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

RESUMO

ABSTRACT

LISTA DE TABELAS

SUMÁRIO

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 valor.

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.

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 usuáiro 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

Para a realização deste trabalho foi utilizado livros, artigos, *papers* e notas técnicas para a criação de sistemas com qualidade. Procuramos por trabalhos que descrevescem o que é qualidade de *softwares* e como construir os produtos com qualidade. Durante a pesquisa foi identificado a immportância dos requisitos não-funcionais o que nos levou a procurar por trabalhos que os apresentassem e demonstrassem técnicas para assegurarar que eles fossem cumpridos.
Foi utilizado como ferramenta de pesquisa o Google Scholar, ResearchGate e o Software Engenieering Institute da Carnegie Mellon University.

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 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

5 PROPOSTA

"Não tente se tornar uma pessoa de sucesso,
prefira tentar se tornar uma pessoa de valor."

-- Albert Einstein

5.1 Como criar produtos com qualidade?

Quando produzimos um *software*, focamos sempre nas funcionalidades que o sistema deve conter, pretendemos entregar o mais rápido possível e respeitar as datas alinhadas com o negócio. Porem, na pressa de colocar as funcionalidades em produção, desconsideramos os requisitos não-funcionais e por mais que a funcionalidade tenha sido entregue conforme o específicado, o usuário não consegue utiliza-la, consequentemente, está entrega não está gerando valor.

Mas como assim, os requisitos foram atendidos mas o usuário não consegue utilizar? Como ignoramos os requisitos não-funcionais, o usuário pode estar sofrendo por diversos problemas, por exemplo, um *layout* confuso, um desempenho baixo, o sistema está forçando ele a realizar uma tarefa que ele não está acostumado a fazer, ou pelo menos não no momento em que apareceu no sistema. Como entregamos vizando a data da entrega e nãp o valor para o usuário, aceitamos o risco de entregar algo que o usuário não vê valor, e por diversas vezes cometemos erros. Ao testar uma funcionalidade, não costumamos ter a visão total do negócio, focamos na funcionalidade, então questões como tempo de resposta, ordem lógica no processo, localização das informações, nos passam despercebidos, não estamos utilizando o sistema diariamente para entender estas questões sozinhos, e o negócio muda frequentemente, então quando finalmente entendemos a realidade do usuário, ela já mudou, e voltamos a estar sucetiveis ao erro.

Como não atendemos a expectativa do usuário, e a funcionalidade não está gerando valor, implementamos diversas melhorias, todas o mais rápido possível, o que acaba gerando *bugs*, como erramos uma vez, precisamos corrigir o erro o mais rápido possível, o que nos faz ignorar novamente requisitos não-funcionais e, por sua vez, ignoramos questões de arquitetura de *software* o que acaba deixando o sistema mais complexo, dificulta a sua manutenção, e deixa o sistema mais sucetível a erros.

Não queremos mais cometer estes erro, queremos cria o produto perfeito que será total-

mente aderente ao usuário e que irá gerar valor para o negócio.
O primeiro passo para gerar um produto de qualidade, que auxilia a empresa a concluir os seus objetivos e que traga valor ao negócio e entender que não existe produto perfeito. Somos humanos, erramos constantemente, realizamos escolhas erradas, nos equivocamos. Para realizar uma entrega de pouco risco e extremamente acertiva, é necessário muito tempo de planejamento e de estudo, como por exemplo a construção de uma avião, ou de um prédio, ou de um equipamento hospitalar, este tipo de produto não pode ter falhas, pois caso tenha, é um risco para a vida das pessoas. Porem, nosso negócio é mais dinâmico, as necessidades muda com mais frequência do que as necessidades de um avião, se utilizarmos tanto tempo para planejar como podemos seguir as mudanças do negócio?

5.1.1 Assumindo riscos

Para conseguirmos acompanhar o negócio e gerar valor para a empresa, evemos escolher os riscos que vamos enfrentar. Mesmo um avião encara riscos em seu projeto, por isso que é implementado diversas redundancias nas funcionalidades mais importantes. Devemos pensar de forma semelhante, qual o objetivo do nosso projeto? Qual é sua funcionalidade mais importante? Quais são os riscos que vamos encarar?
Com base nisso, podemos focar a maior parte de nosso tempo no *core* do produto, descobrindo o objetivo do sistema, podemos alinhar com os objetivos da empresa, assim gerando o retorno esperado. Para realizar o levantamento da funcionalidade é importante considerar o nome da funcionalidade, a sua importância, os objetivos em que ela vai nos auxiliar a alcançar, os riscos que vamos enfrentar com ela e as dependências para disponibilizar a funcionalidade ao usuário.

