Universidade Federal do Triângulo Mineiro

Programa de Mestrado Profissional em Inovação Tecnológica - PMPIT

Mário Sakamoto

# Melhoria contínua da qualidade no processo de produção da camada Back-end de sistemas web em arquitetura REST

Uberaba - MG

2018

Mário Sakamoto

# Melhoria contínua da qualidade no processo de produção da camada Back-end de sistemas web em arquitetura REST

Dissertação apresentada ao Programa de Mestrado Profissional em Inovação Tecnológica da Universidade Federal do Triângulo Mineiro, como requisito para obtenção do título de Mestre.

Orientador: Prof. Dr. Gilberto de Araújo Pereira

Uberaba - MG
2018

**Catalogação na fonte: Biblioteca da Universidade Federal do Triângulo Mineiro**

Sakamoto, Mário
S152m Melhoria contínua da qualidade no processo de produção da camada Back-end de sistemas web em arquitetura REST / Mário Sakamoto. -- 2018.
70 f. : il., fig., graf., tab.

Dissertação (Mestrado Profissional em Inovação Tecnológica) -- Universidade Federal do Triângulo Mineiro, Uberaba, MG, 2018
Orientador: Prof. Dr. Gilberto de Araújo Pereira

1. Arquitetura de software. 2. Representational State Transfer (Arquitetura de software). 3. Controle de qualidade. 4. Processos de fabricação. 5. Controle de processo - Métodos estatísticos. 6. Crosby, Philip B. (Philip Bayard), 1926-2001. I. Pereira, Gilberto de Araújo. II. Universidade Federal do Triângulo Mineiro. III. Título.

CDU 004.413

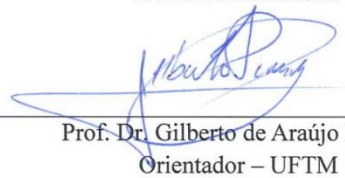
MÁRIO SAKAMOTO

# MELHORIA CONTÍNUA DA QUALIDADE NO PROCESSO DE PRODUÇÃO DA CAMADA BACK-END DE SISTEMAS WEB EM ARQUITETURA REST

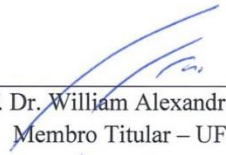
Trabalho de conclusão apresentado ao Programa de Mestrado Profissional em Inovação Tecnológica da Universidade Federal do Triângulo Mineiro, como requisito para obtenção do título de mestre.

Uberaba, 20 de dezembro de 2018

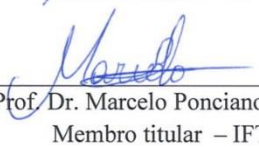
Banca Examinadora:



Prof. Dr. Gilberto de Araújo Pereira
Orientador – UFTM



Prof. Dr. William Alexandre Manzan
Membro Titular – UFTM



Prof. Dr. Marcelo Ponciano da Silva
Membro titular – IFTM

## AGRADECIMENTOS

Agradeço à minha amiga e companheira, Jéssica, pelo incentivo ilimitado ao aprendizado.

Ao orientador Prof. Dr. Gilberto de Araújo Pereira, com quem tive oportunidade de aprender e desenvolver o conhecimento.

À Universidade Federal do Triângulo Mineiro - UFTM, pela oportunidade de realização deste curso.

À banca examinadora, por se dedicarem em contribuir intelectualmente para o aprimoramento deste trabalho.

À FAPEMIG pelo apoio financeiro ao Programa de Mestrado Profissional em Inovação Tecnológica - PMPIT.

Muito obrigado.

## RESUMO

Considerando que na literatura observada não constam relatos de uma ferramenta *web* de gestão da qualidade que ofereça soluções para a avaliação de melhoria contínua da qualidade no processo de produção da camada *Back-end* de sistemas *web* em arquitetura REST. Este trabalho teve como objetivo desenvolver uma ferramenta que gerencie testes de funcionalidades em pontos de acesso da camada *Back-end* através de análise do status de resposta do protocolo de comunicação HTTP para avaliar o impacto da implantação de um processo de produção baseado na filosofia do Zero Defeito de Philip Bayard Crosby, verificando a melhoria da qualidade em uma fábrica de *software* em “nível inicial” segundo o CMMI. Base de dados relacionados a falhas coletadas durante testes na produção da fábrica foi analisada a partir de técnicas do Controle Estatístico de Processo para a validação de controle do processo, medindo os limites superiores e inferiores de controle através da análise do número médio de não conformidades relacionadas às falhas por unidade de amostras de tamanho variável. Para o desenvolvimento da ferramenta *web* de gestão da qualidade foram utilizadas linguagens de programação e marcação de uso livre, tais como HTML, CSS, JavaScript e PHP, além da utilização de banco de dados relacional com o auxílio do gerenciador do MySql. Como resultado, o processo de produção baseado na filosofia de Crosby foi descrito e implantado na organização de uma fábrica, a ferramenta *web* foi desenvolvida e utilizada, e foi observado uma redução de mais de 90% no percentual das não conformidades relacionadas às falhas identificadas na produção após um período de 60 dias de utilização.



**Palavras-chave**: Philip Bayard Crosby. Melhoria contínua da qualidade. Controle estatístico de processo. Gráfico *u*. Camada *back-end*. Sistemas *web*. Arquitetura REST

**ABSTRACT**

*Considering that in the literature observed there are no reports of a web tool of quality management that offers solutions for the evaluation of continuous quality improvement in the production process of the Back-end layer of web systems in REST architecture. The objective of this work was to develop a tool that manages functionality tests on access points of the Back-end layer through analysis of the response status of the HTTP communication protocol to evaluate the impact of the implementation of a production process based on the philosophy of Zero Defect of Philip Bayard Crosby, verifying the improvement of quality in an "initial level" software factory according to the CMMI. Database-related failures collected during factory production tests were analyzed using Statistical Process Control techniques for the validation of process control, measuring the upper and lower control limits by analyzing the mean number of related nonconformities to failures per unit of samples of variable size. For the development of the web tool of quality management were use programming languages and free use markup, such as HTML, CSS, JavaScript and PHP, as well as the use of relational database with the help of the MySql manager. As a result, the production process based on Crosby's philosophy was described and implemented in the organization of a factory, the web tool was developed and used, and a reduction of more than 90% in the percentage of nonconformities related to the failures identified in the after 60 days of use.*

**Keywords**: *Philip Bayard Crosby. Continuous quality improvement. Statistical process control. Graphic u. Back-end layer. Web systems. REST architecture.*

## LISTA DE FIGURAS

**Figura 1** - Diagrama de processo MVC ..... 19
**Figura 2** - Requisição de um serviço na camada *Back-end* ..... 20
**Figura 3** - Composição de uma URI ..... 21
**Figura 4** - Requisição *POST* ..... 22
**Figura 5** - Requisição *GET* ..... 22
**Figura 6** - Requisição *PUT* ..... 23
**Figura 7** - Requisição *DELETE* ..... 24
**Figura 8** - Processo de acesso a uma API ..... 32
**Figura 9** - Divisão de pacotes do projeto ..... 32
**Figura 10** - Processo do ciclo de vida do desenvolvimento de software ..... 35
**Figura 11** - Fluxograma dos 14 passos da filosofia de Zero Defeito de Crosby ..... 37
**Figura 12** - Fluxograma de identificação de não conformidade referente a falha de uma API ..... 38
**Figura 13** - Cálculo da taxa de não conformidades por subgrupos de tamanhos variados ..... 41
**Figura 14** - Cálculo da linha central ..... 41
**Figura 15** - Cálculo da linha superior ..... 41
**Figura 16** - Cálculo da linha inferior ..... 42
**Figura 17** - Carta de controle com os limites superior (LSC), inferior (LIC) e central (LC) e linhas correspondentes aos desvios (σ) ..... 42
**Figura 18** - Exemplos de processos fora de controle estatístico. Adaptado da norma ISO 8258 ..... 43
**Figura 19** - Fluxograma do funcionamento da ferramenta de gestão de testes ..... 45
**Figura 20** - Login administrativo ..... 46
**Figura 21** - Login com inconsistência de informações ..... 46
**Figura 22** - Tela de usuários do sistema ..... 47
**Figura 23** - Tela de falhas por serviço ..... 48
**Figura 24** - Dashboard totalizador dos 12 últimos meses referentes a todos os testes ..... 48

**Figura 25** - Dashboard categorizado por meses ..... 49
**Figura 26** - Gerenciamento dos serviços das APIs ..... 49
**Figura 27** - Gerenciamento dos valores de entrada e as suas regras de negócio ..... 50
**Figura 28** - Gerenciamento dos valores de saída da resposta ..... 50
**Figura 29** - Login do usuário ..... 50
**Figura 30** - Teste de requisição de um serviço ..... 51
**Figura 31** - Requisição com sucesso de resposta ..... 52
**Figura 32** - Requisição com falha de resposta ..... 52
**Figura 33** - Seleção de documentação de API ..... 53
**Figura 34** - Gráfico u de controle referente às falhas ocorridas antes da implantação do processo de produção de software, em 30 dias de testes ..... 53
**Figura 35** - Gráfico u de controle referente às falhas ocorridas após a implantação do processo de produção de software, em 30 dias de testes ..... 54

## LISTA DE TABELAS

**Tabela 1** - Os gráficos de controle, definições e objetivos ..... 40

## LISTA DE ABREVIATURAS E SIGLAS

ABES - Associação Brasileira das Empresas de *Software*

API - Application Programming Interface

ARPA: Advanced Research Projects Agency

CEP - Controle Estatístico de Processo

CMMI - *Capability Maturity Model Integratio*n

CSS - *Cascading Style Sheets*

HTML - *HyperText Markup Language*

HTTP - *Hypertext Transfer Protocol*

LC - Limite de Controle

LIC - Limite Inferior de Controle

LSC - Limite Superior de Controle

PHP - *Hypertext Preprocessor*

REST - *Representational State Transfer*

SPC - *Statistical Process Control*

SQL - *Structured Query Language*

TI - Tecnologia da Informação

URI - *Universal Resource Identifier*

URL - *Uniform Resource Locator*

URN - *Uniform Resource Names*

UX - *User Experience*

**SUMÁRIO**


**1 INTRODUÇÃO** ..... 13
**2 OBJETIVO** ..... 15
2.1 OBJETIVO GERAL ..... 15
2.2 OBJETIVOS ESPECÍFICOS ..... 15
**3 REVISÃO DA LITERATURA** ..... 16
3.1 PROGRAMAÇÃO PARA A INTERNET ..... 16
**3.1.1 Vantagens da programação para a internet** ..... 16
**3.1.2 Complexidade no desenvolvimento de projetos para a internet** ..... 17
**3.1.3 Padrão MVC** ..... 18
**3.1.4 Plataforma Front-end e Back-end** ..... 19
**3.1.5 Webservice em arquitetura REST** ..... 21
3.1.5.1 Método POST ..... 21
3.1.5.2 Método GET ..... 22
3.1.5.3 Método PUT ..... 23
3.1.5.4 Método DELETE ..... 23
**3.1.6 Código de status da resposta de um webservice** ..... 24
3.1.6.1 Server Error ..... 25
**3.1.7 Erro, falta e falha de software** ..... 25
3.2 GESTÃO DA QUALIDADE ..... 27
**3.2.1 Qualidade de software** ..... 27
3.2.1.1 Dívida técnica ..... 27
3.2.1.2 Modelo integrado de maturidade em capacitação ..... 28
**3.2.2 Filosofia da qualidade de Philip Bayard Crosby** ..... 29
**3.2.3 Controle estatístico de processo** ..... 29
**4 MATERIAIS E MÉTODOS** ..... 31
4.1 FERRAMENTA DE GESTÃO DE TESTES ..... 31
**4.1.1 Padronização do desenvolvimento Back-end** ..... 31
**4.1.2 Desenvolvimento da ferramenta** ..... 33
4.1.2.1 Linguagem de marcação de hipertexto ..... 33
4.1.2.2 Folhas de estilo em cascata ..... 33
4.1.2.3 Linguagem de programação JavaScript ..... 34
4.1.2.4 Linguagem de programação PHP ..... 34

4.1.2.5 Gerenciador de banco de dados MySql ..... 34
4.2 PROCESSO DE PRODUÇÃO DA CAMADA BACK-END BASEADO NA FILOSOFIA DA QUALIDADE DE CROSBY ..... 34
**4.2.1 Conformidade com os requisitos** ..... 34
**4.2.2 Prevenção da qualidade** ..... 35
**4.2.3 Padrão Zero Defeito** ..... 38
**4.2.4 Preço da não conformidade** ..... 38
4.3 CONTROLE ESTATÍSTICO DO PROCESSO ..... 39
**4.3.1 Gráficos de controle** ..... 39
**5 RESULTADOS** ..... 44
5.1 DIAGRAMA DE ENTIDADE E RELACIONAMENTO DA FERRAMENTA DE GESTÃO DE TESTES ..... 44
5.2 FLUXOGRAMA DA FERRAMENTA DE GESTÃO DE TESTES ..... 44
5.3 FERRAMENTA DE GESTÃO DE TESTES ..... 45
**5.3.1 Tela de login do administrador** ..... 45
**5.3.2 Tela de cadastro de usuário** ..... 47
**5.3.3 Tela de visualização de falhas por serviço** ..... 47
**5.3.4 Dashboard** ..... 48
**5.3.5 Tela de gerenciamento da documentação de APIs** ..... 49
**5.3.6 Tela de login do usuário** ..... 50
**5.3.7 Tela de teste dos serviços** ..... 51
**5.3.8 Documentação das APIs** ..... 52
5.4 BASE DE DADOS RELACIONADOS AO NÚMERO DE FALHAS DE SOFTWARE ANTES DA IMPLANTAÇÃO DA FILOSOFIA DE CROSBY ..... 53
**5.4.1 Gráfico u relacionado à base de dados antes da implantação** ..... 53
5.5 BASE DE DADOS RELACIONADOS AO NÚMERO DE FALHAS DE SOFTWARE DEPOIS DA IMPLANTAÇÃO DA FILOSOFIA DE CROSBY ..... 54
**5.5.1 Gráfico u relacionado à base de dados depois da implantação** ..... 54
5.6 IMPLANTAÇÃO DO SISTEMA ..... 54
**5.6.1 Dificuldades na implantação** ..... 55
**6 CONCLUSÃO** ..... 56
**REFERÊNCIAS BIBLIOGRÁFICAS** ..... 58
**APÊNDICE A - Diagrama de entidade e relacionamento da ferramenta de gestão de testes** ..... 63

**APÊNDICE B - Demonstrativo do código JavaScript de envio da requisição e recebimento da resposta** ..... 64
**APÊNDICE C - Demonstrativo da visualização da documentação de acesso a uma API** ..... 66
**APÊNDICE D - Demonstrativo das falhas ocorridas antes da implantação do processo de Crosby em 30 dias de testes** ..... 68
**APÊNDICE E - Demonstrativo das falhas ocorridas depois da implantação do processo de Crosby em 30 dias de testes** ..... 69
**APÊNDICE F - Política da qualidade** ..... 70

# 1 INTRODUÇÃO

No processo de produção de *software*, um minucioso levantamento de requisitos e uma comunicação de forma clara, objetiva e integrada entre todas as partes interessadas, são de extrema importância para se chegar a um resultado positivo em relação às expectativas para com a qualidade de um projeto de produção de *software*. No entanto, mesmo sabendo disso, a realidade em algumas fábricas de *software* é outra, onde se encontra uma baixa cultura de gestão de projetos e uma falta de padronização nos processos de produção, o que conduz para um aumento de risco relacionado a uma baixa qualidade da produção. De acordo com uma pesquisa patrocinada pela INNOTAS (2016), 55% dos profissionais de Tecnologia da Informação (TI) disseram ter projetos que fracassaram entre os anos de 2015 e 2016, e como principal causa configura-se as não conformidades, que são o não cumprimento às expectativas para com os requisitos desejados.

De acordo com a Associação Brasileira das Empresas de *Software* (ABES, 2016), em 2015, o Brasil foi responsável por 45% das vendas no setor de TI em toda a américa latina, correspondendo à 60 bilhões de dólares americanos, o que foi um aumento de 9,2% em relação ao seu ano anterior. Considerando a crescente demanda por novos serviços e produtos no setor de TI, e pela grande quantidade de projetos fracassados no setor de TI, especificamente no setor de produção de *software* devido à falta de uma implementação efetiva de um processo de gestão da qualidade, torna-se necessário a busca por melhorias em relação à qualidade e a forma como ela é gerenciada, uma vez que a qualidade é um dos grandes fatores críticos para o sucesso (HIETSCHOLD, REINHARDT, GURTNER, 2014).

Philip Bayard Crosby (1979), define a qualidade como o cumprimento aos requisitos, é conseguida através da prevenção, possui um padrão de zero defeito e é medida pela não conformidade, que são os custos gerados pelo não cumprimento dos padrões da qualidade no processo de produção, tais como a falta de planejamento e documentação, as más práticas de produção e o não cumprimento das especificações do escopo de um projeto (PHILIP CROSBY ASSOCIATES, 2001).

A partir de uma pesquisa em base de dados indexados do ScienceDirect na área de TI e *Software*, utilizando os termos “*software quality AND zero defect*” com a data de publicação a partir do ano de 2013 até o ano de 2017, foram encontrados 63 artigos, que após filtro através de análise sobre o contexto de cada artigo com

conteúdos relacionados à qualidade de *software* e a definição de Philip Bayard Crosby, restou apenas 1 artigo, onde neste artigo é proposto uma solução para a melhoria contínua da qualidade de um *software* com a utilização da análise de estimativas de custos na remoção de não conformidades antes da entrega do *software*, na qual estão incluídos os custos necessários para a eficácia da inspeção, eficácia de teste e eficácia na remoção de não conformidades. Além disso, sugere que para melhorar a qualidade de um *software* poderia ser adotado princípios básicos de suporte à gestão com a utilização da filosofia da qualidade de Philip Bayard Crosby (MARANDI, KHAN, 2015).

