Aprenda com quem faz

A Segunda Maneira: os princípios do feedback

Ana Clara Gonzaga

2023

SUMÁRIO

> Capítulo 1

Capítulo 1. Os princípios do feedback

Introdução - A importância do feedback

Conceituação e história

No mundo globalizado e principalmente pós-pandemia (COVID 19), é comum ouvirmos nos ambientes corporativos ou mesmo em algum post no LinkedIn a expressão "*feedback*" ou "1:1', por exemplo. Mas afinal, o que significa *feedback*? Qual sua importância? Quais valores agrega a uma organização, pessoa ou aplicação? Neste capítulo, serão expostos conteúdos que buscam responder essas e outras perguntas acerca do tema.

Literalmente, a palavra feedback significa reação a um estímulo e/ou efeito retroativo (Oxford Languages, 2021). No contexto organizacional o feedback representa toda e qualquer comunicação que é realizada entre pessoas com o objetivo de levar uma mensagem de um comportamento ou ação que aconteceu. Podem, por exemplo, ser uma necessidade de melhoria ou validação de comportamento apresentado.

A importância do feedback

O momento de feedback gera um valor qualitativo que se torna um viabilizador de melhoria contínua, tanto no contexto de desenvolvimento pessoal, quanto no de desenvolvimento de softwares e aplicações web. É preciso destacar que se torna muito importante a coleta dos dados e/ou insumos obtidos nesse processo de feedback. A periodicidade e excelência na execução também são fatores importantes.

Feedback no contexto DevOps
No contexto Devops

Neste contexto, o feedback pode ocorrer também no ambiente virtualizado por meio de automações. Algumas práticas Devops, como o *deploy* contínuo e o *post mortem* sem culpa, são viabilizadas pelo feedback frequente. Através do monitoramento e da capacidade de tornar as informações visíveis, todos os times envolvidos receberão feedbacks de suas aplicações e infraestrutura *por vários ângulos*.

Existe Devops sem feedback?

Não. O feedback é, definitivamente, um dos componentes mais importantes do DEVOPS. Em um time que atua sob a ótica DEVOPS, o feedback possibilita que as equipes de desenvolvimento e operações planejem, desenvolvam e implementem as codificações com segurança e assertividade. Além disso, permite que mais times consigam acompanhar o funcionamento e desenvolvimento das aplicações e agir adequadamente ao receber um feedback.

Exemplo prático 1: Há um bug na funcionalidade sendo desenvolvida. Os times não podem ter conhecimento deste apenas quando entrar em produção. Ao visualizar que os testes não passaram em um pipeline de implementação, os times de QA e desenvolvimento podem atuar com clareza e assertividade.

Exemplo prático 2: Uma pessoa do time repara uma anomalia no painel do *Cloud Watch.* Investiga e conclui que o *auto scaling group* do ambiente de *stage* não está funcionando e os times começam a reclamar de lentidão. Antes que impacte mais pessoas a pessoa do time já ajustou o problema e a operação continua normal naquele dia.

Nos próximos capítulos, vamos expor várias formas de garantir a cultura do feedback, inclusive a aplicação prática dele no dia a dia dos times.

> Capítulo 2

Capítulo 2. Observabilidade e telemetria

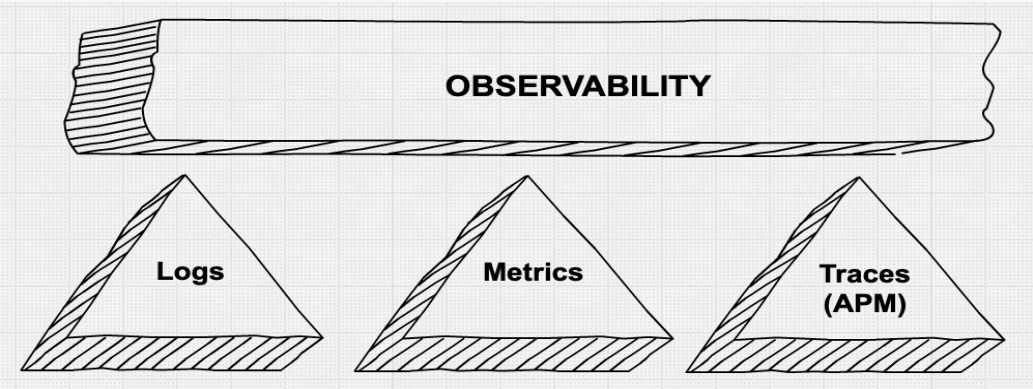
Introdução à observabilidade

Medições, indicadores, métricas, monitoramento, acompanhamento. Essas são palavras que com toda certeza você já ouviu no contexto de Devops e times de alta performance. O termo observabilidade, ou capacidade de observação, vem ganhando destaque ao longo dos anos nas comunidades de tecnologia, principalmente na de SRE. “Em seu sentido mais completo, observabilidade é uma propriedade de um sistema que foi projetado, construído, testado, implantado, operado, monitorado, mantido e evoluído” (Sridharan, 2018). Nela, através de um conjunto de ferramentas e insumos, o time terá informações o suficiente para agir de forma proativa. O time necessita ter a capacidade de garantir que os pontos cruciais que foram mapeados possam ser acompanhados de forma automatizada. Dessa forma, viabilizando que a observabilidade ocorra.

Os 3 pilares

Há 3 pilares na observabilidade que podem ser definidos separadamente, mas que para a viabilização da observabilidade devem ser considerados um conjunto. São eles: Logs, Métricas e Rastreamento distribuído.

Figura 1 – Observability pillars



Fonte: https://www.elastic.co/pt/blog/observability-with-the-elastic-stack

Logs

São registros feitos durante o funcionamento de um software ou a disponibilidade de recursos em uma infraestrutura cloud, por exemplo. Nos logs podemos observar informações como eventos, alertas e erros. Ao ler um log, a pessoa é capaz de observar fatos que aconteceram durante um período de tempo. Os arquivos de log alimentam sistemas que nos fornecem informações para o monitoramento de aplicações.

Métricas

São mensurações que expressam características de funcionamento das aplicações ou da infraestrutura. Há uma grande quantidade de métricas de ambos os cenários que podem ser utilizadas. Um exemplo de métrica de aplicação seria o tempo de resposta em milissegundos(ms) de uma API. Um exemplo de métrica de infraestrutura seria a % de consumo de CPU ou memória de uma instância.

