UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO

ENGENHARIA DE COMPUTAÇÃO

LUCIANO DOS REIS DE SOUSA

USO DA INTEGRAÇÃO CONTÍNUA PARA MELHORIA NA PRODUTIVIDADE E QUALIDADE NO DESENVOLVIMENTO DE SOFTWARE EM UMA EQUIPE MULTIFUNCIONAL:

Um exemplo da equipe de desenvolvimento e manutenção de um software bancário.

São Paulo - SP

2019

LUCIANO DOS REIS DE SOUSA

USO DA INTEGRAÇÃO CONTÍNUA PARA MELHORIA NA PRODUTIVIDADE E QUALIDADE NO DESENVOLVIMENTO DE SOFTWARE EM UMA EQUIPE MULTIFUNCIONAL:

Um exemplo da equipe de desenvolvimento e manutenção de um software bancário.

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação da Universidade Virtual de São Paulo como requisito parcial para obtenção do título de Engenheiro de Computação.

São Paulo - SP

2019

LUCIANO DOS REIS DE SOUSA

USO DA INTEGRAÇÃO CONTÍNUA PARA MELHORIA NA PRODUTIVIDADE E QUALIDADE NO DESENVOLVIMENTO DE SOFTWARE EM UMA EQUIPE MULTIFUNCIONAL:

O exemplo da Centralizadora de TI da Caixa Econômica Federal

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação da Universidade Virtual de São Paulo como requisito parcial para obtenção do título de Engenheiro de Computação.

Aprovado em \_\_\_\_/\_\_\_\_/\_\_\_\_\_\_

Conceito: \_\_\_\_\_\_\_\_\_

BANCA EXAMINADORA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PROFESSOR PRESIDENTE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PROFESSOR MEMBRO

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PROFESSOR MEMBRO

**RESUMO**

Qualidade de software, além de influenciar na percepção dos usuários quanto ao produto, pode significar redução de custos em 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 e terceirizados, é importante que haja domínio sobre os processos e qualidade dos seus sistemas, para auxiliar tomada de decisões e exigir devidamente o 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 tem 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 conta com uma revisão de literatura acerca de temas correlatos à Integração Contínua, Processos de Software Qualidade de Software e Testes Automatizados. É esperado, como projeto futuro a automatização de alguns processos utilizando conceitos, ferramentas de hardware e software livre e de mercado, que foram discutidos durante a graduação.

**Palavras-chave:** Integração Contínua, Jenkins, testes automatizados, Sonar.

**SUMÁRIO**

1 Introdução................................................................................................................06

2 Revisão Bibliográfica ............................................................................................08

3 Objetivos .................................................................................................................10 3.1 Objetivo Geral ......................................................................................................10 3.2 Objetivos Específicos ..........................................................................................10

4 Material e Método ............................................................................................... ...11 4.1 Equipamentos e Ferramentas..............................................................................12

4.2 Análise dos dados ................................................................ ............................. 14 5 Cronograma ........................................................................................................... 15 6 Considerações Finais ............................................................................................ 16 7 Referências Bibliográficas ..................................................................................... 17

1. **INTRODUÇÃO**

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

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.

No caso estudado, uma equipe de desenvolvedores que estão trabalhando em um projeto juntos, utilizam como servidor de versionamento do código-fonte, como o SVN – O Apache Subversion. 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 atrapalha o fluxo de desenvolvimento da equipe, atrasando entregas em tempo de Sprints negociadas com o dono do produto e gera erros advindos da gerência de configuração que podem causar desde pequenos erros à prejuízo financeiro quando o erro não é detectado nos testes e homologação e vai à produção.

Outro exemplo bastante comum, é quando há a necessidade de extrair a última versão de projeto para construir e colocar no servidor de produção. Quando não há um servidor que executa nossos 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 ficarão furiosos 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 foram parar no servidor de versionamento. A descoberta dos erros é feita de forma tardia. Os erros ou falhas de um código problemático aparecem bem depois do que quando foram escritos.

