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 FIGURAS

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 [1], 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 [2], 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

4.2 Qualidade de software

4.2.1 Requisitos funcionais

4.2.2 Requisitos não funcionais

4.2.3 Requisitos inversos

4.2.4 Escalabilidade

4.3 Testes no desenvolvimento de *software*

4.3.1 Testes unitarios

4.3.2 Testes de regrssão

4.3.3 Testes de fumaça

4.3.4 Testes de integração

4.3.5 Testes automatizados

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 classificação 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

Após entendermos os ricos, escolher quais vamos aceitar e definir a prioridade das funcionalidades que queremos desenvolver, devemos começar a refinar cada uma e definir quais são seus requisitos. É importante salientar que os requisitos, assim como foi com os riscos, devem partir do negócio, para assim fazer parte do sistema, a funcionalidade deve solucionar um problema e facilitar a vida dos usuários, agregando valor para o cotidiano deles e consequentemente agregando valor para o negócio.

A funcionalidade deve ser desenvolvida com base no que os usuários já fazem no dia a dia, temos que mapear todas as tarefas que o usuário executa, identificando quais incomodam e quais não incomodam. Identificando quais tarefas o usuário não gosta de fazer, já é um ponto de partida no que o sistema automatizar, uma vez que o usuário não precise fazer determinada funcionalidade, já facilitou o seu cotidiano. Outro ponto de atenção, são com as tarefas que o usuário não se importa em executar, automatizar ou mudar a forma que ele executa pode ser um problema em um promeiro momento, pois ele terá que se acostumar com uma nova forma de trabalho. Muitas vezes esse tipo de mudança pode se originar em uma nova fase da empresa, que está revendo os seus processos e mudando a forma como de trabalho dos colaboradores, neste caso, é importante implementar a funcionalidade de forma que seja um meio termo entre o que os usuários realizavam e a nova forma da empresa, desta forma os usuários começam a se adaptar a nova perspectiva e conforme o projeto for avançando, podemos ir adaptando a funcionalidade a restruturação

da empresa. Devemos lembrar que nenhuma mudança acontece do dia para a noite, e uma mudança brusca na forma de trabalho pode gerar desconforto nos usuários e deixar eles confusos em como executar suas atividades.

Uma funcionalidade deve sofrer melhorias constantes, como o negócio está constantemente sofrendo mudanças, o sistema deve se adaptar para os novos requisitos do mercado, por este motivo, os requisitos devem ser identificados mas não podem ser considerados como certeza, o sistema deve estar pronto para sofrer qualquer tipo de alterações e a qualquer hora porem, sabemos que certas mudanças levam muito tempo e até a alteração, o requisito já sofreu mais uma alteração, por este motivo, devemos classificar nossos requisitos em nível de importância e em nível de mutabilidade. Enquanto nos riscos nós classificamos a funcionalidade como um todo, aqui vamos classificar os requisitos que formam essa funcionalidade, devemos sempre pensar, o que é essencial para a funcionalidade atingir o objetivo? O que pode se tornar uma melhoria para o futuro? Desta forma, conseguimos entregar valor mais rápido para diferentes áreas do negócio, pois se focarmos todo o nosso tempo na funcionalidade mais importante, nunca vamos realizar as outras funcionalidades. Para cada funcionalidade devemos tambem atribuir um nível de mutabilidade, ou seja, qual é a chance dessa funcionalidade sofrer alterações conforme o tempo vai passando, embora o negócio esteja em constante mudança, sabemos que certos processos dificilmente sofrem alterações, pois faz parte do *core* da empresa, é a essência do negócio. A forma de classificar a mutabilidade é através de porcentagem, onde as primeiras classificações serão intuições com base na perspectiva do negócio, mas as seguintes, é importante que conforme o sistema vá sofrendo alterações e melhorias, estas taxas sejam atualizadas por funcionalidade. A primeira regra na hora de classificar é que não existe zero porcento, toda funcionalidade pode sofrer alterações, mesmo que demore anos para isso acontecer, a sociedade muda, novas tecnologias vão surgindo e o que era comum é subtituido por algo novo, que acaba se tornando o novo comum, e se a empresa não estiver preparada para acompanhar essas mudanças, ela acaba se perdendo no tempo. Realizando essa classificação podemos identificar quais processos estão consolidados e quais processos ainda devem ser estruturados, com essa visão sobre os processos, podemos identificar o novo *core* do negócio e desta forma trabalhar em uma tranformação na empresa para atender as novas necessidades do mercado, assim evoluindo o sistema ou até mesmo desenvolvendo um novo, garntindo que a empresa se mantenha sempre atualizada com as tendências do mercado. Na tabela quatro está os atributos que devem ser levados em conta quando calssificarmos um requisito.

