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 conclusão do MBA de Transformações Digítais.

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 conclusão do MBA de Transformações Digítais.

Orientador:

Reginaldo Arakaki

São Paulo
2020

SUMÁRIO

RESUMO

ABSTRACT

1 INTRODUÇÃO

"A confiança perdida é difícil de recuperar. Ela não cresce como as unhas."

-- Brahms, Johannes

Com o passar do tempo, os *softwares* firacam cada vez mais importantes na sociedade e os negócios começaram a depender ainda mais deles. Alem dos computadores, há diversos outros dispositovos que nos permitem acesso a internet como por exemplo os celulares, os *tablets*, *video games*, as televisões, e entre vários outros dispositivos. O que facilitou o acesso a informação e consequntemente, a divulgação da informação. Com os sistemas ficando mais importantes, diversos serviços são realizados pela internet, como compras, negociações, comunicação, tranferências bancarias, entre outros.

Devido a isso, questões como quantidade de acesso, tempo de resposta e segurnaça da informação começaram a ficar cada vez mais importantes na concepção de um *software*. A queda de um sistema por alguns segundos, pode ocasionar em diversos problemas para uma empresa como, a perda de milhões de reais, a desvalorização da marca e causar uma reputação negativa para a empresa. Dependendo do motivo da queda, pode ocasionar o fim do sistema e por consequência o fim de um produto para a empresa. Outro ponto a ressaltar é que devemos entender se mesmo com o sistema funcionando corretamente, ele realmente está trazendo retornos positivos para a empresa? O usuário está gostando do que disponibilizamos para ele? Está falando bem ou mal do produto?

Para se preparar para estas situações inesperadas e manter o *software* funcionando, é importante manter o produto escalável. Não podemos controlar a quantidade de acessos no sistema, mas podemos controlar o quanto de recursos computacionais é disponibilizado para o *software*, em épocas em que os acessos aumentam, podemos aumentar a quantidade de processamento, em épocas que diminuem, podemos diminuir o processamento, assim controlamos o custo do sistema e garantimos sua disponibilidade. Não podemos tambem ter um *feedback* de todos os usuário de como está a experiência na utilização, muitas vezes nem da maioria deles, mas podemos analisar as ações dos usuários para entender como está sendo a sua experiência.

Outro ponto que não deve ser esquecido, é de que problemas vão acontecer, situações inesperadas que não vamos saber lidar no momento, brechas na segurança, um *bug* no

sistema, usuários perdidos na navegação, entre outros casos. Nestes casos devemos aprender com os problemas, descobrindo como identifica-los e corrigi-los. Uma vez descoberto, aplicamos testes automatizados para que eles não voltem a acontecer.

1.1 Motivações

Como podemos notar, criar um *software* se tornou uma tarefa complexa, garantir sua qualidade se tornou uma taréfa difícil, devemos realizar diversas ações para assegurar que nosso sistema vai gerar valor. Foi neste cenário que nos sentimos motivados.
Nossas motivações se definem em criar e manter produtos escaláveis, consequntemente, possibilitar 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 utilização de testes automatizados como um processo de aprendizagem, detecção de falhas e de controle de qualidade.

2 METODOLOGIA DE PESQUISA

3 OBJETIVO

Este trabalho tem como objetivo apresentar uma aborgagem 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 projetos de software

4.1.1 Waterfall

4.1.2 Metodologias ágeis

4.1.3 DevOps

4.1.4 CI / CD

4.1.5 Testes automatizados

4.2 Qualidade de software

4.2.1 Requisitos funcionais

4.2.2 Requisitos não funcionais

4.3 Escalabilidade

4.3.1 Disponibilidade

4.3.2 Falhas no projeto

5 PROPOSTA

"Mais sofreu a mãe do porco espinho."

-- Pai, Meu

Nossa proposta é apresentar uma abordagem para a construção de *software* escaláveis incentivando o descobrimento e o aprendizado de erros e com atenção na disponibilidade do sistema. 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 divido em três partes, o fluxo, *feedback* e, aprendizado continuo 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 telemtria 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 continuo 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 aprendisado 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 realizdas 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

expectivas. 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 vário dominios de redundancia, 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 adaianta 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 simbolo 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 melhora continua. 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.

5.1 Como adquirir escalabilidade?

5.2 Como manter o sistema disponível?

5.3 Como identificar falhas?

5.3.1 O que fazer com as falhas identificadas?

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