Rastreamento distribuído

Os rastreamentos são uma representação dos logs, eles demonstram o fluxo de requisição de ponta a ponta através de um sistema distribuído. Na prática, eles são usados para identificar a quantidade de trabalho em cada camada.

O que é telemetria?

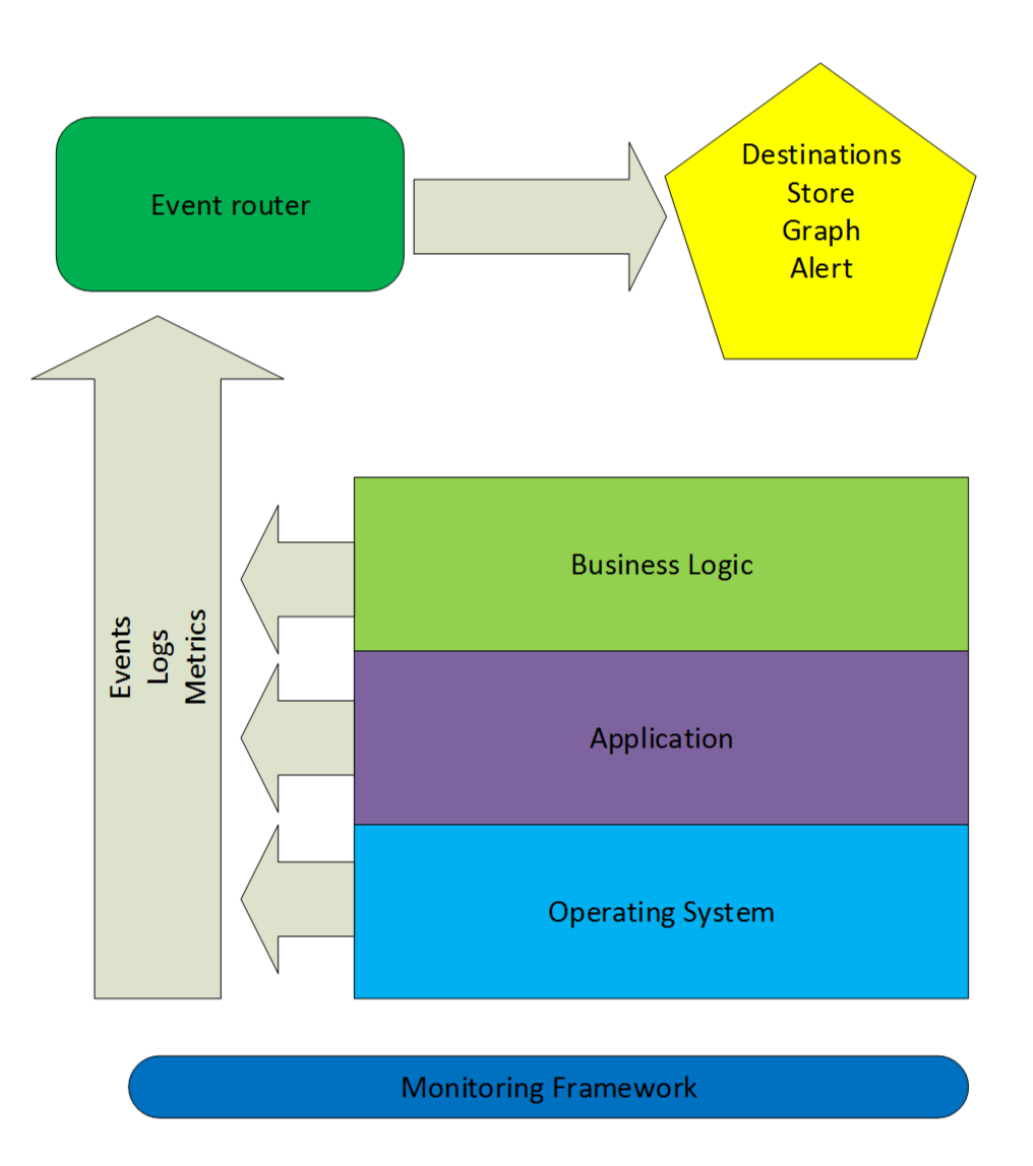
"Telemetry is the feedback you get from your production systems that tells you what's going on in there—feedback that improves your ability to make decisions about your production systems." (Riedesel, 2021). Em tradução literal, Jamie define a telemetria como o feedback que você recebe de seus sistemas em produção, que informa o que está acontecendo lá. É o Feedback que melhora sua capacidade de tomar decisões sobre seus sistemas em produção.

Durante todo o ciclo de vida de uma aplicação, ou de um ecossistema de aplicações, é necessário garantir que sejam criadas telemetrias suficientes tanto para novas partes dos sistemas quanto para seu legado.

Framework de monitoramento.

Este framework a seguir é conteúdo da certificação oficial EXIN Devops Professional, e demonstra visualmente elementos de telemetria e da observabilidade que juntos corroboram para que ocorra o monitoramento.

Figura 2 – Monitoring framework.



Fonte: Turnbull, The Art of Monitoring, edição Kindle, cap. 2.

Existem 3 componentes que englobam algumas métricas de telemetria. São eles: Lógica do negócio, aplicativo e sistema operacional. Esses componentes geram eventos, registros e métricas que vão para um roteador de eventos que são destinados a 3 diferentes esferas: armazenamento, gráfico e alerta. Na esfera 'armazenamento', geralmente as informações são retidas e armazenadas para fins de auditorias ou checagens futuras. Na esfera 'gráfico', são construídos dashboards com essas informações e montadas visões de acordo com a necessidade: visões de dados de segurança, infra, aplicação etc. Na visão de 'alertas', são configurados alertas que vão informar ao time de forma automatizada e proativa se algum indicador passou de um limite definido.

Tipos de logs para telemetria.

Existem diferentes níveis de registro que servirão à telemetria. São eles:

- Nível Informação: Registro das ações realizadas pelos usuários ou sistema. Costuma ser utilizado durante investigações de casos que ocorreram num espaço de tempo passado.
- Nível Depuração: Registro dos eventos de produção. Costuma ser habilitado quando o time está tentando resolver um problema que está ocorrendo naquele espaço de tempo.
- Nível Aviso: Registro de situações incomuns que podem se tornar um erro. Costumam ser visualizados quando há um tempo de resposta maior que o normal por parte do sistema e/ou aplicação, por exemplo.
- Nível Erro: Registros de falhas do sistema e/ou da aplicação.
- Nível Fatal: Registro de interrupção de serviço.