Funcionalidade	Nome da funcionalidade que o requisito faz parte.
Requisito	Nome do requisito que faz parte da funcionalidade.
Importância	O nível de importância do requisito separado em baixo, médio e alto.
Mutabilidade	O quanto este requisito pode sofrer alterações em porcentagem.
Stakeholders	Quais áreas são impactadas com esta funcionalidade.
Descrição	A descrição do requisito da funcionalidade.

Tabela 4: Definição da classificação de requisitos

Agora que temos mapeado os processos que fazem parte de determindada funcionalidade, e sabemos como classificar, podemos começar a levantar eles. Todo requisito deve possuir uma descrição, algo que instrua o time de desenvolvimento a entender o seu propósito e auxilie eles a desenvolver a funcionalidade. Quando analisamos um requisito, outros requisitos vão surgindo pois, a um primeiro momento, quando levantamos os requisitos, ficamos sempre na funcionalidade, e como o sistema deve se comportar mas alem de estrutrar como determinada funcionalidade deve se comportar, devemos utilizar os requisitos como forma de garantir a qualidade do que será desenvolvido. É atrvés de como os usuários vão utilizar o sistema que podemos identifcar o que precisamos garantir que funcione durante a utilização do usuário. Questões como quantidade de acessos, tempo de resposta, usabilidade, métricas, devem ser consideradas como requisitos do sistema, algo diretamente realcionado ao que vai ser desenvido e estabelcer um critério de aceitável ou não para a utilização em produção. Quando identificamos os requisitos devemos tambem identificar quais áreas da empresa será impactada pelo requisito. Para os requisitos funcionais, devemos identificar quais são as áreas que vão utilizar esta funcionalidade, para os requisitos não-funcionais, identificamos as áreas que caso o requisito não seja atendido, serão impactadas, caso o requisito não seja atendido, quais áreas terão seu processo afetado, podendo criar um bloquio ou deixando ele mais longo. Para exemplificar o levantamento dos requisitos, vamos utilizar a funcionalidade de "Cadastro de produtos", utilizado no exemplo de identificação dos riscos.

Para realizar o cadastro de produtos, vamos supor que envolva três áreas, o *marketing*, a equipe técnica e vendas. A equipe de *marketing*, com base em suas análises de mercado e nas solicitações dos vendedores, analisam as tendências dos pedidos do cliente e montam projetos para novos produtos, este projeto é enviado para a área técnica que avalia se é viável ou não a sua execução, podendo aprovar ou recusar a montagem desse produto. Caso o produto o seja recusado, o solicitação é devolvida para a equipe técnica, com os motivos da recusa, onde eles podem realizar as devidas alterações ou cancelar o projeto. Caso seja aceito, a equipe técnica começa a montagem, realizam seu testes e validam se

está aderente com o que foi proposto no projeto, uma vez encerrado está etapa, é anunciado aos vendedores que há um novo produto no catálogo e caso tenha relação à algum pedido de algum vendedor, ele é notificado que determinado produto foi produzido. Há casos, em que o vendedor pega um produto base e este deve ser customizado com base nas especificações do cliente.

Funcionalidade	Cadasto de produtos
Requisito	Notificação de projeto de produto base
Importância	Média
Mutabilidade	45%
Stakeholders	Área técnica
Descrição	Quando um projeto de produto base for cadastrado, o sistema deve enviar uma notificação para a área técnica, que um novo produto foi cadastrdo e agurda aprovação.

Tabela 5: Exemplo de classificação de requisito funcional

Funcionalidade	Cadasto de produtos
Requisito	A notificação deve ser enviada imediatamente para a área técnica
Importância	Média
Mutabilidade	10%
Stakeholders	*Marketing*, área técnica
Descrição	A notificação da criação do projeto deve ser enviada em um intervalo de menos de um segundo para a área técnica.

Tabela 6: Exemplo de classificação de requisito não-funcional

Funcionalidade	Cadasto de produtos
Requisito	O cadastro é destinado somente para produtos base
Importância	Alta
Mutabilidade	7%
Stakeholders	Vendas
Descrição	Os produtos que serão cadastrados nesta funcionalidade deve somente ser somente produtos base, customizações serão feitas na venda do prostudo.