Na literatura estudada, não foram encontrados relatos de uma ferramenta *web* de gestão de testes que ofereça soluções para a análise da melhoria contínua da qualidade no processo de produção da camada *Back-end* de sistemas *web* em arquitetura de Transferência de Estado Representacional (REST - *Representational State Transfer*). Dessa forma, esse trabalho descreve uma proposta de processo baseado na filosofia da qualidade de Philip Bayard Crosby para a produção da camada *Back-end* de sistemas *web* em arquitetura REST, visando a produção baseada em zero defeito, avaliando o impacto sobre a melhoria da qualidade com a implantação do novo processo em uma fábrica de *software* do setor público situada em interior do Estado de Minas Gerais classificada como “nível inicial”, segundo o modelo integrado de maturidade em capacitação (CMMI - *Capability Maturity Model Integratio*n).

Nas próximas sessões são apresentados os objetivos gerais e específicos propostos, a revisão da literatura, os materiais e métodos utilizados, os resultados obtidos e a conclusão sobre o trabalho.

# 2 OBJETIVO

## 2.1 OBJETIVO GERAL

O presente trabalho tem como objetivo geral analisar de que forma a gestão da qualidade auxilia no processo de produção da camada *Back-end* de sistemas *web* em arquitetura REST para avaliação de melhoria contínua da qualidade em uma fábrica de *software* do setor público situada em município do interior do Estado de Minas Gerais classificada como "nível inicial" segundo o CMMI.

## 2.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

a) Desenvolver uma ferramenta *web* de gestão de testes capaz de coletar dados da resposta de uma requisição em um ponto de acesso da camada *Back-end* de sistemas *web* em arquitetura REST, a fim de obter uma base com dados relacionados a falhas de *software*;
b) Descrever e aplicar o processo da qualidade da filosofia do Zero Defeito de Philip Bayard Crosby na produção de *software*;
c) Avaliar o impacto na qualidade da produção de *software* com a implantação do processo da qualidade de Crosby.

# 3 REVISÃO DA LITERATURA

Nesta sessão são abordados tópicos importantes para o entendimento sobre os principais avanços e os problemas encontrados no setor de Tecnologia da Informação na área de desenvolvimento de *software*, bem como práticas de gestão da qualidade relacionados à melhoria contínua da qualidade que são o suporte para o desenvolvimento dos objetivos propostos nesse trabalho.

## 3.1 PROGRAMAÇÃO PARA A *INTERNET*

O surgimento da *internet* ocorreu na Guerra fria, no final dos anos cinquenta a partir de um projeto de pesquisa militar (ARPA - *Advanced Research Projects Agency*), quando o Estados Unidos idealizou um modelo de troca e compartilhamento da informação de forma descentralizada a fim de manter dados ainda que uma de suas bases fosse destruída (BRIGGS, BURKE, 2006). De acordo com Briggs e Burke (2006), no dia 29 de outubro de 1969, foi executada a primeira conexão entre duas universidades (Universidade de Califórnia e o instituto de pesquisa de Stanford), no qual foi um dos momentos mais marcantes da história, uma vez que havia sido enviado o primeiro e-mail, que em seguida, permitiria um acesso livre aos professores e pesquisadores dessa tecnologia.

Nos anos noventa, começou a surgir vários navegadores como Google Chrome, Mozilla Firefox, Internet Explorer e outros, o que popularizou o uso da *internet* entre os usuários (MENDOZA, 2016). Com esse avanço, o mercado desenvolveu uma necessidade por soluções com cada vez mais presença na *internet*, trazendo facilidade de acesso através de diferentes dispositivos de acesso.

### 3.1.1 Vantagens da programação para a *internet*

A *internet* tem se mostrado cada vez mais presente na vida das pessoas, gerando soluções para o compartilhamento de informações e se mostrando como tecnologia chave para o sucesso das empresas. Para o desenvolvimento de código através da programação para a *internet* é possível obter vantagens de padronização de programação, aumento de escalabilidade, que é a forma eficaz de fácil execução para

manipular a capacidade de crescimento de uma aplicação, e acesso da informação, que é a disponibilização da informação em diversos tipos de dispositivos.

A disponibilização da informação através da *internet* é uma estrutura que integra processos de negócio em componentes seguros e padronizados. Ela visa facilitar a integração entre sistemas, orientando a criação e a disponibilização de soluções modulares e fracamente acopladas baseadas no conceito de serviços reutilizáveis (TEICH et al., 2014). Algumas das outras vantagens da utilização da programação para a *internet*, ou sistemas *web* são:

a) Desenvolvimento, manutenção e atualização centralizada da aplicação. Não é necessário instalar a aplicação em todos os equipamentos. É necessário apenas inseri-la no servidor, para que assim, todos possam ter acesso de diferentes tipos de dispositivos.
b) A linguagem de marcação de Hipertexto (HTML - *Hypertext Markup Language*), que é uma linguagem de marcação codificada para a leitura de informações em browsers já se tornou um padrão de desenvolvimento para a *internet*.
c) A exportação de dados entre usuários remotos utilizando o protocolo de comunicação HTTP (*Hypertext Transfer Protocol*).
d) Casa haja a necessidade de aumentar a capacidade de processamento, basta alterar a partir do servidor.
e) Os dados do cliente possuem uma proteção superior contra vírus.

### 3.1.2 Complexidade no desenvolvimento de projetos para a *internet*

Realizar o controle de projetos de desenvolvimento de *software* não é uma tarefa fácil, dessa forma é correto e essencial afirmar que o mesmo apresente o mínimo risco de falhas possíveis, visando a produção de produtos e serviços com a mais alta qualidade.

A grande demanda por novos produtos e serviços relacionados ao desenvolvimento de *softwares*, exerce pressão na concorrência pela busca por soluções com cada vez mais qualidade e aprimoramento da experiência de usuário (UX - *User Experience*). Ao mesmo tempo, para poder suprir as necessidades do mercado e sair na frente da concorrência, o setor de desenvolvimento teve que diminuir o tempo de colocação de um novo produto no mercado, além de diminuir os

orçamentos, o que também impactou no tempo disponível para a garantia da qualidade, e assim, contradizendo com as necessidades pela busca da mais alta qualidade do produto final (BRAUN; ELBERZHAGER; HOLL, 2017).

O desenvolvimento de *software* se torna complexo e desafiador, em um cenário com prazos curtos a serem cumpridos, em vários casos é evidente uma grande pressão na produção do *software*, tal fato acaba gerando problemas relacionados ao estresse, que por fim acaba gerando uma sobrecarrega com o acumulo de trabalho, para se entender o porquê de tal complexidade, Alami (2016) listou 3 itens:

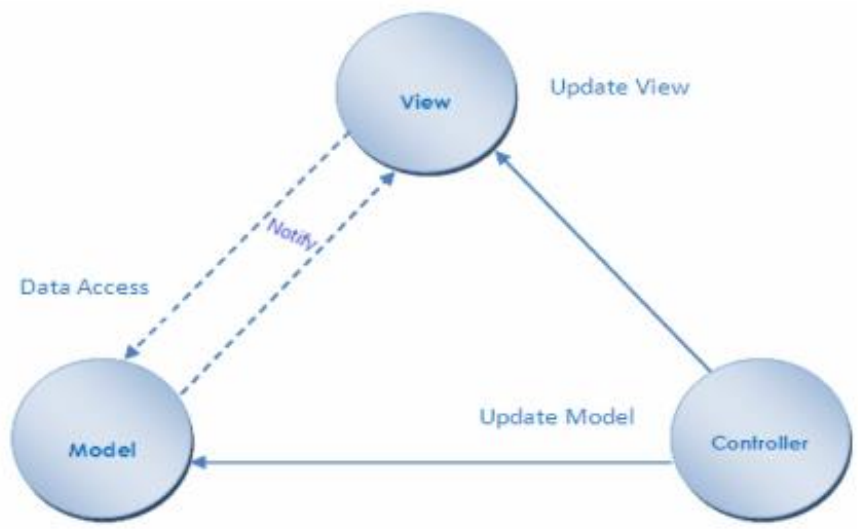
a) Os projetos falham porque não possuem um ecossistema de produção adequado. O ecossistema de um projeto deve ser mantido sobre equilíbrio. Sinais de perturbação devem ser detectadas e geridas em conformidade. Muito tem se feito quanto no sentido de melhorar os processos e práticas de *software*, mas assim como o próprio *software*, estas práticas são relativamente novas e ainda imaturas, é necessário uma padronização e nivelação em todo o ecossistema.
b) As entregas estão muitas vezes relacionadas aos cronogramas, porém o *software* é abstrato, ele não pode ser tocado, normalmente o cronograma para com a produção de um *software* não é alcançada, os requisitos são incompletos e mudanças são inevitáveis. Nenhum *software* é perfeitamente planejado no início, sempre vai haver mudanças necessárias, além de a tecnologia mudar rapidamente. Dessa forma, é necessário a adaptação através de estratégias de entregáveis em *roadmaps* (conjunto de menores funcionalidades em entregas mais rápidas, diminuindo assim a complexidade de desenvolvimento).
c) Falta de gestão de projetos. As tecnologias envolvidas em todo o ciclo de vida do *software*, são muitas e muito vastas, existe uma baixa cultura de gestão de projetos e uma falta de padronização nos processos de produção do *software*.

### 3.1.3 Padrão MVC

Para diminuir a complexidade de código é possível a separação do código em camadas MVC (*Model-View-Controller*), possibilitando o desenvolvimento de codificação semântica de fácil entendimento e manutenção. O MVC é uma abordagem

de *software* que separa a regra de negócio da apresentação (LANDICHO, 2018). É possível fazer uma correlação entre o MVC e, a camada *Front-end* e *Back-end*, onde a camada *View* é associada ao *Front-end*, que é a camada de visualização, e os demais *Model* e *Controller* sendo associados ao *Back-end*, onde o *Model* é a camada de modelagem de dados relacionados à um banco de dados, e o *Controller* é a camada de controle de uma requisição, onde é possível efetuar todo o controle sobre quais modelagens e resultados devem ser acessados e retornados para a *View*. A Figura 1 demonstra uma *View* recebendo dados da modelagem de um *Model* através de controle de requisição e resposta em um *Controller*.

Figura 1 - Diagrama de processo MVC.




Fonte: LANDICHO, 2018

### 3.1.4 Plataforma *Front-end* e *Back-end*

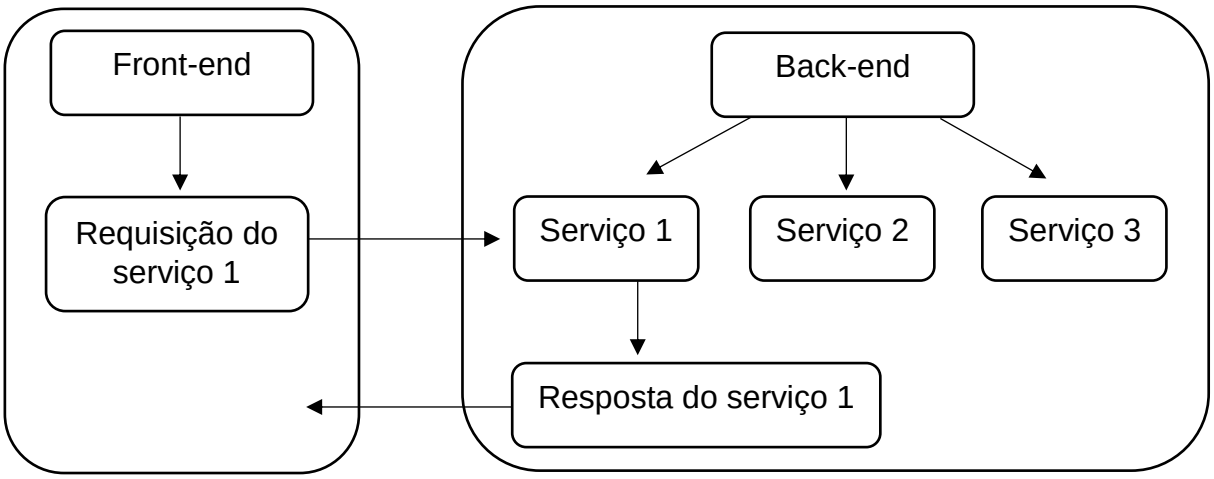
A arquitetura de programação para a *internet* consiste principalmente em dois componentes necessários para o seu funcionamento, sendo uma plataforma *Front-end* e outra *Back-end*. A plataforma *Front-end* é a interação do usuário com um sistema através de uma *interface* de usuário para a obtenção de serviços e a realização de requisições em plataformas *Back-end*. A requisição pode ser efetuada através de um navegador, sistema operacional, celular, *tablet* e outros dispositivos. Exemplos: Android (Sistema Operacional), *smartphone*, Google Chrome (*Browser*) e outros. A plataforma *Back-end* está relacionada a um servidor (*hardware* de alto

desempenho) de armazenamento de dados. O servidor mantém dados e regras de negócio que podem ser acessados pela *internet* através de uma requisição de um cliente (camada de código processado em computador utilizado por um usuário através de um ponto de requisição) (SHARMA, KUMAR, AGARWAL, 2015).

Um *software* desenvolvido para a *web* usa o HTML e o CSS (*Cascading Style Sheets*) para apresentar dados aos usuários e a interação é realizada pela linguagem de programação do JavaScript, essas tecnologias são chamadas de *Front-end,* pois são executadas no lado da camada do cliente. As tecnologias do lado do servidor ou camada *Back-end*, referem-se as tecnologias de armazenamento e processamento de dados que podem utilizar uma arquitetura de serviços reutilizáveis (POP, ALTAR, 2014).

O processo de requisição de uma máquina do tipo cliente é demonstrado na Figura 2, onde ela é o solicitante de um serviço ou informação. Na sequência, o processo do servidor aceita a solicitação do cliente, executa o processamento, reúne as informações, gera a solução para a demanda do cliente e envia uma resposta de solução gerada para o cliente. Após o envio, o servidor de serviços está pronto para aceitar outra demanda, são exemplo de servidores: servidor de impressão, *webservices*, servidor de banco de dados, servidor de bate-papo, servidor FTP (*File Transfer Protocol*) e outros.

Figura 2 - Requisição de um serviço na camada *Back-end*.

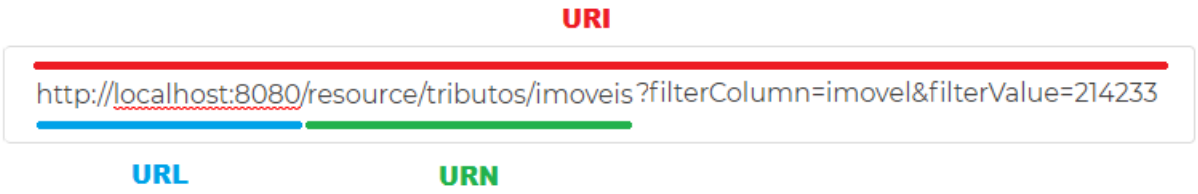



Fonte: SHARMA, KUMAR, AGARWAL, 2015

### 3.1.5 *Webservice* em arquitetura REST

Os *webservices* descrevem técnicas para a integração de *softwares* através de redes de computadores por meio de APIs (*Application Programming Interface*) de serviço da *web* que se comunicam por protocolo HTTP geralmente através de mensagens de envio em XML (*Extensible Markup Language*) ou JSON (*JavaScript Object Notation*) (GRAHL et al., 2017). As APIs de *webservices* podem ser acessadas através de Localizadores Uniformes de Recursos (URL - *Uniform Resource Locators*) e Nomes de Recursos Uniformes (URN - *Uniform Resource Names*), que são pontos de acessos onde se encontram os serviços a serem consumidos por uma aplicação cliente. A URI é composta pela URL, URN e parâmetros de identificação de acordo com a figura 3 (FIELDING, RESCHKE, 2014).

Figura 3 - Composição de uma URI.



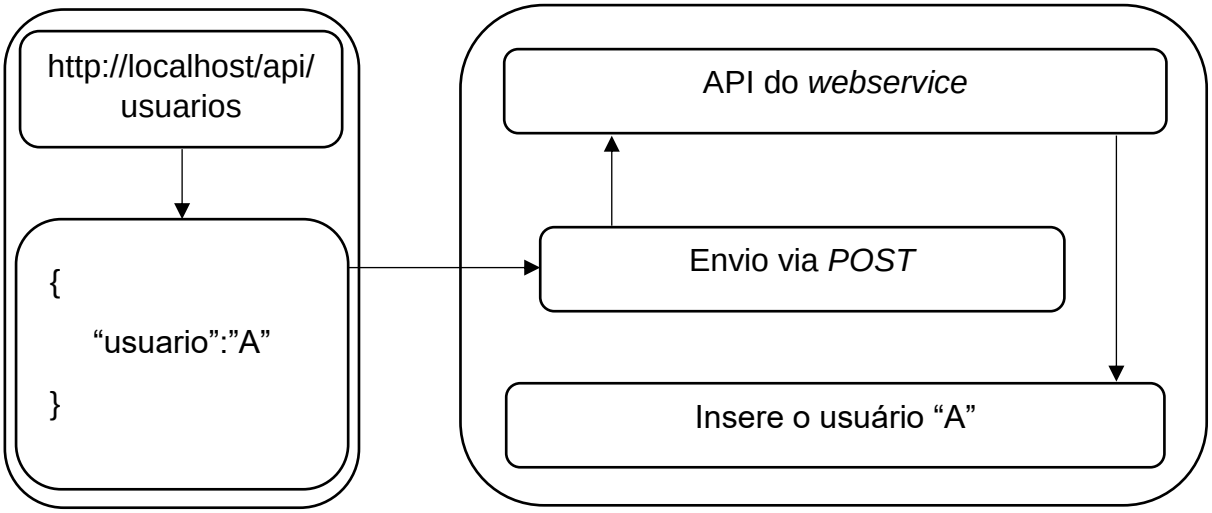

Fonte: Autor, 2018

A API é um conjunto de rotinas que constitui as aplicações, pode ter a característica de uma arquitetura RESTful, que é a capacidade de uma API possuir a implementação dos padrões HTTP (GRAHL et al., 2017). A arquitetura REST é composta por princípios que definem os padrões da *web* com especificações de acesso à métodos *POST*, *GET*, *PUT* e *DELETE* (YUAN et al., 2014).