> Capítulo 3

Capítulo 3. Monitoramento e ferramentas

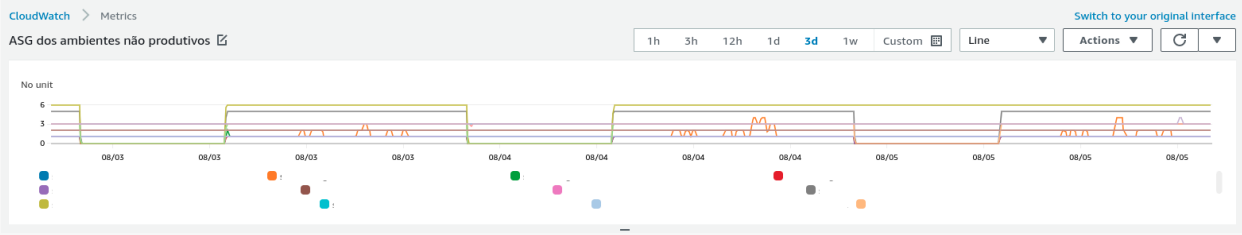
Introdução - Conceituação de monitoramento

Monitoramento é “Coletar, processar, agregar e exibir dados quantitativos em tempo real sobre um sistema, como contagens e tipos de consultas, contagens e tipos de erros, tempos de processamento e tempos de vida do servidor.” (Beyer, 2016). Há diferentes tipos de monitoramento, como o monitoramento da aplicação e o monitoramento da infraestrutura. No monitoramento de aplicações conseguimos observar informações como: tempo de resposta, quantidade de falhas nas respostas etc. No monitoramento da infraestrutura conseguimos observar informações como: Gasto de CPU de uma ou mais instâncias, uso de memória, funcionamento de um *auto scaling group*. Nas figuras 2 e 3, podemos observar recortes de dashboards de monitoramentos de aplicação e infraestrutura respectivamente. Na figura 2 é o monitoramento da quantidade de erros no frontend. Na figura 3 é o monitoramento da escalabilidade dos *asg(auto scaling groups)* de ambientes não produtivos.

Figura 2 – Fragmento de dashboard de monitoramento de Aplicação.

Fonte: https://www.datadoghq.com/product/#dashboards

Figura 3 – Fragmento de dashboard de monitoramento de Infraestrutura.



Fonte: Cedida pela autora.

Diferença entre observabilidade e monitoramento

De forma associativa, a diferença é que a observabilidade é o “O que” a Telemetria o “Meio” e o Monitoramento é o “Como”. A necessidade é que o sistema ou aplicação tenha a característica de ser informativo por meio das métricas e dados que são visualizados e acompanhados no monitoramento.

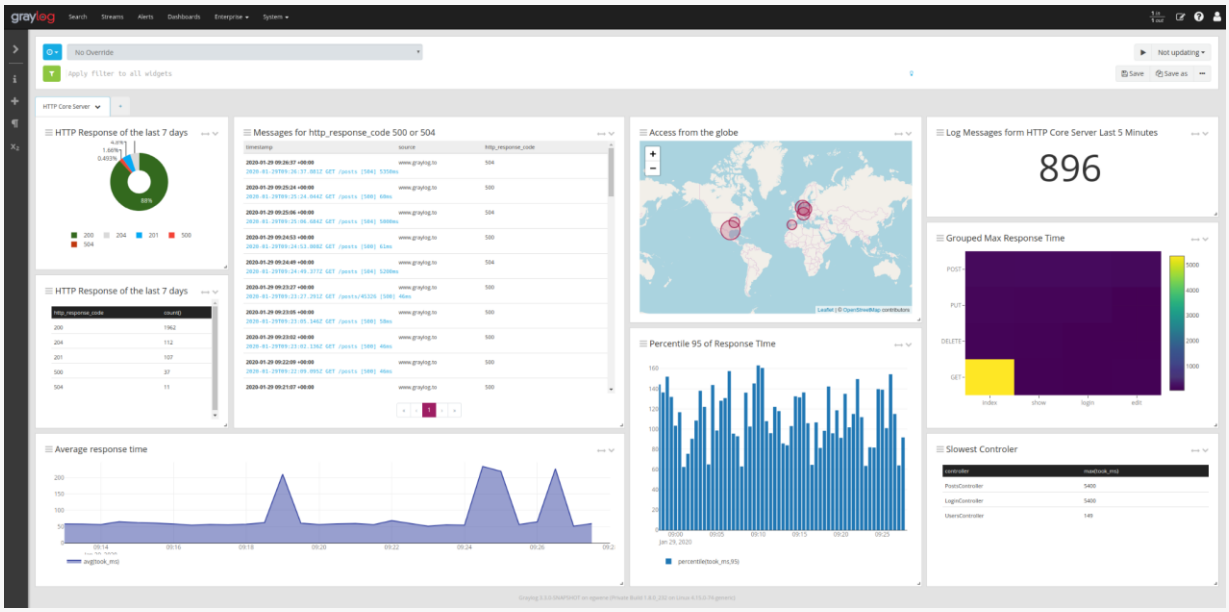
Ferramentas que viabilizam o monitoramento 1
Ferramentas Open Source

Um monitoramento eficaz é viabilizado por meio de ferramentas. A utilização de ferramentas Open Source é muito comum nos times de tecnologia. Devido ao grande trabalho de contribuição da comunidade, temos bastante delas nesse contexto de monitoramento.

Como para cada time e cenário há muitas particularidades, é normal que cada um tenha sua própria stack. Com algumas ferramentas ou conjunto delas conseguimos aglomerar em um dashboard o monitoramento das aplicações e de sua infraestrutura. Mas alguns times escolhem realizar esse monitoramento em ferramentas e dashboards completamente apartados. Abaixo há algumas destas ferramentas, suas principais características e exemplos práticos.

Graylog: É um grande aglomerador de logs oriundos de diferentes fontes. Une todas as informações recolhidas e as consolida em um frontend amigável.

Figura 3 – Dashboard de exemplo de uso do Graylog.