Tabela 7: Exemplo de classificação de requisito inverso

Embora a funcionalidade de "Cadastro de produtos"tanham muitos outros requisitos, vamos exemplificar somente os que estão nas tabelas cinco, seis e sete. O requisito da tabela cinco se trata de um requisito funcional, o requisito de enviar a notificação para a área técnica quando um projeto for criado. Podemos notar que sua mutabilidade é média,

embora o processo dificilmente possa mudar, a área técnica sempre deve ser informada quando um projeto for criado pois, são eles que aprovam se o projeto é válido ou não porem, no futuro é possível que seja criada uma nova área somente para análise de projetos, hoje está concentrado dentro da mesma área que monta o produto, outra alteração é a adição de outra área, hoje o negócio esntende que somente a área técnica deve ser informada porem, os vendedores tambem fazem parte do processo, uma vez que o projeto pode se tratar de uma solicitação deles, por este motivo, a área de vendas pode ser incluída no recebimento da requisição. Embora os vendedores tambem participem desse processo, a notificação é destinada somente para a área técnica, desta forma, somente a else são um *stakeholder* desse requisito. Na tabela seis, temos um requisito não-funcional, é um requisito que está atrelado diretamente a qualidade da fincionalidade, ela está definindo que a notificação deve ser disparado com no máximo um *delay* de um segundo da criação do produto. Para o negócio é importante que a área seja informada que um novo projeto foi criado imediatamene após sua criação, este requisito deve ser utilizado como critério de qualidade e utilizado como cenário de teste, não somente da criação de um projeto, mas para a criação de vários projeto de uma vez só. Na tabela sete, possuímos um requisito inverso, algo que o sistema não deve fazer, que no caso, é que a funcionalidade de "Cadastro de produtos"se destina somete para produtos base, ou seja, as customizações dos vendedores não estão incluidas nessa funcionalidade, é interessante na descrição deste requisito, sempre colocar em qual funcionalidade este requisito se aplica, caso ele se aplique a alguma funcionalidade. Como a customização dos produtos base está sempre atrelada a venda do produto e não ao seu cadastro, podemos ver que dos tês exemplos, este é o de menor mutabilidade, pois este processo dificilmente será mudado e atribuir a responsábilidade ao vendedor de cadastrar um produto, somente por causas das customizações é uma mudança muito grande no processo de como é realizada a venda.

5.1.3 Arquitetura evolutiva

Conforme o produto vai evoluindo, mudanças deverão ser feitas para acompanhar as necessidades do negócio e para isso, devemos construir o nosso sistema de forma que seja fácil implementar novas funcionalidades e aplicar mudanças e melhorias em funcionalidades já desenvolvidas. Para que isso seja possível, devemos desenvolver um *software* com alto nível de abstração, cada parte do sistema deve funcionar de forma quase indivídual, para que desta forma uma alteração realizada em alguma parte do sistema, não afeta as demais e assim, minizamos *bugs* e tempo de manutenção e desenvolvimento de melhorias. O mundo perfeito, realmente é desenvolver um sistema como o descrito anteriormente,

com alto nível de abstração e cada parte do sistema funcionando de forma indivídual, mas para montar determindado sistema, é necessário que o produto e a equipe de desenvolvimento esteja extremamente madura, no começo do projeto ainda não sabemos quais são as partes do sistema, não conseguimos separar o que deve funcionar de forma indívidual e o que faz parte desse indívidual. Para descobrirmos é importante quando desenharmos a arquitetura de nosso sistema, nos atentarmos na mutabilidade de cada requisito, devemos desenvolver cada funcionalidade como se ela pudesse mudar a qualquer hora, mas quando tivermos que tomar a decisão em qual parte deverá ficar mais abstraída do sistema, a mutabilidade dos requisitos deverá ser considerada. Com base na mutabilidade, podemos montar a base de nosso sistema, as funcionalidades que muito dificilmente deverão ser modificadas, e a partir delas evoluir o sistema para as outras funcionalidades, que dependendo do nível de mutabilidade, deverá funcionar de forma quase independente, desta forma, conforme o tempo for passando, podemos ir isolando as partes do sistema cada vez mais, a medida que a mutabilidade dos requisitos vão ficando mais acertivas com a realidade e novos requisitos vão surgindo, pois desta forma, a equipe de desenvolvimento terá um melhor entendimento do negócio e conseguirá abstrair o sistema de forma mais condizente de como o negócio funciona.

Após tivermos um sistema extremamente abstraído e um time de desenvolvimento com um grande entendimento do negócio, podemos evoluir nossa arquitetura para microserviços, de forma que, cada parte do sistema é verdadeiramente um sistema apartado, com seu próprio time e requisitos, onde todas essas funcionalidades se encontram na mesma plataforma. Quando conseguimos evoluir nosso produto para este nível, ele deixa de ser um sistema para se tornar uma plataforma, um lugar onde vários sistemas vão ser executado para cumprir diversos Objetivos relacionados. Lembrando, que embora agora temos vários sistemas, realizando diversas tarefas diferentes, o objetivo da plataforma deve ser o mesmo de quando iniciamos o projeto, nunca devemos esquecer o objetivo do produto ter sido criado, porque é com base nele que as melhorias e os requisitos devem ser levanrados, uma vez que esqucemos desse objetivo, nosso produto começa a realizar diversas ações que não estão relaciondas, sua arquitetura fica cada vez mais complexa e nosso produto perde seu propósito.