#### 3.1.5.1 Método *POST*

O método *POST* é usado para criar um novo recurso de acordo com os dados de entrada (XML e JSON) no corpo de uma requisição. Exemplo: Requisição ao ponto de acesso “http://localhost/api/usuarios” através do método *POST* com dados de entrada JSON com nome de usuário, e como resultado um novo recurso inserido em um banco de dados de acordo com a figura 4.

Figura 4 - Requisição *POST*.

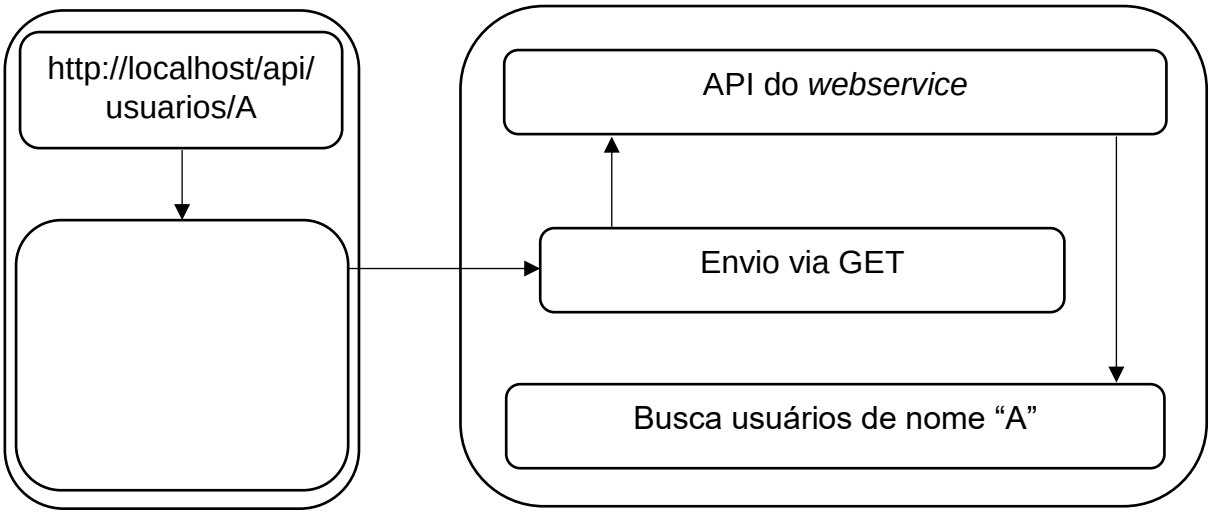



Fonte: YUAN et al., 2014

3.1.5.2 Método *GET*

O método *GET* é usado para recuperar algum recurso através de uma requisição. Exemplo: Requisição em um ponto de acesso “http://localhost/api/usuarios/A” através do método *GET* com dados de identificação de nome de usuário “A”, e como resultado uma busca de usuários de nome “A” de acordo com a figura 5.

Figura 5 - Requisição *GET*.

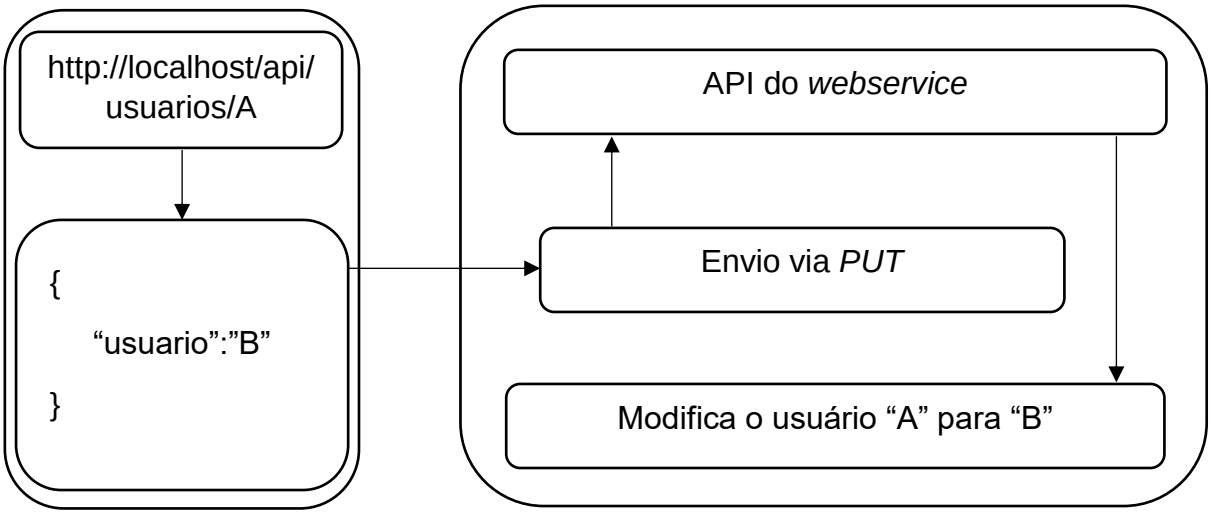



Fonte: YUAN et al., 2014

### 3.1.5.3 Método *PUT*

O método *PUT* é usado para modificar um recurso de acordo com os dados de entrada (XML e JSON) no corpo de uma requisição. Exemplo: Requisição em um ponto de acesso “http://localhost/api/usuarios/A” através do método *PUT* com dados de entrada JSON com um novo nome de usuário, e como resultado um recurso modificado em um banco de dados de acordo com a figura 6.

Figura 6 - Requisição *PUT*.

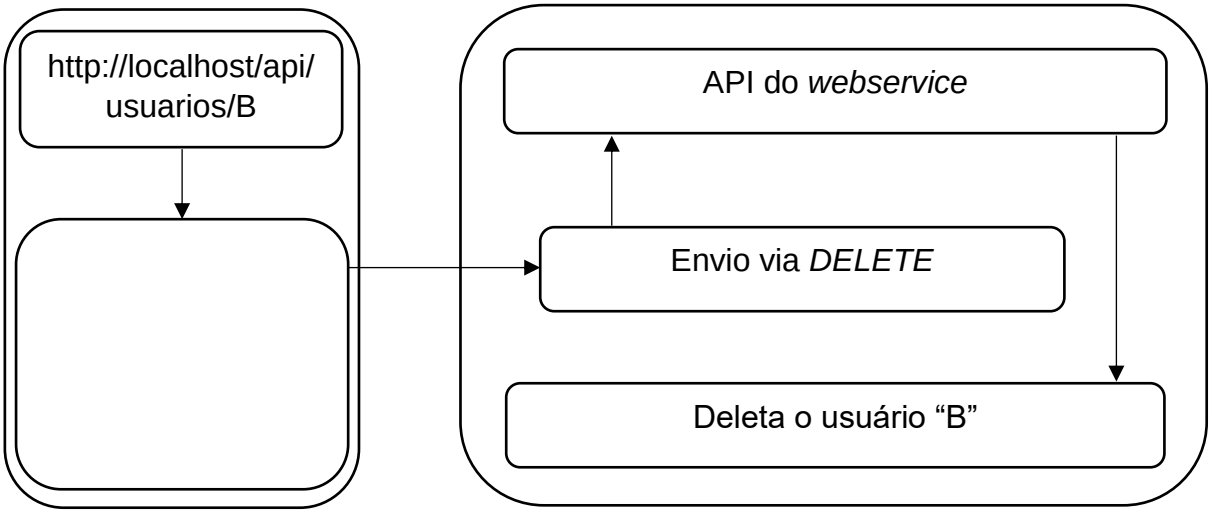



Fonte: YUAN et al., 2014

### 3.1.5.4 Método *DELETE*

O método *DELETE* é usado para deletar um recurso de acordo com a requisição. Exemplo: Requisição em um ponto de acesso “http://localhost/api/usuarios/B” através do método *DELETE* com dados de identificação de nome de usuário “B”, e como resultado um recurso deletado de um banco de dados de acordo com a figura 7.

Figura 7 - Requisição *DELETE*.




Fonte: YUAN et al., 2014

### 3.1.6 Código de status da resposta de um *webservice*

Cada requisição do método de acesso a um *webservice* produz um código de status de resposta, que é um código inteiro de três dígitos, onde fornece uma resposta sobre a tentativa de satisfazer uma requisição. O primeiro dígito do código de status da resposta representa uma classificação, sendo (FIELDING, RESCHKE, 2014):

a) 1xx (*Informational*): A requisição foi recebida e está sendo processada;
b) 2xx (*Successful*): A requisição foi recebida com sucesso, foi entendida e aceita, o que seria o processo natural sem erros e faltas da ação;
c) 3xx (*Redirection*): Indica que a requisição requerida necessita de outras ações para serem concluídas, como por exemplo, um determinado acesso passou a ser acessado por um outro caminho e dessa forma é retornado uma mensagem para uma ação de um redirecionamento;
d) 4xx (*Client Error*): A requisição não foi atendida devido à problemas com dados da requisição do *Front-end*, como por exemplo, dados de entrada da requisição não foram validados ou os dados de entrada não correspondem aos dados necessários à requisição, além de problemas com permissões de acesso, um *Front-end* pode não conter as permissões necessárias de acesso à uma requisição;

e) 5xx (*Server Error*): O *Back-end* não conseguiu atender a uma requisição válida devido à um erro ou uma falta.

#### 3.1.6.1 *Server Error*

O retorno de resposta de tipo *Server Error* trata-se de uma resposta do *Back-end* de que está ciente sobre um erro ou que ela é incapaz de executar a requisição requerida, as classificações de reposta 5xx *Server Error* são (FIELDING, RESCHKE, 2014):

a) 500 *Internal Server Error*: Indica que o servidor encontrou uma condição inesperada devido à um erro interno e impede que a execução da ação seja realizada;
b) 501 *Not Implemented*: Indica que o servidor não é capaz de atender ao método de solicitação da requisição por não ter uma implementação definida;
c) 502 *Bad Gateway*: Indica que o servidor, enquanto atuando como um *gateway* ou *proxy* (Computador que atua como um intermediário entre uma máquina cliente e um servidor, armazenando informações em memória para economizar tempo de acesso, além de filtrar e restringir acessos) recebeu uma resposta inválida de um servidor quando tentava atender a requisição;
d) 503 *Service Unavailable*: Indica que o serviço está temporariamente indisponível por se tratar de uma sobrecarga ou parada programada para manutenção e reparo;
e) 504 *Gateway Timeout*: Indica que o servidor, enquanto atuando como um gateway não recebeu uma resposta em tempo que precisava para a ação ser concluída;
f) 505 HTTP *Version Not Supported*: Indica que o servidor não suporta a versão de comunicação em que a requisição foi solicitada.

### 3.1.7 Erro, falta e falha de *software*

Um *software* pode apresentar problemas em seu funcionamento através de erros, faltas e falhas. De acordo com o *The Institute of Electrical and Electronics Engineers* (IEEE, 1990):

a) Erro: Um erro é uma diferença entre um resultado computado e um resultado correto. O erro é ocasionado por uma ação humana que produz um resultado incorreto, como por exemplo, uma instrução de um código que deveria efetuar validação de campos nulos não é realizada, ocasionando uma exceção quando um determinado cálculo necessitar do campo para um processamento matemático e realizar o processamento com um campo nulo gerando um resultado incorreto, que por consequência produz um mau funcionamento através de um defeito de sistema. O erro pode ser associado ao retorno da resposta de código de status “500 *Internal Server Error”*.
b) Falta: Uma falta é um defeito em um dispositivo de *hardware* ou sistema que pode fazer com que um componente ou sistema falte com a sua responsabilidade ao executar uma funcionalidade requisitada. A falta ocorre devido a um erro (500 *Internal Server Error*) gerado por um sistema fazendo com que uma funcionalidade requisitada produza um defeito no seu funcionamento. A falta também pode ocorrer por um problema através de indisponibilidade da funcionalidade devido a uma sobrecarga de um equipamento, de um circuito elétrico queimado e, de servidores desligados e inoperantes, podendo ser associado ao retorno da resposta de código de status “501 *Not Implemented*”, “502 *Bad Gateway*”, “503 *Service Unavailable”, “*504 *Gateway Timeout” e “*505 HTTP *Version Not Supported”*.
c) Falha: Uma falha é a incapacidade de um sistema ou componente para executar suas funções necessárias dentro dos requisitos especificados. Ela é o desvio de uma entrega, serviço ou resultados esperados. Um sistema pode falhar porque não está em conformidade com a especificação ou porque a especificação não descreveu adequadamente a sua funcionalidade. Uma falta é a parte do estado do sistema que pode causar uma falha subsequente, essa falha ocorre quando uma falta é processada e retornada para a requisição que o requisitou, porém, nem toda a execução de uma falta é uma falha, uma falta pode traçar uma nova rota para tentar solucionar o problema, por exemplo, ao ser requisitado uma informação o servidor retornaria uma resposta de status inoperante, mas ao invés de retornar a resposta, o processo é enviado para uma nova rota tentando acessar a informação através de outra ação e se essa ação ocorrer como desejado então é retornado a resposta de sucesso para o requisitante.

## 3.2 GESTÃO DA QUALIDADE

### 3.2.1 Qualidade de *software*

A qualidade do *software* é determinada pelo processo de produção do *software* (PANE, SARNO, 2015), uma fábrica de *software* bem-sucedida é aquela que fornece *software* de qualidade capaz de atender às necessidades dos requisitos das partes interessadas, e para que o *software* corresponda às expectativas é necessária uma maneira disciplinada de obter os requisitos reais do *software*. Para produzir *softwares* de qualidade e eficiência, com o mínimo de desperdício e de retrabalho, será necessário dispor as pessoas certas, das ferramentas adequadas e do enfoque correto. Para fazer tudo isso de maneira previsível e consistente, com uma avaliação dos custos reais, é necessário um processo gerenciável que permita uma melhoria contínua da qualidade (BOOCH, RUMBAUGH, JACOBSON, 2000).

#### 3.2.1.1 Dívida técnica

Dívida técnica são todas as atividades inerentes à produção que o desenvolvedor ou equipe de desenvolvimento vai deixando para trás inconscientemente ou conscientemente sem controle. As dívidas mais comuns envolvem atividades que os desenvolvedores não gostam de fazer e que não são obrigatórias para se obter um resultado, porém com o aumento dessa dívida as chances de falhas geradas durante e após a produção também aumentam, e com isso, a complexidade do projeto aumenta e o risco de ter uma baixa qualidade do *software* também aumenta. Algumas das atividades da dívida técnica são: documentação, testes, resolução de *warnings* (alerta de código com erro, mas que não impede a execução de uma funcionalidade) indicados pelo compilador (tradutor de códigos em linguagem textual para linguagem de computador), *bugs* (defeitos de *software*) difíceis ou chatos que ninguém quer se responsabilizar e códigos que precisam de uma nova iteração para solucionar algum problema. Essas dívidas ocorrem por desconhecimento do que fazer, desprezo pela atividade ou por falta de condições de realizar as tarefas, seja por falta de recursos (tempo, dinheiro ou pessoas capacitadas) ou por uma situação legada que dificulta a sua realização (falta de entendimento e decisões erradas no gerenciamento ou arquitetura) (HUUMOA, MAGLYAS, SMOLANDER, 2016).

De acordo com Hamdana e Alramounib (2015), para se ter uma integração contínua na produção de um *software* com qualidade é necessário a utilização de boas práticas de codificação como:

a) Produção e disponibilização para o usuário final com entregáveis menores (Disponibilização do *software* com menos funcionalidades) porém frequentes;
b) Não disponibilizar códigos quebrados (Códigos que contenham falhas);
c) Ter uma rápida ação de reparos de códigos quebrados;
d) Efetuar e documentar testes;
e) Validar através de verificação e aprovação de todos os testes.

3.2.1.2 Modelo integrado de maturidade em capacitação

O CMMI é uma estrutura para o avanço do processo de desenvolvimento de *software*. Os modelos CMMI oferecem referências que podem ser usadas para se identificar o nível de maturidade dos processos na produção de um *software* (PANE, SARNO, 2015). Os níveis de maturidade do CMMI são:

a) Nível inicial, o processo é caracterizado como sendo imprevisível e ocasionalmente caótico. Poucos processos são definidos e o sucesso depende de esforços individuais e, muitas vezes, heroicos;

b) Nível gerenciado, processos básicos de gerenciamento de projeto são estabelecidos para controle de custos, prazos e escopo. A disciplina de processo permite repetir sucessos de projetos anteriores em aplicações similares;

c) Nível definido, um processo composto por atividades de gerenciamento e engenharia é documentado, padronizado e integrado em um processo padrão da organização. Todos os projetos utilizam uma versão aprovada e adaptada do processo organizacional para desenvolvimento e manutenção de produtos e serviços tecnológicos;

d) Nível quantitativamente gerenciado, métricas detalhadas dos processos e dos projetos são coletadas. Tanto os processos como os projetos são quantitativamente compreendidos e controlados;

e) Nível em otimização, a melhoria contínua do processo é estabelecida por meio de sua avaliação quantitativa, e da implantação planejada e controlada de tecnologias e ideias inovadoras.

### 3.2.2 Filosofia da qualidade de Philip Bayard Crosby

Crosby (1979), define a qualidade como o cumprimento aos requisitos e ela é medida pela não conformidade, que são os custos gerados pelo não cumprimento dos padrões da qualidade no processo de produção, tais como a falta de planejamento e documentação, as más práticas de produção e o não cumprimento das especificações do escopo de um projeto (PHILIP CROSBY ASSOCIATES, 2001), sendo possível correlacionar o custo da não conformidade com os itens que geram a dívida técnica do desenvolvimento de *software*, podendo a qualidade ser medida através do custo necessário para suprir e/ou reparar uma falha da produção das funcionalidades de um *software*. Crosby foi um praticante de gestão da qualidade por mais de 35 anos, suas contribuições foram um despertar global ao longo das últimas duas décadas, elas são reconhecidas por toda a comunidade mundial de qualidade e sua influência se estendeu ao mundo até líderes de negócios internacionais. Ele desenvolveu conceitos que são considerados elementos fundamentais do corpo de conhecimento de qualidade, incluindo os Quatro Absolutos da Qualidade da Gestão:

a) Qualidade significa conformidade com os requisitos: Tende-se a fazer o certo desde a primeira vez;
b) Qualidade é conseguida através da prevenção: Vacinação é a rota para prevenir o desastre organizacional. Prevenção se origina do treinamento, da disciplina, do exemplo e da liderança;
c) Qualidade tem um padrão de Zero Defeito: A qualidade deve ter um padrão de zero defeito e não níveis de qualidade aceitáveis;
d) Qualidade é medida pelo preço da não conformidade: A medição deve ser feita através dos custos relacionados aos reparos de problemas com as falhas encontradas, e não através de índices.

