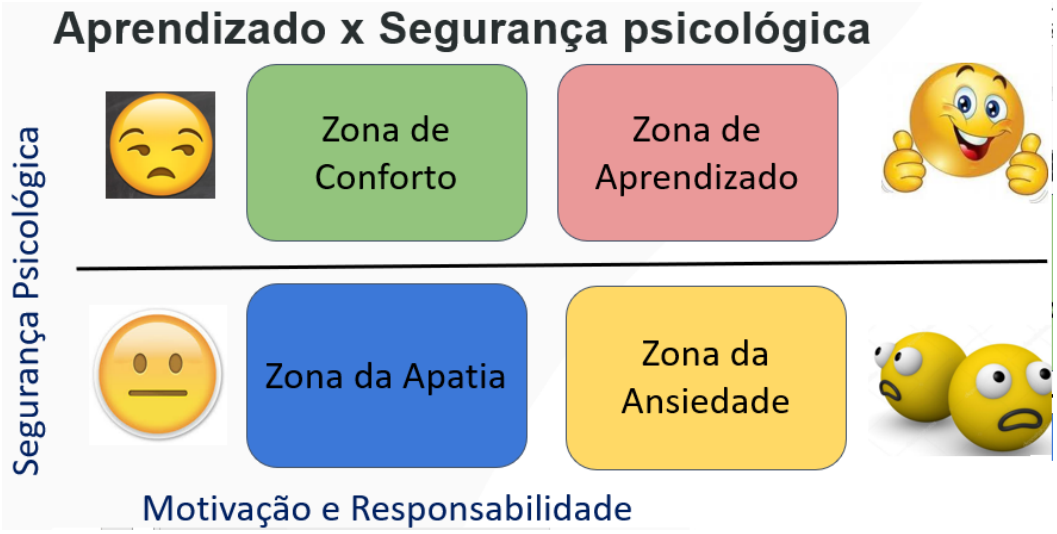
A importância da Terceira Maneira

Para que seja possível criar aprendizado e implantar o loop de feedback contínuo, as práticas da Terceira Maneira são imprescindíveis para a manutenção da colaboração e da automação. Sem tempo para melhoria contínua e aprendizado, os times acabam não evoluindo na primeira e na segunda maneira.

Para que esse aprendizado seja possível, precisamos correr riscos, ter uma cultura justa e buscar fatos e dados para tomar qualquer tipo de decisão.

Para que possamos correr riscos, aprender e não ativar nosso lado “reptiliano”, precisamos de segurança psicológica. Conforme ilustrado na Figura 3, segurança psicológica e responsabilidades andam juntos para que o aprendizado aconteça.

Figura 3 – Aprendizado x Segurança Psicológica.



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

Segundo Muniz e Irigoyen no Livro Ágil e Digital (2020), os indivíduos podem estar inseridos em quatro zonas:

- **Zona de Conforto**: o funcionário está com o nível de Segurança Psicológica excelente, porém sua motivação e responsabilidade deixam a desejar, o tornando assim um trabalhador raso.

- **Zona de Apatia**: os níveis de segurança psicológica estão baixos, a responsabilidade e a motivação estão aquém do desejável, tornando assim o indivíduo triste, desmotivado e sem ânimo para inovar e desenvolver suas atividades.

- **Zona de Ansiedade:** o indivíduo possui bastante Motivação e Responsabilidade, quer inovar, implantar novas práticas. No entanto, a segurança psicológica do ambiente de trabalho dele é bastante negativa, tornando o funcionário ansioso e sem força para realizar tudo o que realmente deseja.

- **Zona de Aprendizado**: é considerada o melhor cenário, visto que o indivíduo se sente seguro, feliz, capaz de inovar, consegue arriscar sabendo que caso aconteça qualquer problema ele não será criticado, humilhado e ridicularizado no seu ambiente de trabalho, porque as falhas serão vistas como oportunidades de aprendizado. Consegue, assim desenvolver cada vez mais o seu trabalho.

Antes de iniciar a implementação das práticas da terceira maneira, se assegure que a organização em que você trabalha tenha um ambiente propício para a aprendizagem.

A terceira maneira e o profissional DevOps do tipo T-shaped

Além da terceira maneira impulsionar as organizações a ter ambientes seguros e uma cultura justa, estas práticas permitem que os profissionais DevOps envolvido nelas se tornem profissionais do tipo “T-Shaped”.

Segundo Andy Boynton e William Bole destacaram em artigo da Forbes em 2011, não há nada de errado em ser um profissional em forma de I, desde que você também possa ser um T em algum sentido significativo. A maioria dos profissionais tem uma área de especialização, porém é mais provável que enriqueçam suas ideias se tiverem um pé fora de seu mundo habitual.

> Como é esperado atualmente que os profissionais tenham pensamento disruptivo e foco em experimentação e adaptação, a jornada Ágil e Digital depende de profissionais generalistas, que estejam dispostos a pensar e agir fora de seus próprios silos com conhecimento profundo e habilidades ampliadas. (MUNIZ, 2019)

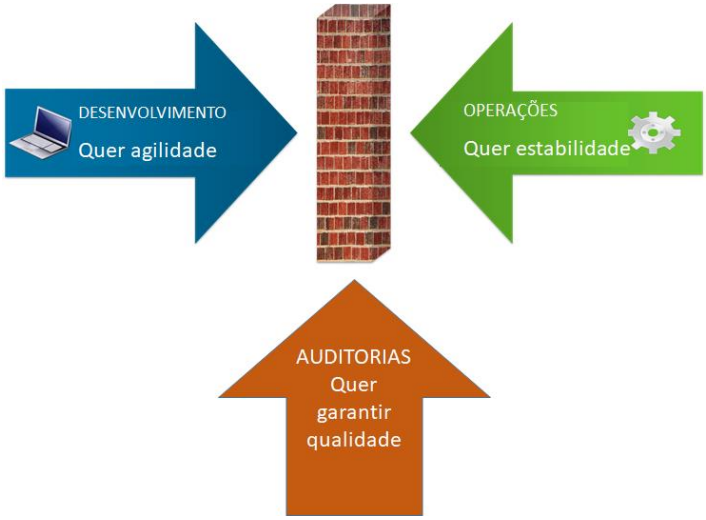
As práticas de aprendizagem multidisciplinares da terceira maneira tornam cada vez mais o profissional DevOps especialista-generalista, conhecendo e tendo experiência em diversas áreas e se especializando nas áreas com as quais se identifica mais.

Introdução à segurança e gestão de mudanças

- **DevOps é para todos.**

A Figura 4 representa que objetivos diferentes - tanto de desenvolvimento, quanto de operações, quanto das áreas de compliance - podem, em primeira mão, ser entendidos como um obstáculo ao DevOps.

Figura 4 – DevOps é para todos.

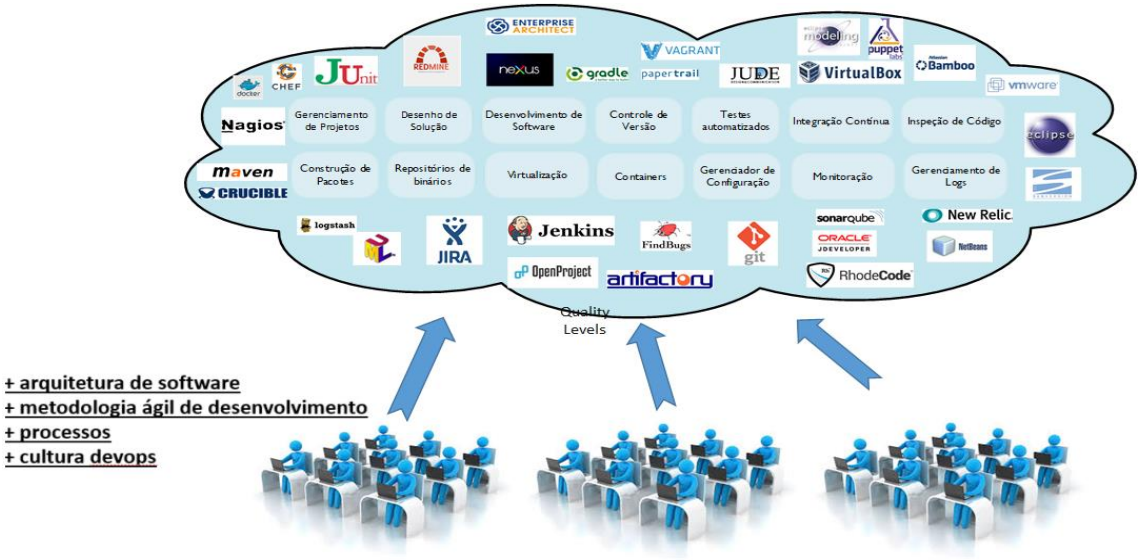


No entanto, é exatamente ao contrário. Com as automações temos diversos benefícios, dentre eles:

- Melhor organização e colaboração entre os profissionais: com a automação cria-se uma cultura de interdependência durante a realização das atividades.
- Definição de níveis de acesso: com a automação cada profissional e cada equipe consegue operar dentro de determinadas atividades, o que evita confusões e ruídos de comunicação, por exemplo.
- Maior controle de horas de atividade: com a automação, as métricas são criadas de forma automática, gerando mais confiabilidade para melhorias e análises de causa-raiz.
- Redução de custos e aumento de produtividade: com a automação de processos repetitivos, tem-se mais produtividade, qualidade e menos retrabalho. Os times ficam focados e motivados por trabalhar em tarefas que exigem mais criatividade. Além de ferramentas específicas, que detectam vulnerabilidades de segurança e falta de qualidade mínima do código.
- Gerenciamento mais inteligente: o gestor consegue criar relatório gerenciais em poucos cliques para acompanhar os processos, o que facilita a tomada de decisões em tempo hábil.

Que auditor ou responsável por segurança da informação não será agradado por tantos benefícios e pelo aumento da qualidade e da produtividade, com evidências e logs automáticos de todas as atividades realizadas, com pouca intervenção humana? Conforme a Figura 5, para orquestrar (integrar) todas estas ferramentas, é necessário que se exista um bom processos automatizado e muito conhecimento de: Arquitetura, Gestão, Qualidade, Engenharia de Software, Serviços e Inovação.

Figura 5 – DevOps e Engenharia de Software.



A importância da terceira maneira

- **Um exemplo de processo definido por trás do DevOps.**

Abaixo, na Figura 6, está representado um processo definido DevOps e os passos para a sua adoção.

É possível perceber que existe um planejamento não só em relação a que ferramentas vão ser utilizadas, mas a necessidade das integrações entre elas. Por trás deste exemplo existe um processo e um fluxo automatizado, bem como: muito treinamento, o acompanhamento de um projeto piloto e as adequações necessárias a cada tipo de produto da organização.

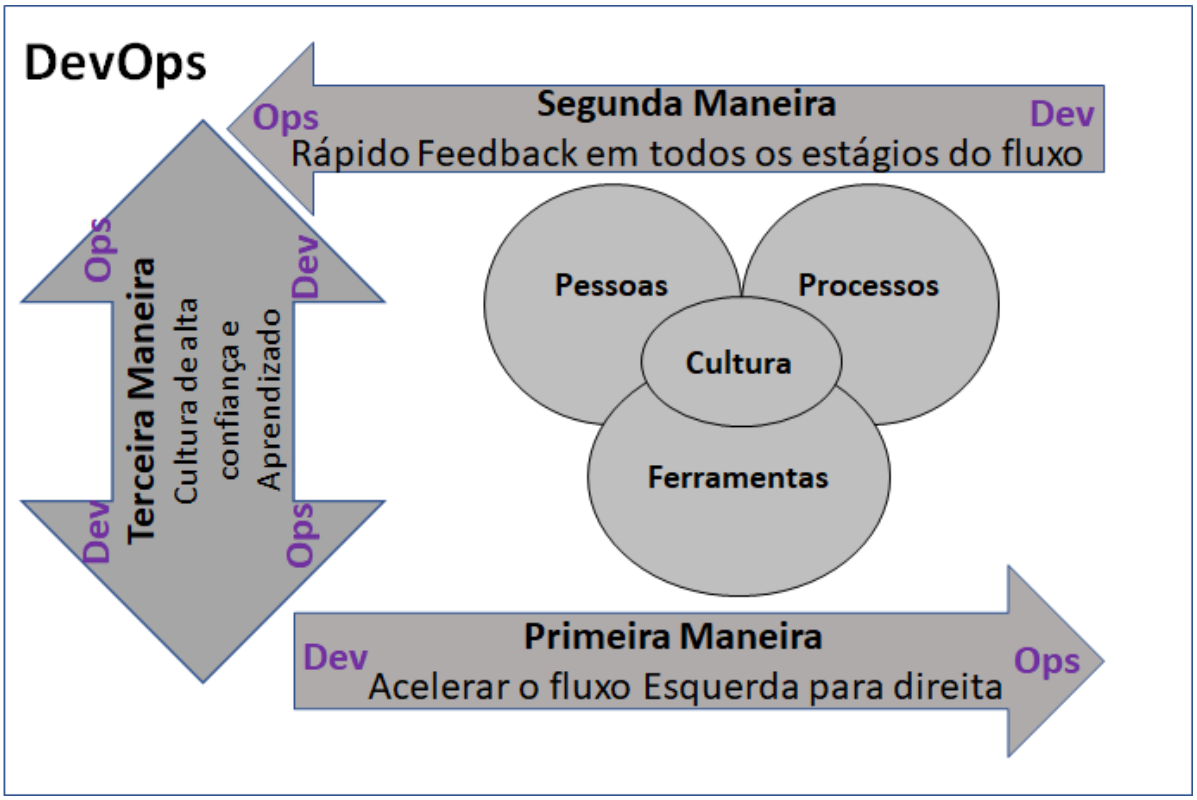
Nos próximos capítulos veremos como o DevOps pode garantir segurança desde o início do ciclo de desenvolvimento e uma maior quantidade de mudanças de baixo risco.

Figura 6 – Exemplo de processo com DevOps.

Cada implementação de pacote envolve:

1. Planejamento da melhor solução para o cliente
2. Implementação das ferramentas
3. Treinamento da cultura DevOps
4. Treinamento da equipe no uso das ferramentas
5. Treinamento de arquitetura de software orientado para as ferramentas
6. Acompanhamento de um projeto piloto
7. Adequação do processo da empresa de forma a levar em consideração as ferramentas em todo o processo de desenvolvimento

Figura 7 – DevOps e as três maneiras.



Alvin Toffler, em uma das suas frases mais conhecidas, diz que “os analfabetos do futuro não serão os que não sabem ler ou escrever, mas os que não sabem aprender, desaprender e reaprender”. Não é difícil entender como criar uma cultura de experimentação e aprendizado é imprescindível para qualquer empresa.

Uma das formas de estimular este aprendizado é utilizar os erros que ocorrem de uma forma mais frequente, para gerar um aprendizado e ter uma cultura livre de culpa, multiplicando conhecimentos e aprendizados continuamente.

Todas estas formas serão detalhadas nos próximos capítulos: Exército de Macacos Símios, Reunião post-mortem, Dias de Jogos e Descobertas.

Capítulo 2. Aprendizagem e experimentação

Neste capítulo abordaremos as seguintes práticas de aprendizagem e experimentação:

- Tipos de Macaco do exército simiano (Case Netflix).
- Reunião post-mortem livre de culpa.
- Dias de Jogos.
- Descobertas.

Case da Netflix

Conforme detalhado no Livro Jornada DevOps (2020): o time de engenheiros da Netflix criou um processo para injeção de falhas chamado Chaos Monkey, que foi criado em resposta ao movimento de mover a aplicação de uma infraestrutura física para uma infraestrutura física na nuvem, provida pela AWS (Amazon Web Services). Na época, eles precisaram garantir que a perda de uma instância da Amazon não poderia afetar a experiência de streaming da Netflix. Essa técnica ficou bastante conhecida quando em 2011 houve uma grande indisponibilidade no AWS (serviço de nuvem da amazon), que impactou muitas empresas no mundo. A grande surpresa do mercado foi que este acontecimento não gerou grandes impactos nos serviços da Netflix, isso porque suas equipes já tinham aumentado a resiliência da infraestrutura com o aprendizado adquirido na injeção de falhas. O resultado principal disso foi a forma como eles implementaram o sistema, que permite o aprendizado contínuo das equipes sem a cultura de medo e punição.

Este “Chaos Monkey” evoluiu para novos macacos, chamado “Exército Símio”, sendo eles:

- **Chaos Gorilla:** simula a falha de **uma zona inteira** de disponibilidade AWS (Serviço Web Amazon).

- **Chaos Kong:** simula indisponibilidade em **regiões inteiras** da AWS (ex.: Europa, América do Norte, África etc.)

