UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO
ENGENHARIA DE COMPUTAÇÃO

LUCIANO DOS REIS DE SOUSA

Uso da integração contínua e testes automatizados para melhoria na qualidade de software: proposta para uma equipe de desenvolvimento e manutenção de softwares bancários

SÃO PAULO - SP
2019

LUCIANO DOS REIS DE SOUSA

Uso da integração contínua e testes automatizados para melhoria na qualidade de software: proposta para uma equipe de desenvolvimento e manutenção de softwares bancários

Trabalho de Conclusão de Curso apresentado para a Universidade Virtual do Estado de São Paulo como parte dos requisitos para a graduação em Engenharia de Computação.

Orientadora: Msc. Sandra Souza Rodrigues
Coorientadora: Barbara Milan Martins

SÃO PAULO - SP
2019

FOLHA DE APROVAÇÃO

[Ficha Catalográfica]

DEDICATÓRIA

Dedico este trabalho à Deus, minha família, meus amigos e meus professores. É impossível esquecer de pessoas que no campo de batalha lutaram junto a mim, tornando-se meus irmãos.

AGRADECIMENTOS

Agradeço primeiramente a Deus que criou todas as condições e me permitiu estar aqui e aos meus pais, que sempre acreditaram e lutaram pelo bem-estar de seu filho. Minha gratidão a todos os profissionais da UNIVESP por sua competência, dedicação, paciência e disponibilidade em orientar que resultaram em encorajamento e frases de valorização a esse trabalho.

“Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrependе.”

Leonardo da Vinci

RESUMO

A qualidade de software, além de influenciar na percepção dos usuários quanto ao produto, pode significar redução de custos de manutenção e prevenção de prejuízos financeiros. Na Centralizadora de Desenvolvimento de TI da CEF – CAIXA Econômica Federal, onde os serviços de tecnologia da informação são realizados por funcionários públicos em conjunto com terceirizados, é importante o domínio sobre os processos e qualidade dos seus sistemas, para auxiliar tomada de decisões e manutenção do nível de qualidade esperado. Conforme norma interna TE107 da CAIXA todo código entregue deve ser submetido à testes e avaliação de qualidade buscando a melhoria contínua. Caso o código entregue não obtenha a qualidade mínima e passe em 80% dos testes unitários a fábrica terceirizada poderá ser penalizada mediante contrato e a CAIXA ter sua imagem prejudicada junto aos clientes devido à falha em seus sistemas. Este trabalho teve por objetivo propor uma melhoria no processo de desenvolvimento de software dos sistemas de canais da empresa - MTC, por meio da implementação de protótipo em ambiente pré-testes funcionais usando a Integração Contínua, testes automatizados, e conscientização dos stakeholders. A proposta foi elaborada a partir de um estudo de caso junto à equipe de desenvolvimento da empresa, que utiliza a abordagem ágil. Para coleta de dados foram realizadas entrevistas do tipo estruturas qualitativas no próprio ambiente de trabalho com os colaboradores com o objetivo de avaliar a percepção, dores, conhecimentos dos entrevistados e os problemas relacionados à qualidade das entregas realizadas. Este trabalho tem sua fundamentação teórica ancorada na área de Integração Contínua, Processos de Software, Qualidade de Software e Testes Automatizados. Como resultado constatou-se que uso o de plugins atualizados da ferramenta de software Jenkins abre uma janela de oportunidades de pesquisa de ferramentas integradas para a melhoria contínua. Constatou-se ganho significativo em agilidade com a prototipação realizada em ambiente apartado com o Raspberry Pi, tendo em vista que não há a necessidade de alterações no ambiente de desenvolvimento da empresa, disponibilização de servidores e solicitações de liberação de regra de firewall. Como constatação final foi avaliada a ferramenta SonarQube tendo a mesma impedido compilar um pacote de código defeituoso, validando seus benefícios.

Palavras-chave: Integração Contínua, Jenkins, testes automatizados, SonarQube.

ABSTRACT

The quality of software, in addition to influencing users' perception of the product, can mean reducing maintenance costs and preventing financial losses. At the IT Development Center of CEF - CAIXA Econômica Federal, where information technology services are performed by public employees in conjunction with outsourced companies, it is important to master the processes and quality of their systems, to assist in decision making and maintenance of the expected quality level. According to CAIXA internal standard TE107, all delivered code must be submitted to quality tests and evaluation for continuous improvement. If the delivered code does not obtain the minimum quality and passes in 80% of the unit tests, the outsourced factory may be penalized under a contract and CAIXA will have its image impaired with the customers due to the failure of their systems. This work aimed to propose an improvement in the software development process of the company's channel systems - MTC, through the implementation of a prototype in a functional pre-test environment using Continuous Integration, automated testing, and stakeholder awareness. The proposal was drawn up from a case study with the company's development team, which uses the agile approach. In order to collect data, qualitative structure interviews were carried out in the working environment with the employees in order to evaluate the perception, pains, knowledge of the interviewees and the problems related to the quality of the deliveries. This work has its theoretical foundation anchored in the area of Continuous Integration, Software Processes, Software Quality and Automated Testing. As a result it has been found that using the upgraded plugins of the Jenkins software tool opens a window of search opportunities for integrated tools for continuous improvement. It was observed a significant gain in agility with the prototyping performed in a remote environment with Raspberry Pi, considering that there is no need for changes in the company's development environment, provision of servers and requests for the release of firewall rule. As a final verification, the SonarQube tool was evaluated, which prevented the compilation of a defective code package, validating its benefits.

Keywords: Continuous Integration, Jenkins, automated testing, SonarQube**.**

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Qualidade é um tema que faz parte do dia a dia de todos nós e desempenha um papel importante nas organizações em geral.
Na literatura computacional, qualidade é definida como uma das áreas de conhecimento que tem como objetivo garantir a conformidade do software com as expectativas dos clientes, através da definição e normatização dos processos de desenvolvimento.

De acordo com MOLINARI:

> "Se você olhar na literatura computacional, verá dois significados gerais de qualidade: O primeiro é que a qualidade é algo de estado binário. A qualidade do produto existe ou não existe. No segundo, qualidade define-se como um produto ou serviço que faz o que o usuário precisa. Na prática diz-se também pronto para usar. (MOLINARI, 2015, p. 20)"

Devido à alta competitividade as empresas de tecnologia se viram num cenário de busca de resultados melhores, com menor custo. Isto têm significado mudança de processos, tecnologia e uma mudança estrutural nas organizações dando um papel essencial para a tecnologia e inovação.