Fonte: https://docs.graylog.org/docs/dashboards

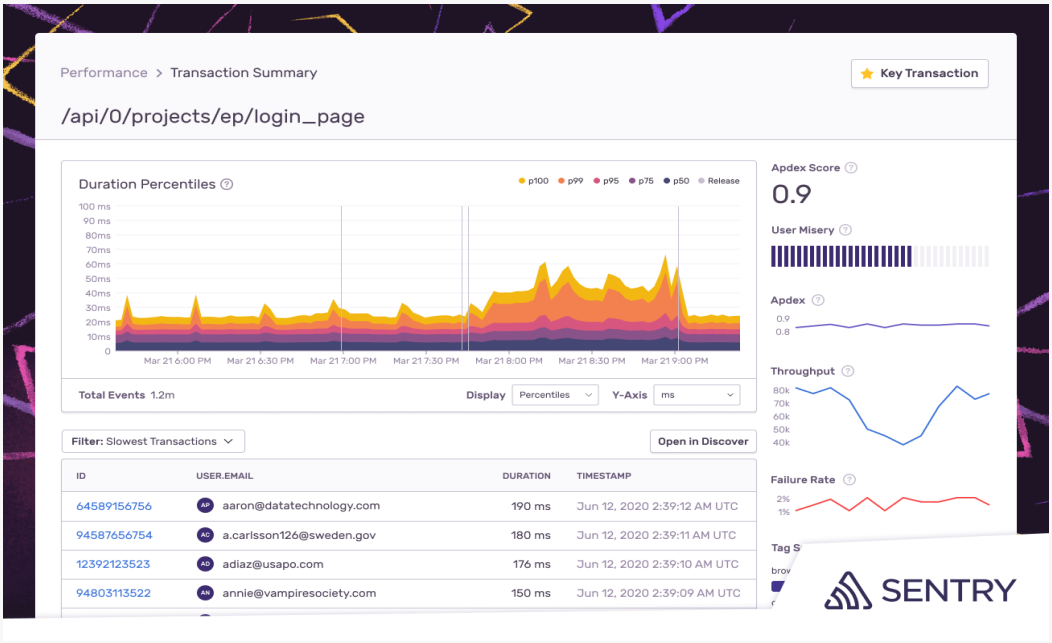
Prometheus + Grafana: São duas ferramentas diferentes, mas que são constantemente utilizadas em conjunto. O Prometheus coleta os dados de aplicações e sua infraestrutura e as expõe em um endpoint que pode ser consumido por aplicações de dashboards inteligentes, que são atualizados com frequência como o Grafana.

Figura 4 – Case de implementação de Prometheus + Grafana

Fonte: shorturl.at/GJLZ2

Sentry: Usado para fazer tracing das aplicações. Monitora os erros em tempo real. Ele tem suporte em mais de 100 linguagens e frameworks.

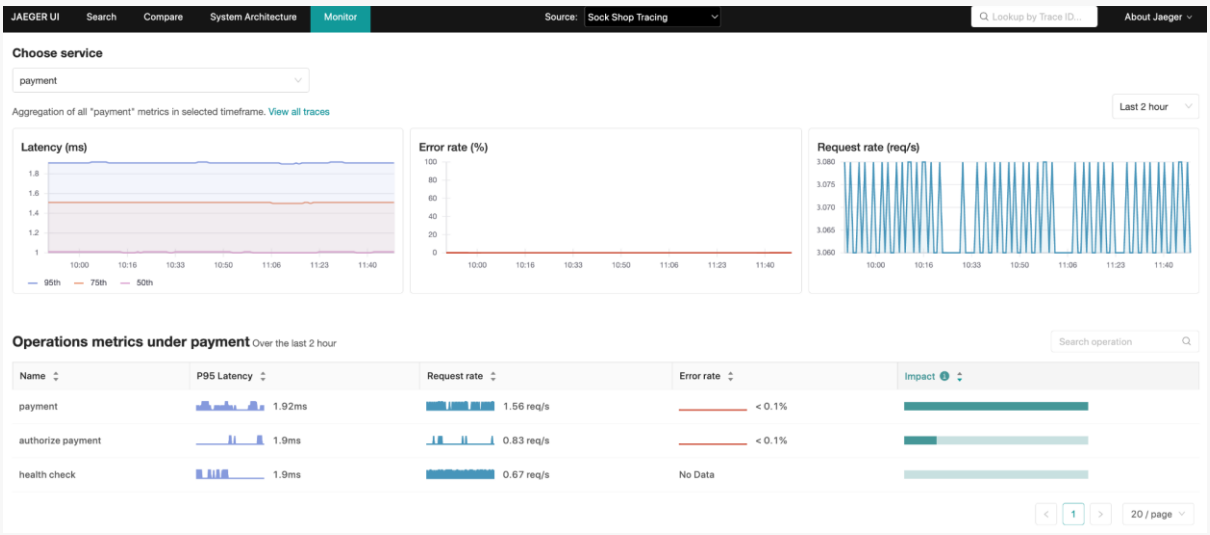
Figura 5 – Dashboard de exemplo no Sentry.



Fonte: https://sentry.io/about/media-resources/

Jaeger: É um software para rastreamento de transações. Geralmente utilizado em sistemas de alta complexidade.

Figura 6 – Service Performance Monitoring View.



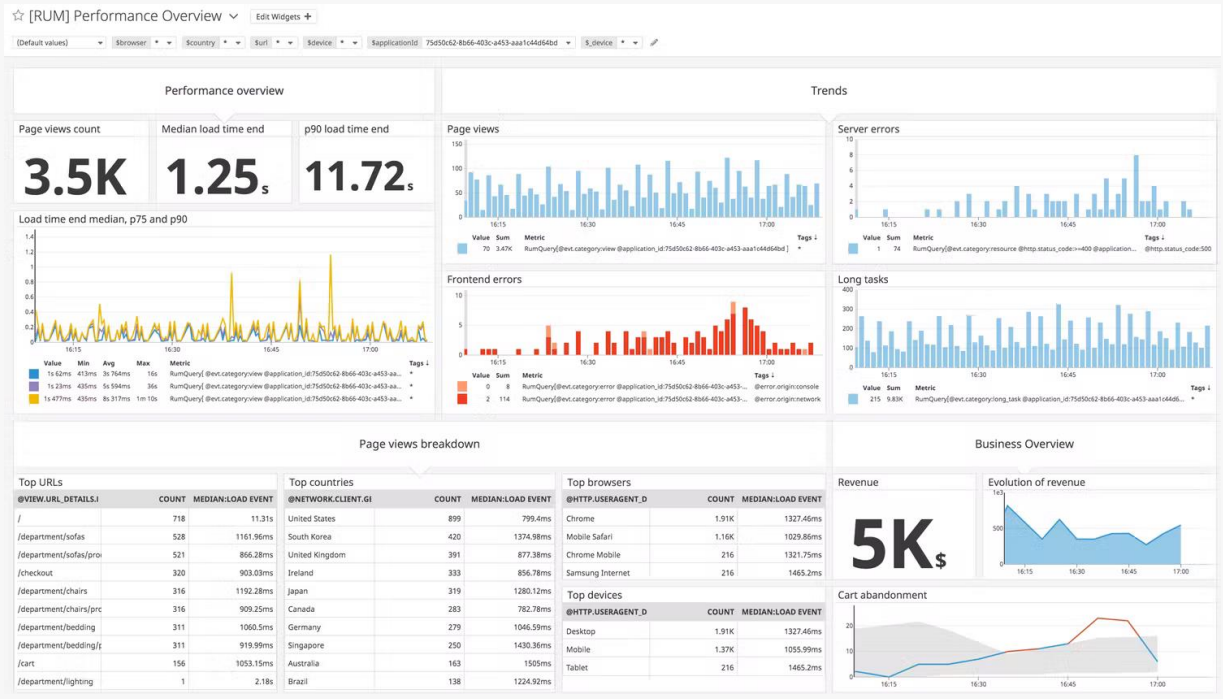
Fonte: https://www.jaegertracing.io/docs/1.36/#screenshots