- **Macaco de Latência:** causa atrasos ou paralisações artificiais, simula a **degradação de serviço** para garantir que serviços dependentes respondam de forma adequada.

- **Macaco Janitor:** responsável em garantir que o ambiente esteja livre de desperdício e desorganização. Procura recursos não utilizados e os desativa.

- **Macaco de Conformidade**: localiza e desliga instâncias AWS que não seguem as melhores práticas (Ex.: falta de e-mail para alerta), o Macaco de Segurança é uma extensão para vulnerabilidades.

- **Macaco Doutor:** verifica a integridade de cada instância, desliga instâncias não íntegras quando o responsável não resolve a causa raiz no tempo combinado.

Abaixo destaco importantes reflexões sobre o aprendizado com falhas:

- Para o verdadeiro aprendizado, é necessário combater a teoria da maçã podre, que busca “eliminar as pessoas que causaram os erros”.

- Segundo Dekker, o erro humano é a consequência do projeto de ferramentas que as pessoas recebem para trabalhar: **Deve-se buscar a causa sistêmica dos erros**.

- Um engenheiro do Google confessou: “Eu estraguei uma linha de código e isso nos custou um milhão de dólares em receita”, e não foi demitido.

Reunião post-mortem livre de culpa

É uma reunião que deve focar no problema, ou seja, o foco não está em descobrir quem fez ou contribuiu, mas sim no aprendizado constante da equipe e na melhoria dos processos desde o início da demanda até a descontinuidade do produto, envolvendo assim todas as áreas da empresa (desenvolvimento, operação, pessoas, segurança e negócio).

Uma cultura justa é tão importante quanto a segurança psicológica dentro de uma organização. Ao realizar qualquer tipo de análise de causa-raiz ou reunião, siga estas dicas para que seja possível criar um ambiente propício de solução de problemas e não de acusações:

- Nunca nomear ou envergonhar quem é o culpado da falha.
- Incentivar quem compartilha problemas do sistema.
- Criar confiança para que equipe aprenda com problemas.
- Lições aprendidas sem culpa e injeção controlada de falhas em produção.

É muito comum que esta reunião aconteça após a solução de um incidente em produção, mas também pode ser realizada ao final da sprint, em uma retrospectiva para discutir a ocorrência de defeitos recorrentes, por exemplo. O principal objetivo desta reunião é entender e divulgar:

- Ações que deram certo.
- Ações que podem ser aprimoradas.
- Erros e medidas para evitar sua recorrência.

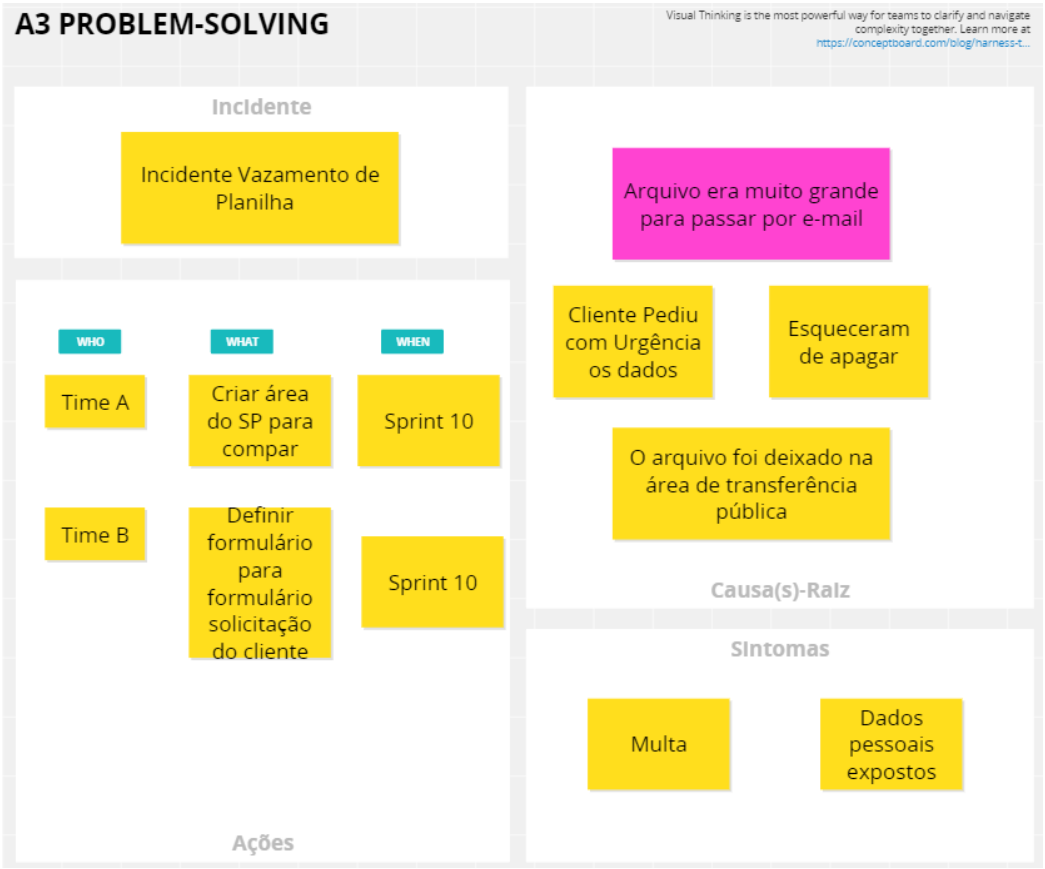
Destaco algumas etapas importantes desta reunião:

- Agendar a reunião com as pessoas envolvidas.
- Realizar a reunião.
- Publicar o resultado da reunião.

Exemplo de uma retrospectiva de causa-raiz após um incidente

O exemplo abaixo retrata um exemplo de causa-raiz após o incidente de um vazamento de uma planilha que continha dados pessoais e sensíveis. Foi utilizado o concept board (https://app.conceptboard.com/) para essa reunião, e os resultados foram compartilhados na wiki.

Figura 8 – Reunião post-mortem.

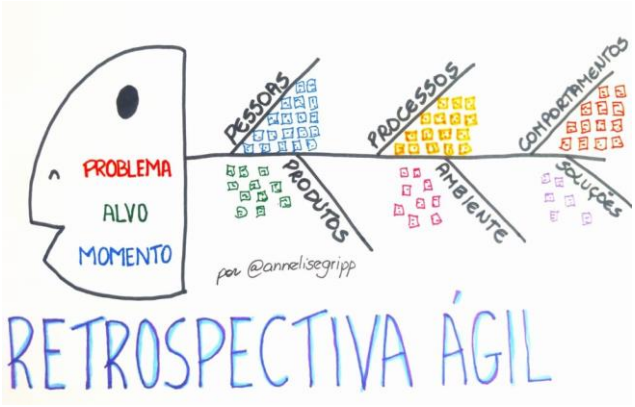


Nesta reunião seguimos os seguintes passos:

- Descobrir Sintomas (1).
- Chegando a causa-raiz.
- Planos de Ação e seus responsáveis.

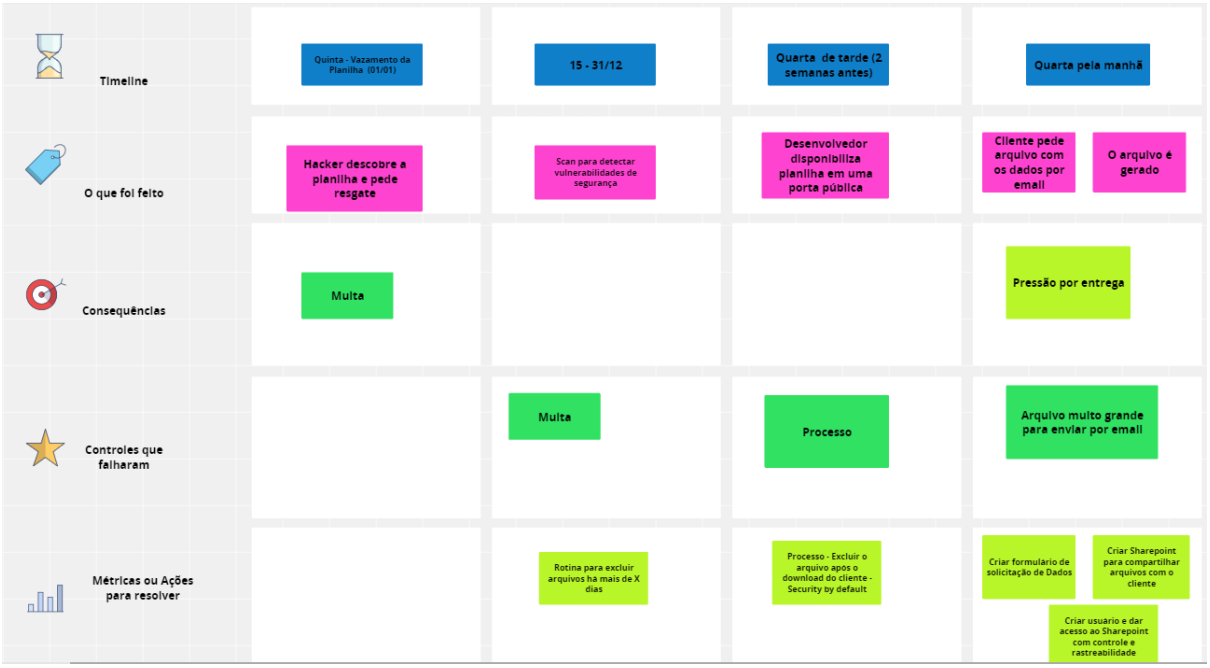
As Figuras 9 e 10 representam outras duas formas de publicar os resultados de uma reunião post-mortem.