Segundo ANICHE:

> "É natural que, em um projeto que muitos desenvolvedores do time trabalhem em paralelo no desenvolvimento de funcionalidades e correções. Muitas vezes, podem acabar por alterar os mesmos arquivos gerando conflitos, ou até mesmo alterar arquivos diferentes, que depois de combinadas alterações, o software apresente um comportamento inadequado. De qualquer forma, de tempos em tempos, as alterações precisarão ser integradas em uma base comum de código (ANICHE, 2017)."

No caso estudado, uma equipe de desenvolvedores que estão trabalhando em incrementos de funcionalidades no projeto MTC, utilizam como servidor de versionamento do código-fonte, como o Apache Subversion (SVN).

Neste cenário é comum, por engano, distração, ou até mesmo falta de treinamento que o programador envie um código que não compila para o servidor de

versionamento ou até mesmo que se baixe código fonte de uma pasta desatualizada do mesmo servidor. Os outros desenvolvedores, quando forem obter o código mais recente, podem acabar obtendo o código quebrado, que havia sido enviado por engano ou até mesmo a versão base errada para desenvolver uma nova funcionalidade. Isto impacta no fluxo de desenvolvimento da equipe, nas entregas em tempos negociados com o dono do produto e gera erros advindos da gerência de configuração que podem causar prejuízo financeiro ou de imagem quando o erro não é detectado nos testes e homologação e vai à produção.

Um estudo hipotético foi apresentado por Black (2010) para proporcionar um melhor entendimento sobre o retorno de investimento proporcionados pelos testes.

Figura 1: Opções para Investimento em testes

Opções para Investimento em Testes (Adaptado de Rex Black,2010)				
TESTES		**Sem Processo Formal**	**Processo Manual**	**Teste Automatizado**
Equipe		$ 0	$ 60.000	$ 60.000
Infraestrutura		$ 0	$10.000	$10.000
Ferramentas e Automatização		$ 0	$ 0	$12.500
Total		$ 0	$ 70.000	$ 82.500
Desenvolvimento				
Defeitos Encontrados		250	250	250
Custo da Correção		2500	2500	2500
Testes				
Defeitos Encontrados		0	350	500
Custo da Correção		0	35.000	50.000
Suporte ao usuário				
Defeitos Encontrados		750	400	250
Custo da Correção		750.000	400.000	250.000
Custo da Qualidade				
Investimento		$ 0	$ 70.000	$ 82.500
Custo da Correção		$ 752.500	$ 437.500	$ 302.500
Total		$ 752.500	$ 507.500	$ 385.000
Retorno do Investimento		**N/A**	**350%**	**445%**

Fonte: Black, 2010

Segundo ele "Um software implantando em um cliente possui uma nova versão liberada a cada 3 meses. Em média, cada nova versão possui 1000 novos defeitos. A equipe desenvolvedora desse software encontra em média 250 erros antes de liberar

a versão, sendo que o restante (750) e encontrado pelo cliente. Tendo como base a estimativa anterior do próprio Black (2010), que um erro encontrado por um desenvolvedor custa em média $10 e pelo cliente custa $ 1000, esse cenário, que não possui um processo formal de teste, gera um custo de $750.000 por versão, e uma insatisfação imensa no cliente. Para tentar melhorar essa situação, o gerente do projeto consegue investir $ 70.000 por versão em testes manuais para minimizar os impactos. Sendo assim, a equipe dedicada para testes identifica mais 350 defeitos que são corrigidos antes de lançar a versão. Partindo da estimativa que cada defeito encontrado por um testador custa em média $100, o retorno do investimento e 350%, conforme apresentado na tabela a seguir. Percebendo o retorno do investimento, o gerente do projeto consegue pleitear um investimento de mais $12.500, por versão, para investir em automatização dos testes, encontrando 150 defeitos a mais. Dessa forma, o retorno de investimento obtido seria de 445%."

Outro exemplo bastante comum é quando há a necessidade de extrair a última versão de projeto para construir e disponibilizar à produção. Quando não há um servidor que executa os testes, o desenvolvedor responsável por colocar o código em produção só vai descobrir se o mesmo possui um bug ou não quando executar os testes em sua própria máquina. Pode assim acontecer que o dono do produto que fica aguardando receber um software funcional, mas na véspera do lançamento, quando deveria ser enviado para produção que os bugs são descobertos. Tanto o dono do produto quanto o desenvolvedor responsável por esta tarefa terão suas expectativas frustradas pela falta de visibilidade do processo de entrega de software.

Um importante aspecto na avaliação destes cenários é que, o código defeituoso, seja por conta de erro de algum erro de compilação ou pelos testes não passarem foi parar no servidor de versionamento. A descoberta dos erros é feita de forma tardia e o um código problemático aparece bem depois do que quando foi escrito.

1.1 PROBLEMA DE PESQUISA

“Como melhorar o processo de desenvolvimento com a utilização Integração Contínua e testes automatizados na equipe MTC da empresa Caixa Econômica Federal?”

1.2 OBJETIVO GERAL E ESPECÍFICOS

1.2.1 Objetivo Geral:

Este estudo teve como objetivo propor melhorias para o desenvolvimento de software utilizando os princípios da integração contínua aplicada ao contexto da equipe de desenvolvimento e manutenção do software bancário MTC, da empresa Caixa Econômica Federal.

1.2.2 Objetivos específicos:

Os objetivos específicos deste TCC foram:

- Avaliar dos benefícios da Integração contínua e Testes Automatizados e aplicar ao processo de desenvolvimento de software da equipe de desenvolvimento do sistema MTC.

- Prototipação com baixo custo utilizando um Raspberry PI como um protótipo de servidor de integração contínua.

1.3 JUSTIFICATIVA

As equipes de desenvolvimento de software têm aumentado para acompanhar o mercado e a qualidade do produto deve acompanhar esse crescimento.

Dentre outras falhas, em abril de 2019 foi enviado erroneamente à produção um pacote do software MTC que não havia sido alterado com base na versão mais atual de produção.

