

**RODRIGO AGUIAR ORDONIS DA SILVA**

**ABORDAGEM PARA O DESENVOLVIMENTO  
DE SOFTWARES ESCALÁVEIS FOCANDO EM  
DISPONIBILIDADE E DETECÇÃO DE FALHAS**

São Paulo  
2020

**RODRIGO AGUIAR ORDONIS DA SILVA**

**ABORDAGEM PARA O DESENVOLVIMENTO  
DE SOFTWARES ESCALÁVEIS FOCANDO EM  
DISPONIBILIDADE E DETECÇÃO DE FALHAS**

Trabalho apresentado à Escola Politécnica  
da Universidade de São Paulo para a con-  
clusão do MBA de Transformações Digitais.

São Paulo  
2020

**RODRIGO AGUIAR ORDONIS DA SILVA**

**ABORDAGEM PARA O DESENVOLVIMENTO  
DE SOFTWARES ESCALÁVEIS FOCANDO EM  
DISPONIBILIDADE E DETECÇÃO DE FALHAS**

Trabalho apresentado à Escola Politécnica  
da Universidade de São Paulo para a con-  
clusão do MBA de Transformações Digitais.

Orientador:

Reginaldo Arakaki

São Paulo  
2020

# SUMÁRIO

**Resumo**

**Abstract**

<b>1</b>	<b>Introdução</b>	<b>7</b>
1.1	Motivações . . . . .	9
<b>2</b>	<b>Metodologia de pesquisa</b>	<b>11</b>
<b>3</b>	<b>Objetivo</b>	<b>12</b>
<b>4</b>	<b>Fundamentos conceituais</b>	<b>13</b>
4.1	Análise geral sobre desenvolvimento de software . . . . .	13
4.1.1	Metodologias ágeis . . . . .	13
4.1.2	DevOps . . . . .	13
4.1.3	CI / CD . . . . .	13
4.1.4	Testes automatizados . . . . .	13
4.2	Qualidade de software . . . . .	13
4.2.1	Requisitos funcionais . . . . .	13
4.2.2	Requisitos não funcionais . . . . .	13
4.2.3	Escalabilidade . . . . .	13
4.2.4	Disponibilidade . . . . .	13
4.2.5	Falhas no projeto . . . . .	13
<b>5</b>	<b>Proposta</b>	<b>14</b>
5.1	Como criar produtos com qualidade? . . . . .	15

5.1.1	Assumindo riscos . . . . .	15
5.1.2	Entendendo os requisitos de uma funcionalidade . . . . .	15
5.1.3	Arquitetura evolutiva . . . . .	15
5.1.4	Estruturando o fluxo de entrega . . . . .	15
5.1.5	Automatizando testes de qualidade . . . . .	15
5.2	Como adquirir escalabilidade? . . . . .	15
5.2.1	Descobrimos os limites do sistema . . . . .	15
5.2.2	Desenvolvendo estratégias de escalabilidade . . . . .	15
5.3	Como manter o sistema disponível? . . . . .	15
5.3.1	Utilizando o negócio para definir disponibilidade . . . . .	15
5.3.2	Analisando o desempenho do sistema . . . . .	15
5.3.3	Falando com os usuários . . . . .	15
5.3.4	Utilizando métricas para assegurar a disponibilidade . . . . .	15
5.3.5	Aplicando testes automatizados . . . . .	15
5.4	Como identificar falhas? . . . . .	15
5.4.1	Sentindo o cheiro do produto . . . . .	15
5.4.2	Utilizando o usuário para identificar falhas . . . . .	15
5.4.3	Solucionando falhas . . . . .	15
5.4.4	Aprendendo com os erros . . . . .	15
5.4.5	Testes automatizados como ferramenta de aprendizado . . . . .	15
5.4.6	Divulgando as descobertas de forma global . . . . .	15
<b>6</b>	<b>Resultados da proposta</b>	<b>16</b>
6.1	Um produto escalável . . . . .	16
6.1.1	Um produto com custo dinâmico . . . . .	16
6.2	Um produto disponível . . . . .	16
6.3	Um produto com falhas planejadas . . . . .	16

<b>7</b>	<b>Conclusão</b>	<b>17</b>
7.1	Resultados em relação ao objetivo . . . . .	17
7.2	Trabalhos futuros . . . . .	17
<b>8</b>	<b>Referência Bibliográfica</b>	<b>18</b>

# RESUMO

# ABSTRACT



# 1 INTRODUÇÃO

*“A confiança perdida é difícil de recuperar. Ela não cresce como as unhas.”*

-- Brahms, Johannes

Acreditamos que um produto escalável e uma equipe que se preocupe em aprender com erros e manter o sistema sempre disponível, é capaz de criar produtos de alta qualidade e grande sucesso.

Como já é apresentado na Engenharia de Confiabilidade do Google ([\*]), devemos esquecer a cultura de culpar as pessoas por erros cometidos e adquirir uma cultura de aprendizado, utilizar do erro para entender como ele ocorreu e assim produzir um produto com maior qualidade e adquirir o conhecimento para que o erro não ocorra novamente ou ocorra em outros produtos. Mas, adquirir essa maturidade e esse *mindset* é um processo, não é algo imediato, e é necessário entender em qual momento está a equipe e como implementar estes processos.