Figura 9 – Retrospectiva Causa-Raiz (Espinha de Peixe).



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

Figura 10 – Retrospectiva Causa-Raiz (Espinha de Peixe).



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

Dias de Jogos

Esse conceito 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.

Foi popularizado por Jesse Robbins pelo trabalho que fez na Amazon para garantir a disponibilidade do site. Ele ficou conhecido como o “Mestre do Desastre” e defende que **“Um serviço não está realmente testado até o estragarmos em produção”**.

Dias de Jogos – Etapas

As seguintes etapas devem ser planejadas nos Dias de Jogos:

1. Planejar a interrupção. Ex.: simular perda completa de datacenter.
2. Adotar medidas. Ex.: criar site de contingência.
3. Testar medidas. Ex.: usar site de contingência.
4. Executar interrupção. Ex.: avaliar resultados da contingência.
5. Seguir plano e aprender. Ex.: avaliar situações não previstas.

Games Days: como é na AWS?

1. Defina seu cenário e planeje:
   - Requisitos de Ambiente necessário (HW/SW).
   - Seleção de pessoas e comunique as que vão jogar e as que vão só observar o jogo.
   - Um processo para divulgação dos jogos.

Não se esqueça:

- Pessoas que vão “jogar” precisam conhecer ferramentas e processos.
- Múltiplos times vão precisar de múltiplos observadores.

Dicas de cenários: falhas anteriores conhecidas fraquezas de processos e times, tarefas sazonais que são feitas sob demanda etc.

Fonte: https://wa.aws.amazon.com/wat.concept.gameday.en.html.

Dias de Jogos: como é na AWS?

- Games Days: como é na AWS?

Na preparação do jogo:

- Criar ambientes que devem ser “iguais” ao de produção.
- Pensar nas permissões, indicadores críticos e automatizar as verificações (runbooks).

Dicas de Runbooks, o que documentar?

- **Requisitos**: permissões, ferramentas e suas configurações, acesso e conectividade.
- **Restrições**: janelas de manutenção, recursos impactados e identificar conflitos entre atividades de negócio e operações.
- **Procedimentos e saídas**: vulnerabilidades, fraquezas e/ou melhorias registradas.
- **Procedimentos de escalamento**: jogadores não participarem, timebox, para quem escalar, o que e quando.
- Tomadores de decisão (antes, durante e após os jogos).

Dicas sobre as reuniões:

- Defina um cronograma;
- Divulgue;
- Obtenha o comprometimento.

Dicas sobre as simulações:

- Execute a simulação;
- Use uma sala separada;
- Anuncie que os jogos vão começar;
- Verifique se o runbook está sendo executado;
- Ouça o feedback dos observadores para decisões de adiamento das simulações;
- Anuncie o fim do jogo.

Ao final do jogo:

- Analise o dia de jogo;
- Faça uma retrospectiva e anote melhorias nos processos, ferramentas e runbook;
- Analise necessidades adicionais de treinamentos, ferramentas ou automações.
- Documente outras áreas para os próximos dias de jogos.

Descobertas:

Quando falhas são descobertas e soluções são dadas, é muito importante que exista mecanismos capazes de divulgar este aprendizado para a organização como um todo. Além disso, esse conhecimento precisa estar explícito para que, quando pessoas novas entrem na empresa, elas saibam como agir e o que fazer nestas situações.

Empresas de alto desempenho obtém os mesmos resultados (ou melhores) refinando as operações diárias, introduzindo tensão continuamente para aumentar o desempenho, e assim gerando mais resiliência nos seus sistemas.

Como divulgar e compartilhar aprendizados (descobertas) entre os times, então?

Existem diversas formas de explicitar este conhecimento, e as que vamos abordar neste capítulo são:

- A utilização de requisitos não funcionais (NFR) (codificados) para projetar as operações.
- Como elaborar histórias de usuários de operações reutilizáveis com base no desenvolvimento.
- Quais objetos devem ser armazenados no repositório de códigos-fonte de compartilhamento simples.
- Como transformar descobertas locais em melhorias globais.

Usando os requisitos não funcionais (RNF) para projetar operações:

Os requisitos não funcionais (RNF) mais comuns para projetar operações são:

- Telemetria completa de produção: verifica através de métricas se o sistema está comportando conforme o esperado.
- Capacidade de monitorar dependências: verifica se os serviços que fornecem dados ao sistema estão operacionais, funcionando e na capacidade adequada.
- Serviços resilientes que degradam quando necessário: serviços que podem ser degradados de uma forma planejada, para que a aplicação não caia devido a uma carga excessiva no sistema.
- Compatibilidade com todas as versões: verifica se o sistema é compatível com outros sistemas nos quais ele é integrado. Geralmente aqui são implementados testes de contrato.

Como transformar melhorias locais em globais

O livro Jornada DevOps (Muniz et al, 2020) descreve algumas ações importantes de transformação de melhorias locais em globais (Figura 11), são elas:

1. **Chat automatizado**: as conversas com o time são registradas de forma dinâmica e todos podem colaborar e dar sua opinião. Além disso, algumas operações no sistema, como o deploy, pode ser automatizado via chat-bots, onde você digita o comando no chat e ele se comunica com a aplicação executando uma instrução pra subir aquela branch do código para algum ambiente de testes, ou até mesmo para a produção. Desta forma não existe a necessidade de realizar reuniões formais e escrever atas, já que tudo é conversado e resolvido dinamicamente.

2. **Automatizar atividades em software**: ao invés de utilizar documentos estáticos que ficam passíveis de nunca serem atualizados, você documenta no próprio código. Os engenheiros da General Eletrics criaram o conceito de ArchOps, onde os diagramas são automatizados e atualizados dinamicamente, de forma qualquer pessoa do time possa ter uma visão clara da arquitetura.

3. **Código fonte em repositório único**: mesmo usando microsserviços, onde os repositórios são separados em partes, a ideia aqui é que todos tenham acesso a todo o código. Sendo assim, é possível ter uma compreensão maior do sistema como um todo e não apenas uma parte específica. As configurações de infra podem estar nesse repositório, os padrões de testes e segurança, e as configurações do pipeline de implantação da ferramenta de integração contínua utilizada. Padrões de codificação e tutoriais também podem estar no repositório.

4. **Testes automatizado como documentação e comunidade de prática**: quando são criados os testes automatizados, existem exemplos de como a aplicação deve funcionar. Neste caso, as regras de negócio também são documentadas usando testes, como uma documentação viva, que nada mais são do que uma fonte rica das entradas e saídas possíveis de cada ponto de

comunicação do software. As comunidades de prática se resume às pessoas que se reúnem para discutir o comportamento do sistema e que definem de fato esse comportamento usando ferramentas de discussão ou bate-papo, como Slack, Skype da Microsoft, Gooogle Hangouts, Appear.in, dentre outras.

5. **Codificar registro não funcional de OPs**: criar mecanismos de monitoramento de requisitos não funcionais dentro do próprio código servem como um tipo de "estetoscópio" da aplicação. Assim como o cardiologista usa aparelhos de medição para estudar a frequência cardíaca, o software deve ter aparelhos de medição do time de operações instalados dentro do código-fonte para monitorar a "saúde" da aplicação.