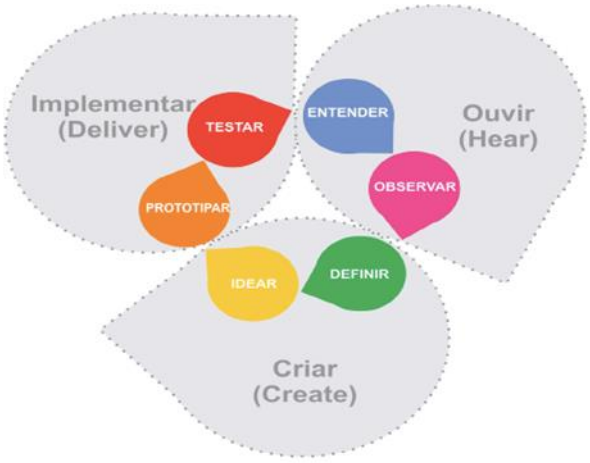
Houve o acompanhamento durante a madrugada da equipe de operações e desenvolvimento e em torno das duas horas da manhã a implantação terminou com registro de sucesso pelos envolvidos. Não obstante, às 7 horas da manhã do mesmo dia o dono do produto foi até uma agência e não conseguiu utilizar uma importante funcionalidade que necessitava de validação biométrica e a partir deste momento cerca de 30 pessoas entre técnicos, coordenadores e demais envolvidos da matriz da empresa foram acionados. O incidente somente foi resolvido às 10h da manhã.

Uma outra justificativa é que o código entregue deve ter uma qualidade mínima, passando acima de 80% dos testes unitários, podendo ocorrer penalização mediante contrato celebrado entre a fábrica terceirizada e a Caixa Econômica Federal.

1.4 DELIMITAÇÃO DO ESTUDO

O trabalho proposto segue a ótica do processo de Design Thinking que é organizada em uma estratégia cíclica que permite a evolução gradativa da solução como um todo, conforme sugere a figura 2, proposta por Cavalcanti (2015).

Figura 2: Ilustração do processo de Design Thinking



Fonte: Cavalcanti, 2015

1.5 ESTRUTURA DO TRABALHO

Há um direcionamento claro para a prática do protótipo que materializa em aprendizado efetivo e real para própria evolução do projeto.

O desafio é estabelecer uma estratégia centrada no código com integração entre os processos. Assim, problema tratado neste trabalho foi:

“Como melhorar o processo de desenvolvimento com a utilização Integração Contínua e testes automatizados na equipe MTC da empresa Caixa Econômica Federal?”

Este estudo se caracteriza importante quando alguma pequena falha de software tem um alto impacto como por exemplo uma mudança em ambiente produtivo faça transparecer que o seu saldo está inconsistente, fazendo com que correntistas relatem à imprensa o “sumiço” de dinheiro em suas contas após ocorrer um atraso de processamento de algumas transferências em conta via TED.

2 REFERENCIAL TEÓRICO

O gerenciamento de configuração de software pode ser definido como as diferentes formas de juntar partes do software para montar o pacote final e é fundamental gerenciar os sistemas em evolução para não perder o controle de quais mudanças e versões de componentes foram incorporadas em cada versão do sistema (SOMMERVILLE, 2011)

Martin Fowler (2015) definiu a integração contínua e seus respectivos benefícios:

> "Integração Contínua é uma prática de desenvolvimento de software onde os membros de uma equipe integram seu trabalho frequente, geralmente pelo menos uma vez por dia. Cada integração é verificada por um build automatizado (incluindo testes) para detectar erros e devolver um feedback o mais rápido possível. Muitas equipes viram que essa abordagem reduz significativamente problemas com integração e permite que a equipe desenvolva software mais coeso de forma mais rápida." (FOWLER, 2015)

.

O termo IC (Integração Contínua) passa a ideia de algo junto(integrado) e feito sem parar (continuamente). Começamos criando código, depois commitamos (guardamos) tudo em um repositório (em sua maioria SVN ou Git), testamos a aplicação, depois publicamos o sistema gerado desse código em algum lugar e, finalmente, passamos para o usuário usar. Depois, temos alterações no sistema, que podem ser melhorias ou correções de bugs, onde voltamos a criar código. E assim estamos de volta à primeira etapa do ciclo de desenvolvimento. (BOAGLIO, 2016)

Paul Durvall (2018) que definiu Integração Contínua como sendo:

> "Um processo realizado em intervalos regulares, que resulta em um executável que incorpora o aumento funcional do sistema", além disso ele sugere que este processo seria uma espécie de "marco de projeto onde a gerência poderia realizar medições para controlar melhor o projeto e atacar os riscos em uma base contínua" (PAUL, 2018).

Com o advento das Metodologias Ágeis diversas técnicas e práticas que permitem dinamizar o desenvolvimento de software ganharam destaque. O conceito de Integração Contínua foi aprimorado (PAUL, 2015), passando a ser recomendado

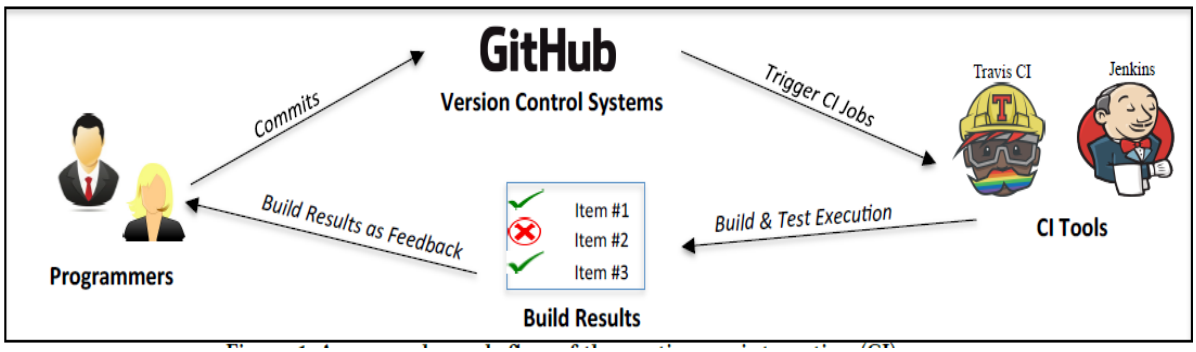
não apenas em intervalos regulares, mas sempre que possível, por viabilizar um ambiente coeso e propício para redução significativa de problemas de integração, além de atenuar uma das características mais críticas de software, a invisibilidade (FOWLER, 2015).