### 3.2.3 Controle estatístico de processo

O Controle Estatístico de Processo (CEP) envolve um conjunto de atividades para garantir que produtos e serviços atendam aos requisitos necessários e que a qualidade se mantenha em constante melhoria (VIEIRA, 2012). A gestão do CEP é caracterizada pelas diversas ferramentas da qualidade, e cada uma delas tem a sua própria utilização dependendo dos dados coletados e dos dados históricos

disponíveis, permitindo assim detectar aspectos cruciais para solução de problemas e melhorias de processos.

De acordo com Vieira (2012), algumas ferramentas utilizadas para gestão da qualidade são:

a) Folha de verificação: tabela ou planilha estruturada nos quais os dados coletados são registrados;
b) Estratificação: técnica onde se classifica em grupos semelhantes itens semelhantes e relevantes para a análise dos dados;
c) Diagrama de Pareto: permite visualizar as causas e variações por ordem de importância;
d) Histograma: mostra graficamente a maneira como os dados se distribuem, descrevendo a frequência com que variam os processos evidenciando a distribuição dos dados como um todo;
e) Diagrama de causa e efeito ou diagrama de Ishikawa: representação gráfica que possibilita a identificação das possíveis causas para um problema e organiza “ideias soltas” em categorias;
f) Diagrama de dispersão: gráficos que permitem a identificação entre causas e efeitos, para avaliar o relacionamento entre as variáveis;
g) Gráficos de controle: gráficos que permitem examinar se o processo está ou não sob controle.

# 4 MATERIAIS E MÉTODOS

Esta sessão apresenta os matérias e métodos utilizados referente ao desenvolvimento da ferramenta *web* de gestão de testes que é a ferramenta de coleta de dados relacionados as falhas de *software*, ao processo da filosofia do Zero Defeito de Crosby e a avaliação do impacto sobre a qualidade de acordo com os dados coletados a partir da implantação do novo processo.

## 4.1 FERRAMENTA DE GESTÃO DE TESTES

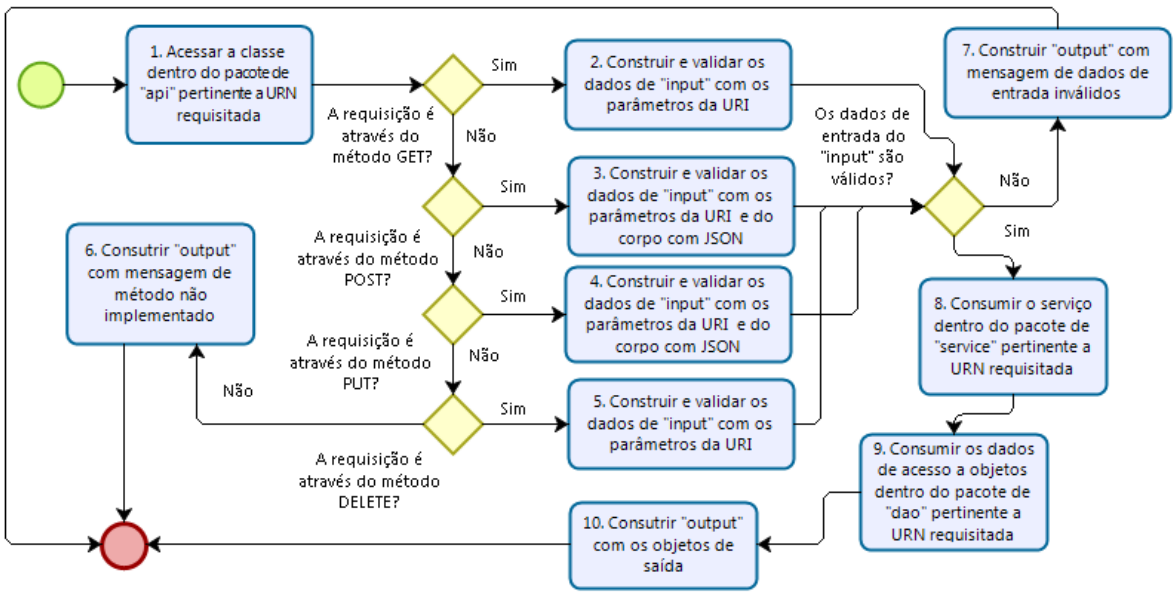
A ferramenta desenvolvida para a gestão dos testes utiliza o protocolo de comunicação HTTP enviando e recebendo dados de *inputs* (entrada de dados) e *outputs* (saída de dados) respectivamente. Os dados são verificados através de técnica de validação a partir do status de retorno da resposta processada por uma requisição. A técnica foi aplicada para testar as funcionalidades da produção na camada *Back-end* de sistemas *web* em arquitetura REST através do consumo em APIs de *webservices* e coletar informações relacionadas à identificação de falhas de *software*.

Além disso, foi utilizado o modelo de direção do CMMI para identificar o nível de maturidade dos processos de produção em uma fábrica de *software* do setor público situada em município do interior do Estado de Minas Gerais, dessa forma, a fábrica foi avaliada em nível inicial por não possuir padrões de processos bem definidos. Sendo assim, foi proposto uma padronização do desenvolvimento de *software* e os processos de produção para todo o ciclo de vida dos projetos.

### 4.1.1 Padronização do desenvolvimento *Back-end*

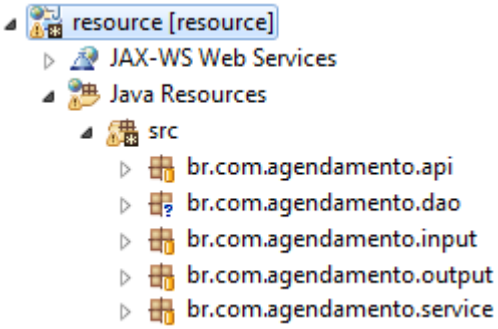
Para a padronização que se refere ao desenvolvimento *Back-end* do *software*, foi definido um processo conforme a figura 8 para facilitar a manutenção e diminuir a complexidade de código, sendo os pacotes de projeto divididos em “api”, “input”, “service”, “dao” e “output” conforme a figura 9.

Figura 8 - Processo de acesso a uma API.




Fonte: Autor, 2018

Figura 9 - Divisão de pacotes do projeto.




Fonte: Autor, 2018

O pacote de api é responsável por identificar as URNs das requisições, podendo elas serem acessadas por método *GET*, *POST*, *PUT* e *DELETE*, tendo assim, a responsabilidade em chamar os respectivos serviços. O pacote de *input* é responsável pelas regras de negócio relacionado aos dados de entrada para o bom funcionamento da API. O pacote de *service* é o *controller* da camada MVC, que controla as requisições e consome a modelagem de dados. O pacote de *model* é a camada de modelagem do MVC, que faz acesso aos dados de uma base. E o pacote *output* é responsável na conversão de dados da modelagem em dados de saída para a resposta. Com os requisitos de entrada e saída de dados bem definidos, fica de fácil

acesso e manutenção das definições referentes as regras de negócio. As APIs produzidas seguindo essa padronização foram o objeto de análise para a verificação da melhoria da qualidade com a implantação da gestão baseada na filosofia de Crosby.

### 4.1.2 Desenvolvimento da ferramenta

Para alcançar o objetivo de desenvolvimento de uma ferramenta *web* de gestão de testes que colete dados categorizados em status de falhas de respostas de requisições em pontos de acesso na camada *Back-end* de sistemas *web* em arquitetura REST, foram utilizados materiais de desenvolvimento de acesso livre. Para o desenvolvimento *Front-end* da ferramenta foi utilizado o HTML, CSS e JavaScript, para o desenvolvimento *Back-end* da ferramenta foi utilizado a linguagem de programação PHP (*Hypertext Preprocessor*) e para o banco de dados foi utilizado estrutura relacional de linguagem SQL (*Structured Query Language*) com o auxílio do gerenciador do MySQL para a criação e gerenciamento das tabelas necessárias.

#### 4.1.2.1 Linguagem de marcação de hipertexto

HTML é uma linguagem de marcação para criar páginas da *web* e aplicativos da web (ABHIJIT, JOSHIB, 2015). Ele descreve a estrutura de páginas da *web* usando marcação com elementos HTML, que são os blocos de construção de páginas HTML, esses elementos são representados através de *tags* que descrevem conteúdos como títulos, subtítulos, parágrafos e tabelas. Essa linguagem de *script* é usada devido ao suporte a vários navegadores, ela também tem a capacidade de fornecer desde animações a gráficos (LANDICHO, 2018).

#### 4.1.2.2 Folhas de estilo em cascata

CSS é uma linguagem que descreve o estilo de um documento HTML. Ele descreve como os elementos devem ser estilizados e exibidos na tela (SAPUTRA, AZIZAH, 2013). Esta linguagem de folha de estilo é usada para o design do site, onde oferece compatibilidade de navegador que torna o design mais rápido e fácil (LANDICHO, 2018).

#### 4.1.2.3 Linguagem de programação JavaScript

O JavaScript é a linguagem de programação que descreve o comportamento de uma página da *web*. Todas as interações de usuário e máquina são feitas através do JavaScript e as suas execuções são processadas pelo computador do usuário (código processado do lado do cliente) (GRAHL et al., 2017). É usado para manipulação de elementos HTML, principalmente efetuar sobreposições de elementos DOM (mapa de hierarquia em elementos HTML) (LANDICHO, 2018).

#### 4.1.2.4 Linguagem de programação PHP

PHP foi criado em 1994 por Ramus Ledorf e é uma linguagem de *script* de código aberto amplamente utilizada, estudos mostram que mais de 80% dos projetos *web* são escritos em PHP (PROKOFYEVA, BOLTUNOVA, 2017).

#### 4.1.2.5 Gerenciador de banco de dados MySql

MySql é um gerenciador de banco de dados SQL relacional, através dele é possível criar, acessar, editar e excluir dados (CIORANUA, CIOCAB, NOVAC, 2015). O sistema de banco de dados MySQL é *open source*, sem custo e de fácil aprendizado, além de fácil conexão em uma integração de APIs com o PHP (LANDICHO, 2018).

## 4.2 PROCESSO DE PRODUÇÃO DA CAMADA BACK-END BASEADO NA FILOSOFIA DA QUALIDADE DE CROSBY

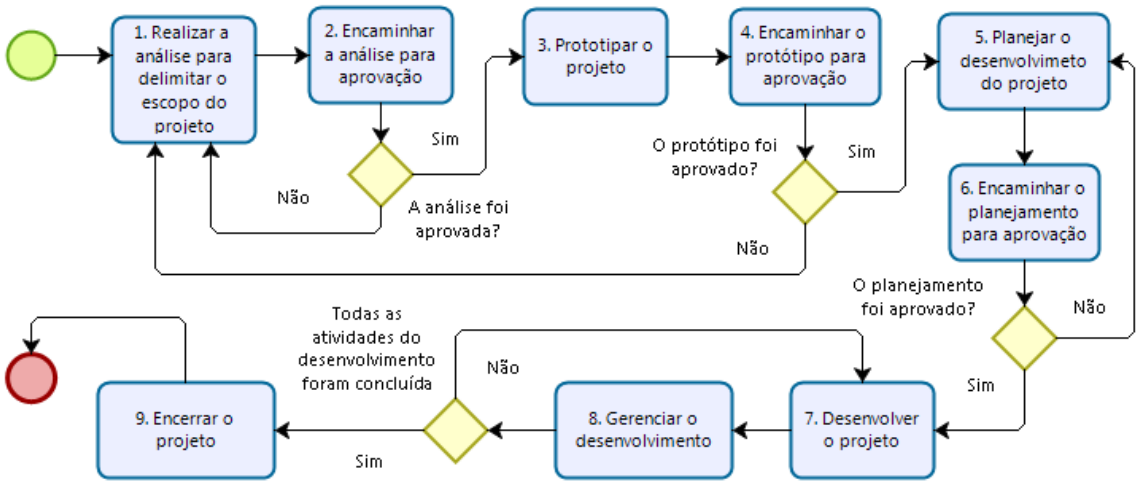
Para alcançar o objetivo de implantação da filosofia da qualidade de Crosby foi proposto processos para atender aos Quatro Absolutos da Qualidade.

### 4.2.1 Conformidade com os requisitos

Para a atender o primeiro item dos 4 absolutos de Crosby que se refere as conformidades com os requisitos, foi proposto fases de aprovações para todas as ações antes de iniciar qualquer desenvolvimento, e uma vez definido os requisitos,

tende-se a fazer o certo desde a primeira vez, não gerando retrabalhos e realizando entregáveis de acordo com as expectativas conforme a figura 10.

Figura 10 - Processo do ciclo de vida do desenvolvimento de *software*.




Fonte: Autor, 2018

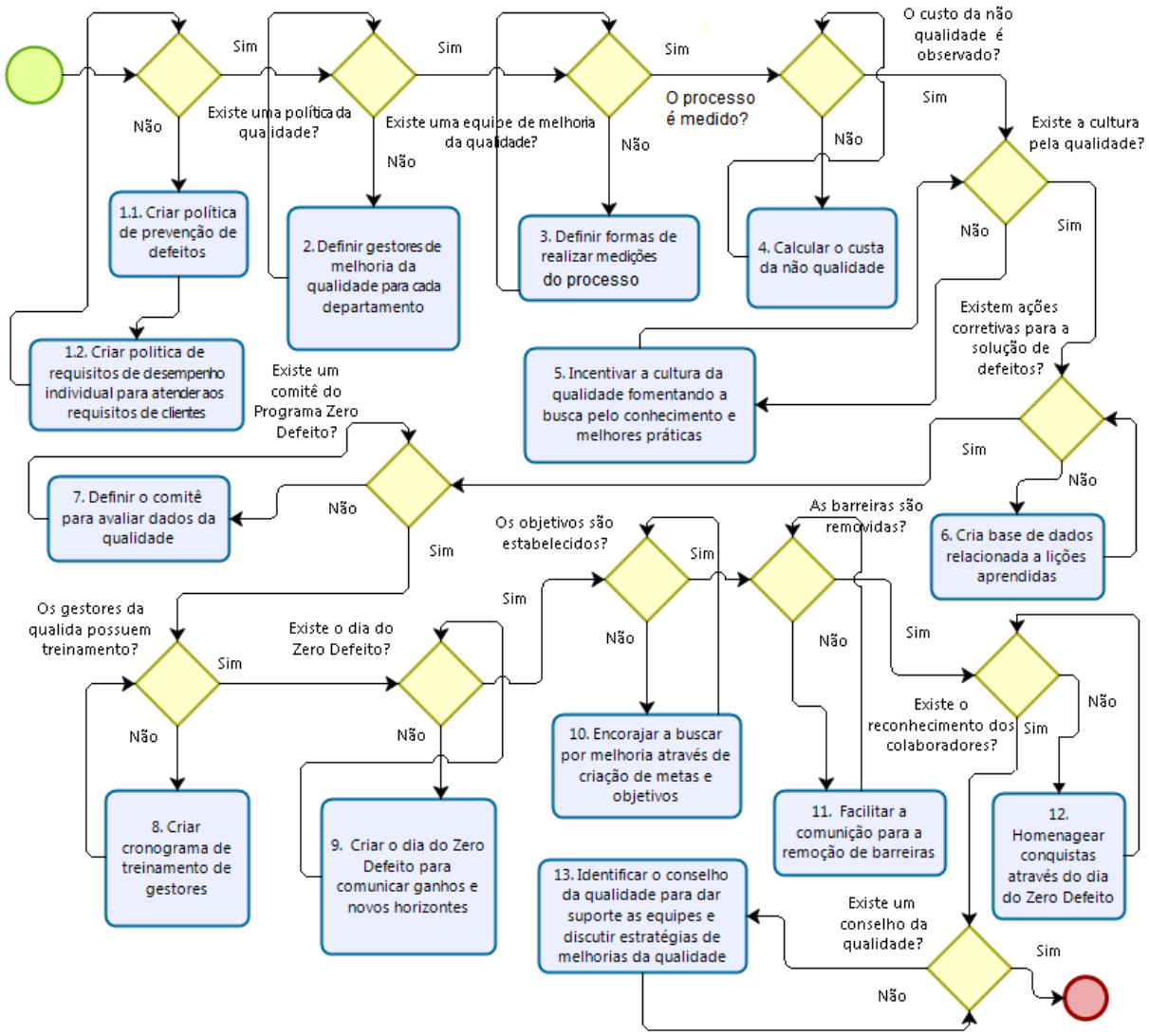
### 4.2.2 Prevenção da qualidade

Para atender ao segundo item dos 4 absolutos de Crosby que se refere as prevenções, foi seguido o programa de 14 etapas para a melhoria da qualidade de Crosby conforme a figura 11, onde se demonstra como conseguir o envolvimento e o apoio de todos os seus funcionários para a melhoria da qualidade, e os ajuda a entender e apreciar o verdadeiro significado de qualidade (ASQ, 2005), eles são separados em:

1) Pessoal de gerenciamento comprometido com a qualidade: Todo o pessoal de gestão, de todas as áreas da empresa, deve estar comprometido com qualidade, e uma política clara de execução das atividades devem ser divulgadas;
2) Formação de times gerenciais da qualidade: Times gerenciais da qualidade devem ser formados, cuja finalidade é discutir e patrocinar ações para melhoria da qualidade;
3) Medição dos processos: Deve ser estabelecida medidas dos processos para identificar os problemas de qualidade atuais e potenciais, para cada área;

4) Análise do custo da não qualidade: A avaliação do custo da não qualidade deve levar a empresa a identificar onde e em quais ações devem ser tomadas para a melhoria da qualidade;
5) Formação de cultura da qualidade: Deve ser criada uma cultura para a qualidade, conscientizando cada colaborador sobre a importância da qualidade, através de treinamento e compartilhamento de informações;
6) Ações corretivas: Os problemas encontrados de qualidade devem ser corrigidos rapidamente através de ações corretivas, de modo que os colaboradores percebam que existe um sistema que realmente promove a melhoria;
7) Estabelecer um comitê do Programa Zero Defeitos: Deve ser estabelecido um comitê, cuja função é avaliar a situação atual e monitorar os progressos, através de ações de conscientização da necessidade de que todos executem suas funções conforme os requisitos;
8) Treinamento dos supervisores: Todos os gestores devem ser treinados sobre os conceitos do programa e sobre gestão de melhoria;
9) Dia do Zero Defeitos: Deve ser criado um "dia do zero defeitos", cuja finalidade é promover o Programa Zero Defeitos, reavivando a necessidade das melhorias, comunicando ganhos e informando novas perspectivas;
10) Estabelecimento de objetivos: Todos os gestores devem encorajar seu pessoal a estabelecer objetivos de melhoria, com prazos de 30 a 90 dias;
11) Remoção de barreiras: Todos os gestores devem orientar seu pessoal que descrevam barreiras existentes que impedem o trabalho de ser feito com qualidade;
12) Reconhecimento: É importante haver o devido reconhecimento aos participantes que atingem metas ou que tem desempenho notável com um prêmio (evitando-se prêmios de ordem financeira, mas sim, de honraria), de modo a estimular a atuação e engajamento dos demais;
13) Conselhos da qualidade: Devem ser criados "Conselhos da qualidade", formados por profissionais da qualidade e líderes, cuja finalidade é avaliar e discutir ações de melhoria;
14) Recomeçar: Após o encerramento do programa, deve haver um relançamento, com alterações de representantes e times, com o objetivo de recomeçar e dar novo fôlego ao programa.