6. **Histórias de usuários de OPs usadas em DEV**: é necessário criar histórias de tudo o que o time de operação realiza no dia a dia, para "ensaiar" todas as ações junto ao time de desenvolvimento. Assim, se mantém todo o time preparado para imprevistos.

7. **Escolhas de tecnologia para os objetivos de negócio**: cada tecnologia tem características que influenciam tanto no esforço de aprendizado e desenvolvimento da aplicação, quanto no impacto dos requisitos não-funcionais para o pleno funcionamento da aplicação. Um exemplo disso são tecnologias assíncronas, que enviam um request e não têm uma previsão de retorno da resposta ao sistema quando utilizados os conceitos de jobs e filas. É interessante alinhar se este tipo de estratégia está cumprindo os requisitos de negócio também para ter uma transparência ao usuário final, que está utilizando a aplicação.

Transformar processos que estão documentados em editores de texto em *workflows* e/ou *scripts* automatizados é uma outra forma de transformar um conhecimento local em organizacional, permitindo a reutilização e a sua ampla utilização, fornecendo valor agregado para todos os que usam.

Figura 11 – Como transformar descobertas locais em globais.

Fonte: Livro Jornada DevOps (Muniz et al,2020)

Descobertas – Histórias de operações reutilizáveis

Histórias de operações reutilizáveis são utilizadas quando existem atividades operacionais que, de alguma forma, não podem ser automatizadas em um primeiro momento, para tornar visível para todos o time como realizá-las. Além do mais, estando documentadas, podem ser priorizadas para que o time de desenvolvimento possa automatizá-las em alguma sprint/release.

É possível destacar os benefícios desta prática:

- Expõe o trabalho de Operações de TI passíveis de reprodução, de forma que aparece ao lado do trabalho de desenvolvimento, permitindo melhor planejamento e resultados mais passíveis de reprodução.
- Manter as configurações de ambientes automatizadas. Sendo assim, sempre que preciso é possível criar um ambiente novo de forma rápida e confiável,

considerando que as mesmas configurações serão aplicadas em ambiente de produção.

Como começar então?

Atividades: levantar quais são as atividades do time de operação que o time de DEV pode ajudar a automatizar.

Recursos Necessários: determinar o que é preciso em termos de infra para que o sistema venha a funcionar em produção.

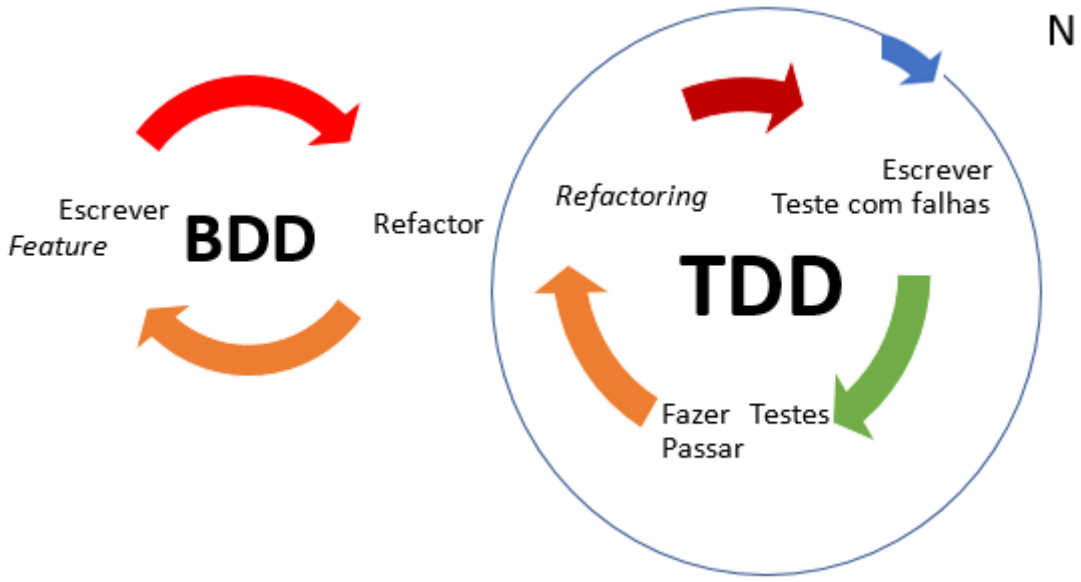
Etapas planejadas: para cada etapa do projeto ou do ciclo de desenvolvimento do software, é necessário entender em quais etapas o time de operação atua para agir sempre em conjunto com a evolução do software. Ou seja, realizar o hand-off das atividades, seja documentar de forma simples quando há atuação do desenvolvimento ou da operação, e em que momento isso ocorre.

Ferramentas: disseminar o conhecimento acerca das ferramentas que o time de operações utiliza para poder atuar junto com eles seja no monitoramento, seja na infra ou em outras configurações necessárias. Embora cada time tenha autonomia para escolher suas ferramentas, é interessante disseminar o conhecimento sobre as ferramentas e tentar alinhar ferramentas que possam ser utilizadas pelo time de desenvolvimento e operações.

Testes automatizados como documentação (BDD):

“Behavior-driven development é sobre implementar uma aplicação através da descrição de seu comportamento pela perspectiva de seus stakeholders” (Dan North)

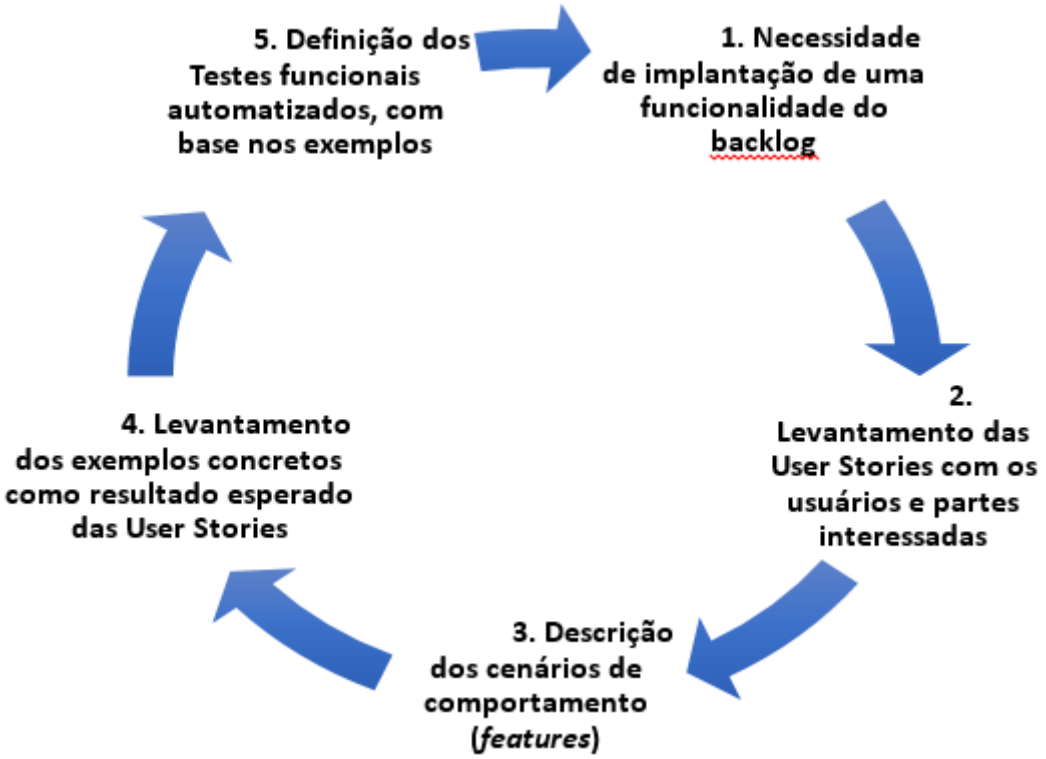
Figura 12 – BDD.



O BDD:

- É uma técnica de desenvolvimento Ágil.
- Encoraja colaboração entre desenvolvedores, setores de qualidade e pessoas não-técnicas ou de negócios num projeto de software.
- É uma resposta ao TDD que é direcionado unicamente aos desenvolvedores.
- Possui frameworks de automatização conhecidos: Jbehave(Java), Rspec(Ruby), Cucumber, Nbehave(.Net) e SpecFlow (.Net).

Figura 13 – Passo do BDD.