A Integração Contínua é um passo inicial para uma outra prática fortemente encorajada, a “Entrega Contínua” em inglês *Continuous Delivery*, que se trata de uma disciplina de desenvolvimento de software onde o sistema é construído de tal forma que pode ser liberado para produção em qualquer momento (FOWLER, 2015). Os principais benefícios de entregar continuamente são reduzir risco de deploy, melhoria na confiança do progresso do projeto (aumento de visibilidade) e adiantamento do feedback do usuário (antecipação de mudanças) (FOWLER, 2015).

Atualmente diversas ferramentas promovem integração contínua identificando mudanças no repositório do projeto, verificando o código e executando um conjunto de procedimentos para verificar se a mudança é boa e não irá prejudicar alguma parte do projeto. Algumas ferramentas bastante difundidas para integração contínua são “Cruise Control”, “Continuum”, “Team City”, “Hudson” e “Jenkins”. Entre elas o Jenkins conseguiu maior alcance na comunidade open-source, consequentemente tem vantagem na identificação e correção de bugs, implementações de melhorias, bem como no desenvolvimento de plugins compatíveis (SMART, 2011). Portanto o Jenkins foi escolhido para a realização do trabalho.

A ferramenta Jenkins é originária do mesmo projeto da ferramenta Hudson, cuja teve sua primeira versão disponibilizada em 2005. Alguns impasses relacionados aos direitos e decisões de projeto envolvendo a empresa Oracle culminaram na “separação” dos projetos e adoção do nome Jenkins em 2011 (JENKINS, 2011). Jenkins é um software livre e open-source para integração contínua, desenvolvido em Java. Além de monitorar, integrar (através de compilação e testes) e fornecer feedback sobre os projetos, ele permite que sejam customizados diversos procedimentos adicionais através de plugins como por exemplo o “SonarQube plugin”, que permite a execução remota de inspeções de qualidade de código utilizando o Sonar (JENKINS, 2015).

Figura 3: Um exemplo de fluxo de trabalho do processo de integração contínua (CI)



FONTE: SIQUEIRA, 2018

Teste automatizado é a prática de tornar os testes de software independentes da intervenção humana, criando scripts ou programas simples de computador que exercitam o sistema em teste, capturam os efeitos colaterais e fazem verificações, tudo automática e dinamicamente (MESZAROS; WESLEY, 2007). Os testes automatizados afetam diretamente a qualidade dos sistemas de software, portanto, agregam valor ao produto final, mesmo que os artefatos adicionais produzidos não sejam visíveis para os usuários finais dos sistemas (BERNARDO, 2011).

3 PROCEDIMENTOS METODOLÓGICOS

A identificação do problema objeto da pesquisa deve ser validada conforme a metodologia do Design Thinking junto à local definindo uma vez que a proposta de solução deve atender aos desejos reais dos envolvidos no projeto e em suas manutenções.

A inovação em Design ocorre através do entendimento e antecipação das necessidades dos usuários, criando produtos ou serviços que atendam aos seus desejos, criando, em contrapartida, vantagens para os negócios. Isso ocorre quando existe a confluência entre os aspectos Tecnológicos (praticabilidade), de Valores Humanos (usabilidade e desejabilidade) e de Negócios (viabilidade) - sendo este último o campo mais negligenciado nos assuntos envolvendo Design.

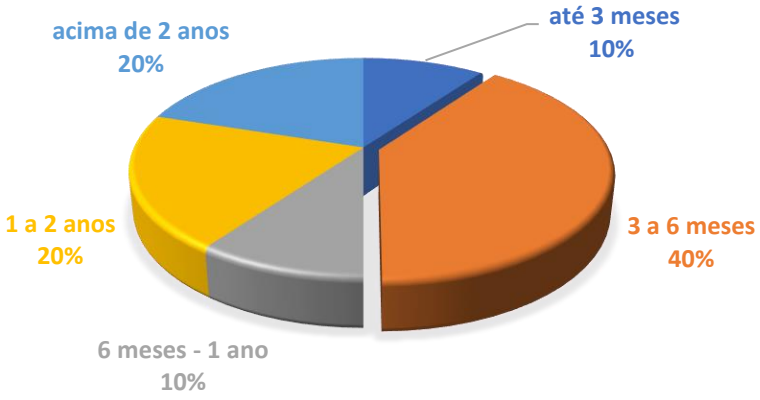
Entre 19 e 22/04/2019 na sala de reuniões 1.1 do primeiro andar do edifício foram distribuídos formulários de pesquisa individuais, contendo questões voltadas à equipe que realiza quinzenalmente uma retrospectiva dos trabalhos feitos e metas, pontos fortes e pontos de melhoria.

Para coleta de dados foram realizadas 9 entrevistas do tipo estruturas qualitativas no próprio ambiente de trabalho com o objetivo de avaliar a percepção, dores e conhecimentos dos entrevistados e os problemas relacionados à qualidade das entregas realizadas.

O roteiro de perguntas foi inserido em uma planilha eletrônica para tabulação de dados resultando nos gráficos 1 a 5 apresentados abaixo:

1. Quanto tempo você trabalha no projeto MTC?

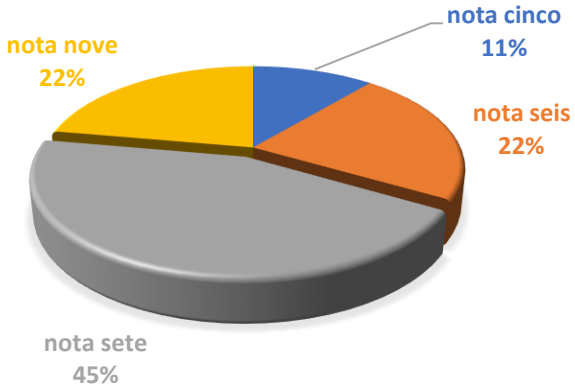
Gráfico 1



Fonte: Próprio autor

2. Qual a nota que você atribui para a qualidade do software que entregamos ao cliente?

Gráfico 2



Fonte: Próprio autor

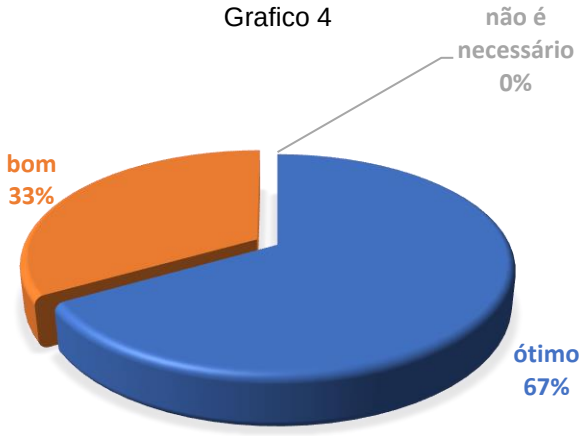
3. Possui conhecimento de integração contínua e testes automatizados?