De acordo com o The DevOps Handbook ([\*]), esse processo de transformação é dividido em três partes, o fluxo, *feedback* e, aprendizado contínuo e experimentação. A primeira parte é sobre definirmos o nosso fluxo de trabalho, os nossos processos e com isso automatiza-los, com a criação do *pipeline*, realização de testes automatizados, estruturação da arquitetura do *software* e conseguindo realizar entregas de baixo risco. A segunda parte é adquirir a prática do *feedback*, cujo objetivo é o de aprender com os nossos erros, tentar solucionar o mais rápido possível e logo após, realizar uma reunião com todos os envolvidos e interessados para adquirir entender o que ocorreu, quais foram as medidas tomadas e o que fazer para não ocorrer novamente, sempre lembrando que o objetivo não é encontrar um culpado, mas sim aprender com o ocorrido. Outro fator importante é o de investir tempo em melhorar a telemetria que temos sobre o sistema, trabalhar em quais informações precisamos para supervisionar o funcionamento do sistema e evitarmos que problemas ocorram, como por exemplo, manter o controle da performance de cada servidor utilizado para o funcionamento da aplicação, quando um deles estiver com o nível de processamento abaixo do esperado, podemos analisar se ele não está sendo aproveitado, e assim realocar recursos para melhorar o desempenho do sistema, ou se ele está apresentando sinais de mau funcionamento. A terceira parte é o de aprendizado contínuo e experimentação, ela

nos incentiva a realizar experimentos no nosso produto afim de adapta-lo e gerarmos conhecimento para o negócio, um exemplo de experimento, unificar três telas em uma só e vêr como o usuário reage a esta mudança, dependendo do resultado, podemos aumentar a produtividade do usuário no sistema, ou descobrir que ele isso o deixou mais confuso, em ambos os casos aprendemos sobre o comportamento do nosso usuário e isso pode ser utilizado para futuras demandas e resoluções de erros. Outro ponto importante sobre essa parte é a de gerar conhecimento, assim que algo é descoberto, devemos apresentar para todos do time e caso seja algo que possa ser utilizados em outros projetos, escalar esse aprendizado para a empresa, onde devemos considerar que devemos demonstrar todo tipo de conhecimento, seja uma descoberta comportamental do usuário, a resolução de algum defeito, um erro cometido durante o projeto, qualquer aprendizado deve ser levado em consideração e, devido a isso, não podemos culpar as pessoas por erros cometidos. Quando culpamos alguém, inibimos a pessoa de demonstrar o erro, consequentemente, perdemos uma importante fonte de aprendizagem, desta forma, perdemos a oportunidade de melhorarmos nosso produto, nossa equipe e nosso trabalho.

Um dos problemas mais urgentes em nosso sistema e que precisamos procurar evitar o máximo possível são os de disponibilidade. Um sistema que não está disponível, perde credibilidade, o negócio é impactado de forma direta e os usuários que necessitam dele, acabam se estressando por não poder fazer nada. Lidar com a disponibilidade do sistema está diretamente relacionada com assumir riscos, entender que haverá momentos críticos no sistema que precisamos nos preparar e que sempre há a chance de falharmos e do sistema cair. Precisamos através das necessidades do negócio e da utilização do usuário aprender como podemos manter a disponibilidade. Primeiramente, disponibilidade não pode ser considerado simplesmente como "no ar ou fora do ar", uma funcionalidade que demore para ser executada, pode motivar o usuário a não utiliza-la, o que prova que ela não está disponível, vamos supor, que estamos em um sistema que sejam realizadas negociações, o usuário está negociando com um cliente e propõe um desconto para o produto que ele está tentando vender, nosso sistema tem uma função de simulação de desconto, então nosso vendedor logo tenta simular, porem, como é um produto grande, essa simulação está sendo processada já a cinco minutos, e o cliente já está cansado de esperar, como o ele não teve a simulação em tempo hábil e o vendedor dependia desse valor para continuar a negociação, decidiram continuar a conversa outro dia, já com o valor em mãos. Após este episódio, nosso vendedor perdeu a confiança no sistema e decidiu não mais utilizar a funcionalidade de negociação, embora a simulação tenha funcionado muito bem para produtos pequenos, quando o nosso vendedor mais precisou, ele não atendeu as expectativas. Cinco minutos foi o suficiente para indisponibilizar essa funcionalidade para

esse usuário, que como está aborrecido, vai passar esta frustração para outros usuários. Se não tivermos a informação de que há este problema no sistema, seja por *feedback* ou por alguma métrica, jamais saberemos deste erro e com o passar do tempo esta funcionalidade será esquecida.