O trabalho proposto segue a ótica do processo para Design Thinking que é organizada em uma estratégia cíclica que permite a evolução gradativa da solução como um todo, conforme sugere a figura 1, proposta por Cavalcanti (2015).

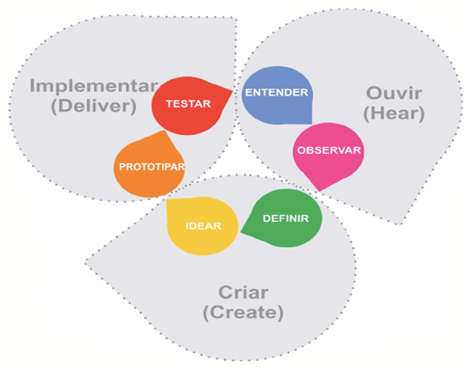


Figura 1: Ilustração para o processo cíclico e evolutivo sugerido para Design Thinking

Ainda nesse sentido, 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, a problemática tratada neste trabalho é:

“Baseado em problemas de Gerência de Configuração de Software, o que pode ser feito para melhorar o processo de desenvolvimento com a utilização Integração Contínua 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.

1. **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 (Marcelo Fantinato).

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)

Integração Contínua é uma prática de desenvolvimento de software onde membros da equipe incorporam mudanças ao software frequentemente, utilizando processos de compilação e testes que asseguram a integridade do projeto (FOWLER, 2016). Segundo Durvall Paul (2018) um dos primeiros autores a utilizar o termo Integração Contínua foi Grady Booch, que atribuiu 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 (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).

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).

**3. OBJETIVOS**

3.1 Objetivo Geral:

Esta pesquisa tem como objetivo propor melhorias para o desenvolvimento de software utilizando os princípios da integração contínua aplicado ao contexto um exemplo da equipe de desenvolvimento e manutenção do software bancário MTC.

3.2 Objetivos específicos:

Os objetivos específicos deste TCC foram:

* Avaliar os benefícios da Integração contínua e Testes Automatizados e aplicar as ao processo de desenvolvimento de software da equipe de desenvolvimento do sistema MTC da empresa CAIXA ECONOMICA FEDERAL.
* Prototipação com baixo custo utilizando um Raspberry PI como um protótipo de servidor de integração contínua.

**4. MATERIAL E MÉTODOS**

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. Foram distribuídos formulários de pesquisa individuais, contendo questões voltada 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. As respostas foram inseridas em uma planilha eletrônica para tabulação de dados cujas algumas perguntas são apresentadas abaixo:

**4.1 EQUIPAMENTOS E FERRAMENTAS**

Em busca de baixo custo e uso de softwares livres serão 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;
* SAIDA HDMI E RJ43;
* 4 ENTRADAS USB; MICRO SLOT SD;
* 40 PINOS PARA CIRC ELETRÔNICO;
* ENTRADAS PARA VIDEO, AUDIO, CAM.