Gráfico 3

Fonte: Próprio autor

4. O que você acha de ser informado periodicamente (email) sobre alterações realizadas no repositório central?

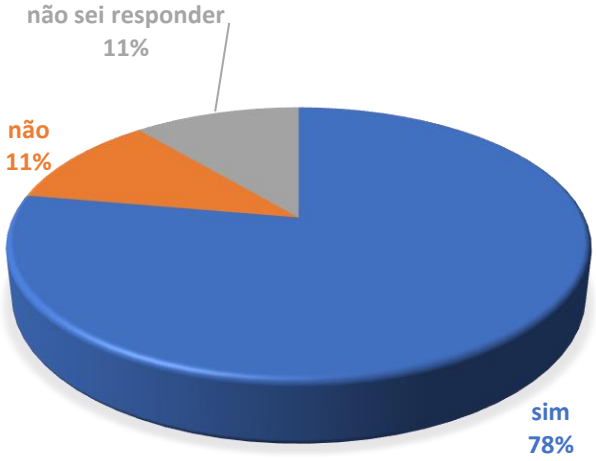
Grafico 4



Fonte: Próprio autor

5. O que você acha de ser informado periodicamente (e-mail) sobre alterações realizadas no repositório central?

Gráfico 5



Fonte: Próprio autor

4 RESULTADOS E DISCUSSÕES

Em busca de baixo custo e uso de softwares livres foram utilizadas as ferramentas:

- Cartão SD classe 10.
- Raspberry Pi 3 B com Sistema Operacional Linux. A distribuição é Debian tendo em vista é o que mais se aproxima da estrutura atual para construção de servidor de integração de modo a não afetar a estrutura existente.

  Configuração do hardware Raspberry Pi 3 B:

  - ✓ 1 GB DE RAM;
  - ✓ SAÍDA HDMI E RJ43;
  - ✓ 4 ENTRADAS USB; MICRO SLOT SD;
  - ✓ 40 PINOS PARA CIRC ELETRÔNICO;
  - ✓ ENTRADAS PARA VÍDEO, ÁUDIO, CAM.

Figura 4: Visão superior do Hardware Raspberry Pi B e seus componentes

Fonte: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

- Softwares:

  a) Controle de versão: Apache Subversion e GIT;

b) Servidor de automação: Jenkins;

c) Cobertura código**:** EclEmma, SonarLinte JUnit integrados ao Eclipse;

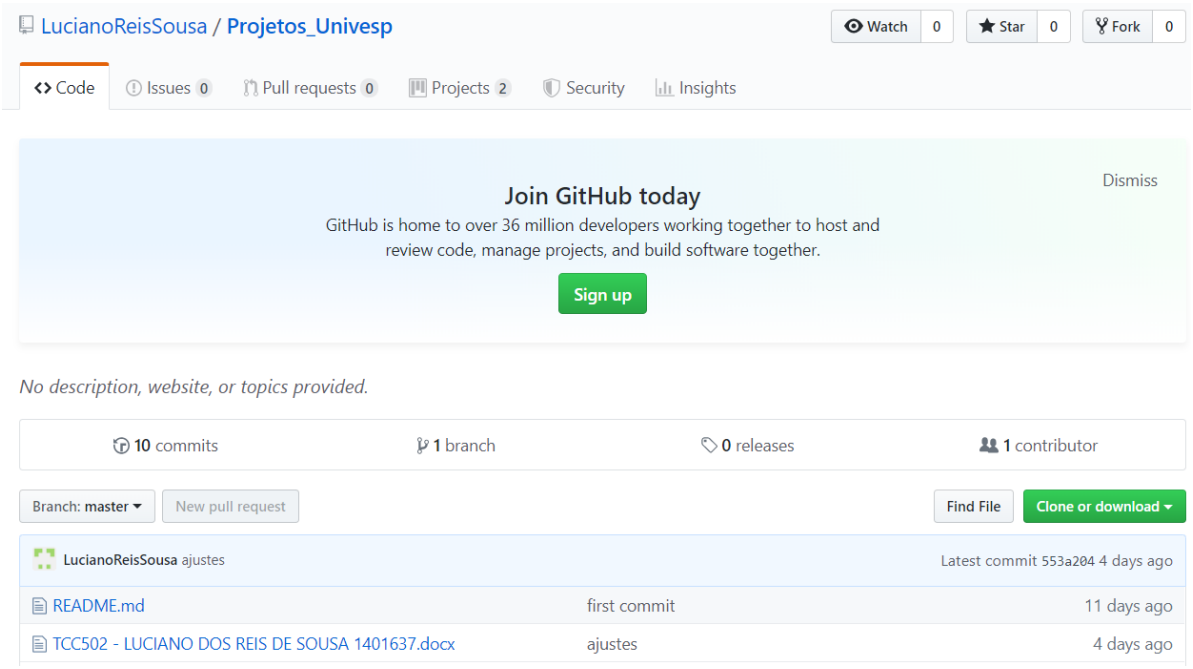
d) Testes: SonarQube integrado ao Maven;

e) Conexão remota ao servidor: WINSCP.

Foi criado um novo Projeto no Github para simular o armazenamento e gerenciamento dos artefatos durante a construção desta proposta.