Uma vez que nosso produto esteja separado em vários sistemas, podemos reaproveitar as funcionalidades que desenvolvemos em outros produtos, e como cada sistema da plataforma tem a sua própria equipe, as análises realizadas no produto serão realizdas em cada parte de valor do que foi desenvolvido, com esses dados, podemos analisar qual parte do negócio está sofrendo maior alteração, qual está gerando mais valor, qual precisa ser reestruturada, qual precisa de mais investimento, quais áreas podem ser dividas, entre

outras. Como cada parte da nossa plataforma está se preocupando com uma determindada parte do negócio e como, nosso time já conseguiu separar essas partes de uma forma que esteja completamente aderente a como o negócio funciona, conseguimos ter a visão de todas as atuações da empresa no mercado. Conseguimos ter a visão do todo separada em várias equipes diferentes, preocupadas em evoluir a sua parte e contribuir com as outras. Com essa visão, podemos evoluir nossos sistemas para outras plataformas, com objetivos diferentes, e agregrando valor para o negócio de forma distinta. Podemos através das necessidades do mercado, e com os requisitos do sistema, reformular a forma que a empresa atua no mercado, identificar atuações secundárias da empresa, que tem potencial para se tornar primárias, fazer os sócios e diretores enxergarem cada pedaço que gera valor para a empresa e definir estratégias não tradicionais, algo diferente, para adquirir uma possível vantagem no mercado.

5.1.4 Estruturando o fluxo de entrega

Para alcançarmos nossos objetivos e desenvolvermos as nossas funcionalidades em tempo ágil, com qualidade e se preocupando em manter nossa arquitetura simple e evolutiva, precisamos estruturar o nosso fluxo de entrega, ou seja, precisamos definir como será o processo de desenvolvimento, verificação de qualidade, homologação e subida em produção.

Para cada requisito, devemos escrever os critérios de aceite, para que eles sejam considerados cumpridos, para que desta forma, quem for testar as funcionalidades, consigam compreender o que devem verificar e para que os desenvolvedores, compreendam como devem desenvolver a funcionalidade. Desta forma, alem de termos desenvolvedores e testadores cientes de como deve o sistema deve funcionar, conseguimos criar um padrão de qualidade, que será documentado e revisado conforme o produto for evoluindo pois, conforme o tempo for passando, novos requisitos forem surgindo e os requisitos já mapeados forem mudando. Com os critérios de aceite escritos, podemos mapear quais testes deverão ser feitos para cada funcionalidade, teste de regressão, teste de fumaça, teste de integração, teste em massa, teste unitário, cada requisito vai exigir um determindao tipo diferente de teste, e com o critério de aceite é importante mapearmos não somente quais testes, mas tambem em quais cenários e como eles devem ser executados, pois é nos testes que devemos procurar simular os momentos mais importantes para o negócio, os momentos em que o sistema não pode falhar. Embora cada requisito tenha o seu próprio conjunto de testes, é importante padronizar que todas as novas funcionalidades tenham sempre testes unitários e em massa, e para melhorias ou mudanças seja sempre

realizado testes de regressão, caso uma nova funcionalidade esteja relacionada com outra, é importante tambem realizar os testes de regressão nas funcionalidades relacionadas. Os critérios de aceite devem ser escritos se baseando diretamente nos riscos que aceitamos, quando mapeamos as funcionalidades, nossos testes, alem de garantir que os requisitos da funcionalidade sejam cumpridos, deve direcionar os nossos testes para que realizemos as situações que mapeamos como risco, embora os riscos tenham sido aceite, precisamos verificar como o sistema vai se comportar nestes cenários, desta forma conseguimos antes de disponibilizar para o usuário, averiguar se realmente faz sentido correr esse risco ou não.

Para exemplificar a escrita dos critérios de aceite, vamos utilizar o requisito da tabela seis.

Funcionalidade	Cadasto de produtos
Requisito	A notificação deve ser enviada imediatamente para a área técnica
Critério	Deve ser disparado a notificação para todos os produtos que forem criados.
Critério	Na notificação deve conter o nome do produto e quando ele foi criado.
Critério	A área técnica deverá identificar se uma notificação já foi lida ou não.
Critério	A notificação somente deve ser marcada como lida caso o usuário da área técnica acesse a notificação.

Tabela 8: Exemplo de especificação de critérios de aceite

Na tabela oito, podemos verificar os critérios de aceite para o requisito de notificação da funcionalidade de "Cadastro de produtos". Podemos notar que cada requisito descreve como a funcionalidade deve funcionar, assim como auxilia como este requisito deve ser testado, conseguimos identificar tambem que este requisito deve ser testado de forma massiva, pois um dos critérios é de que cada produto deve disparar obrigatóriamente uma notificação, desta forma, se eu criar nove produtos de uma vez, deve ser gerada uma notificação para os nove produtos.

