Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Implantação de práticas de Integração Contínua: Um relato de experiência em um laboratório de pesquisa e desenvolvimento de *software*

William Santos Nunes

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Igor Muzetti Pereira

COORIENTAÇÃO:
Vicente José Peixoto de Amorim

Setembro, 2017
João Monlevade–MG

William Santos Nunes

Implantação de práticas de Integração Contínua: Um relato de experiência em um laboratório de pesquisa e desenvolvimento de *software*

Orientador: Igor Muzetti Pereira

Coorientador: Vicente José Peixoto de Amorim

Monografia apresentada ao curso de Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Setembro de 2017

N972i Nunes, William Santos.
Implantação de práticas de integração contínua [manuscrito]: um relato de experiência em um laboratório de pesquisa e desenvolvimento de software / William Santos Nunes. - 2018.

63f.: il.: color; grafs.

Orientador: Prof. MSc. Igor Muzetti Pereira.
Coorientador: Prof. MSc. Vicente José Peixoto de Amorim.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Departamento de Computação e Sistemas de Informação.

1. Desenvolvimento ágil de software. 2. Software de aplicação. 3. Automação. 4. Android (Recurso eletrônico). I. Pereira, Igor Muzetti. II. Amorim, Vicente José Peixoto de. III. Universidade Federal de Ouro Preto. IV. Titulo.

Catalogação: ficha@sisbin.ufop.br CDU: 004.9

ANEXO V – Folha de Aprovação

Curso de Sistemas de Informação

FOLHA DE APROVAÇÃO DA BANCA EXAMINADORA

Implementação de práticas de Integração Contínua: Um relato de experiência em um laboratório de pesquisa e desenvolvimento de software

William Santos Nunes

Monografia apresentada ao Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto como requisito parcial da disciplina CSI499 – Trabalho de Conclusão de Curso II do curso de Bacharelado em Sistemas de Informação e aprovada pela Banca Examinadora abaixo assinada:

Igor Muzetti Pereira

Prof. MSc. Igor Muzetti Pereira
DECSI - UFOP

Prof. MSc. Vicente José Peixoto de Amorim
DECSI - UFOP

Gleiph G. L. Muniz

Prof. Dr. Gleiph Giotto
DECSI – UFOP

Fernando B Oliveira

Prof. Dr. Fernando Bernardes de Oliveira
DECSI – UFOP

João Monlevade, 04 de setembro de 2017

ANEXO IV - Ata de Defesa

ATA DE DEFESA

Aos 04 dias do mês de setembro de 2017, às 16 horas, na sala C203 do Instituto de Ciências Exatas e Aplicadas, foi realizada a defesa de Monografia pelo aluno **William Santos Nunes,** sendo a Comissão Examinadora constituída pelos professores: Prof. MSc. Igor Muzetti Pereira, Prof. MSc. Vicente José Peixoto de Amorim, Prof. Dr. Gleiph Giotto e Prof. Dr. Fernando Bernardes de Oliveira.

O candidato apresentou a monografia intitulada: *"Implementação de práticas de Integração Contínua: Um relato de experiência em um laboratório de pesquisa e desenvolvimento de software"*. A comissão examinadora deliberou, por unanimidade, pela aprovação do candidato, com nota 10 (dez), concedendo-lhe o prazo de 15 dias para incorporação das alterações sugeridas ao texto final.

Na forma regulamentar, foi lavrada a presente ata que é assinada pelos membros da Comissão Examinadora e pelo graduando.

João Monlevade, 04 de setembro de 2017.

Igor Muzetti Pereira

Prof. MSc. Igor Muzetti Pereira
Professor Orientador/Presidente

Prof. MSc. Vicente José Peixoto de Amorim
Professor Coorientador

Gleiph G. L. Menezes

Prof. Dr. Gleiph Giotto
Professor Convidado

Fernando Oliveira.

Dr. Fernando Bernardes de Oliveira
Professor Convidado

William Santos Nunes

William Santos Nunes
Graduando

TERMO DE RESPONSABILIDADE

Eu, **William Santos Nunes** declaro que o texto do trabalho de conclusão de curso intitulado "***Implantação de práticas de Integração Contínua: Um relato de experiência em um laboratório de pesquisa e desenvolvimento de software***" é de minha inteira responsabilidade e que não há utilização de texto, material fotográfico, código fonte de programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos respectivos autores.

João Monlevade, 04 de setembro de 2017

William Santos Nunes

Dedicado à minha família, amigos e todos os envolvidos.

Agradecimentos

Agradeço à minha família e meus amigos que sempre me apoiaram e não deixaram de acreditar nesta grande realização, compreendendo os programas e compromissos adiados.