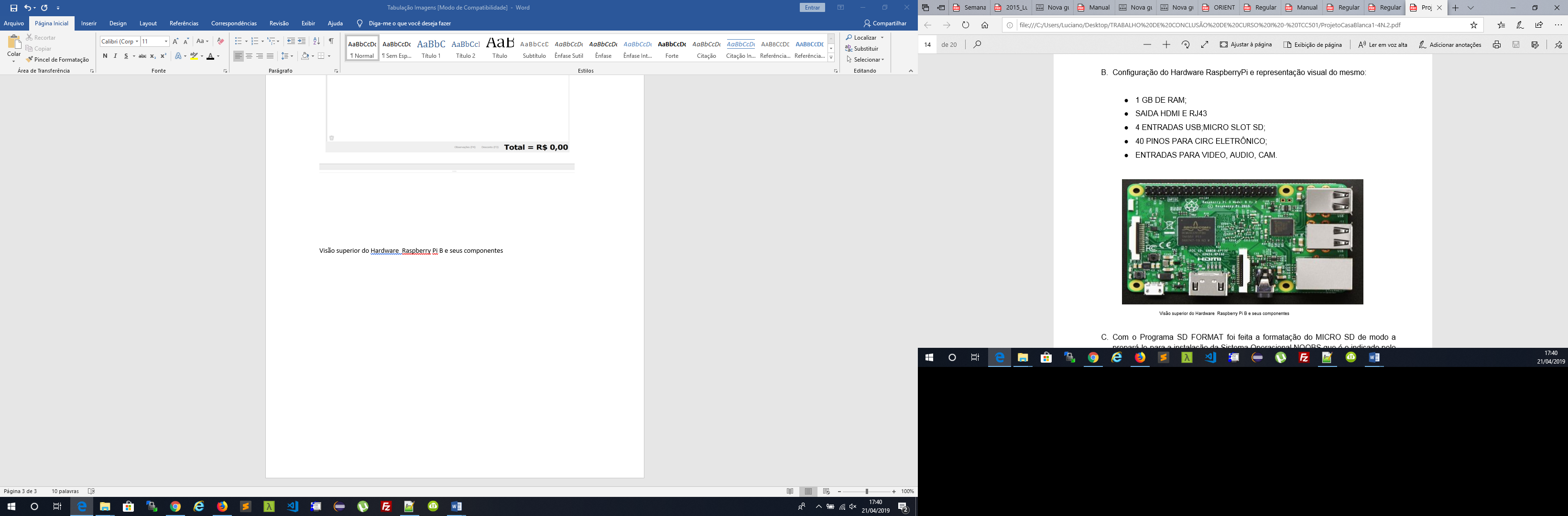


Figura 1: Visão superior do Hardware Raspberry Pi B e seus componentes

* Softwares:

1. Controle de versão: Apache Subversion e GIT;
2. Servidor de automação: Jenkins;
3. Cobertura código**:** EclEmma, SonarLinte JUnit integrados ao Eclipse;
4. Testes: SonarQube integrado ao Maven;
5. 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. Abaixo o endereço para baixar (clonar) o repositório:

<https://github.com/LucianoReisSousa/Projetos_Univesp>

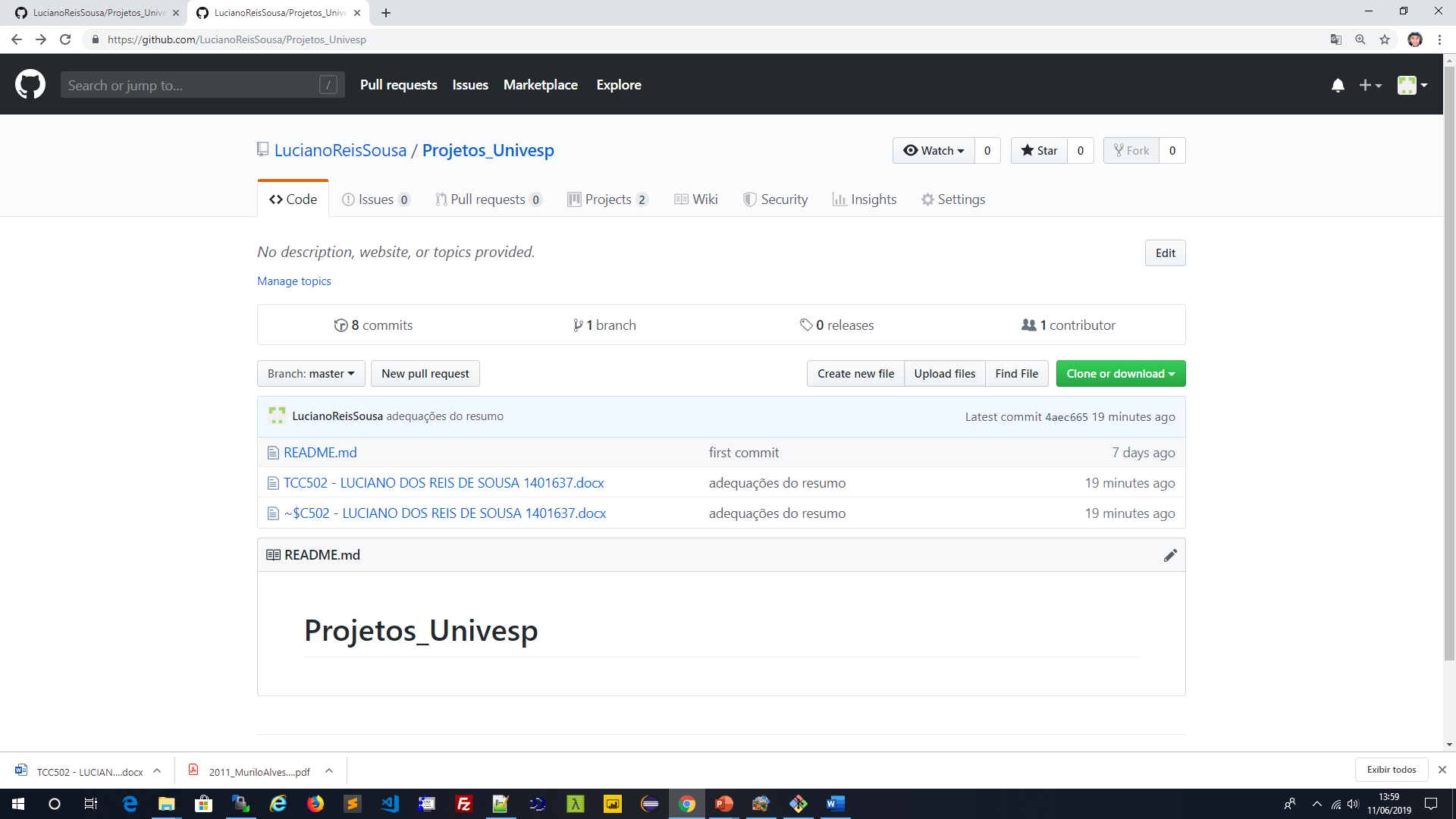