Agora que definimos como nosso requisito será desenvolvido e como ele será testado, precisamos definir em qual momento cada uma dessas ações deverá ser realizada e em qual ambiente. Quando estamos desenvolvendo uma funcionalidade devemos sempre testar ela por completo, mas para sermos ágeis, não podemos esperar todos os requisitos serem desenvolvidos para iniciarmos os testes, da mesma forma que os desenvolvedores não podem ficar aguardando todos os testes serem realizados para iniciarem um novo desenvolvimente. Enquanto os testadores só podem iniciar um teste assim que um desenvolvedor

finalizar uma atividade e os desenvolvedores não podem alterar a funcionalidade para não influenciar nos testes, é interessante realizar estas ações em ambientes separados. Os desenvolvedores realizam os seus desenvolvimento em um ambiente exclusivo para desenvolvimento, enquanto os testadores realizam seus testes em um ambiente exclusivo para testes. Cada critério, cada cenário deverá ser simulado neste ambiente, devido a isso a qualidade dos dados deste ambiente deverá ser o mais fiel possível ao que os usuários irão utilizar, por outro lado, os desenvolvedores necessitam de dados somente para verificar se a funcionalidade está funcionando e os critérios foram atendidos, devido a isso, é importante ter uma boa qualidade nos dados, mas não é preciso investir tanto tempo nos dados deste ambiente. Vamos chamar o ambiente dos desenvolvrdores de **DEV** (desenvolvimento) e o ambiente dos testadores de **QA** (*Quality Assurance*).

Os desenvolvedores tem a obrigação de garantir que todos os critérios do requisito desenvolvido seja atendido, ou seja, só é permitido disponibilizar o requisito para ser testado no ambiente de **QA** após o desenvolvedor testar todos os crtérios no ambiente de **DEV**. O requisito disponibilizado em **QA**, o testador deverá testar todos os critérios nos mais diversos cenários possíveis, dando foco nos riscos que foram mapeados na especificação do requisito e conforme mais requisitos vão sendo disponibilizados, ele deverá testar novamente todos cenários novamente, mas agora validando os critérios dos outros requisitos. Desta forma, conseguimos garantir que as situações de maior necessidade do negócio está sendo contemplada. Para que possamos gerar valor de forma ágil, é importante que na primeira iteração do time, seja desenvolvida somente as funcionalidades de maior importância, pois dessa forma conseguimos integrar uma funcionalidade de forma rápida, embora ela realizando o minimo planejado, já podemos como será a reação dos usuários e identificar se há a necessidade de mudanças ou melhorias para serem desenvolvidas na próximas iteração, além de reavaliar a importância dos requisitos mapeados.

Logo após todos os testes em todos os cenários de risco da funcionalidade, podemos disponibilizar a funcionalidade para homologação, ou seja, devemos realizar todas as atividades que o usuário irá executar, com o objetivo de verificar se o que foi desenvolvido está aderente com o negócio, podemos tambem realizar apresentações para os usuários, afim de adiantar para eles o que será entregue e preparar uma documentação de como determinada funcionalidade funciona, com o intuíto de disponibilizar esta informação para os usuários e para novos integrantes do projeto. Todo esse processo deve ser realizado em um ambiente separado que vamos nomear de **HOMOL** (homologação). Este ambiente deve estar separado pois os dados que contem nele, devem ser preparados para apresentações, como ele será utilizado para compreender o que será entregue para o usuário e para apresentar uma prévia para eles, as funcionalidades não devem ser influênciadas por funcionalidades

em desenvolvimento e os dados presente nelas, não devem ser influênciadas pelos testes realizados pelos testadores. Os dados neste ambiente deve ser o mais limpo possível, ele deve ser o mais próximo possível do que os usuários irão utilizar em produção, e as funcionalidades que estão neste ambiente devem estar completamente testadas e com todos os critério garantidos.

Após realizarmos a homologação da nossa funcionalidade, devemos nos preparar para entregar essa funcionalidade para os usuários, para isso, vale realizarmos uma ultima validação, em um ambiente que vamos nomear de **PREPROD** (pré-produção). Este ambiente deve ser uma cópia perfeita de produção, nele devemos realizar novamente todos os princípais testes que já realizamos em **QA** e **HOMOL**, para verificar que todos os critérios continuam sendo cumpridos, alem de aplicarmos treinamentos para os usuários da nova funcionalidade, permitindo que eles utilizem esta funcionalidade de forma controlada, para que eles aprendam a utilizar o sistema antes de colcoar ele no ar. Com essa abordagem, antes de disponilizar o sistema em produção, conseguimos verificar a reação deles com o que será entregue, alem de ter uma prévia de como o sistema será utilizado, desta forma já conseguimos identifcar melhorias para as próximas iterações.