Figura 11 - Fluxograma dos 14 passos da filosofia de Zero Defeito de Crosby.




Fonte: Autor, 2018

Como estratégia para o início da cultura pela qualidade foi criado a política da qualidade conforme o apêndice F, que demonstrando a importância da gestão dentro da fábrica de *software*. Equipes de qualidade foram criadas para discutir processos gerenciáveis da qualidade, para analisar os resultados e para facilitar a comunicação entre as partes interessadas. Com a criação do dia do Zero Defeito foi possível alinhar os objetivos a serem alcançados, demonstrar ganhos e homenagear as conquistas de toda a organização, engajando assim todos os envolvidos com os projetos. Além disso, foi possível definir cronogramas de treinamentos e alinhamentos dos conhecimentos para que a fábrica caminhe de forma saudável e em conjunto através de um ambiente colaborativo e produtivo. Todos os avanços foram documentados em

base de dados de lições aprendidas para que sirvam de informação para novos projetos e que se tornem a base para a melhoria contínua da qualidade.

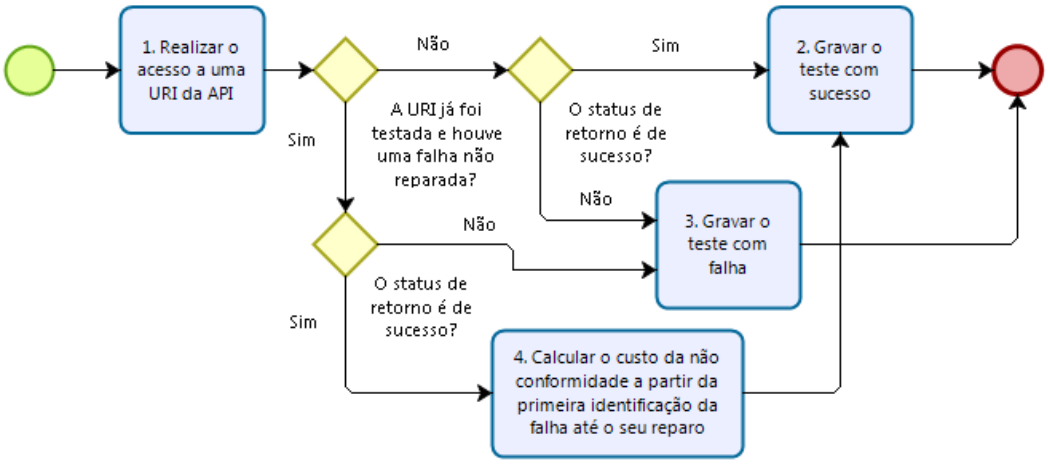
### 4.2.3 Padrão Zero Defeito

Para atender ao terceiro item dos 4 absolutos de Crosby que se refere ao padrão Zero Defeito, foi utilizado a ferramenta *web* de gestão de testes da camada *Back-end* para a coleta de informações relacionadas à identificação de falhas de *software* e analisadas em Gráficos *u* para a avaliação da melhoria da qualidade na busca pelo padrão Zero Defeito.

### 4.2.4 Preço da não conformidade

Para atender ao último item dos 4 absolutos de Crosby que se refere ao preço da não conformidade, foi proposto a análise dos dados obtidos pela ferramenta de gestão de testes a partir da identificação de uma falha até o seu reparo, todos os testes produzidos dentro da ferramenta foram gravados com a data, horário e o colaborador que a executou, dessa forma, foi possível medir e demonstrar o preço da não conformidade de acordo com o tempo gasto até que um teste com falha seja reparado. É importante orientar toda a equipe de produção sobre a importância da qualidade, pois uma vez atendido aos requisitos desejados desde a primeira vez, é gerado uma redução de custos e tempo com a eliminação de retrabalhos conforme a figura 12.

Figura 12 - Fluxograma de identificação de não conformidade referente a falha da API.




Fonte: Autor, 2018

## 4.3 CONTROLE ESTATÍSTICO DO PROCESSO

Para o objetivo que se refere a avaliar a melhoria da qualidade que é a realização de um esforço contínuo para a melhoria de produtos, serviços e processos (GEJDOS, 2016), foi utilizado uma das sete ferramentas do controle estatístico de processo que é a análise entre Gráficos *u*, sendo que em um primeiro momento foram analisados os dados referentes as falhas coletados pela ferramenta de gestão de testes por um período de 30 dias antes da implantação do processo da filosofia de Crosby, e em um segundo momento foram analisados outros dados coletados por um período de 30 dias após a implantação do processo de Crosby, a análise do impacto da qualidade foi através da comparação entre as duas amostragens de dados convertidas em Gráficos *u*. Os dados coletados foram classificados em unidades por URNs de identificação de APIs testadas por dia e o número de não conformidades foram associados ao número total de falhas verificadas nessas unidades inspecionadas. O objeto de análise foi composto por toda a produção referente a camada Back-end de sistemas *web* em arquitetura REST de todo o portfólio de uma fábrica de *software* do setor público situada em município do interior do Estado de Minas Gerais classificada como "nível inicial" segundo o CMMI pela razão de se investigar qual é o impacto na qualidade ao se utilizar o processo da filosofia do Zero Defeito de Crosby na busca pela melhoria contínua da qualidade em uma fábrica aonde não tenha os seus processos bem definidos.

### 4.3.1 Gráficos de controle

Os gráficos de controle, também chamados de cartas de controle, foram desenvolvidos e aplicados por Walter A. Shewhart por volta de 1942 (COSTA, 2007), eles são utilizados para fazer a análise dos dados obtido e identificar as causas especiais existentes em um processo. A partir dos dados obtidos na análise destes gráficos pode-se tomar decisões preventivas e controlar possíveis desvios de variabilidade no processo produtivo (MICHEL, 2002).

Segundo Mayer (2004), os gráficos de controle permitem entender e visualizar os resultados dos processos, caso os processos estejam fora de controle, os próprios operadores dos processos têm a oportunidade de atuarem imediatamente sobre as causas especiais, contribuindo para o ajuste e a estabilização do processo.

Existem diferentes tipos de gráficos de controle, dentre eles os gráficos de controle para atributos e cartas de controle para variáveis, a escolha de qual utilizar dependerá da característica da qualidade a ser controlada. As cartas de controle para atributos se caracterizam por características qualitativas (proporção de não conformes, número de não conformes (defeitos) e número médio de não conformidades por unidade), já as cartas de controle para variáveis se caracterizam por características quantitativas (valor individual, média, amplitude e desvio padrão).

Para o controle qualitativo, existem quatro tipos de cartas de controle conforme descritas na Tabela 1, onde cada tipo de carta de controle se aplica melhor a uma determinada situação.

Tabela 1 - Os gráficos de controle, definições e objetivos.

| Gráficos | Definição |
|---|---|
| **Gráfico de Controle *np*** | Número de itens não conforme na amostra de mesmo tamanho. |
| **Gráfico de Controle *c*** | Número de defeitos ou não conformidades por unidade em amostras de tamanho constante. |
| **Gráfico de Controle *p*** | Porção de itens não conforme por unidade de amostras de tamanho constante. |
| **Gráfico de Controle *u*** | Número médio de defeitos por unidade de amostras de tamanho variável. |

Fonte: Mayer, 2004, p. 29

Os gráficos de controle para atributos referem-se à característica da qualidade que pode estar, ou não, conforme as especificações. É comum utilizar-se os termos “conforme” e “não conforme”.

O Gráfico *u*, ou Carta de Controle *u*, é um gráfico de controle de atributos usado com dados coletados de unidades de amostragem com subgrupos de inspeção de tamanhos variados. As não conformidades observadas nos subgrupos de unidades

da amostragem determinam se o processo é estável, além de monitorar os impactos na melhoria da qualidade a partir da implantação de um novo processo. A análise é feita de acordo com os limites superiores e inferiores estabelecidos pelo cálculo a partir da coleta de amostras relacionadas à uma quantidade de não conformidades observadas (MONTGOMERY, 2009).

Para se calcular a taxa de não conformidades por unidade utilizou-se a fórmula conforme a Figura 13, onde D*i* é a quantidade de não conformidades observadas e o *ni* é o número de subgrupos de tamanho variados.

Figura 13 - Cálculo da taxa de não conformidades por subgrupos de tamanhos variados.

$$u_i = \frac{D_i}{n_i}$$

Fonte: MONTGOMERY, 2009

Para identificar a linha central é calculado através do *u* barra, onde se é dado pela soma de todas as taxas de não conformidades divididas pelo número de unidades de amostragem (Figura 14).

Figura 14 - Cálculo da linha central.

$$\overline{u} = \frac{\sum_{i=1}^{k} D_i}{\sum_{i=1}^{k} n_i}$$

Fonte: MONTGOMERY, 2009

Para se calcular as linhas superior e inferior são calculados usando as fórmulas correspondentes as Figuras 15 e 16. Onde *m* é um multiplicador (geralmente definido como 3) para controlar a probabilidade de sinais fora de controle quando o processo estiver no controle.

Figura 15 - Cálculo da linha superior.

$$\overline{u} + m\sqrt{\frac{\overline{u}}{n_i}}$$

Fonte: MONTGOMERY, 2009

Figura 16 - Cálculo da linha inferior.

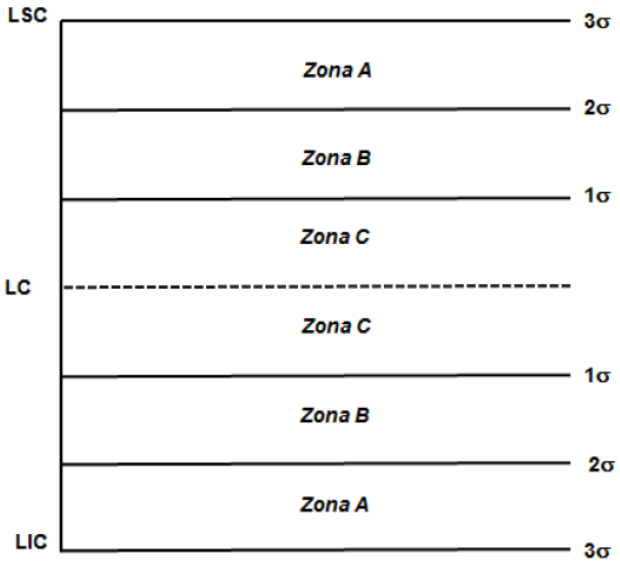
$$\bar{u} - m\sqrt{\frac{\bar{u}}{n_i}}$$

Fonte: MONTGOMERY, 2009

A interpretação dos gráficos de controle é verificada através dos pontos que estão fora dos limites superior e inferior. A norma ISO 8258 - Shewhart Control Charts, estabelece os seguintes critérios de decisão em gráficos de controle de acordo com a Figura 17 (OLIVEIRA et al., 2013):

a) 1 ou mais pontos acima do LSC ou abaixo do LIC;
b) 9 pontos consecutivos na zona C ou no mesmo lado do LC;
c) 6 pontos consecutivos, todos aumentando ou todos diminuindo;
d) 14 pontos consecutivos alternando para cima e para baixo;
e) 2 de 3 pontos consecutivos na zona A ou além dela;
f) 4 de 5 pontos consecutivos na zona B ou além dela;
g) 15 pontos consecutivos na zona C (tanto acima quanto abaixo do LC);
h) 8 pontos consecutivos na zona B.

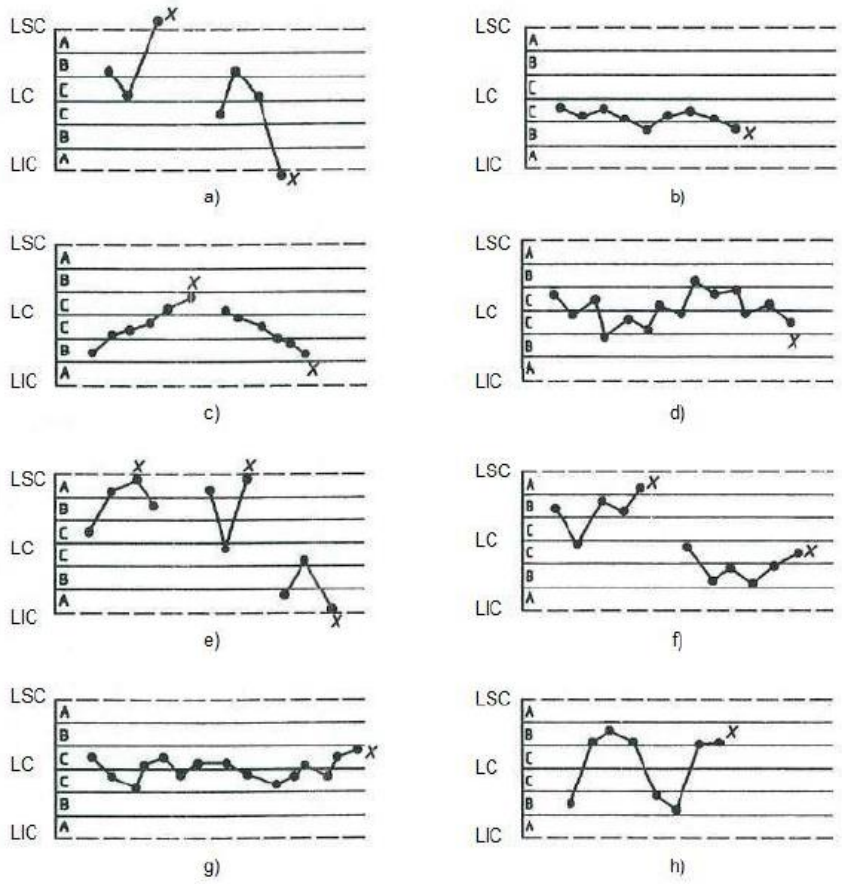
Figura 17 - Carta de controle com os limites superior (LSC), inferior (LIC) e central (LC) e linhas correspondentes aos desvios (σ).




Fonte: OLIVEIRA et al., 2013

A Figura 18 exemplifica a avaliação dos oito critérios estabelecidos na norma ISO 8258 para a interpretação das cartas de controle de Shewhart.

Figura 18 - Exemplos de processos fora de controle estatístico. Adaptado da norma ISO 8258.




Fonte: OLIVEIRA et al., 2013

# 5 RESULTADOS

Esta sessão apresenta os resultados obtidos em relação a ferramenta *web* de gestão de testes que recebe o nome de “Back-IO” e a análise da qualidade da produção da fábrica antes e após a implantação do processo baseado na filosofia da qualidade de Crosby.

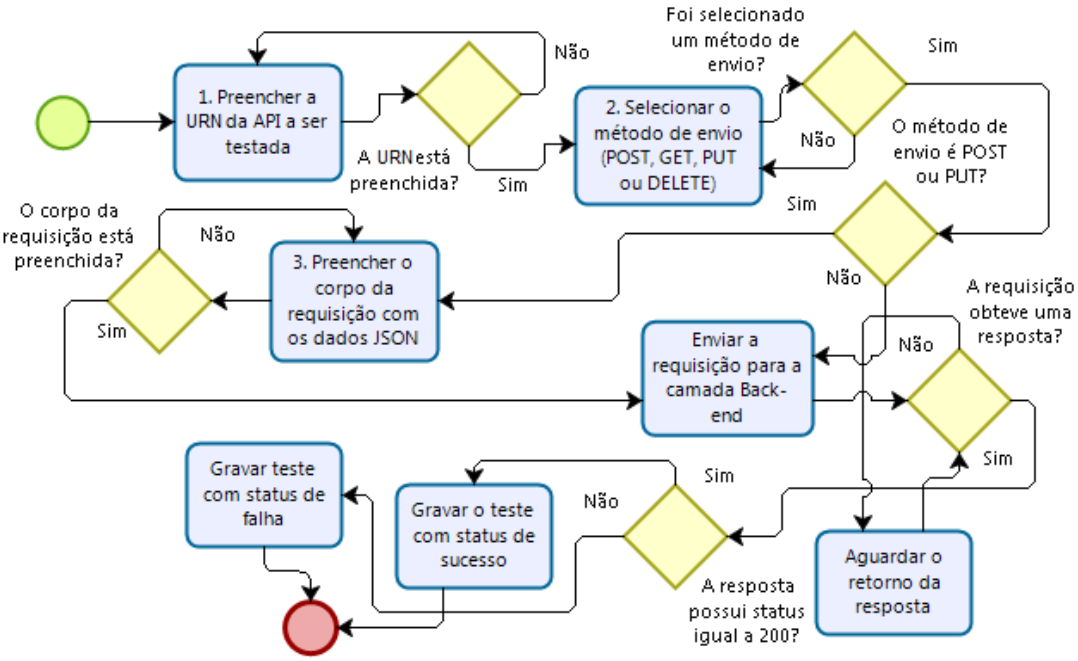
## 5.1 DIAGRAMA DE ENTIDADE E RELACIONAMENTO DA FERRAMENTA DE GESTÃO DE TESTES

O apêndice A demonstra as tabelas e colunas do banco de dados utilizados para o desenvolvimento da ferramenta, bem como os relacionamentos entre as tabelas com suas chaves primárias e chaves estrangeiras. As chaves são índices com valores únicos para identificação de cada um dos registros em banco de dados.