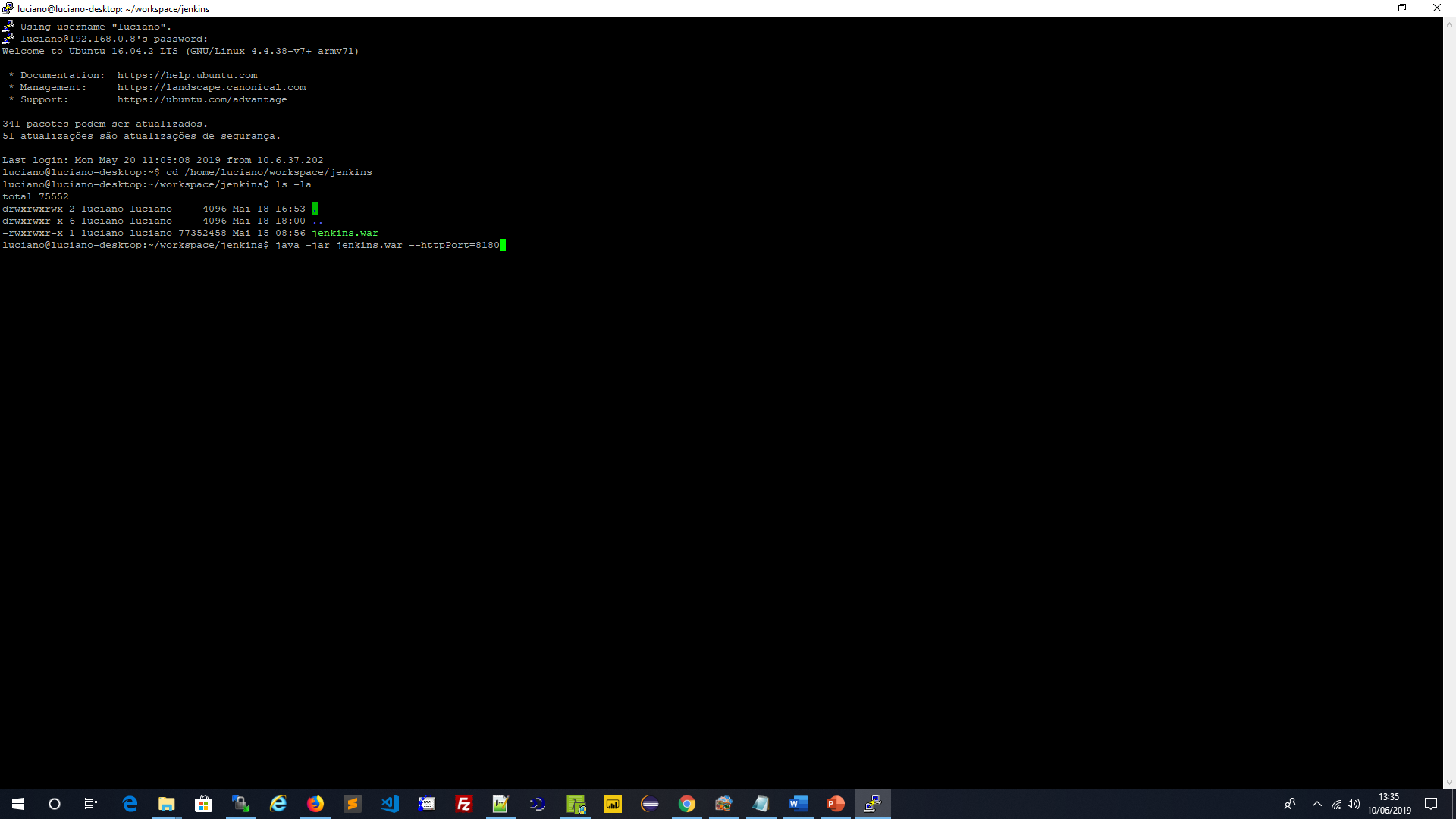
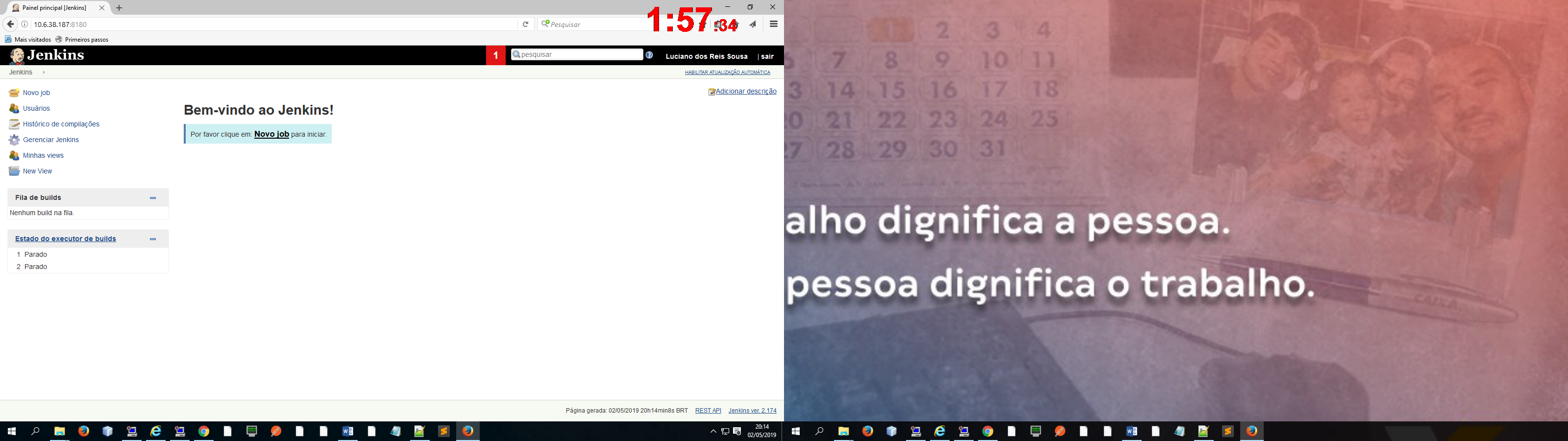