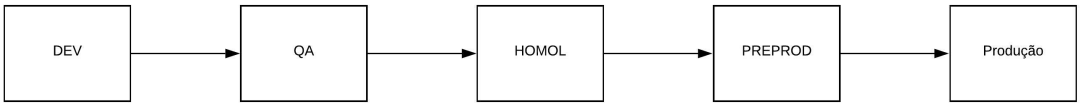

Figura 1: Fluxo de ambientes

5.1.5 Automatizando testes de qualidade

Para otimizar a execução dos testes, e facilitar o trbalho dos testadores, podemos automatizar os testes que realizamos no sistema, para que sempre que uma funcionalidade nova, uma melhoria, uma alteração ou uma correção for implementada podemos realizar novamente todos os testes no sistema automaticamente. Desta forma, não consumimos tanto tempo dos testadores e garantimos que o tudo no nosso sistema continua funcionando da forma que esperamos. A fase de automação dos testes, é importante ser realizada assim que o requisito for homologado, com a homologação, estamos mais certos que o requisito está aderente com o negócio e por causa da etapa de testes, temos todos os princípais cenários que presisam ser garantidos, desta forma podemos montar testes automatizados para após homologado, qualquer alteração que seja realizada nos ambientes de **QA** e adiante, inicie os testes e garanta que os cenários de risco continuem não apresentando

problemas para o sistema.
Os testes devem estar diretamente vinculados com o *deploy* para os ambiente, ou seja, é importante que seja um processo do *pipeline*, onde todo *deploy* realizado, rode novamente todos os testes, em cada um dos ambientes, pois desta forma, conseguimos averiguar se alguma alteração no sistema impactou em alguma parte que não estavamos prevendo e com essa descobertra, conseguimos uma visão mais detalhada do quanto o nosso sistema está estruturado, auxiliando nas decisões de arquitetura.

5.2 Como adquirir escalabilidade?

Conforme o nosso sistema vai aumentando, mais usuários vão utilizando ele, mais funcnionalidade ele possui, e mais poder computacional ele necessita. Garantir que um sistema se mantenha no ar é uma tarefa complicada, embora estejamos desenvolvendo o sistema da melhor forma possível, realizando testes na maioria dos cenários e evoluindo a arquitetura para se adequar a atual realidade do sistema, não conseguimos prever tudo, e muitas vezes, quando mais precisamos que ele funcione, ele apresenta algum problema. Muitas vezes podemos subestimar alguma situação ou superestimar outra situação, e se não estivermos preparados para os imprevistos, a qualidade de nosso produto pode cair e a confiança dos nossos usuários diminuir.
Vamos imaginar, que a empresa que utilizamos como exemplo anteriormente, conseguiu criar um computador base, que atendende muitas necessidades do seus clietes, devido a isso, o número de pedidos aumentou, e a empresa teve que contratar mais vendedores, mais colaboradores da área técnica e a equipe de *marketing* está com várias idéias para novos produtos. O sistema nessa situação vai estar constantemente executando vendas, assim como vários projetos serão criados, e várias notificações deverão ser disparadas. Certo dia, a utilização no sistema foi tão intensa, que o sistema não enviou mais notificações durante dois minutos, estes dois minutos foi o suficiente para que a área técnica não analisasse três projetos, estes projetos que a equipe de *marketing* tinha uma grande expectativa, o tempo passou foi gerado o conflito entre as duas áreas, e foi identificado este erro no sistema, agora o momento para a produção dos produtos base se perdeu, este conflito abalou as duas áreas e a confiança com o sistema pelos usuários diminuiu.

5.2.1 Descobrindo os limites do sistema

Para nos prepararmos para as situações inesperadas, precisamos descobrir os limites do nosso sistema, precisamos tentar fazer o sistema falhar em um ambiente controlado,

antes que ele falhe em produção, quando os nossos usuários estiverem utilizando. Entender que nosso sistema tem limites e que ele está sucetível a erros é o primeiro passo para descobrirmos como podemos encontrar estes erros, devemos utilizar os riscos que mapeamos, e simular diversas vezes estes cenários, o que levantamos em nossos requisitos não-funcionais, devem ser levado ao extremo, então caso levantamos que um determinado requisito deve ter um tempo de resposta de no máximo dois segundos, devemos foçar o processamento do nosso sistema para que o tempo de resposta supere dois segundos e neste cenário, rodar novamente todos os testes novamente, desta forma nós conseguimos ver se o fato do requisito não ser atendido tem algum impacto nos critérios que levantamos. Uma vez conseguido simular este cenário, podemos mapear os limites do sistema, como nos esforçamos para forçar o sistema a funcinoar no seu limite, agora podemos documentar qual é o seu limite e analisar se devemos aumentar este limite ou se aceitamos o risco que ele representa.