Funcionalidade	Nome da funcionalidade que será desenvolvida.
Importância	A importância que a funcionalidade representa para o negócio, separa entre alta, média e baixa.
Objetivos	Os objetivos que está funcionalidade irá auxiliar à alcançar.
Riscos	Quais os riscos que essa funcionalidade pode gerar para o negócio.
Dependências	Quais são as dependências para a utilização e desenvolvimento dessa funcionalidade.

Tabela 1: Definição da tabela de funcionalidaes

Toda funcionalidade deve ter um nome que represente o que será desenvolvido pois, durante o desenvolvimento, é através do nome que os desenvolvedores vão se comunicar, sem um nome claro, que represente bem o negócio, os desenvolvedores não vão conse-

guir se comunicar de forma clara para entender as necessidades do negócio. Termos e nomeclaturas devem ser estabelecidas, para que o usuário entenda como o sistema deve ser utilizado e para os desenvolvedores compreenderem como o sistema será utilizado. Através desta comunicação, o time de desenvolvimento consegue extrair os requisitos de forma mais fácil e formular um padrão de qualidade.

É necessário colocar a importância da funcionalidade que será desenvolvida pois, com base nela podemos entender a ordem que vamos iniciar o desenvolvimento e quais os riscos podemos enfrentar e através da importância podemos definir o rigor dos testes que devemos realizar no desenvolvimento da funcionalidade e o padrão de qualidade para cada nível de importância. Embora colocamos somente três níveis de importância (alto, médio e baixo), cada negócio pode ter mais níveis, porém recomendamos não exagerar e passar de cinco níveis, pois o principal objetivo desta classificação é forçar escolhermos quais funcionalidades são realmente mais importantes, se colocarmos muitos níveis, a menor classificação será "alta".

As funcionalidades que vamos desenvolver tem que estar relacinada com um objetivo da empresa, pois é isso que dá propósito para ela, e a nossa base para verificar se ela está gerando valor. Com base no uso da funcionalidade, podemos verificar se o objetivo está sendo alcançado, e com base nesse retorno, podemos decidir quais serão os nossos próximos passos.

Quando pensamos em uma funcionalidade, devemos sempre considerar os riscos para o negócio, a utilização de um sistema implica em com a operação da empresa trabalha, e toda mudança gera riscos para a operação. Tudo que é novo, deve ser ensinado, por mais que a funcionalidade reflita exatamente o que os usuários já faziam, eles vão começar a eecutar essas atividades em um outro lugar, o que no começo pode gerar dúvidas e frustrações. Devemos sempre levantar os riscos com base na perspectiva do usuário, pois assim podemos entender quais serão as reações delese e quais estratégias devemos utilizar para engajar o uso da ferramenta e buscar por melhorias.

Com base nos objetivos, nas funcionlidades e nos riscos, podemos mapear as pendências das funcionalidades, onde conseguimos traçar quais os passos que devemos tomar para iniciar o desenvolvimento até disponibilizar a funcionalidade para o usuário.

Para o levantamento dessas informações recomendamos utilizar o formato de tabela. A tabela possibilita consolidar de forma clara cada tópico e sempre que a empresa tiver um novo objetivo, é mais fácil analisar o que já foi listado, possibilitando ver se o sistema se encaixa com essa nova necessidade ou se será necessário uma nova funcionalidade, ou se será necessário um novo produto, ou uma reformulação do negócio. Quando listamos tudo que estamos fazendo junto com o que o negócio pretende alcançar, podemos tomar