Ferramentas que viabilizam o monitoramento 2
Ferramentas Comercializadas

Para muitos cenários as ferramentas *Open Source* fazem sentido, mas estas demandam pessoas dedicadas no time para sua manutenção e funcionamento. Por esse e outros motivos, as empresas buscam por ferramentas de monitoramento comercializadas como serviço. Abaixo há algumas destas ferramentas, suas principais características e exemplos práticos.

DataDog: É um *software* que entrega monitoramento de toda a aplicação e sua infraestrutura. Tem uma alta quantidade de integrações com outras ferramentas e *frameworks*. Suporta ambientes *multi-cloud*.

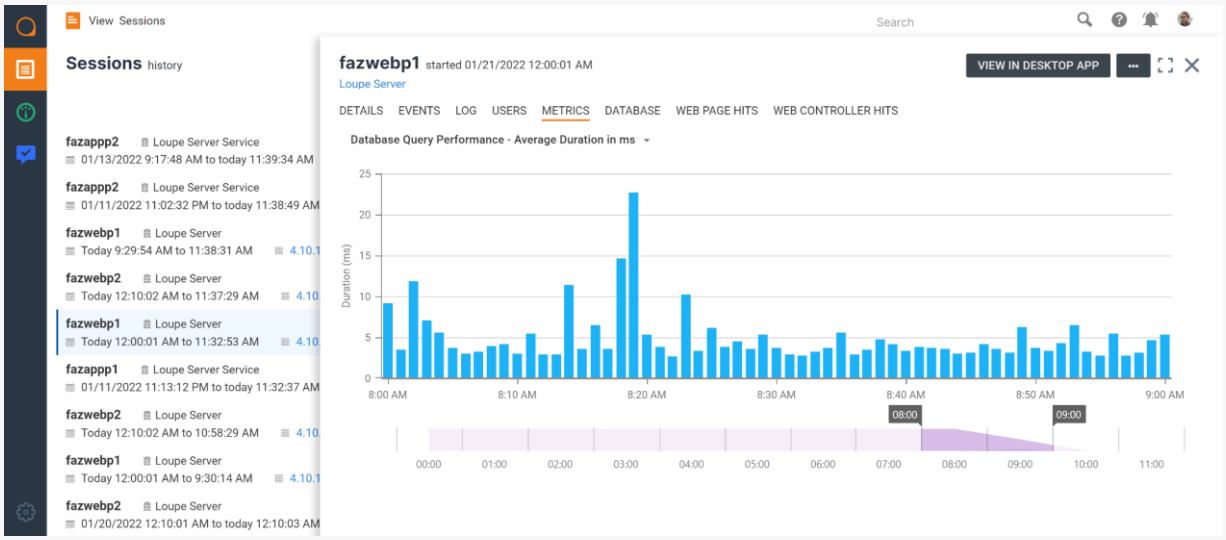
Figura 7 – Datadog Dashboard



Fonte: https://www.datadoghq.com/product/#dashboards

Loupe: A ferramenta pode ser utilizada em cliente *web* ou *desktop*. Tem foco em clientes empresariais e uma de suas principais funcionalidades é a retenção de logs e dashboards inteligentes.

Figura 8 – Loupe Dashboard



Fonte: https://onloupe.com/blog/loupe-5-view-introduction/

A importância da manutenção do monitoramento.

É de extrema importância que os times mantenham o funcionamento, a estabilidade e a confiabilidade das ferramentas de monitoramento em seus contextos de software e/ou aplicações. O próprio time técnico ou de negócio pode demandar novas necessidades de medição e/ou monitoramento. Portanto, regularmente há ajustes e melhorias a serem realizados. "Como Adrian Cockcroft ressaltou: "O monitoramento é tão importante que nossos sistemas de monitoramento precisam ser mais disponíveis e escalonáveis que os sistemas monitorados". (Gene Kim, 2018).

> Capítulo 4

Capítulo 4. Correção progressiva e reversão

Introdução

Num ciclo de vida de um *software*, por mais que práticas sejam aplicadas para que tenham qualidade, nenhum estará livre de *bugs*. Sendo assim, na ocorrência de *bugs* nas novas versões dos códigos, há técnicas que são aplicadas para retornar ao “estado sem defeito”.

Correção ou *Fix Forward*

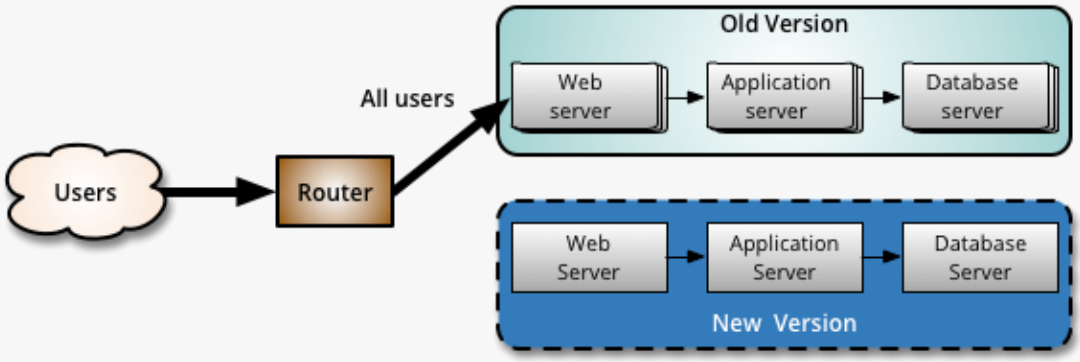
Essa técnica é, basicamente, corrigir o problema em produção. Após o deploy, quando o time identifica que há um problema, executa a correção em produção mesmo, sem reverter o incremento de versão que foi feito. Essa técnica deve ser utilizada por times de alto desempenho técnico e que tenham uma maturidade Devops considerável, tendo implantados processos de implantação rápida, alta cobertura de testes automatizados, e uma telemetria excelente.