Definir limites para o sistema é definir novamente os riscos que pretendemos aceitar, quando estávamos especificando as nossas funcnionalidades, os riscos que mapeamos era com a pespectiva de negócio, nosso objetivo era garantir que o que iria ser desenvolvido, gerasse valor e não teria impactos negativos para os usuários, agora devemos utilizr estes riscos, em conjunto com os nossos requisitos, para mapearmos os limites do sistema. Estamos utilizando o negócio para aprender até onde o sistema deve aguentar, e por outro lado, estamos utilizando o sistema para aprender o que o negócio precisa e quais riscos podemos enfrentar.

Quando mapeamos os limites do sistema, é importante separarmos os limites por funcinalidae e não buscar valores absolutos, mas sim valores estatísticos. Por mais testes que realizamos e por mais cenários que simulamos, não podemos garantir que em um situação extrema, o sistema vai se comportar sempre da mesma forma. Como estamos buscando o extremo, e esperando que o que desenvelvemos pare de funcinoar, por mais que tenhamos uma idéia de qual parte do sistema vai falhar, não é garantido que as nossas expectativas sempre se cumpram, realizando o mesmo teste mais de uma vez, podemos ter diversos retornos diferentes. Por este motivo, é interessante, com base na importância da funcionalidade, estipular uma quantidade de testes para cada situação extrema, com base nos retornos gerados, podemos documentar o que aconteceu e qual foi a frequência deste ocorrido, pois caso algum dia venha a ocorrer em produção, já temos mapeado os princípais problemas ou uma vez que, produção comece a trablhar no limite do sistema, já sabemos o que é mais provável de ocorrer. Para armazenarmos os testes que ocorreram, é importante colcoar anotar a funcionalidade que testamos, o cenário que estava o sistema e quais foram resultados com as porcentagens. Na tabelas nove e dez, exemplificamos está

análise utilizando a funcnionalidade exemplo de "Cadasto de produtos".

Funcionalidade	Cadasto de produtos
Cenário	O processamento do servidor utilizado para a funcnionalidade se encontra com 95% de uso.
Resultado	Em 89% do cadastro novos projetos, a notificação demorou aproximadamente três segundos para aparecer.
Resultado	Em 96% do cadastro dos novos projetos, levou um aproximadamente dois segundos para realizar a criação do projeto.

Tabela 9: Exemplo de análise de limite de sistema - limite de processamento

Funcionalidade	Cadasto de produtos
Cenário	Realizado 500 criações de novos projetos ao mesmo tempo.
Resultado	Em 98% dos cadastros a concexão com o banco de dados foi perdida ao tentar inserir os produtos.

Tabela 10: Exemplo de análise de limite de sistema - limite de requisições

As análises devem ser feitas em um ambiente que seja uma cópia perfeita de produção, sempre que algo for entregue em produção, deve ser automaticamente replicado para este ambiente, vamos chamá-lo de **CHAOSPROD** (produção caótica).
Estas análises são prevendo cenários adversos, imaginando situações inesperadas, por este motivo, é interessante realizar estas análises sempre após que um requisito for entregue em produção, desta forma, conseguimos maior produtividade na entrega dos requisitos em desenvolvimento, e uma vez que ele tenha sido homologado, e temos o retorno de como os usuários estão utilizando o sistema, conseguimos realizar as análises nos baseando não somente no que mapeamos como risco, mas também com base em como o usuário está utilizando o sistema e no *feedback* que estamos tendo com eles, seja esse *feedback* dito diretamente pelo usuário ou através de alguma análise realizada em produção.

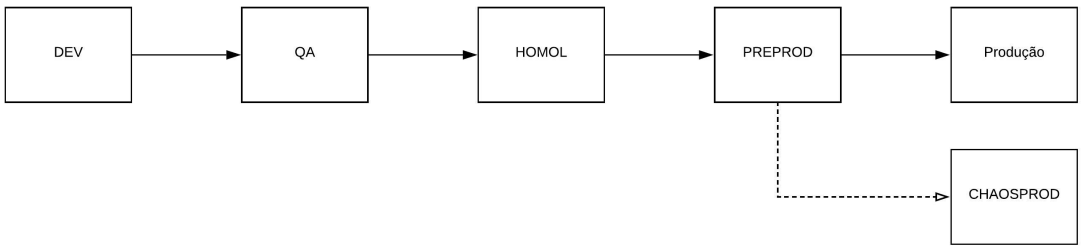

Figura 2: Fluxo de ambientes com CHAOSPROD

5.2.2 Desenvolvendo estratégias de escalabilidade

Uma vez identificado os nossos limites, devemos criar estratégias para lidar com elas, ser aumentar o poder de processamento dos nossos servidores, migrar a nosso aplicação para um banco de dados mais robusto, melhorar o desenempendo da funcnionalidade desenvolvida através do código, existem várias ações que podemos tomar, para podermos enfrentarmos os riscos que mapeamos.