Figura 2: Visão do Repositório do Projeto criado no GITHUB

* Instalação do Jenkins utilizando a porta 8180.



* Jenkins disponível na rede da empresa após a instalação.



* Configurações realizadas no Maven e download do jacoco-maven-plugin para cobertura dos testes

λ mvn -version

Apache Maven 3.6.1 (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-04T16:00:29-03:00)

Maven home: C:\Users\Luciano\Documents\workspace\apache-maven-3.6.1\bin\..

Java version: 1.8.0\_211, vendor: Oracle Corporation, runtime:

Default locale: pt\_BR, platform encoding: Cp1252

OS name: "windows 8.1", version: "6.3", arch: "amd64", family: "windows"

Downloaded from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/15.0/guava-15.0.jar (2.2 MB at 108 kB/s)

[INFO]

[INFO] --- jacoco-maven-plugin:0.7.6.201602180812:prepare-agent (default)

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

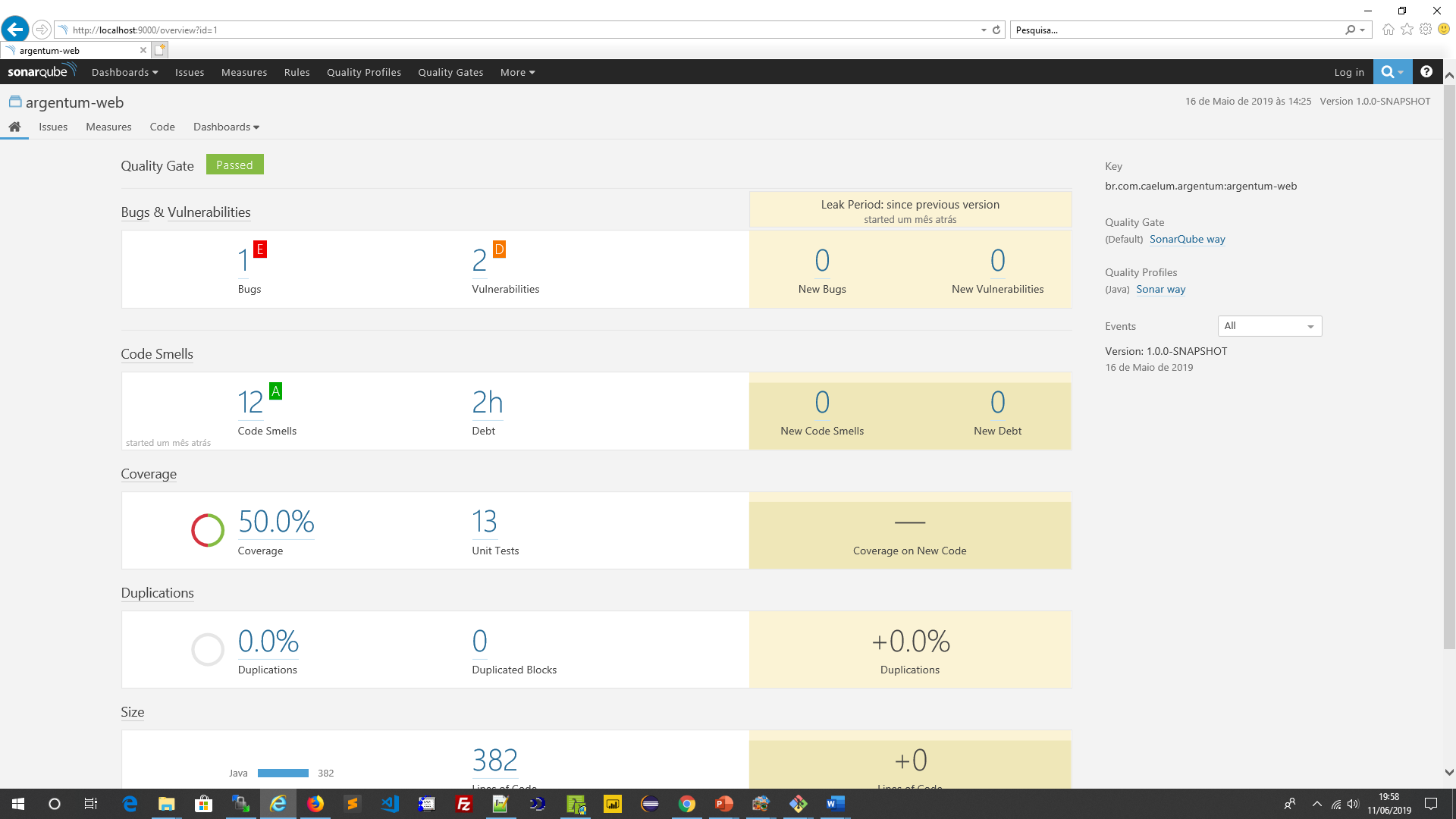
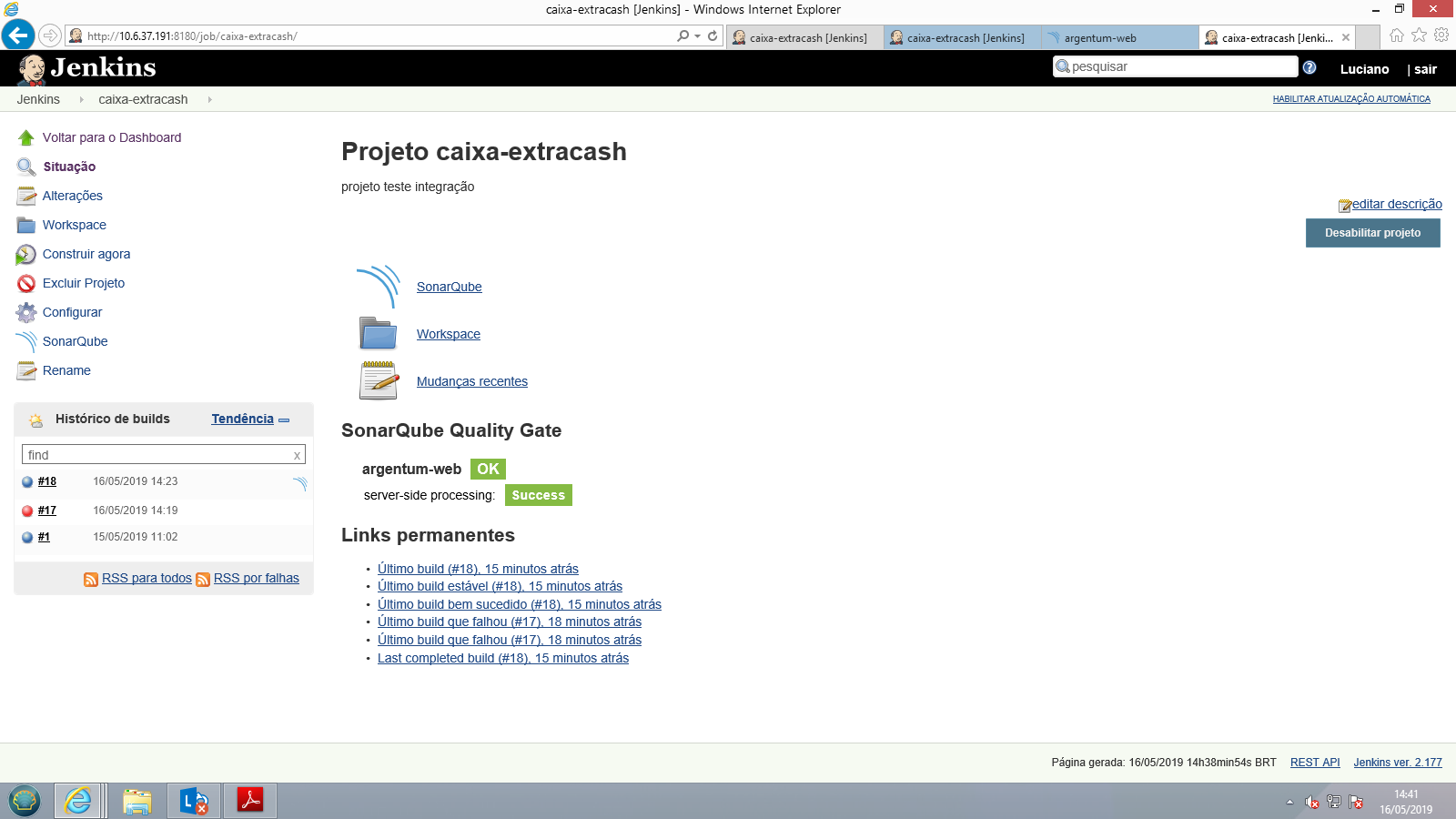