Outro ponto a ser considerado é o preço do *software*, podemos manter nosso sistema sempre disponível se tivermos uma grande quantidade de servidores, armazenamento infinito, e vários domínios de redundância, o custo para manter todos esses recursos é inviável, precisamos sempre controlar o que o sistema precisa e o que a empresa pode pagar. Controlar os custos é uma tarefa desafiadora, precisamos ver o qual o retorno que estamos recebendo com a aplicação e quais são os riscos que podemos enfrentar. Não adianta montar um sistema indestrutível se ele não gera valor para o negócio, e não adianta deixar o sistema vulnerável para controlar custos, a queda de um produto deve ser considerado como uma descredibilidade da empresa, pois é uma se torna um símbolo de falta de qualidade. O meio termo seria montar um produto escalável, conseguir controlar o quanto de recurso disponibilizar, saber quais são as épocas em que precisamos do sistema funcionando, quais são as funcionalidades mais críticas e quais riscos podemos aceitar. Tomar as nossas decisões com base nos riscos, nos dá segurança e preparo para as mais adversas situações, como planejamos que pode acontecer erros, conseguimos nos antecipar e solucionar o mais rápido possível, gerando pouco impacto para o negócio. Sabendo onde estão os pontos fracos do nosso produto, podemos prever que um problema ocorra antes que ele aconteça, e acompanhando as métricas diariamente, quando os primeiros sinais de que vai ocorrer um erro, podemos reavaliar os riscos e assumir uma postura de mitigação ou solução. Admitindo que nosso produto não é perfeito e aceitando falhas, podemos trabalhar na melhoria contínua. Com a possibilidade de controlar os riscos, podemos investir financeiramente conforme a necessidade do negócio, muitas vezes é mais viável comunicar que determinada funcionalidade vai para de funcionar por um período, do que realocar recursos para manter em funcionamento.

## 1.1 Motivações

Nossas motivações se definem em criar e manter produtos escaláveis, possibilitando o controle de seu custo com base na utilização dos usuários e na situação da empresa, demonstrando como utilizar a interação dos usuários para aprender como melhorar e engajando a criação de testes automatizados como um processo de aprendizagem, detecção de falhas e de controle de qualidade.

Pretendemos demonstrar técnicas para a identificação de erros e como aprender com eles, possibilitando que o seu produto evolua e que os aprendizados auxilia na criação de outros produtos. Desta forma, a busca por valor não se limita a apenas a um sistema, beneficiando o negócio com outras visões e ampliando a inteligência de mercado na empresa. A abordagem apresentada não tem como objetivo criar produtos perfeitos, sem erros, e que vão conluir todos os objetivos da empresa, mas utilizar do produto e da interação do usuário para montar *sfotware* de qualidade aproveitando o aprendizado obtido para auxiliar na evolução do negócio, gerar valor para a empresa e auxiliar na criação de novos produtos.

## **2 METODOLOGIA DE PESQUISA**

### 3 OBJETIVO

Este trabalho tem como objetivo apresentar uma abordagem de como construir sistemas escaláveis, com foco em assegurar performance e detecção de falhas. Criando *softwares* com qualidade, gerando valor e que com base nos retornos do sistema, possibilitando novas visões de negócio e ocasionando no desenvolvimento de novos produtos.

## 4 FUNDAMENTOS CONCEITUAIS

### 4.1 Análise geral sobre desenvolvimento de software

#### 4.1.1 Metodologias ágeis

#### 4.1.2 DevOps

#### 4.1.3 CI / CD

#### 4.1.4 Testes automatizados

### 4.2 Qualidade de software

#### 4.2.1 Requisitos funcionais

#### 4.2.2 Requisitos não funcionais

#### 4.2.3 Escalabilidade

#### 4.2.4 Disponibilidade

#### 4.2.5 Falhas no projeto

## 5 PROPOSTA

*“Software funcionando é a principal medida do progresso.”*

-- Manifesto Ágil



## 5.1 Como criar produtos com qualidade?

5.1.1 Assumindo riscos

5.1.2 Entendendo os requisitos de uma funcionalidade

5.1.3 Arquitetura evolutiva

5.1.4 Estruturando o fluxo de entrega

5.1.5 Automatizando testes de qualidade

## 5.2 Como adquirir escalabilidade?

5.2.1 Descobrindo os limites do sistema

5.2.2 Desenvolvendo estratégias de escalabilidade

## 5.3 Como manter o sistema disponível?

5.3.1 Utilizando o negócio para definir disponibilidade

5.3.2 Analisando o desempenho do sistema

5.3.3 Falando com os usuários

5.3.4 Utilizando métricas para assegurar a disponibilidade

5.3.5 Aplicando testes automatizados

## 5.4 Como identificar falhas?

5.4.1 Sentindo o cheiro do produto

5.4.2 Utilizando o usuário para identificar falhas

5.4.3 Solucionando falhas

5.4.4 Aprendendo com os erros

5.4.5 Testes automatizados como ferramenta de aprendizado

5.4.6 Divulgando as descobertas de forma global

## **6 RESULTADOS DA PROPOSTA**

### **6.1 Um produto escalável**

#### **6.1.1 Um produto com custo dinâmico**

### **6.2 Um produto disponível**

### **6.3 Um produto com falhas planejadas**

## 7 CONCLUSÃO

### 7.1 Resultados em relação ao objetivo

### 7.2 Trabalhos futuros

## 8 REFERÊNCIA BIBLIOGRÁFICA