## 5.2 FLUXOGRAMA DA FERRAMENTA DE GESTÃO DE TESTES

A figura 19 demonstra o funcionamento para a coleta de dados relacionado às falhas identificadas nos testes através do consumo em APIs. Para a execução do fluxograma foi utilizado a linguagem JavaScript, que efetua uma chamada da requisição via AJAX, que é uma técnica utilizada na camada *Front-end* para criar requisições assíncronas, tornando possível o recebimento da resposta da requisição sem que seja necessário o carregamento da página *web*. O apêndice B demonstra o código JavaScritpt para atender ao fluxograma proposto.

Figura 19 - Fluxograma do funcionamento da ferramenta de gestão de testes.




Fonte: Autor, 2018

## 5.3 FERRAMENTA DE GESTÃO DE TESTES

O sistema consiste em testar os serviços produzidos na camada *Back-end* de um *software* ou um conjunto de *softwares* a partir de um teste de requisição em um ponto de acesso, que é um caminho específico que localiza um serviço a ser consumido e aguarda uma resposta, quando essa resposta possui uma falha, é então registrado em banco de dados a quantidade de falhas encontradas para análise da qualidade do *software* através da geração de Gráfico *u* para posterior análise sobre a melhoria da qualidade com a implantação do processo da filosofia do Zero Defeito de Crosby na produção de *software*. A ferramenta também contém formulário para cadastro da documentação relacionada ao escopo de cada uma das requisições com os dados de entrada da requisição com suas regras de negócio e os dados saída da resposta.

### 5.3.1 Tela de *login* do administrador

O sistema possui dois módulos, sendo um módulo administrativo, onde é possível o cadastro de usuários do sistema, além de cadastro e liberação de perfis de acesso,

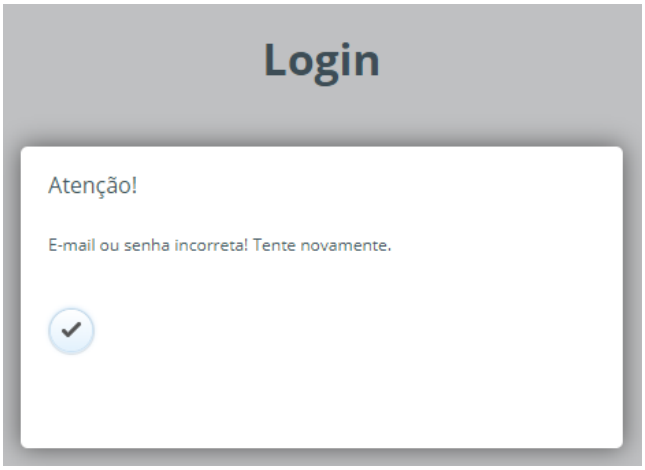
também é possível o gerenciamento de todas as APIs e de todos os testes, e outro módulo do usuário, que é o módulo de testes das APIs produzidas.

A tela de *login* (Figura 20) do módulo administrativo possui dois campos de entrada, sendo e-mail e senha. Ao informar um usuário válido, o sistema irá redirecionar para as telas administrativas, caso contrário, é mostrado uma mensagem de dados de *login* incorretos (Figura 21).

Figura 20 - Login administrativo.


Fonte: Autor, 2018

Figura 21 - *Login* com inconsistência de informações.

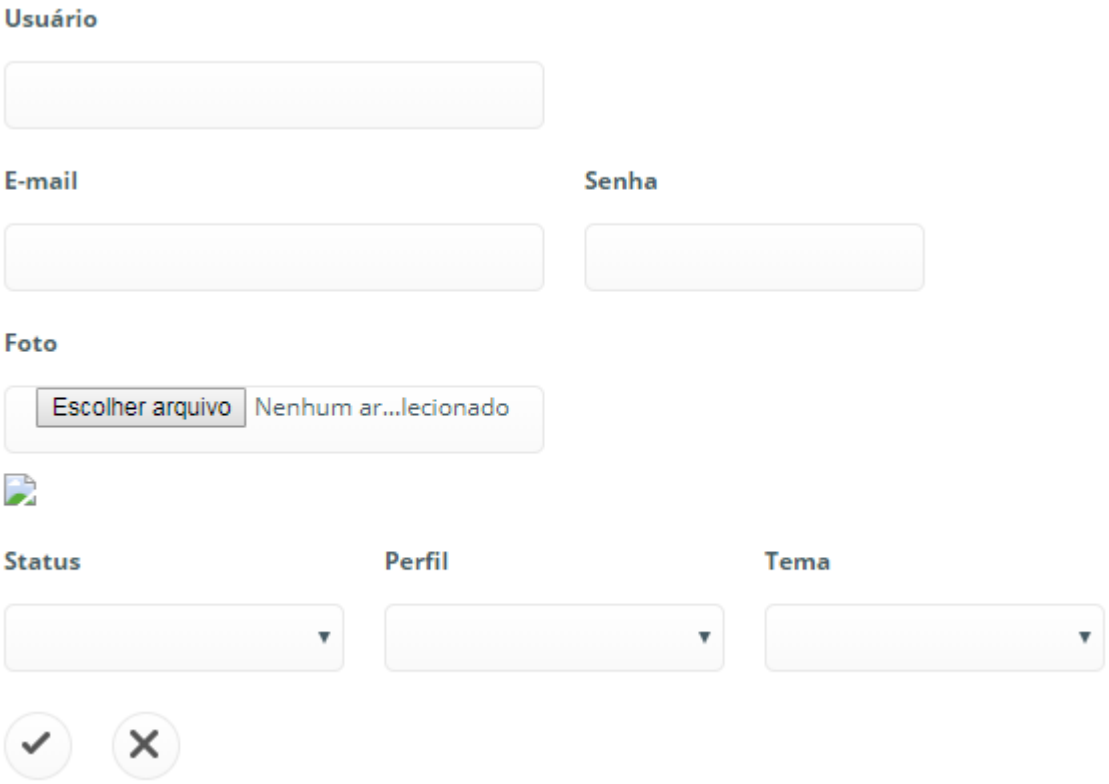



Fonte: Autor, 2018

### 5.3.2 Tela de cadastro de usuário

É possível gerenciar cadastros de todos os usuários do sistema através da tela de usuários com operações de criação (Figura 22), visualização, alteração e exclusão.

Figura 22 - Tela de usuários do sistema.

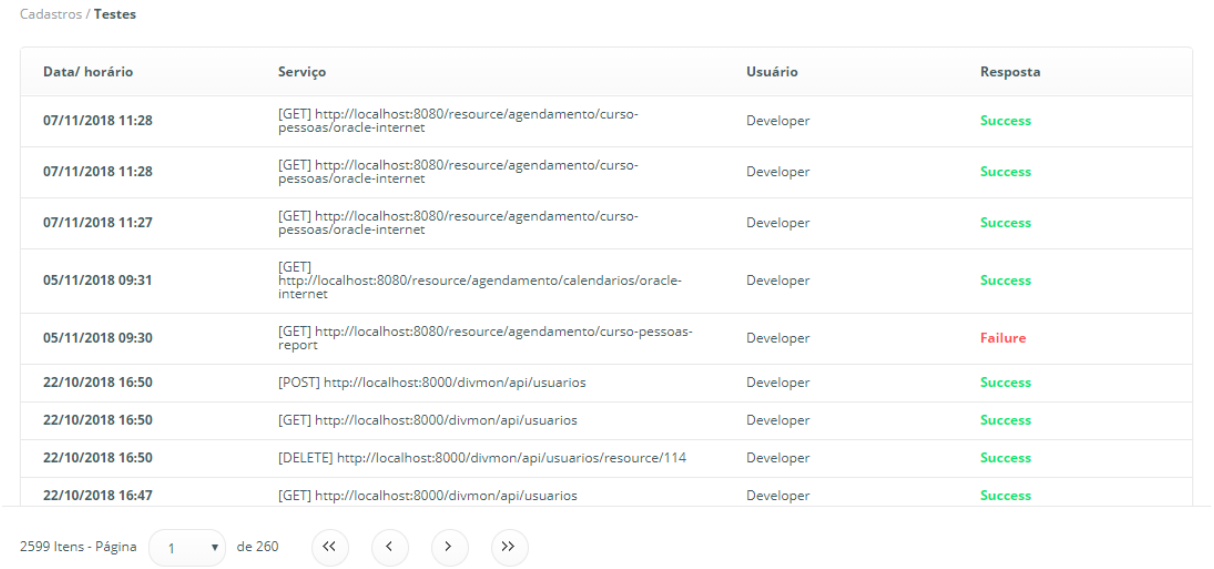



Fonte: Autor, 2018

### 5.3.3 Tela de visualização de falhas por serviço

Lista um histórico com todas as falhas encontradas por ponto de acesso aos serviços de APIs verificados (Figura 23). A tabela detalha informações sobre a data e horário do processamento do teste, bem como a API consumida, o usuário que testou e o valor da resposta sendo de sucesso ou falha.

Figura 23 - Tela de falhas por serviço.




Fonte: Autor, 2018

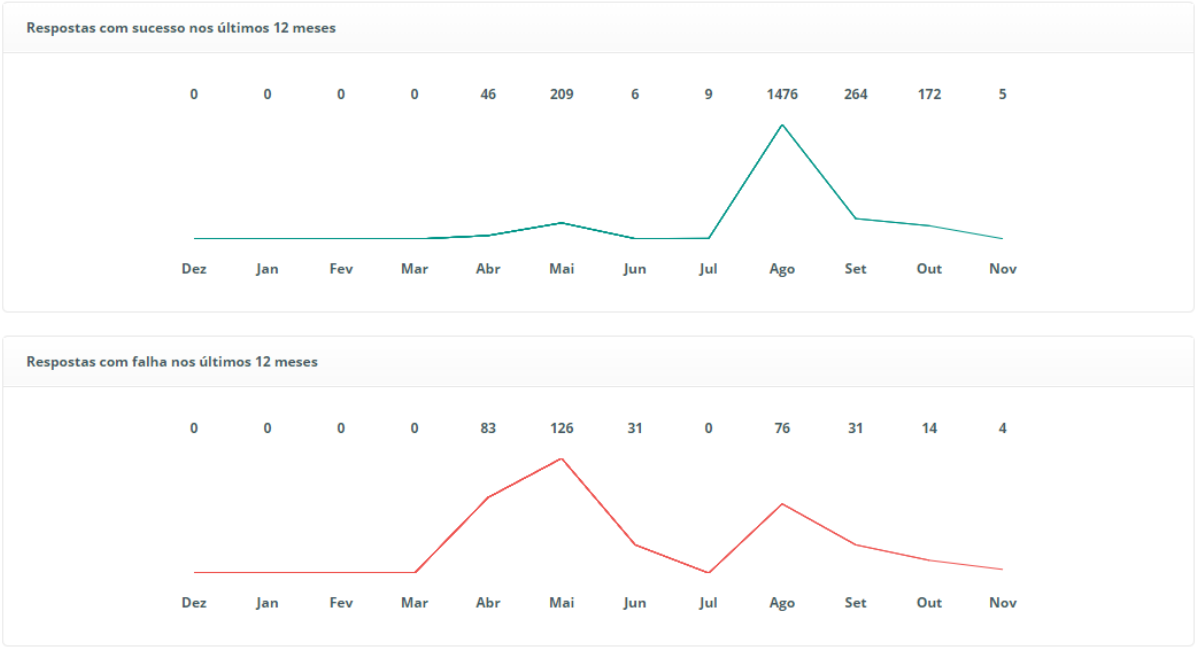
### 5.3.4 *Dashboard*

O *dashboard* é composto por dois componentes de gráficos, sendo um primeiro composto por um gráfico de pizza totalizando todo o histórico referente aos 12 últimos meses de testes com sucesso e com falha, demonstrando a proporção entre os dois resultados (Figura 24). E um segundo composto por gráfico categorizado pela quantidade de testes processados em cada mês nos 12 últimos meses, podendo assim ter uma ampla visão sobre a melhoria contínua da qualidade na produção da fábrica (Figura 25).

Figura 24 - *Dashboard* totalizador dos 12 últimos meses referentes a todos os testes.


Fonte: Autor, 2018

Figura 25 - *Dashboard* categorizado por meses.




Fonte: Autor, 2018

### 5.3.5 Tela de gerenciamento da documentação de APIs

É possível o cadastro e gerenciamento de todas as APIs com as suas respectivas documentações referentes aos pontos de acesso para consumo, o método de requisição, os valores de entrada e as suas regras de negócio, e os valores de saída da resposta conforme as figuras 26, 27 e 28.

Figura 26 - Gerenciamento dos serviços das APIs.


Fonte: Autor, 2018

Figura 27 - Gerenciamento dos valores de entrada e as suas regras de negócio.

Subpasta / **Entradas**

| Entrada | Tipo | Obrigatoriedade | Regras |
|---|---|---|---|
| **EmpresasInput** | Object | **Obrigatório** | Regras |
| **resource** | Integer | **Obrigatório** | Regras |

Fonte: Autor, 2018

Figura 28 - Gerenciamento dos valores de saída da resposta.

Subpasta / **Saídas**

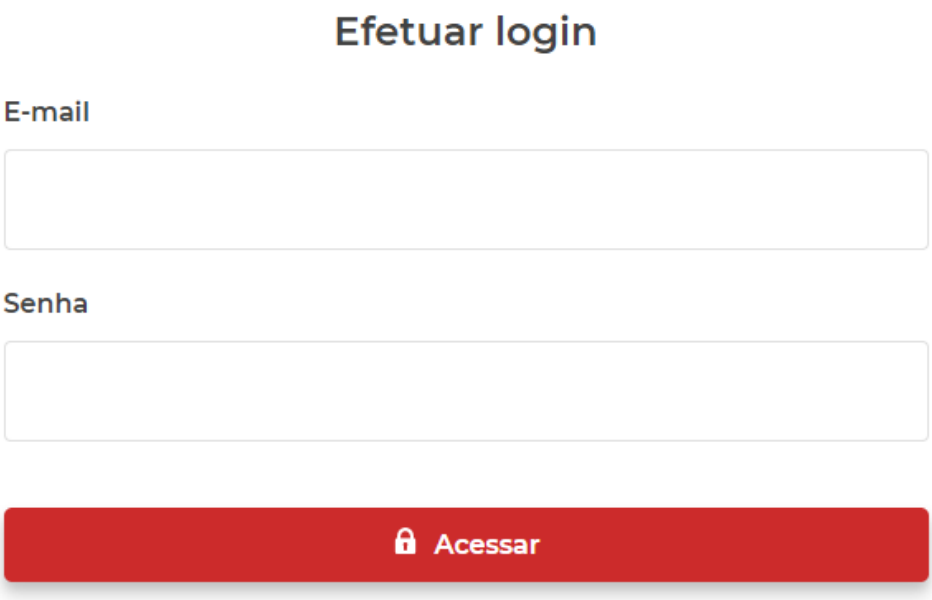
| Saída | Descrição | Tipo |
|---|---|---|
| **message** | A mensagem de erro. | String |
| **status** | O status do protocolo HTTP. | Integer |
| **size** | A quantidade de registros impactos. | Integer |
| **response** | O objeto de EmpresasOutput. | Object |

Fonte: Autor, 2018

### 5.3.6 Tela de *login* do usuário

Registra um acesso a um usuário que possui permissão para a execução de testes de serviços (Figura 29).

Figura 29 - Login do usuário.

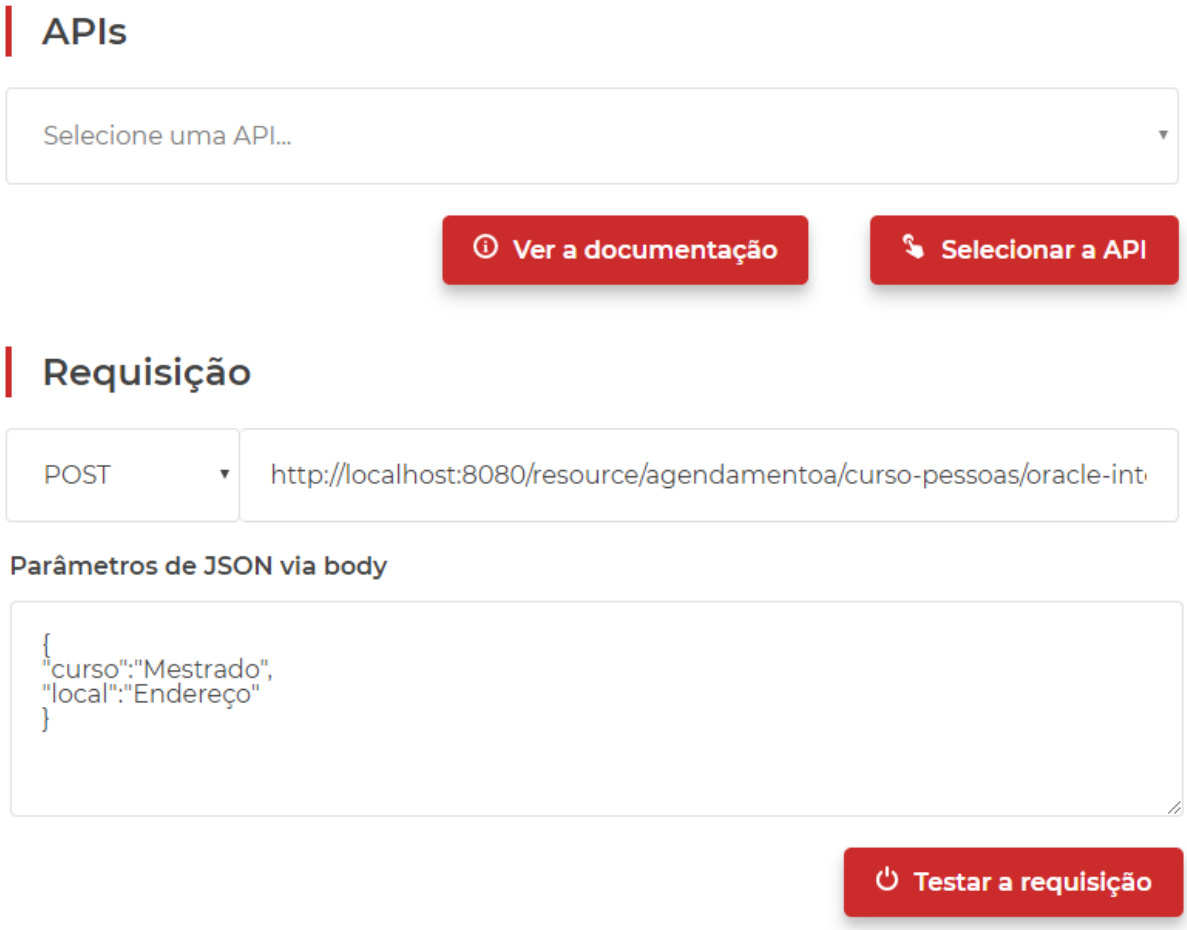



Fonte: Autor, 2018

### 5.3.7 Tela de teste dos serviços

É possível realizar uma requisição de teste com parâmetros de entrada (Figura 30), sejam eles via URI, que são os parâmetros passados pela URI, ou JSON (*JavaScript Object Notation*), que é um formato de dados simples amplamente utilizado para codificação de dados em requisições e respostas de serviços. Os dados JSON são compactos e fáceis de analisar na maioria das linguagens de programação (GRAHL et al., 2017).