decisões mais acertivas, entender dependências e otimizar o valor que será desenvolvido. Muitas vezes focamos em desenvolver uma funcionalidade que não irá gerar muito valor para o negócio, o que nos faz aceitar um risco que não precisamos no momento, toda funcionalidade gera riscos, então é melhor nos concentrarmos somente naqueles que podem gerar mais valor.
Uma vez definido nossas funcionalidade, sua importância e os objetivos que pretendemos com elas, podemos escolher quais vamos implementar primeiro e quais riscos nós vamos enfrentar.
Para exemplificar a utilização da tabela proposta vamos considerar uma empresa que possuí vendores que entram em contato com outras empresas para negociar a venda de computadores. Estes computadores podem ser customizados pelo cliente, solicitando maior espaço de memória, se deseja um computador com *SSD* ou *HD*, qual o processador, entre outras ecolhas. Embora já existam computadores pré-montados, eles podem ser customizados somente aproveitando a base do produto.
Hoje a venda é feita sem a utilização de um sistema, a especificação que o cliente deseja é feita de forma indivídual por cada vendedor e controlado por meio de planilhas, onde cada vendedor possui a sua forma de organizar. Somente o pedido final é colocado em um sistema de controle de pedidos, para emitir a nota. A comunicação entre os vendedores e a área técnica é feita por meio de telefone e email, os produtos base e os produtos vendidos são compartilhados através de planilhas enviadas por email. Como cada vendedor negocia de uma forma, nã há Padronização nas planilhas, o que acaba gera confusões na área técnica na hora da montagem dos produtos. Outro ponto a destacar é que a área administrativa não tem visibilidade de como as negociações estão sendo realizadas o que dificulta o entendimento do negócio como um todo e a cração de estratégias.
Com base nessas necessidades, a empresa decidiu começar um projeto para a criação de um sistema para os vendedores realizar as suas negociações. Onde o produto base seria disponibilizado na ferramenta e os pedidos finais seriam montados conforme a negociação avança. Outra necessidade, é que a negociação deve ser faseada, para que seja possível rastrear os passos do vendedor e ter uma melhor visão do negócio.
Neste cenário, o primeiro passo é levantar as princípais funcionalidades para atender o negócio, já colocando a sua importância e os objetivos que pretendemos alcançar.

Na tabela dois, representamos três funcionalidades, o cadastro de produtos, a venda de produtos, e o *chat* interno. Podemos notar que a maior necessidade da empresa é a de padronizar a venda do produto, onde a negociação deve ser feita de forma mais uniforme e que facilite os vendedores a venderem produtos mais aderentes com o que a área

Funcionalidade	**Importância**	**Objetivos**
Cadastro de produtos	Alta	Padronização na criação de produtos; Reduzir o cadastro incorreto de produtos.
Venda de produtos	Alta	Venda faseada do produto; Melhor rastreabilidade da venda; Padronização da negociação.
Chat interno	Baixa	Otimização da comunicação entre a área técnica e os vendedores.

Tabela 2: Exemplo de levantamento de funcionalidades

técnica consegue produzir. Com base nestas necessidades, foi levantado as funcionalidades de cadastro de produtos e venda de produtos, onde o cadastro está relacionada com a padrinização da criação do produto e a venda está relacionada com a padronização da venda. Porem, a funcionalidade de *chat* interno, não é uma prioridade para o negócio, as áreas estão se comunicando, o problema é que cada vendedor trabalha de um jeito, o que dificulta as decisões da área técnica. Embora um dos objetivos seja melhorar a comunicação entre as áreas, este não é o maior dos problemas que precisamos resolver, então os riscos que desta funcionalidade não precisam ser corridos até o desenvolvimento das outras duas.

Outro ponto de destaque é a identificação das dependências, quando mapeamos as dependências, conseguimos ver qual funcionalidade precisamos realizar primeiro, como listado no exemplo, não é possível vender os produtos sem antes cadastrá-los. Mas alem da dependências entre funcionalidades, podemos listar o que precisamos fazer para disponibilizar a funcionalidade para o usuário, como apresentado no exemplo, todas as funcionalidades precisam de treinamento, como o sistema é novo e cada vendedor realiza as negociações do seu jeito, é importante explicar como foi montado o processo de vendas e apresentar como realizar as atividades no sistema.

Cada funcionalidade tem os seus riscos, porem é através de sua importância que podemos criar estratégias para enfrentar estes riscos e marcar estas ações que definimos em nossa estratégia como dependências para liberar o uso da funcionalidade, é importante atrelar estas ações como dependências, pois desta forma os usuários não são surprendidos, e é através dos treinamentos e das apresentações que podemos pegar o *feedback* dos usuário e começar a mapear novas funcionalidades e melhorias. Podemos ver o mapeamento dos riscos e das dependências das funcionalidades exemplo na tabela três.

Funcionalidade	Riscos	Dependências
Cadastro de produtos	Processo de cadastro de produtos mais demorado; Criação de produtos menos flexível.	Treinamento para cadastrar os produtos no sistema; Explicação das regras utilizadas para o cadastro dos produtos.
Venda de produtos	Venda menos flexível; Mudança na forma de negociação dos vendedores;	Funcionalidade de cadastro de produtos; Treinamento para realizar vendas no sistema; Explicação das fases na venda.
Chat para clientes	Não adoção do *chat* como princípal meio de comunicação; Utilização do *chat* para assuntos sem relação ao trabalho;	Treinamento para a utilização do *chat*; Treinamento sobre como se comunicar por *chat*.

Tabela 3: Exemplo de levantamento dos riscos e dependências por funcionalidade

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?

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