O BDD sugere que os analistas/testadores escrevam os cenários antes mesmo dos testes serem implementados, e desta forma os desenvolvedores terão uma visão geral do objetivo do projeto antes de codificá-lo.

Em BDD, um desenvolvedor, algum profissional do setor de qualidade ou até mesmo o cliente, podem esclarecer os requisitos quebrando-os em exemplos específicos.

Exemplo de requisito: “Itens reembolsados ou substituídos devem ser retornados para o estoque”:

- **Cenário 1**: Itens reembolsados devem retornar para o estoque:

  Dado que um cliente compra um jumper preto

  E eu tenho três jumpers pretos no estoque,

  Quando ele retorna com o jumper preto para reembolso

Então eu devo ter quatro jumpers pretos no estoque.

- **Cenário 2**: Itens substituídos devem ser retornados ao estoque:

  Dado que uma cliente compra um vestido azul

  E eu tenho dois vestidos azuis no estoque

  E eu tenho três vestidos pretos no estoque,

  Quando ela retorna com o vestido para uma troca por um preto

  Então eu devo ter três vestidos azuis no estoque

  E dois vestidos pretos no estoque.

Gherkin:

O **Gherkin** é uma linguagem semiformal, cujo objetivo é descrever cenários de teste inteligíveis, automatizáveis e em conformidade com uma necessidade de negócio. Além disso, é orientada a espaços, usando indentação para definir a estrutura. Os fins de linha encerram as declarações (Passos) e espaços ou tabs também podem ser usados para endentação

O **Gherkin** possui palavras reservadas. São elas:

Feature -> Funcionalidade.

Scenario -> Cenário.

Given -> Dado.

When -> Quando.

Exemplos Gherkin:

Figura 13 – Exemplos de cenários.

```
12
13  Cenário: Digitei letras no campo de número do cartão
14      Dado que eu escolhi os itens que vou comprar
15         E que estou prestes a digitar os dados do meu cartão de crédito
16    Quando eu insiro letras no campo de número do cartão
17     Então nada aparece escrito
18         E uma mensagem me orienta com o formato correto de um número de cartão de crédito
19
20  Cenário: Número do cartão de crédito com poucos dígitos
21      Dado que eu escolhi os itens que vou comprar
22         E que estou prestes a digitar os dados do meu cartão de crédito
23    Quando eu insiro um número de cartão que tem apenas 15 digítos
24     Então o botão para confirmar a compra fica desabilitado
25         E ao tentar clicar nesse botão, uma mensagem me orienta com o formato correto
```

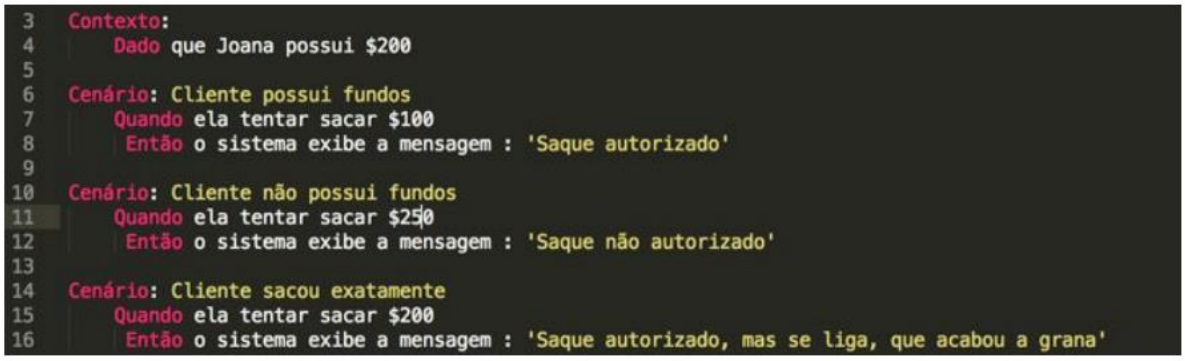
Fonte: https://medium.com/idexo-developers/bdd-behavior-driven-development-e-a-qualidade-de-software-d04b06f54ec1.

Figura 14 – Exemplos de contextos.

```
8   # assim incluímos comentários no arquivo
9   Contexto:
10      Dado que eu escolhi os itens que vou comprar
11         E que estou prestes a digitar os dados do meu cartão de crédito
12
13  Cenário: Digitei letras no campo de número do cartão
14    Quando eu insiro letras no campo de número do cartão
15     Então nada aparece escrito
16         E uma mensagem me orienta com o formato correto de um número de cartão de crédito
17
18  Cenário: Número do cartão de crédito com poucos dígitos
19    Quando eu insiro um número de cartão que tem apenas 15 digítos
20     Então o botão para confirmar a compra fica desabilitado
21         E ao tentar clicar nesse botão, uma mensagem me orienta com o formato correto
```

Fonte: https://medium.com/idexo-developers/bdd-behavior-driven-development-e-a-qualidade-de-software-d04b06f54ec1.

Figura 14 – Exemplos de contextos.



```
Contexto:
    Dado que Joana possui $200

Cenário: Cliente possui fundos
    Quando ela tentar sacar $100
     Então o sistema exibe a mensagem : 'Saque autorizado'

Cenário: Cliente não possui fundos
    Quando ela tentar sacar $250
     Então o sistema exibe a mensagem : 'Saque não autorizado'

Cenário: Cliente sacou exatamente
    Quando ela tentar sacar $200
     Então o sistema exibe a mensagem : 'Saque autorizado, mas se liga, que acabou a grana'
```

Fonte: https://medium.com/idexo-developers/bdd-behavior-driven-development-e-a-qualidade-de-software-d04b06f54ec1.

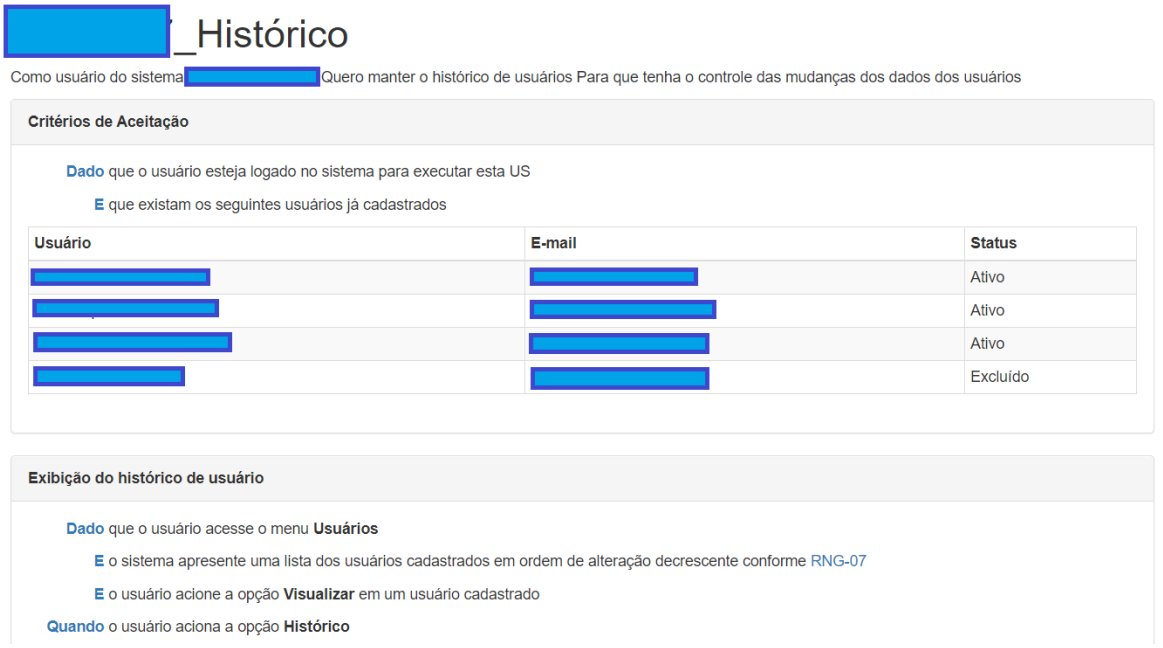
EPE (Especificação por Exemplo):

Segundo o Livro Jornada DevOps, Gojko Adzic, 2011, no livro Specification by Example, os testes de software servem como uma documentação viva de como o software precisa se comportar. E pelo fato de ser executável, esse tipo de documentação nunca fica desatualizada (Muniz et al, 2020).

Martin Fowler, no artigo “Specification By Example”, descreve a importância da especificação por exemplo e de como documentação útil no desenvolvimento de software é uma forma de estimular a cultura de testes automatizados de ponta a ponta nos times.