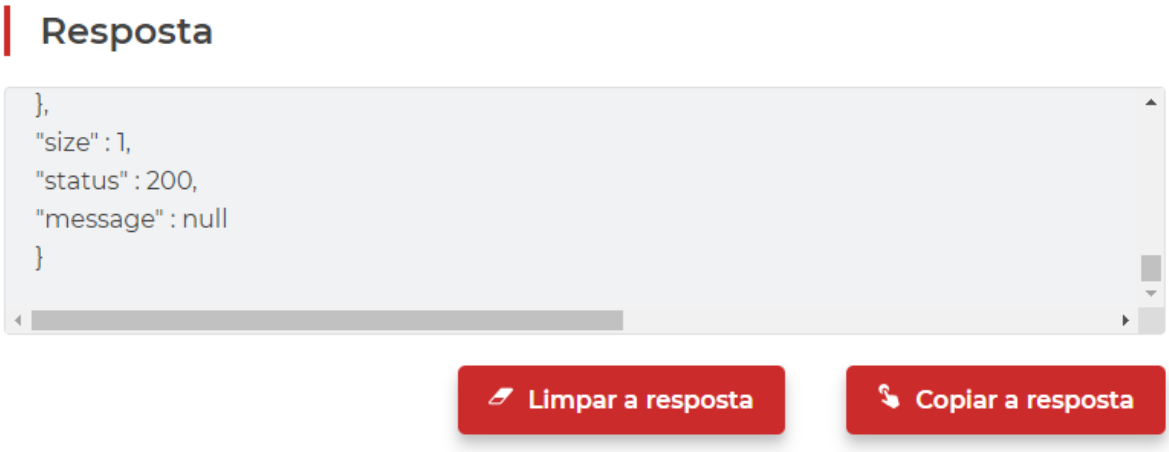
Figura 30 - Teste de requisição de um serviço.




Fonte: Autor, 2018

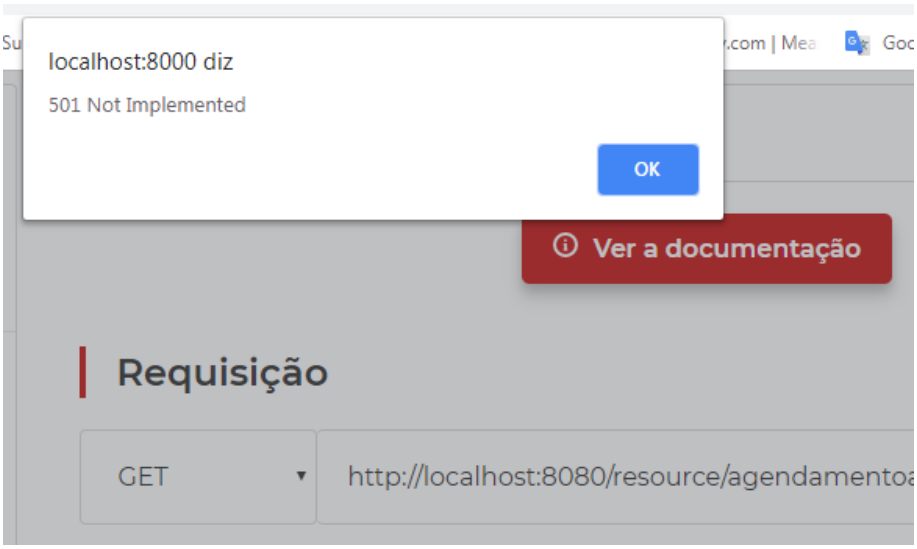
Ao processar a requisição é recebido uma resposta contendo os dados de retorno, quando essa requisição está válida, é listado um retorno com sucesso (Figura 31), caso contrário é mostrado uma mensagem de falha na execução (Figura 32).

Figura 31 - Requisição com sucesso de resposta.




Fonte: Autor, 2018

Figura 32 - Requisição com falha de resposta.

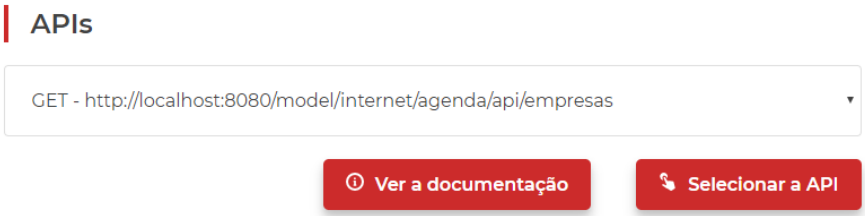



Fonte: Autor, 2018

### 5.3.8 Documentação das APIs

Após cadastro da documentação das APIs pelo sistema administrativo, é possível fazer a visualização selecionando uma API e clicando em Ver a documentação de acordo com a figura 33. O apêndice C demonstra uma documentação preenchida.

Figura 33 - Seleção de documentação de API.




Fonte: Autor, 2018

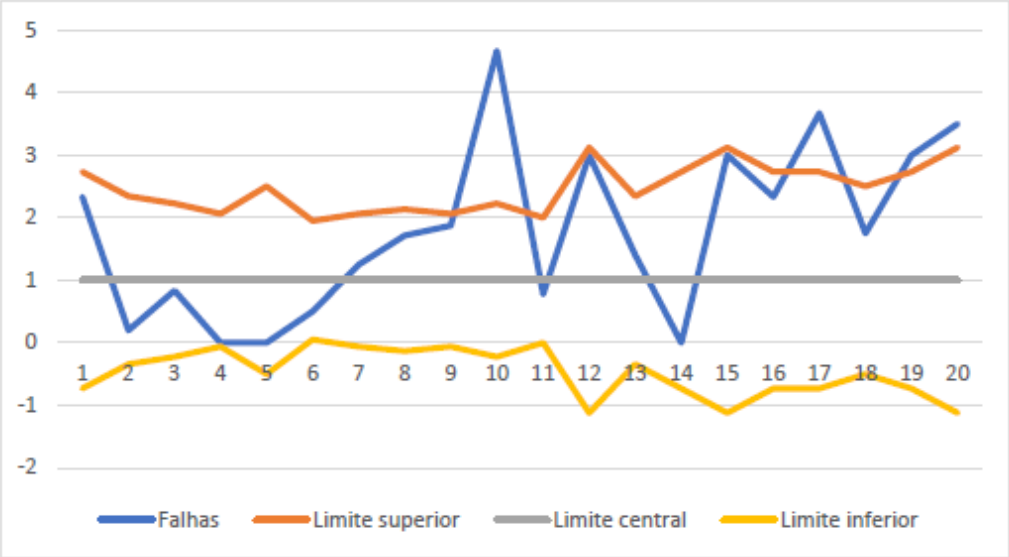
## 5.4 BASE DE DADOS RELACIONADOS AO NÚMERO DE FALHAS DE SOFTWARE ANTES DA IMPLANTAÇÃO DA FILOSOFIA DE CROSBY

O apêndice D demonstra a coleta de dados observada em 30 dias de utilização da ferramenta de testes em toda a produção da fábrica antes da implantação da filosofia de Crosby. Foram totalizadas 101 unidades de pontos de acesso inspecionadas e identificadas 150 falhas durante os testes.

### 5.4.1 Gráfico *u* relacionado à base de dados antes da implantação

A figura 34 demonstra o gráfico de controle do processo de produção de software de acordo com a amostragem referente ao apêndice D.

Figura 34 - Gráfico *u* de controle referente às falhas ocorridas antes da implantação do processo de produção de *software*, em 30 dias de testes.




Fonte: Autor, 2018

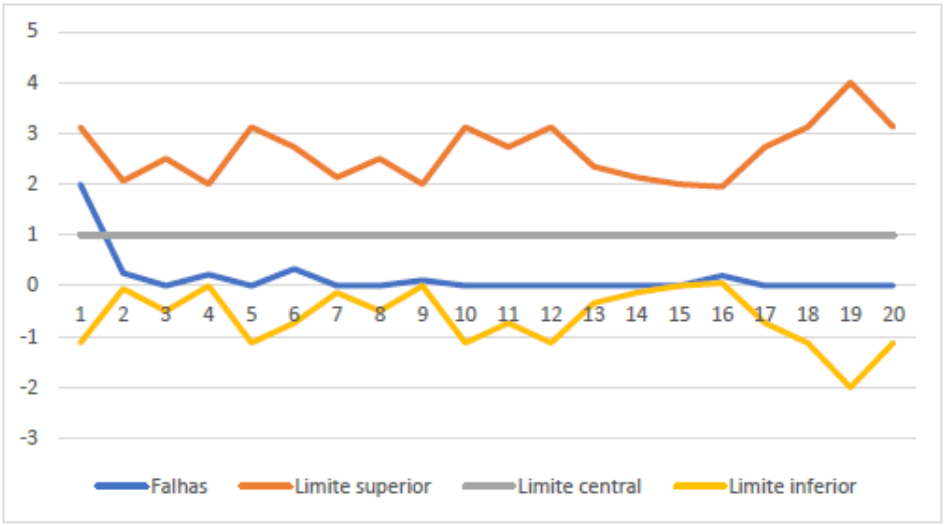
## 5.5 BASE DE DADOS RELACIONADOS AO NÚMERO DE FALHAS DE SOFTWARE DEPOIS DA IMPLANTAÇÃO DA FILOSOFIA DE CROSBY

O apêndice E demonstra a coleta de dados observada em 30 dias de utilização da ferramenta de testes em toda a produção da fábrica depois da implantação da filosofia de Crosby. Foram totalizadas 94 unidades de pontos de acesso inspecionadas e identificadas 12 falhas durante os testes.

### 5.5.1 Gráfico *u* relacionado à base de dados depois da implantação

A figura 35 demonstra o gráfico de controle do processo de produção de *software* de acordo com a amostragem referente ao apêndice E.

Figura 35 - Gráfico *u* de controle referente às falhas ocorridas após a implantação do processo de produção de *software*, em 30 dias de testes.




Fonte: Autor, 2018

## 5.6 IMPLANTAÇÃO DO SISTEMA

O sistema foi implantado em uma fábrica de *software* do setor público situada em interior do Estado de Minas Gerais com “nível de classificação inicial” segundo o modelo integrado de maturidade em capacitação. O sistema coletor de dados foi utilizado por um período de 30 dias, após esse período foi elaborado o Gráfico *u* refletindo o nível de controle da qualidade sobre o processo de produção da camada

*Back-end*. Após esse período, foi implantado o processo de produção da filosofia de Crosby e utilizado o sistema coletor por mais 30 dias para a elaboração de um novo Gráfico *u*.

### 5.6.1 Dificuldades na implantação

Foi encontrado dificuldade em disseminar a importância da qualidade devido à alta resistência à mudança, as equipes não entendem a importância da qualidade e a dívida técnica que é gerada com a falta de gestão da qualidade. A adaptação com o novo processo de produção impactou com algumas insatisfações pela mudança no ambiente de trabalho e com a criação de novas responsabilidades. Porém com a demonstração através de resultados foi possível estimular a busca pela cultura da qualidade, atendendo as necessidades dos requisitos com validações contínuas foi possível um grande avanço em relação a eliminação de retrabalhos.

# 6 CONCLUSÃO

Comprovou-se com a implantação do novo processo e o desenvolvimento de uma ferramenta *web* de gestão de testes, o auxílio no levantamento de dados relacionados ao número de falhas de *software* a fim de observar se o processo de produção está sob controle de acordo com o cálculo estatístico de uma das sete ferramentas da qualidade do Controle Estatístico de Processo através do Gráfico de Controle *u* ou Carta de Controle *u*. Com a análise dos dados obtidos, resultou-se na agilidade para a tomada de decisão, observando as não conformidades da produção de *software* visando pontos importantes para a melhoria contínua da qualidade.

O desenvolvimento e utilização da ferramenta *web* de gestão de testes da produção de *software* a partir de indicadores de qualidade como a quantidade de falhas, pôde contribuir para discussão com gestores da fábrica no sentido de solucionar problemas de qualidade a fim de reduzir custos, reduzir retrabalhos, reduzir frustrações de acordo com a expectativa versus a realidade das partes interessadas e melhorar o poder de tomada de decisão através da visualização de cenários em forma de gráficos de controle.

Mesmo com as dificuldades encontradas na implantação da ferramenta *web* e com a identificação da permanência de falhas, a proposta se mostra uma relevante estratégia para melhoria da qualidade, bem como para a implantação de um processo de gestão da qualidade da filosofia do Zero Defeito de Crosby em uma fábrica de software do setor público em interior do estado de Minas Gerais em “nível inicial” segundo o CMMI.

A ferramenta *web* de gestão de testes desenvolvida para a gestão em busca da melhoria da qualidade visa identificar falhas ainda em produção na camada *Back-end* o mais rápido possível de forma a prevenir posteriores problemas na entrega de funcionalidades do software ao cliente.

Os Gráficos de controle *u* gerados a partir dos dados coletados de acordo com a quantidade de falhas identificadas demonstraram uma melhoria em relação a qualidade da produção após a implantação do processo de produção da filosofia do Zero Defeito de Crosby, fazendo com que a produção fosse entregue de forma mais ágil ao cliente e com a eliminação de alguns retrabalhos com reparos de falhas, reduzindo custos e tempo com a produção. Após os 60 dias de observação foi

verificado uma redução de mais de 90% na quantidade de falhas identificadas em toda a produção da fábrica de software.

Após a implantação da filosofia de Crosby, foi possível um avanço segundo o CMMI, atingindo o nível de otimização na organização, com processos padronizados e documentados, além de métricas definidas e gerenciadas para a melhoria contínua da qualidade.

Como sugestão para um trabalho futuro seria possível desenvolver um *software* capaz de categorizar falhas e vincular uma solução para o reparo da falha, e dessa forma alimentar uma base de dados com possíveis soluções caso uma falha aconteça, além disso, também seria possível automatizar testes que já foram realizados de maneira manual, podendo definir cenários de desempenho como por exemplo, medir quantos testes determinada API conseguiria se manter com determinado desempenho, simulando cenários para a agilidade na tomada de decisão.

## REFERÊNCIAS BIBLIOGRÁFICAS

ABES, Associação Brasileira das Empresas de Software. **Mercado Brasileiro de Software: panorama e tendências, 2016.** 1ª ed. São Paulo: ABES – Associação Brasileira das Empresas de Software, 2016, cap. 7, p. 100-101. Disponível em: <http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados%202011/ABES-Publicacao-Mercado-2016.pdf>. Acesso em 13 jan 2018.

ABHIJIT, A. H.; JOSHIB, A. R. A. **Novel Approach to HTML Page Creation Using Neural Network**. Procedia Computer Science, 2015, v. 45, p. 197-204. Doi: 10.1016/j.procs.2015.03.122

ALAMI, A. **Why Do Information Technology Projects Fail?** Procedia Computer Science, 2016, v. 100, p. 62-71. Doi: 10.1016/j.procs.2016.09.124

ASQ, American Society for Quality. **Crosby´s 14 Steps to Improvement**. 2005, p. 62-64. Disponível em <http://www.agiledevelopment.org/download/qp1205crosby.pdf>. Acesso em 13 jan 2018.

BRAUN, S.; ELBERZHAGER, F.; HOLL, K. **Automation Support for Mobile App Quality Assurance – A Tool Landscap**. Procedia Computer Science, 2017, v. 110, p. 117-124. Doi: 10.1016/j.procs.2017.06.129

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML GUIA DO USUÁRIO**. 9ª tiragem. Rio de Janeiro: Editora Campus Ltda, 2000, cap. 1, p. 3-5.

BRIGGS, A.; BURKE, P. **Uma história social da mídia: de Gutenberg à internet**. 2ª edição. Rio de Janeiro: Zahar, 2006, cap. 1, p. 300-302

CIORANUA, C.; CIOCAB, M.; NOVAC, C. **Database Versioning 2.0, a Transparent SQL Approach Used in Quantitative Management and Decision Making**. Procedia Computer Science, 2015, v. 55, p 523-528. Doi: 10.1016/j.procs.2015.07.030

COSTA, A. F. B. **Gráficos de controle de Shewhart: Duas décadas de pesquisa. 2007**. 163 f. Tese (livre docência) - Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2007. Disponível em: <http://hdl.handle.net/11449/116107>.