Reversão ou Rollback (canary deploy. voltar a versão anterior.)

Durante um deploy, é inserida uma nova versão do software em produção. De repente alguém encontra um problema. O time rapidamente decide “dar um *rollback*" e voltar a versão anterior. O cenário descrito é comum em times de tecnologia, principalmente nos que estão ainda em busca de maturidade. Nesse cenário o usuário final deixa de aproveitar a nova versão até que ela seja corrigida e retornada ao ambiente produtivo.

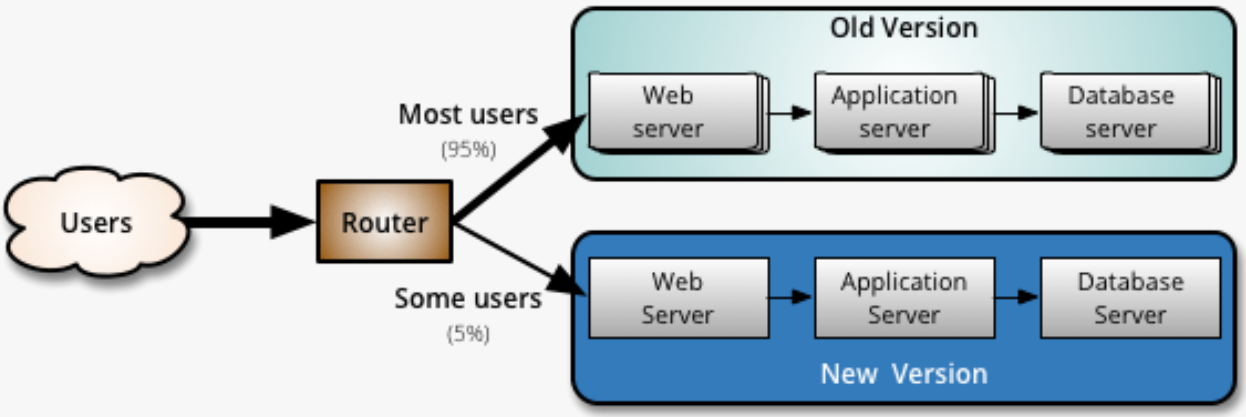
Uma estratégia mundialmente conhecida e regularmente aplicada nos deploys para evitar o *rollback,* e também o *fix forward* é o *canary deploy.* Nesta técnica, a nova versão é liberada apenas para parte da base de usuários e é observado o seu comportamento. Enquanto isso, a outra parte fica utilizando a versão estável. Caso não tenha problemas a versão pode ser promovida e liberada para o restante da base. Caso haja problema, a versão pode ser rejeitada ficando apenas a versão estável. Abaixo há imagens que demonstram a explicação.

Figura 9 – Atual versão do software sendo utilizada por 100% dos usuários.
Nova versão do software sendo disponibilizada.



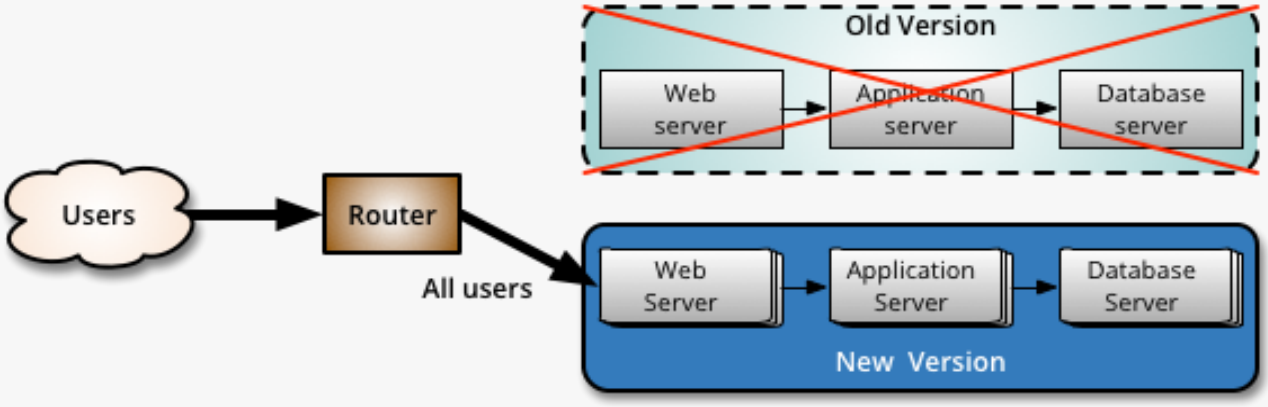
Fonte: https://martinfowler.com/bliki/CanaryRelease.html?ref=wellarchitected

Figura 10 – Atual versão do software sendo utilizada por 95% dos usuários.
Nova versão do software sendo utilizada por 5% dos usuários



Fonte: https://martinfowler.com/bliki/CanaryRelease.html?ref=wellarchitected

Figura 11 – Não ocorreram problemas na nova versão. Foi mudado o apontamento e a nova versão foi promovida a versão atual.