Para cada cenário que mapearmos precisamos criar um plano, para caso esta situação venha a ocorrer. Este plano, deve estar atrelado diretamente a importância da funcionalidade e os impactos que sua execução irá gerar, devemos considerar os gastos financeiros, o tempo que o plano vai demorar para ser executado, os times que irão atuar nele, as áres de negócio que vão ser impactadas, como devemos avisar os usuários, entre outras ações. Com a realização deste plano, nos preparamos para o imprevisto, caso algum dos cenários que mapeamos aconteça, já vamos estar preparado, embora não sabemos exatamente o que vai acontecer, temos a probabilidade de cada situação. Podemos tambem nos antecipar a esse limite, caso a funcnionalidade seja muito importante para o negócio, e identificamos que o limite identificado pode ser atingido, podemos trabalhar para aumentá-lo, como sabemos aonde precisamos melhorar, basta realizar uma exploração mais aprofundada da melhor forma que podemos aumentar este limite. Outro ponto de destaque, é que conseguimos adaptar o nosso sistema conforme as necessidades de negócio, caso um determinada funcnionalidade necessite de mais processamento, podemos alocar mais servidores para esta funcnionalidade, caso um não tenhamos os servidores disponíveis, podemos aumentar nosos servidores de forma horizontal, caso não haja verba disponível, para aumentar a quantidade de servidores, podemos desativar uma funcnionalidade menos importante para realocar os recursos para a funcionalidade de maior importância, desta forma, assim que este periodo de maior necessidade passar, reativamos a funcnionalidade antiga.

Para cada caso devemos estruturar uma estratégia e devemos utilizar de métricas no ambiente de produção para supervisionar a saúde do ambiente e anteciparmos algum tipo de erro. Devemos monitorar o armazenamento já utilizado pelo nosso sistema, para caso ele esteja no fim, podemos aumentar o nível de armazenamento ou excluir registros antigos, devemos monitorar os nossos servidores, para caso algum deles estejam começando a apresentar defeito, ou algum conjunto de servidores estejam sendo mais utilizados que outros, assim podemos realocar as nossas funcnionalidades em servidores diferentes e aprendemos qual parte do sistema utiliza maior quantidade de recursos dos nossos servidores. Podemos utilizar diversas acompanhamentos e implementar diversas métricas em nosso

produto, mas é importante sempre lembrar, que devemos gerar as nossas métricas e basear os nossos acompanhamentos com base em como o sistema está sendo utilizado, ou seja, é através das necessidades do negócio, que aprendemos aonde devemos olhar. Na tabela dez, específicamos como estruturar uma estratégia.

Funcionalidades	Nome das funcnionalidades que foram impactadas
Cenário	Descrição do cenário em que ocorreu os erros.
Stakeholders	Quais são as áreas de negócio que serão impactadas no cenário descrito.
Times	Quais são os time realcionados para a execução deste plano.
Plano	O que deverá ser feito caso produção se encontre neste cenário.
Tempo de Execução	Tempo para a execução do plano.

Tabela 11: Especificação para estruturação de estratégias

Com o ambiente de **CHAOSPROD**, conseguimos descobrir os limites do sistema, com o ambiente de produção, conseguimos descobrir como o sistema é utilizado, é através dos usuários, e da persepção de valor gerado para eles e para o negócio que devemos basear as nossas prioridades e ajustar a importância das nossas funcionalidades. Com está visão podemos tomar a decisão aonde precisamos testar mais, e também onde precisamos de maior supervisão, embora estamos constantemente procurando por erros, não estamos livres de falhas, erros em produção vão acontecer. É nestas horas que as ferramentas que implementamos para supervisionar o sistema tem sua maior utilidade, é através delas que vamos descobrir o que aconteceu e uma vez descoberto, vamos utilizar este ocorrido como aprendizado, alem de resolver o problema, vamos identificar o porquê ele aconteceu, se realmente resolvemos o problema ou mitigamos, se ele pode acontecer e como podemos nos preparar para ele. Este erro pode se tornar mais um teste automatizado para ser realizado durante os desenvolvimentos ou pode se tornar mais uma análise a ser feita em **CHASOPROD**. De toda forma, ele se tornou um aprendizado para o produto e agora faz parte de um cenário que estamos prevendo, pode ser que ele aconteça novamente mas desta vez estaremos preparados.

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

REFERÊNCIAS

1 BEYER, B. *Site Reliability Engineering*. [S.l.]: O'Reilly, 2018.

2 KIM, G. et al. *The DevOps Handbook*. [S.l.]: IT Revolution, 2016.