Endereço: https://github.com/LucianoReisSousa/Projetos_Univesp

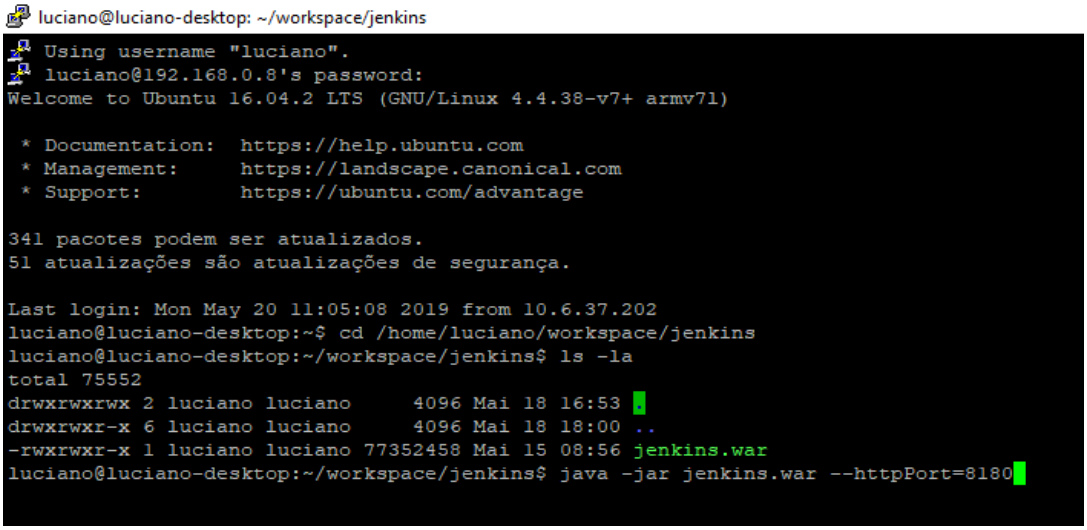
Figura 5: Visão do Repositório do Projeto criado no GITHUB



Fonte: próprio autor

Foi realizada a Instalação do Jenkins utilizando a porta 8180 para evitar conflito com outras aplicações.

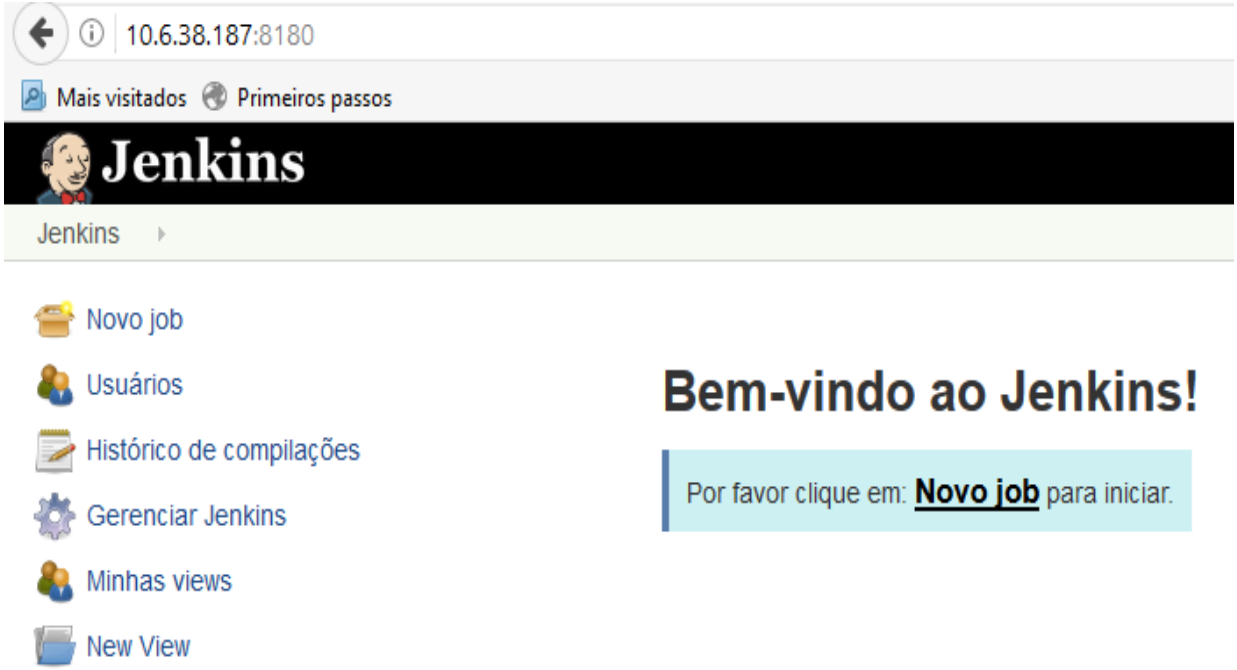
Figura 6: Instalação do Jenkins via terminal do Linux



Fonte: próprio autor

Abaixo o Jenkins disponível na rede corporativa após a instalação:

Figura 7: Jenkins disponível após instalação



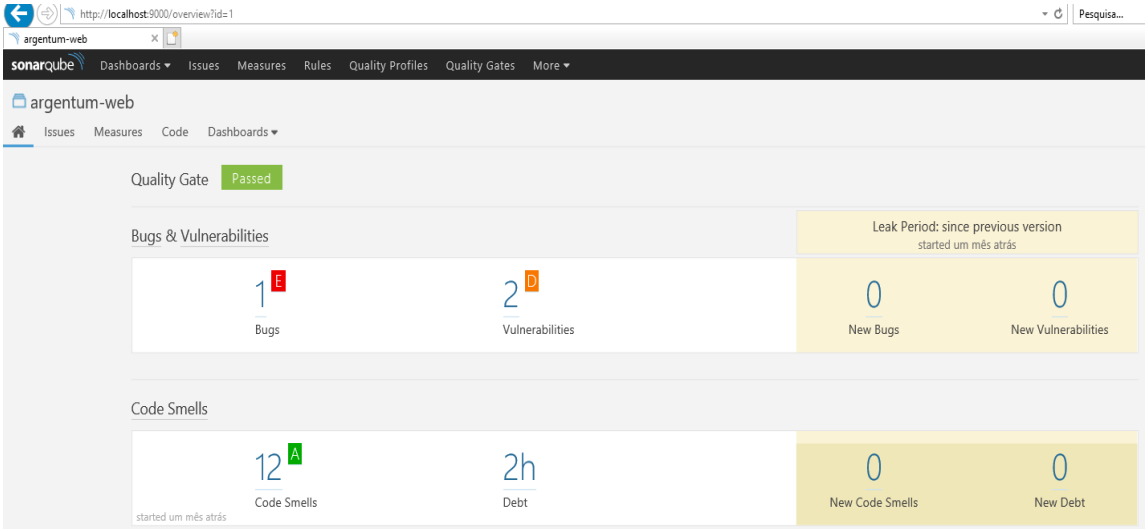
Após a instalação do MAVEN 3 e configurações foi realizado o download do jacoco-maven-plugin para cobertura dos testes.