Exemplos:

Figura 15 – Exemplo EPE.



[redacted]_Histórico

Como usuário do sistema [redacted] Quero manter o histórico de usuários Para que tenha o controle das mudanças dos dados dos usuários

Critérios de Aceitação

Dado que o usuário esteja logado no sistema para executar esta US

E que existam os seguintes usuários já cadastrados

Usuário	E-mail	Status
[redacted]	[redacted]	Ativo
[redacted]	[redacted]	Ativo
[redacted]	[redacted]	Ativo
[redacted]	[redacted]	Excluído

Exibição do histórico de usuário

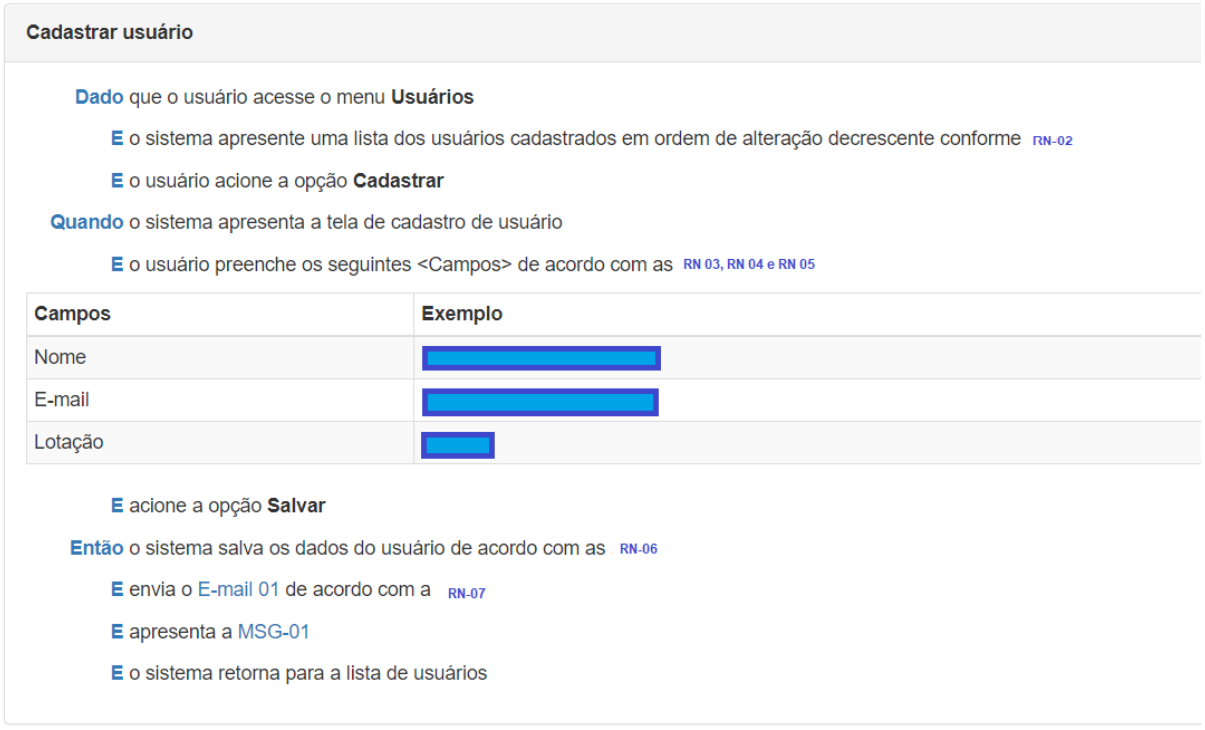
Dado que o usuário acesse o menu **Usuários**

E o sistema apresente uma lista dos usuários cadastrados em ordem de alteração decrescente conforme RNG-07

E o usuário acione a opção **Visualizar** em um usuário cadastrado

Quando o usuário aciona a opção **Histórico**

Figura 16 – Exemplo EPE.



Cadastrar usuário

Dado que o usuário acesse o menu **Usuários**

E o sistema apresente uma lista dos usuários cadastrados em ordem de alteração decrescente conforme RN-02

E o usuário acione a opção **Cadastrar**

Quando o sistema apresenta a tela de cadastro de usuário

E o usuário preenche os seguintes <Campos> de acordo com as RN 03, RN 04 e RN 05

Campos	Exemplo
Nome	[redacted]
E-mail	[redacted]
Lotação	[redacted]

E acione a opção **Salvar**

Então o sistema salva os dados do usuário de acordo com as RN-06

E envia o E-mail 01 de acordo com a RN-07

E apresenta a MSG-01

E o sistema retorna para a lista de usuários

Capítulo 3. Segurança e Gestão de Mudanças

O que é Segurança da Informação

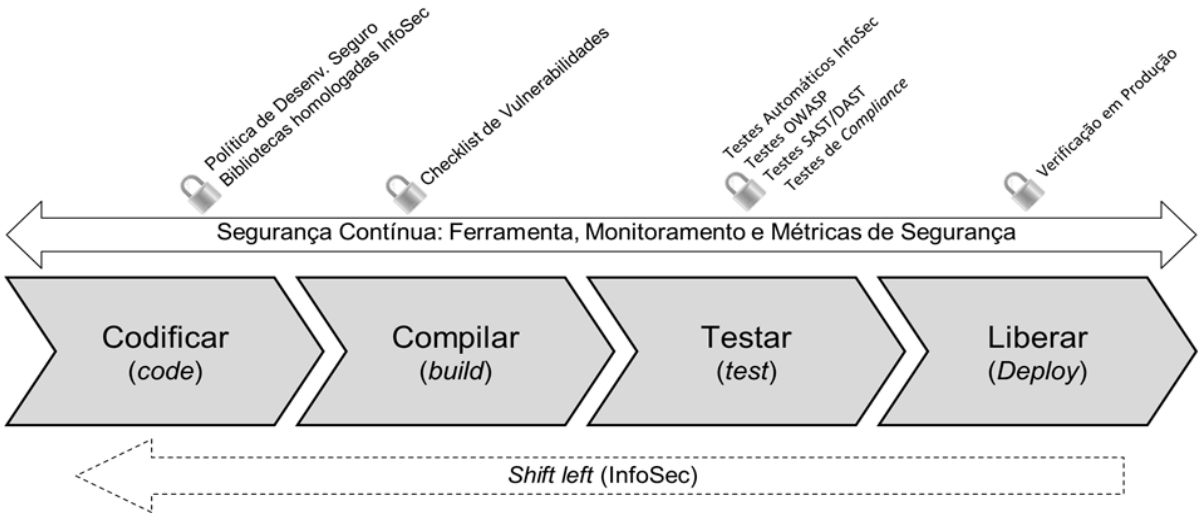
Resumidamente, a segurança da informação trata de três princípios:

- **Integridade**: propriedade da exatidão e completeza da informação.
- **Confidencialidade**: a informação não é disponibilizada ou divulgada para indivíduos, entidades ou processos autorizados.
- **Disponibilidade**: propriedade de ser acessível e utilizável sob demanda, por uma demanda de uma entidade autorizada.

O gargalo da Segurança da Informação no final do ciclo:

O objetivo é que a equipe de segurança participe ativamente desde o início do ciclo de desenvolvimento, com grande foco em automatização dos controles, exercitando a conformidade por demonstração. Desta forma, evitamos o gargalo da segurança ao final do ciclo de desenvolvimento não autorizando mudanças por problemas no ambiente de produção que poderiam ter sido alinhados desde o início do desenvolvimento da aplicação.

Figura 17 – Exemplo EPE.



A Figura 17 sugere algumas ações que devem ser tomadas ao longo do ciclo. São elas:

- Codificar (code):
  - Política de desenvolvimento seguro.
  - Bibliotecas homologadas de InfoSec.
- Compilar (build):
  - Checklist de vulnerabilidades.
- Testar (test):
  - Testes automáticos InfoSec.
  - Testes OWASP.
- Testes SAST/DAST:
  - Testes de Compliance.
- Liberar (Deploy):
  - Verificação em produção.

Figura 18 – Sete maneiras para melhor integração do Sec com o Dev e Ops.

Fonte: MUNIZ; SANTOS; IRIGOYEN; MOUTINHO. Jornada DevOps. Brasport, 2019.

Dicas de Segurança da Informação:

A seguir seguem algumas dicas do Livro Jornada DevOps para a garantia da segurança ao longo do ciclo de desenvolvimento, implantação e operação do produto.