Fonte: https://martinfowler.com/bliki/CanaryRelease.html?ref=wellarchitected

> Capítulo 5

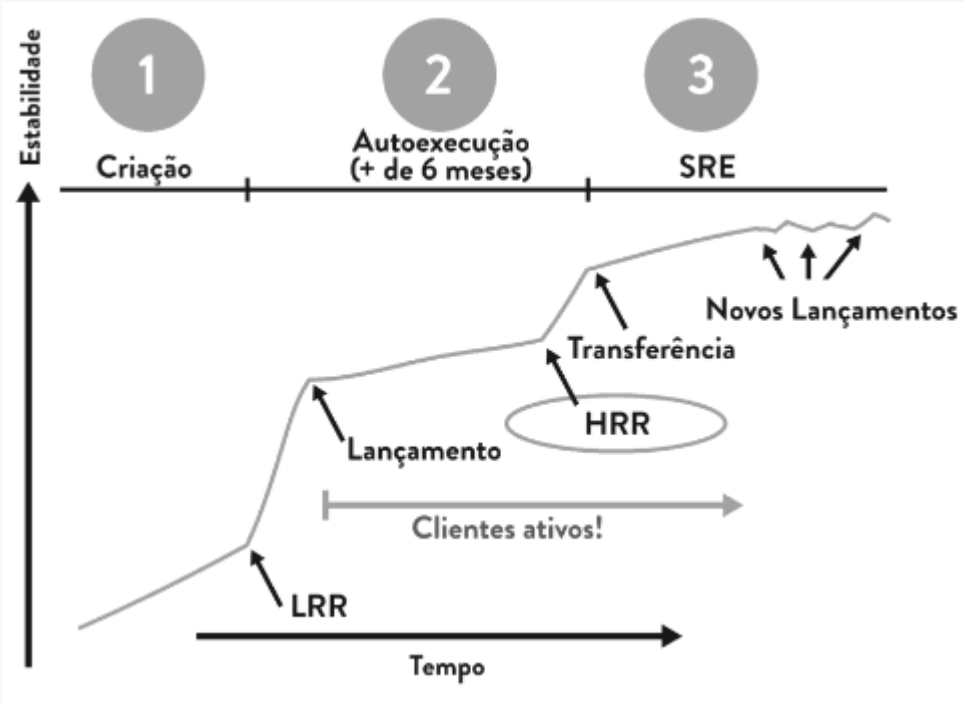
Capítulo 5. Verificações de segurança

Introdução

Ao lançar um novo serviço, a Google realiza duas verificações de revisão, são elas: LRR e HRR. A utilização de ambas as verificações requer do time colaboração e empatia, o que auxilia na extinção dos silos e aproxima os times de desenvolvimento e operações. Sendo que a HRR é mais rígida pois ela força os desenvolvedores a experimentar o trabalho de operações para melhorar o padrão que a transição de serviço faz desde a criação até o lançamento.

Verificações LRR e HRR.

Figura 12 – HRR e LRR na prática.



Fonte: "SRE@Google: Thousands of DevOps Since 2004", vídeo do YouTube, 45:57, postado por UNESIX, 12 de janeiro de 2012, https://www.youtube.com/watch?v=iIuTnhdTzK0.

Revisão de Prontidão de Lançamento (LRR) é uma lista de verificação de autorrelato que as equipes de produto fazem antes que o serviço seja lançado para os clientes.

Revisão de Prontidão sem Intervenção (HRR) acontece quando o serviço faz a transição para o gerenciamento de operações.

Há um estudo de caso da Google executando LRR e HRR, que foi publicado no livro Manual Devops em 2018. O mesmo pode ser visualizado em:
https://www.google.com.br/books/edition/Manual_De_DevOps/lFbzDwAAQBAJ?hl=pt-BR&gbpv=1&dq=manual%20devops&pg=PA239&printsec=frontcover

> Capítulo 6

Capítulo 6. User experience

Introdução e conceituação de UX

Os softwares que desenvolvemos e publicamos, em sua grande maioria são baseados na necessidade de um ou mais indivíduos. Do início ao fim precisamos nos basear em pessoas e em como gerar valor para elas.

User experience é um termo de origem nos anos 90 e se trata da experiência que o usuário tem quando usufrui de algo ou alguma coisa. Isso diz respeito desde a utilização de um caixa eletrônico para sacar dinheiro até a tentar editar um *heels* no *Instagram*.

Importância

Existem várias áreas de especialização envolvidas no ciclo de vida de um software. Sem dúvida a UX é extremamente importante nesse processo pois está ligada ao que as pessoas sentirão ao utilizá-lo. Estudar o usuário, o contexto em que o software está inserido, a linguagem que "fala", a forma como será visualizado, a necessidade que será atendida, entre outros pontos são responsabilidade dessa área de especialização. Por isso, a cada ano ela vem ganhando visibilidade no mercado e nas comunidades de tecnologia e produto.

Áreas de UX

Por ser uma área de conhecimento muito ampla a UX se fragmentou, ao longo dos anos, em áreas que juntas garantem uma boa experiência do usuário. Abaixo vamos sintetizar algumas delas.

Figura 13 – Diagrama de UX.

Fonte: UX Writing: How to do it like Google with this powerful checklist

- UX Writing: "... é escrever, revisar e otimizar textos com apoio de pesquisas e testes." (Velo,2022). Garantindo assim que os textos da aplicação seguirão a voz da marca.
- UX Design: Responsável por desenhar toda a jornada de experiência do usuário.
- Research: Pesquisar necessidades, características e dores dos usuários que são o público de uma aplicação. Por meio dos insumos das pesquisas, como o feedback, garantem melhoria contínua na experiência.

> Capítulo 7

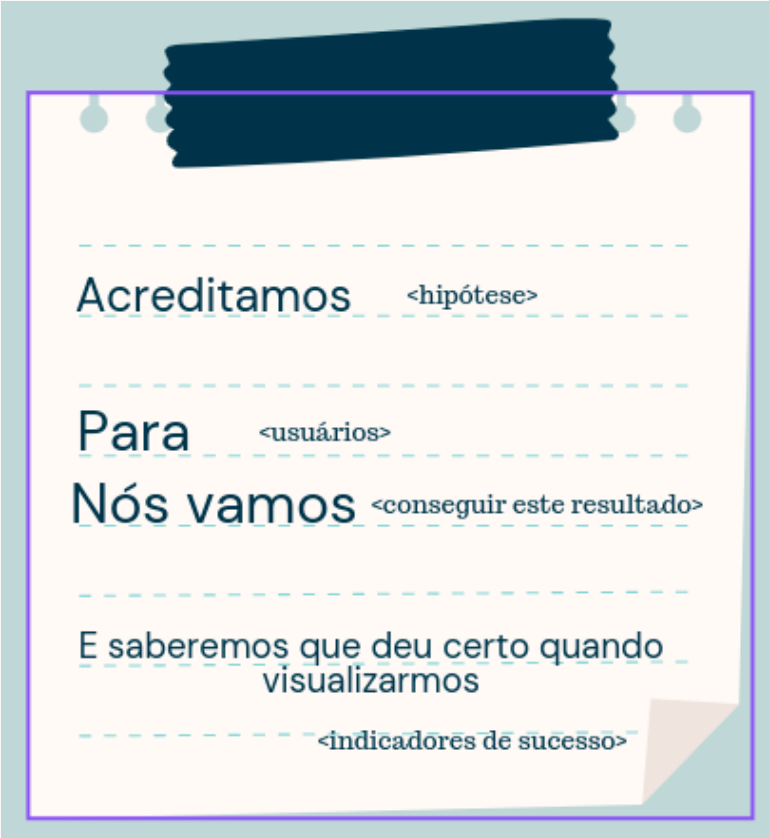
Capítulo 7. Desenvolvimento orientado à hipótese

Introdução

Em seus estudos, você já deve ter observado que o mundo de TI é uma grande sopa de letrinhas e já deve ter passado por uma série de siglas. Bom, aqui vai mais uma. O HDD (Hypothesis-Driven Development), que traduzido intitula esse capítulo. Segundo Flewelling (2018) "[...]nos permite adotar uma abordagem científica para entregar valor. [...]".