Figura 8 – Configuração do Maven exibida via terminal

Fonte: Próprio autor

Relatório atual de inspeção contínua gerado pelo Sonar após realizados os testes unitários e testes automatizados:

Figura 9: Modelo de relatório de cobertura de testes gerado pelo SonarQube

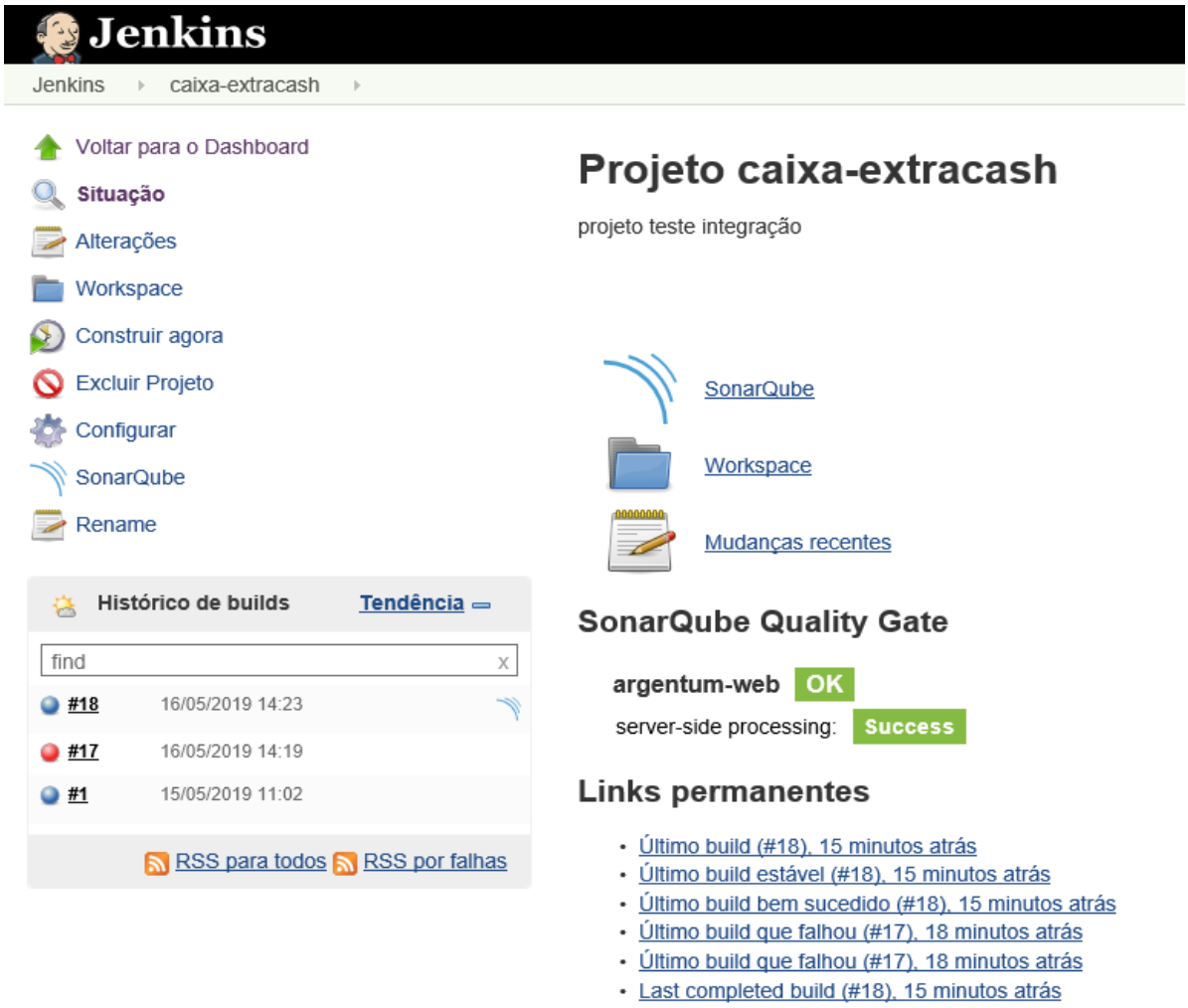


Fonte: Próprio Autor

A análise dos dados apontou que a equipe de projeto apresenta alta rotatividade, não obstante anseia e reconhece que melhorias podem ser realizadas com foco na qualidade. O conhecimento da equipe sobre integração contínua é bom, porém não se usa muito a estrutura legada, que utiliza a ferramenta SVN junto ao Jenkins com uma versão desatualizada e sem plugins de integração instalados Apenas dois dos entrevistados sabiam qual a versão de produção do software na data da coleta de dados.

Após a instalação do SonarQube o mesmo foi integrado por meio de plugin ao Jenkins automatizando as análises de qualidade e geração de relatórios.

Figura 10: Jenkins integrado ao SonarQube



Fonte: Próprio autor

Conforme evidenciado pelos arquivos de log do Jenkins quando alguma alteração no código é confirmada pelo programador ocorre o "*start*" do processo de construção do pacote passando pelos testes com JUNIT e Jacoco e avaliação da qualidade com geração de relatórios pelo Sonar:

```
-------------------------------------------------------

T E S T S

-------------------------------------------------------

Running br.com.caelum.argentum.indicadores.MediaMovelPonderadaTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.234 sec

Running br.com.caelum.argentum.indicadores.MediaMovelSimplesTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.047 sec

Running br.com.caelum.argentum.modelo.CandlestickFactoryTest

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec

Running br.com.caelum.argentum.modelo.NegociacaoTest

Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 sec

Running br.com.caelum.argentum.reader.LeitorXMLTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.433 sec

Results :

Tests run: 13, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time:  51.718 s

[INFO] Finished at: 2019-04-27T21:25:09-03:00

[INFO] ------------------------------------------------------------------------
```

Essa sequência de procedimentos automatizados garante que a alteração realizada em uma funcionalidade não tenha efeito colateral em outra, diminuindo a probabilidade de erro humano e sistêmico.

Os testes manuais e homologação pelo cliente continuam sendo indispensáveis, porém são menos onerados (SIQUEIRA, 2018).

A prototipação com o Raspeberry PI se mostrou uma maneira rápida e fácil de trabalhar a melhoria contínua sem atrapalhar processos legados.

Um aspecto importante a ser considerado é que após a subida do SonarQube houve quebra da construção automática demonstrando que o código não atingiu o mínimo de qualidade necessária e deve ser revisto.

Figura 11- Demonstração de falha na construção provocada pelos testes do Sonar

Fonte: próprio autor

Em nova reunião realizada na sala 1.1 do Bloco 9 da Centralizadora a apresentação a solução foi bem recebida pela equipe em geral e teve feedbacks positivos.