Desenvolvimento:

1. Teste de código.
2. Revisão de código.
3. Teste de penetração.
4. Servidores de IC como código.
5. Análise estática pode ser realizada considerando critérios específicos de segurança, garantindo a remoção dos erros antes mesmo da execução do *build (SONAR,* Veracode, Gauntlet *e manual).*

Evitar que usuários sem autorização acessem o ambiente:

1. Controle das configurações de ambiente.
2. Teste de ataques de injeção de SQL.
3. Use credenciais de IC somente leitura.
4. Use VM isoladas.

Usar telemetria em ambiente para detectar possíveis falhas:

- Alterações de componentes e infraestrutura na nuvem.
- Alterações em configurações diversas (Puppet, Chef, entre outros).
- Alteração de usuários nos grupos privilegiados de admin e InfoSec.
- Erros http no servidor web, como os 4xx e 5xx, ex. 401 - Não autorizado; 403 - Permissão Negada; 502 - Bad Gateway.
- Abertura e fechamento de portas.

O uso do SONAR para garantir a segurança:

O SONAR é uma ferramenta de análise de código estática que detecta vulnerabilidades, code smells e cobertura de testes automatizados.

As dez principais vulnerabilidades que o SONAR verifica (Figura 19), de acordo com a OWASP, são as abaixo:

- Injection.
- Broken Authentication and Session Management.
- Cross-Site Scripting (XSS).
- Broken Access Control.
- Security Misconfiguration.

- Sensitive Data Exposure.
- XML External Entities (XXE).
- Insecure Deserialization.
- Using Components with Known Vulnerabilities.
- Insufficient Logging & Monitoring.

O OWASP é um Projeto Aberto de Segurança em Aplicações Web, e disponibiliza gratuitamente vários artefatos acerca da segurança em aplicações WEB (artigos, metodologias, documentação e ferramentas).

Figura 19 – OWASP x SONAR.

Categories	Vulnerabilities		Security Hotspots		
			Open	In Review	Won't Fix
A1 - Injection	0	A	43	0	0
A2 - Broken Authentication	5	E	0	0	0
A3 - Sensitive Data Exposure	2	C	13	0	0
A4 - XML External Entities (XXE)	0	A	0	0	0
A5 - Broken Access Control	0	A	0	0	0
A6 - Security Misconfiguration	4	E	0	0	0
A7 - Cross-Site Scripting (XSS)	0	A	5	0	0
A8 - Insecure Deserialization	0	A	1	0	0
A9 - Using Components with Known Vulnerabilities	0	A	0	0	0
A10 - Insufficient Logging & Monitoring	0	A	0	0	0
Not OWASP	656	D	0	0	0

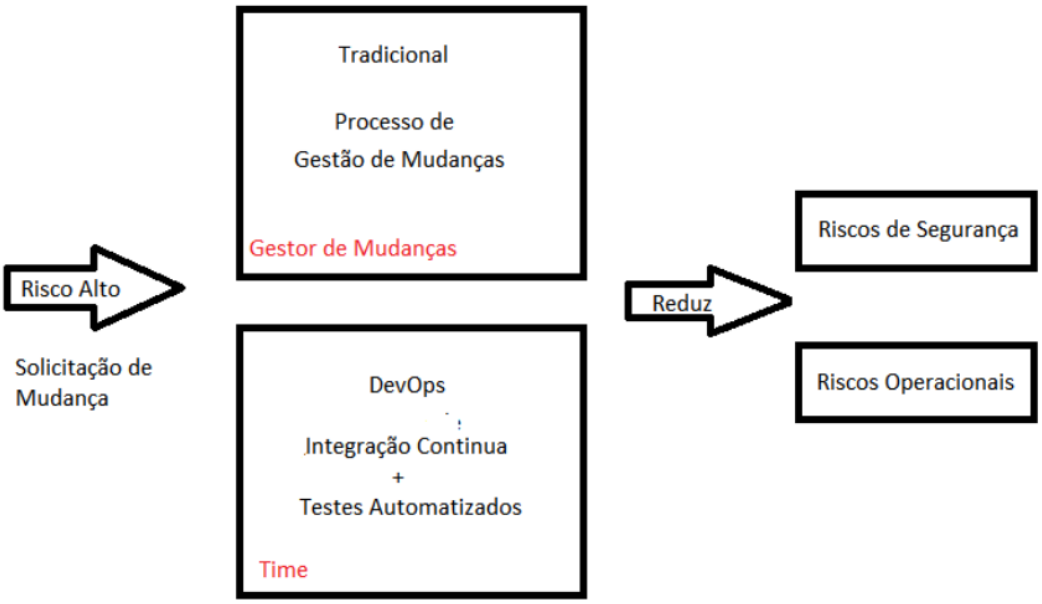
Gestão de mudanças:

Normalmente, nas empresas temos diversos tipos de mudança:

- **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.

A utilização de práticas DevOps, como um pipeline de implementação adequado, permite que a grande maioria das mudanças seja classificada como baixo risco (Figura 20).

Figura 20 – Mudanças com e sem DevOps.



Fonte: Livro Jornada DevOps.

Simplifique a gestão de mudanças:

- Crie uma rotina de visita local (gemba), para sempre conversar com os envolvidos e propor mudanças para a eliminação de desperdícios.
- Avalie o formulário de mudanças e a real necessidade de todos os campos.
- Negocie a redução de aprovações pelo Pull Request.
- Negocie com a auditoria a existência de evidências nas ferramentas e não nos artefatos.

Gere transparência:

- Lista de mudanças realizadas nos últimos três meses.

- Compartilhar a lista completa dos problemas encontrados nestas mudanças.
- Compartilhar os indicadores de mudanças, por exemplo: tempo médio entre falhas ou tempo médio para reparar uma mudança.
- Demonstrar o controle do ambiente, onde são realizadas as implementações e testes automatizados.
- Demonstrar como os erros são controlados e os processo de correção. Neste ponto é importante mostrar as ferramentas de automação e o controle de acesso, quando necessário.
- Automatizar o máximo possível as solicitações e gestão de mudanças, para que seja fácil mostrar as evidências automatizadas do controle e gerar confiança no processo DevOps para os stakeholders gestor de compliance, auditores e gestor de mudanças. Muitas vezes o diálogo e o treinamento de algumas ferramentas podem gerar a confiança necessária.

Capítulo 4. AIOPS

A Inteligência Artificial na Operação (AIOPS)

Os principais objetivos da Inteligência artificial são:

- Automação de processos repetitivos.
- Observar e aprender sobre comportamentos.

Uma aplicação AIOPs, segundo o Gartner, deve ser capaz de:

- Consumir dados de um conjunto de tipos e fontes diferentes.
- Analisar dados em tempo real e a qualquer momento depois (dados históricos).
- Ativar o armazenamento de dados para acesso a qualquer momento.
- Permitir acesso aos dados de forma segura e em qualquer momento baseado no cargo ou função na empresa.
- Aproveitar o aprendizado de máquina (machine learning) para analisar dados gerados por máquinas e seres humanos a utilizá-lo para aprendizagem e fornecimento de análises.
- Capacidade de aproveitar as análises para automação e ação proativa.
- Capacidade de apresentar as análises em contexto para a pessoa ou equipe funcional.

Alguns exemplos de uso da inteligência artificial encontrados em ferramentas do mercado, são:

- Tempos de respostas reduzidos.
- Detecta dependências de outros serviços (atuais e futuras).
- Classifica com estatística (quem teve o maior peso na degradação).
- Saturação de CPU em um dos hosts Lin.
- Vão do mainframe aos servidores em nuvem (full stack).

Referências

DWECK, C. S. *Mindset*: the new psychology of success. London: Robinson, 2017.

EDMONDSON, Amy C. *Strategies for Learning from Failure*. Harvard Business Review. 2011.

EDMONDSON, Amy C. *The Competitive Imperative of Learning*. Harvard Business Review. 2008.

KIM, Gene et al. *The DevOps Handbook*: how to create world-class agility, reliability, and security in technology organizations. Portland: IT Revolution Press, 2016.

MUNIZ, Antonio et al. *Jornada DevOps*. Editora Brasport, 2019.

MUNIZ, Antonio; IRIGOYEN, Analia. *Jornada Ágil e Digital.* Editora Brasport, 2019.

PRESSMAN, R. S. *Engenharia de Software*. 5. ed. Rio de Janeiro: McGraw-Hill, 2011.