FIELDING, R. T.; RESCHKE, J. F. **Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**. Internet Engineering Task Force, 2014, v. 7231, p 47-64. Doi: 10.17487/RFC7231

GEJDOS, P. **Continuous Quality Improvement by Statistical Process Control**. Procedia Economics and Finance, 2015, v. 34, p. 565-572. Doi: 10.1016/S2212-5671(15)01669-X

GRAHL, M. et al. Archive WEB API: **A web service for the experiment data archive of Wendelstein 7-X**. Fusion Engineering and Design, 2017, v. 123, p. 1015-1019. Doi: 10.1016/j.fusengdes.2017.02.047

HAMDANA, S.; ALRAMOUNIB, S. **A Quality Framework for Software Continuous Integration**. Procedia Manufacturing, 2015, v. 3, p. 2019-2025. Doi: 10.1016/j.promfg.2015.07.249

HIETSCHOLD, N.; REINHARDT, R.; GURTNER, S. **Measuring critical success factors of TQM implementation successfully–a systematic literature review**. International Journal of Production Research, 2014, v. 52, n. 21, p. 6254-6272. Doi: 10.1080/00207543.2014.918288

HUUMOA, J. Y.; MAGLYAS, A.; SMOLANDER, K. **How do software development teams manage technical debt? – An empirical study**. The Journal of Systems and Software, 2016, v. 120, p. 195-218. Doi: 10.1016/j.jss.2016.05.018

IEEE. **IEEE Standard Glossary of Software Engineering Terminology**. IEEE Std 610.12-1990, 1990, p. 31-32. Doi: 10.1109/IEEESTD.1990.101064

INNOTAS. **The Project and Portfolio Management Landscape**. 2016. p. 7 Disponível em: <https://www.innotas.com/wp-content/uploads/2016/05/The-Project-and-Portfolio-Management-Landscape-Report_2016_final.pdf>. Acesso em 13 jan 2018.

LANDICHO, J. A. **A web-based geographical project monitoring and information system for the road and highways**. Journal of Electrical Systems and Information Technology, 2018, v. 5, p. 252-261. Doi: 10.1016/j.jesit.2016.10.011

MARANDI, A. K.; KHAN, D. A. **An Impact of Linear Regression Models for Improving the Software Quality with Estimated Cost**. Procedia Computer Science, 2015, v. 54, p. 335-342. Doi: 10.1016/j.procs.2015.06.039

MAYER, P. C. **Redução da Variabilidade em uma linha de produção de chapas de corpo de silos de grãos de corrugação 4'' através da implantação do controle estatístico do processo**. Tese de mestrado. Programa de Pós-Graduação em Engenharia. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2004. Disponível em: < http://hdl.handle.net/10183/6109>.

MENDOZA, D. K. O. **Antitrust in the new economy case Google Inc. against economic competition on web**. Mexican Law Review, 2016, v. 8, p. 1-29. Doi: 10.1016/j.mexlaw.2016.07.001

MEYERS, M. A. **A new and important step: Inclusion in ScienceDirect**. Journal of Materials Research and Technology, 2013, v. 2, p. 205. Doi: 10.1016/j.jmrt.2013.07.001

MICHEL, R.; FOGLIATTO, F. S. **Projeto econômico de cartas adaptativas para monitoramento de processos**. Revista Gestão da Produção, v9, n.1, p 17-31, 2002.

MONTGOMERY, D. C. **Introduction to Statistical Quality Control**. 6ª edição. Arizona State University: Editora WILEY, 2009, cap. 7, p. 314-316.

OLIVEIRA, C. C. et al. **Manual para elaboração de cartas de controle para monitoramento de processos de medição quantitativos em laboratórios de ensaio**. 1ª edição. São Paulo: Instituto Adolfo Lutz, 2013, p.18-19.

PANE, E. S.; SARNO, R. **Capability Maturity Model Integration (CMMI) for Optimizing Object-Oriented Analysis and Design (OOAD)**. Procedia Computer Science, 2015, v. 72, p. 40-48. Doi: 10.1016/j.procs.2015.12.103

PHILIP CROSBY ASSOCIATES. **Biography**. 2001. Disponível em: <http://www.philipcrosby.com/25years/crosby.html>. Acesso em 13 jan 2018.

PMI. **A Guide to the Project Management Body of Knowledge**. 5ª edição. Pennsylvania: 14 Campus Boulevard.

POP, D. P.; ALTAR, A. **Designing an MVC Model for Rapid Web Application Development**. Procedia Engineering, 2014, v. 69, p. 1172-1179. Doi: 10.1016/j.proeng.2014.03.106

PROKOFYEVA, N.; BOLTUNOVA, V. **Analysis and Practical Application of PHP Frameworks in Development of Web Information Systems**. Procedia Computer Science, 2017, v. 104, p. 51-56. Doi: 10.1016/j.procs.2017.01.059

SAPUTRA, D. G.; AZIZAH, F. N. **A Metadata Approach for Building Web Application User Interface**. Procedia Technology, 2013, v. 11, p. 903-911. Doi: 10.1016/j.protcy.2013.12.274

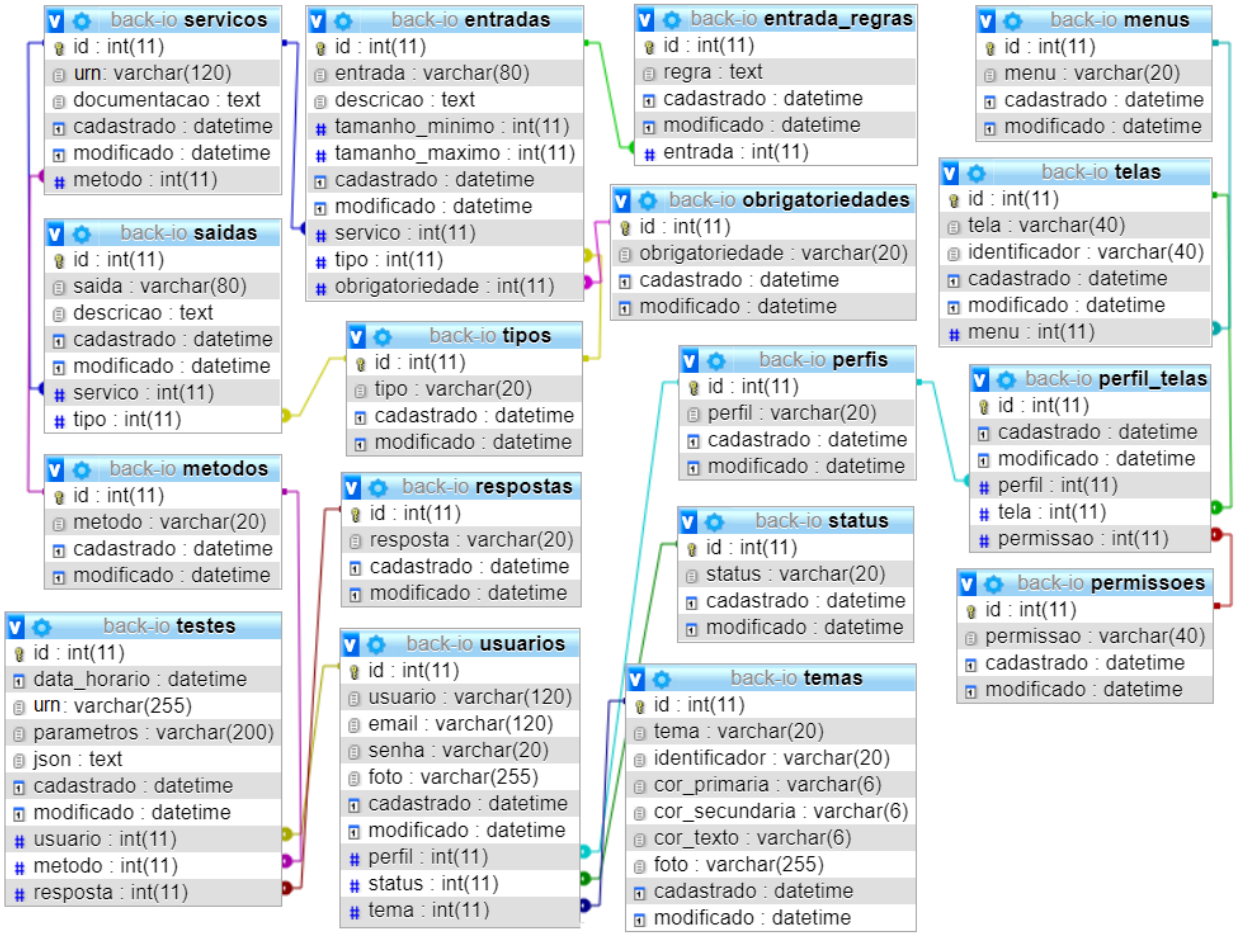
SHARMA, A.; KUMAR, M.; AGARWAL, S. **A Complete Survey on Software Architectural Styles and Patterns**. Procedia Computer Science, 2015, v. 70, p. 16-28. Doi: https://doi.org/10.1016/j.procs.2015.10.019

TEICH, T. et al. **Concept for a Service-oriented Architecture in Building Automation Systems**. Procedia Engineering, 2014, v. 69, p. 597-602. Doi: 10.1016/j.proeng.2014.03.031

VIEIRA, S. **Estatística para a qualidade**. 2ª edição. Rio de Janeiro: Elsevier, 2012, p. 5-6, p.135-136.

YUAN, T. et al. **Formalization and Verification of REST on HTTP Using CSP**. Electronic Notes in Theoretical Computer Science, 2014, v. 309, p. 75-93. Doi: 10.1016/j.entcs.2014.12.007

## APÊNDICE A - Diagrama de entidade e relacionamento da ferramenta de gestão de testes.



## APÊNDICE B - Demonstrativo do código JavaScript de envio da requisição e recebimento da resposta.

```javascript
/**
 * Back-IO.
 *
 * @author Mário Sakamoto <mskamot@gmail.com>
 * @license MIT http://www.opensource.org/licenses/MIT
 * @see http://mariosakamoto.com/getz
 * @since 26 Jan 2018
 * @version 1.0.0
 */

/**
 * Envia os dados para o teste da requisição.
 */
function test() {
    var method =
    document.getElementById("method").options[gI("method").selectedIndex].text;
    var urn  = document.getElementById("urn").value;
    var json = document.getElementById("json").value;
    if (urn == "") {
        alert("Atenção! É necessário informar uma URN para continuar!");
    } else if (method == "POST" || method == "PUT") {
        if (json == "") {
            alert("Atenção! É necessário informar o JSON para continuar!");
        } else {
            request(method, urn , json, "testResponse");
        }
    } else {
        request(method, urn , json, "testResponse");
    }
}

/**
 * Efetua a requisição para um webservice.
 *
 * @param {String} method O método
 * @param {String} urn A URN
 * @param {String} json O corpo JSON da requisição
 * @param {String} callBack A função a ser chamada após o retorno da resposta
 */
function request(method, urn , json, callBack) {
    var data = "";
    var request = initRequest();
    request.open(method, urn , true);
    request.setRequestHeader("Accept-Language", "pt-BR");
    request.setRequestHeader("Content-Type", "application/json; charset=UTF-8");
    request.onreadystatechange = function() {
        if (request.readyState == 4 && request.status == 200) {
            eval(callBack + "('" + request.responseText + "')");
        } else if (request.readyState == 4 && request.status != 200) {
            eval(callBack + "('" + request.status + "')");
        }
    };
    if (method == "POST" || method == "PUT") {
        data = "request=" + JSON.stringify(json));
    }
    request.send(data);
}

/**
 * Inicializa a configuração da requisição.
 */
function initRequest() {
    if (window.XMLHttpRequest) {
        if (navigator.userAgent.indexOf("MSIE") != -1) {
            isIE = true;
        }
        return new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
```

```
69          try {
70              return new ActiveXObject("Microsoft.XMLHTTP");
71          } catch(e) {
72              try {
73                  return new ActiveXObject("Msxml2.XMLHTTP");
74              } catch(e) { }
75          }
76      }
77  }
78
79  /**
80   * Valida o retorno da resposta da requisição.
81   */
82  function testResponse(response) {
83      if (response == "500") {
84          alert("500 Internal Server Error");
85      } else if (response == "501") {
86          alert("501 Not Implemented");
87      } else if (response == "502") {
88          alert("502 Bad Gateway");
89      } else if (response == "503") {
90          alert("503 Service Unavailable");
91      } else if (response == "504") {
92          alert("504 Gateway Timeout");
93      } else if (response == "505") {
94          alert("505 HTTP Version Not Supported");
95      } else {
96          var normalized_enters = response.replace(/\r|\n/g, "\r\n");
97          var text_with_br = normalized_enters.replace(/\r\n/g,
            "<br>");
98          var json = JSON.parse(text_with_br);
99          var str = JSON.stringify(json);
100         document.getElementById("response").innerHTML = "<p>" +
101                 str.replace(/,\"/g, ",<br>\"").replace(/{/g, "{<br>").
102                 replace(/}/g, "<br>}").replace(/":/g, "\" : ") + "</p>";
103     }
104 }
```

**APÊNDICE C - Demonstrativo da visualização da documentação de acesso a uma API.**

## 1 Model/internet/agenda/api/empresas

**Método**: GET

**URN**: http://localhost:8080/model/internet/agenda/api/empresas

**Descrição**: Busca registros de empresas.

## 2 Dados de entrada

| Campo | Descrição | Min | Max | Required |
|---|---|---|---|---|
| filterColumn | Coluna para o filtro do WHERE. | 0 | 100 | Não obrigatório |

**Regra**: Enviar friendly para filtrar pela coluna amigável da tabela.

| Campo | Descrição | Min | Max | Required |
|---|---|---|---|---|
| filterValue | Valor para o filtro do WHERE. | 0 | 100 | Não obrigatório |

**Regra**: Quando filterColumn for igual à friendly, enviar o valor do filtro com lowercase e espaçado por hífen.

| Campo | Descrição | Min | Max | Required |
|---|---|---|---|---|
| orderColumn | Coluna para a ordenação do ORDER BY. | 0 | 100 | Não obrigatório |

| Campo | Descrição | Min | Max | Required |
|---|---|---|---|---|
| orderValue | Valor para a ordenação do ORDER. | 0 | 100 | Não obrigatório |

| Campo | Descrição | Min | Max | Required |
|---|---|---|---|---|
| page | Página da leitura. | 1 | 10 | Obrigatório |

| Campo | Descrição | Min | Max | Required |
|---|---|---|---|---|
| pageSize | Quantidade de registros por página. | 1 | 10 | Obrigatório |

## 3 Dados de saída

| Campo | Descrição | Tipo |
|---|---|---|
| message | A mensagem de erro. | String |

| Campo | Descrição | Tipo |
|---|---|---|
| response.empresasOutputList | A lista de objetos de EmpresasOutput. | List |

| Campo | Descrição | Tipo |
|---|---|---|
| size | A quantidade de registros encontrados. | Integer |

| Campo | Descrição | Tipo |
|---|---|---|
| status | O status do protocolo HTTP. | Integer |

**APÊNDICE D - Demonstrativo das falhas ocorridas antes da implantação do processo de Crosby em 30 dias de testes.**

| Amostra | Unidades | Falhas identificadas |
|---|---|---|
| 1 | 3 | 7 |
| 2 | 5 | 1 |
| 3 | 6 | 5 |
| 4 | 8 | 0 |
| 5 | 4 | 0 |
| 6 | 10 | 5 |
| 7 | 8 | 10 |
| 8 | 7 | 12 |
| 9 | 8 | 15 |
| 10 | 6 | 28 |
| 11 | 9 | 7 |
| 12 | 2 | 6 |
| 13 | 5 | 7 |
| 14 | 3 | 0 |
| 15 | 2 | 6 |
| 16 | 3 | 7 |
| 17 | 3 | 11 |
| 18 | 4 | 7 |
| 19 | 3 | 9 |
| 20 | 2 | 7 |

**APÊNDICE E - Demonstrativo das falhas ocorridas depois da implantação do processo de Crosby em 30 dias de testes.**

| Amostra | Unidades | Falhas identificadas |
|---|---|---|
| 1 | 2 | 4 |
| 2 | 8 | 2 |
| 3 | 4 | 0 |
| 4 | 9 | 2 |
| 5 | 2 | 0 |
| 6 | 3 | 1 |
| 7 | 7 | 0 |
| 8 | 4 | 0 |
| 9 | 9 | 1 |
| 10 | 2 | 0 |
| 11 | 3 | 0 |
| 12 | 2 | 0 |
| 13 | 5 | 0 |
| 14 | 7 | 0 |
| 15 | 9 | 0 |
| 16 | 10 | 2 |
| 17 | 3 | 0 |
| 18 | 2 | 0 |
| 19 | 1 | 0 |
| 20 | 2 | 0 |

# APÊNDICE F - Política da qualidade.

**POLÍTICA DA QUALIDADE**

As empresas estão mudando o seu jeito de trabalhar através do trabalho compartilhado e colaborativo, os chamados “ambientes produtivos”. Dessa forma, é de extrema importância envolver todos os colaboradores, alinhar as ideias e delimitar os objetivos de forma engajadora e organizada.

Crosby inspira o ambiente produtivo de forma colaborativa, onde todos são importantes para a execução produtiva de resultados com a mais alta qualidade. É importante pensar nos colaboradores como clientes internos, se eles não acreditam na qualidade do que fazem, eles irão transparecer essa verdade aos clientes externos.

As pessoas são a parte mais importante de uma empresa, e prezar pela qualidade é saber ouvir e entender suas necessidades, tanto as partes interessadas que colaboram para o desenvolvimento dos objetivos quanto as partes interessadas que requisitam o desenvolvimento. Propor soluções criativas, produtivas e com qualidade é uma obrigatoriedade para atender as expectativas do cliente.

Através de levantamento e análise dos requisitos, a engenharia de software é uma aliada para a execução de projetos de software, mas, às vezes, o cliente ainda não sabe o que realmente deseja e se frustra ao receber um produto de qualidade inferior ao que se imaginava. Dessa forma, é preciso estar ao lado do cliente, definir requisitos e coletar feedbacks através de aprovações é imprescindível. Além disso, o desenvolvimento de software é dinâmico e mutável, funcionalidades são repensadas e novos requisitos são inseridos a todo momento, é necessário a maturidade para atender as mudanças, inovar todos os dias e fornecer soluções que realmente gerem valor ao cliente.

O avanço na qualidade é preciso, a gestão da qualidade de software deve ser maior e mais observada.