Foi questionado a avaliação o código legado tendo em vista o tamanho esforço necessário para realizar todas as melhorias e em comum acordo foi decidido a versão base e as metas para melhoria gradual.

5 CONSIDERAÇÕES FINAIS

Com o objetivo de agilizar o processo de desenvolvimento, a integração contínua é com toda certeza essencial no dia-a-dia de um grupo de desenvolvimento de software, principalmente se a equipe tem como objetivo realizar entregas com mais qualidade em menores períodos de tempo.

Executamos essa integração num fluxo simples onde nosso projeto de testes automatizados é separado da aplicação, mas, pensando num processo de desenvolvimento de software comum nas empresas hoje, os fluxos de trabalho podem estar presentes realizando testes automatizados desde o início do desenvolvimento do projeto, até a realização da entrega da aplicação em vários ambientes, estando apto a ir inclusive para produção, garantindo assim que a entrega de aplicações de software seja mais rápida e segura; esta é a abordagem de desenvolvimento chamada de Entrega Contínua (*Continuous Delivery* em inglês). Parte em que é retomado o passo a passo do desenvolvimento do estudo, com os respectivos resultados e o consequente desfecho.

Houve ganho em agilidade com a prototipação realizada em ambiente apartado com o Raspberry PI, tendo em vista que não há a necessidade de alterações no ambiente de desenvolvimento da empresa, disponibilização de servidores e solicitações de liberação de regra de firewall.

O uso de plugins atualizados do Jenkins abriu um leque de oportunidades de pesquisa de ferramentas integradas para a melhoria contínua no processo de desenvolvimento de software na equipe MTC.

06 REFERÊNCIAS

ANDRADE, A. S. de Oliveira e Fernando Souza de. Sistemas Embarcados. Hardware e Firmware na Prática. [S.l.]: Érica, 2010. ISBN 8536501057.

BROWN, Tim. **Design thinking**: uma metodologia poderosa para decretar o fim das velhas ideias. Rio de Janeiro: Elsevier, 2010.

DEITEL, H.M. **Sistemas Operacionais** - São Paulo - SP: Pearson Prentice Hall, 2015.

PAUL Duvall, STEPHEN M. Matyas,and Andrew Glover..**ContinuousIntegration**: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series). Addison-WesleyProfessional, 2007.

CASALE, A. **Aprendizagem Baseada em Problemas** – desenvolvimento de competências para o ensino em engenharia. Tese de Doutorado. Universidade de São Paulo, São Paulo, 2013. 162p.

GIMENEZ, SALVADOR PINILOS. **Microcontroladores**: Teoria do hardware e do software Aplicações em Controle Digital Laboratório e Simulação - São Paulo - SP: Pearson Education do Brasil, 2012.

MARTIN, Roger. **Design de negócios**: por que o design thinking se tornará a próxima vantagem competitiva dos negócios e como se beneficiar disso. Rio de Janeiro: Elsevier, 2010.

MOLINARI, Leonardo. **Testes de Software**: Produzindo Sistemas Melhores e Mais Confiáveis. Editora Érica, 2015, 2a Edição, São Paulo.

NEMETH, EVI. **Manual Completo do Linux** - São Paulo - SP: Pearson Makron Books, 2014.

NIELSEN, Jakob. **Designing Web Usability**: The Practice of Simplicity. New Riders Press, 1999.

PEARSON Academia. **Criatividade e inovação**. São Paulo: Pearson Books, 2011.

PINHEIRO, Tennyson; ALT, Luis. **Design Thinking Brasil**: empatia, colaboração e experimentação para pessoas, negócios e sociedade. São Paulo: Elsevier, 2012.

TANENBAUM, A. S., WOODHULL, A. S. Sistemas Operacionais: Projeto e Implementação. Porto Alegre: Bookman 2018.

SIQUEIRA, R. et al. **Continuous delivery**: Building trust in a large-scale, complex government organization. IEEE software, IEEE, 2018.

SOMMERVILLE, I. **Software Engineering**. 9. ed. [s.l.] Addison-Wesley, 2011.

HUMBLE, J. FARLEY, D. **Entrega Contínua**: Como Entregar Software de forma rápida e confiável, 2014.

SMART, F. J. **Jenkins**: The Definitive Guide. O'Reilly Media, Inc, Sebastopol, 2011.

MELO, C. DE O.; FERREIRA, G. R. Adoção de métodos ágeis em uma Instituição Pública de grande porte-um estudo de caso. In: Workshop Brasileiro de Métodos Ágeis, Porto Alegre. (2019).

CAVALCINTI, C. M. C. **Contribuições do Design Thinking para concepção de interfaces de Ambientes Virtuais de Aprendizagem centradas no ser humano**. Tese de Doutorado. Universidade de São Paulo, São Paulo, 2015. 253 p.

PFLEEGER, S. L. **Engenharia de software**: teoria e prática, 2a ed., Pearson, 2004.

PUPIM, E.K; MORALES, A.G; BERNARDO, C.H.C; LOURENZANI, W.L. **Manual para apresentação de monografias, dissertações e teses**. Tupã: Unesp – campus de Tupã, 2016.

Associação Brasileira de Normas Técnicas - ABNT - **NBR 14724** - Informação e documentação – Trabalhos acadêmicos – Apresentação. 2011.

UNIVERSIDADE ESTADUAL PAULISTA. Coordenadoria Geral de Bibliotecas. **Normas para publicações da Unesp**. São Paulo: Editora Unesp, 1994. 4v., v.3, Preparação e Revisão de Textos, Unesp (2010).

UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO. **Manual de normalização de trabalhos acadêmicos**. São Paulo: Univesp, 2018.

MESZAROS, G.x Unit Test Patterns:Refatoring Test Code, Adilson Wesley, Edition by Gerard Meszaro ,2017

ANICHE, MAURÍCIO. **Testes automatizados de software**: Um guia prático. São Paulo: Casa do Código - Livros para o programador, 2017.

UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO. **Manual de normalização de trabalhos acadêmicos**. São Paulo: Univesp, 2018.

UNIVERSIDADE DE SÃO PAULO. Sistema Integrado de Bibliotecas. **Diretrizes para apresentação de teses e dissertações da USP**: documento eletrônico - parte I (ABNT). 2.ed. São Paulo, 2009.102p. Disponível em: http://www.usp.br/prolam ABNT_2011.pdf>. Acesso em: 23 fev. 2016.