Figura 3: Modelo de relatório de cobertura de testes gerado pelo SonarQube.

**4.2 ANÁLISE DOS DADOS**

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 sonar o mesmo foi integrado ao Jenkins automatizando as análises de qualidade e geração de relatórios.



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 e

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.

Em consonância com que o professor Marcelo Fantinato falou em aula os testes manuais continuam sendo indispensáveis porém são menos onerados.

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

**5. CONSIDERAÇÕES FINAIS**

**7. REFERÊNCIAS BIBIOGRAFICAS**

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

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

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.

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

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

Site Raspberry​, Documentação​, disponível no endedereço: <https://www.raspberrypi.org/>

PRESSMAN, R. S. Engenharia de Software. [s.l.] McGraw Hill Brasil, 2011.

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.

SONAR. SonarQube Disponível em: <http://www.sonarqube.org/>. Acesso em: 18 abril. 2019.

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. Anais...2010. Disponível em: <http://agilcoop.org.br/files/WBMA\_Melo\_e\_Ferreira.pdf>. Acesso em: 13 abril. 2019.

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.

Cavalcanti, 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.

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. Disponível em: <https://www.tupa.unesp.br/Home/ensino/pos-graduacao/agronegocioedesenvolvimento/pa-manual-normalizacao-pgad.pdf. Acessado em 02/06/2017>.

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.

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

GERÊNCIA E QUALIDA DE SOFTWARE – Testes de Unidade Aula 08. Professor responsável : Fábio Levi Siqueira. In: Gerência e Qualidade de Software. Produção: Universidade Virtual do Estado de São Paulo. Local: Produtora do vídeo, ano. Disponível em: <http://>. Acesso em: dia mês

ano.

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.

Metodologia Científica

Informática - Aula 14 - Recursos necessários para apresentações públicas

Engenharia de Software aula 16.

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

**E. Medeiros, Desenvolvendo software com UML 2.0: definitivo, Pearson, 2004.**