Nele, baseado em uma hipótese, o time inicia o desenvolvimento de uma feature acreditando que coletarão alguns resultados prédeterminados.

Figura 14 – HDD no contexto de user storie.



Fonte: Adaptado e traduzido pela autora, do livro *The Agile Developer's Handbook* de Paul Flewelling, publicado em 2018.

Valor agregado. Qual valor que o desenvolvimento orientado à hipótese traz?

É uma abordagem complexa que segue em constante evolução e que no geral tem um potencial de gerar intangíveis valores para organização ou produto em que se aplica. Respondendo diretamente à pergunta que intitula esse subcapítulo, um dos valores é a oportunidade de validar uma ou mais hipóteses que foram baseadas em dados concretos e muitas vezes em percepções e vivência dos usuários.

Teste A/B

O teste A/B consiste em disponibilizar para seus usuários 2 versões diferentes de uma mesma funcionalidade e através da observação, analisar qual versão converteu mais resultado. O planejamento do teste deve ser feito por times multidisciplinares que tenham em mãos os indicadores que desejam acompanhar e o que pretendem comprovar.

Na prática temos um exemplo de um aplicativo de delivery de comida. São disponibilizadas duas versões para os usuários, uma delas contém um carrossel sugerindo restaurantes específicos logo no início da tela, já a outra exibe esse calendário ao final da primeira tela. Até o final do teste é observado qual das duas versões converte mais vendas para os restaurantes exibidos.

“Técnicas como desenvolvimento guiado por hipóteses, definição e medida do funil de aquisição de clientes e testes A/B nos permitem fazer experiências com os usuários com segurança e facilmente, possibilitando pôr em ação criatividade e inovação, criando ainda aprendizado organizacional. “(KIM, 2018).

> Capítulo 8

Capítulo 8. Pair programming, code review e refatoração

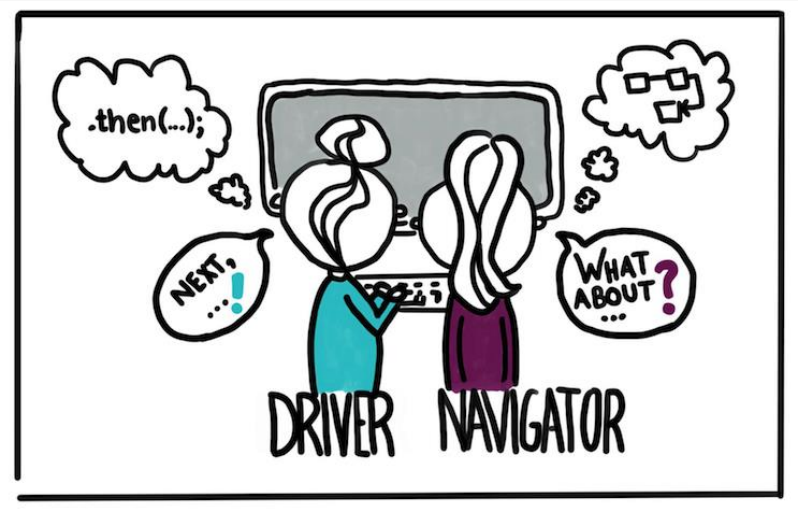
Introdução

Ao longo da história da tecnologia, diversas normas, práticas, técnicas, frameworks e abordagens foram sendo criadas e adotadas pelos times para viabilizar uma melhoria contínua das entregas, garantir a qualidade e promover o aprendizado coletivo do time. Algumas destas técnicas serão apresentadas neste capítulo.

Pair Programming

Muito presente na metodologia ágil Extreme Programing (XP), essa prática é a programação em pares. Ocorre na união de 2 pessoas desenvolvedoras para programarem juntas. Seja para resolver um problema oportuno ou concluir uma tarefa. As pessoas são divididas entre o *driver* e o *navigator.* O *driver* é o programador da vez e toma o controle da codificação, enquanto o *navigator* detecta erros, contribui com *insights*, etc.

Figura 15 – Nome da Tabela.



Fonte: Artigo On Pair Programming, disponível em: https://martinfowler.com/articles/on-pair-programming.html

Code Review

A revisão de código é quando uma pessoa “sobe” o código para o repositório e outras pessoas do time podem contribuir, questionar ou sugerir edições e melhorias. Na prática, ao abrir um MR, outras pessoas do time podem contribuir através de uma revisão.

> “[...]Muitas organizações exigem que o código seja revisado por outro engenheiro antes que possa ser enviado à base de código. Isso é um pouco como revisar o código; um segundo par de olhos geralmente detecta problemas que o autor do código não percebeu.”
>
> Long (2021)

Na figura 16 pode-se visualizar uma inserção de código com um primeiro comentário que era uma sugestão de edição. A autora explicou que para aquele contexto não fazia sentido essa edição. A thread foi resolvida e o MR aprovado.

Figura 16 – Exemplo prático de code review iterativo.

Fonte: Imagem cedida pela autora.

Refatoração

"Como muitos termos em desenvolvimento de software, o termo ''refatoração" com frequência é usado de forma bastante flexível pelos seus adeptos..." (Fowler, 2020). É dado que as necessidades dos usuários, tecnologias, cenário mundial e ideias mudam. Partindo disto, sabemos que a necessidade de refatoração de código será uma das certezas que surgem após o mesmo ser submetido na *branch* principal. Não há definição de quando, mas com certeza a refatoração será necessária em algum momento.

Refatorar também é tida pelo ato de editar o código existente com o objetivo de melhorar algum resultado. Alguns exemplos de melhorias são: melhoria de segurança, de estabilidade, de escalabilidade, de qualidade e de sintaxe.