Agradeço ao meu orientador, Prof. Igor Muzetti, pelo incentivo, gerência e paciência, ao Prof. Vicente J. P. Amorim pelos conselhos e discussões de ideias, juntamente aos amigos de laboratório iMobilis. Ambos possibilitaram a execução deste trabalho e o amadurecimento acadêmico e profissional.

Agradeço a todos que participaram desta minha jornada acadêmica, oferecendo apoio, conselhos, ótimos momentos de descontração e estudo.

"Estamos irrevogavelmente em um caminho que nos levará às estrelas. A não ser que, por uma monstruosa capitulação ao egoísmo e à estupidez, acabemos nos destruindo."

Carl Sagan (1934–1996).

Resumo

A intensa condição sobre a indústria de *software* para conseguir demonstrar resultados mais rapidamente vem revolucionando seu processo de desenvolvimento. O que no passado era sinônimo de atrasos na entrega e aumento nos custos do projeto, atualmente é referência em métodos que propõem agilidade aliada inclusive à qualidade do *software*. Dentro deste segmento do desenvolvimento ágil é nítida a importância da Integração Contínua, que promove um *feedback* ao incorporar mudanças no *software*, mostrando à equipe o que funciona e o que está pendente, constituindo a real noção de completude de um projeto, além de alavancar a realização de outras atividades que compõe o processo de desenvolvimento que podem ser "deixadas de lado" durante todo esse ritmo acelerado. A partir de um projeto real desenvolvido dentro do ambiente do laboratório acadêmico iMobilis, observa-se uma necessidade de reduzir os riscos do projeto e obter um retorno ágil do comportamento das mudanças que ocorrem durante o seu desenvolvimento. Este trabalho apresenta um relato de experiência ao implantar a Integração Contínua no projeto de pesquisa acadêmico *MobMine*, agregando ao conjunto de demais práticas de engenharia de *software*, bem como o *BOPE process*, já adotadas no laboratório e no projeto, o qual consiste em uma aplicação Android que auxilia na criação de soluções e automação de alguns processos internos de uma organização do ramo de mineração, obtendo um resultado prévio satisfatório, observado pelas respostas da equipe por meio de um questionário aplicado em duas fases, antes e depois do início da implantação, além da análise quantitativa sobre o aumento de *Merges* realizados pela equipe no projeto. As dificuldades encontradas durante a implementação da prática neste ambiente são discutidas, além do ponto de vista inicial da equipe após a implantação da prática e os impactos iniciais no projeto.

Palavras-chaves: Integração Contínua. Ambiente Acadêmico. Desenvolvimento Ágil. Android. Projeto de Pesquisa.

Abstract

The intense condition about the software industry to be able to demonstrate results more quickly has revolutionized its development process. What in the past was synonymous with delays in delivery and increase in project costs, is currently a reference in methods that propose agility, allied to the quality of software. Within this segment of agile development, it is clear the importance of Continuous Integration promoting feedback by to incorporating changes in the software, showing the team what works and what is pending, constituting the real notion of completeness of a project, as well as leveraging the realization of other activities that make up the development process that can be "laid aside" during this accelerated pace. As of a real project developed within the environment of an academic laboratory, an Android application that assists in the creation of solutions and automation of some internal processes of a mining organization, is observed a need to reduce the risks of the project and obtain an return agile of the behavior of the changes that occur during its development. This work presents an experience report by implementing Continuous Integration in the academic research project MobMine, adding to the set of other software engineering practices, as well as the BOPE process, already adopted in the laboratory and in the project, which consists of an Android application that helps in the creation of solutions and automation of some internal processes of an organization of the mining branch, obtaining a satisfactory prior result, observed by the team's answers through a questionnaire applied in two phases, before and after the beginning of the implantation, besides the increase in the number of Merges observed in the project. The difficulties encountered during the implementation of the practice in this environment are discussed, in addition to the team's initial point of view after the implementation of the practice and the initial impacts on the project.

Key-words: Continuous Integration. Academic Environment. Agile Development. Android. Research Project.

Lista de ilustrações

Lista de abreviaturas e siglas

ADB Android Debug Bridge

APK Android Package

AVD Android Virtual Device

GCS Gerência de Configuração de Software

IC Integração Contínua

ICEA Instituto de Ciências Exatas e Aplicadas

IDE Integrated Development Environment

JDK Java Development Kit

RAM Random Access Memory

SDK Software Development Kit

UFOP Universidade Federal de Ouro Preto

USB Universal Serial Bus

XP Extreme Programming

Sumário

1 Introdução

A indústria de *software* tem lidado com grandes desafios como entregas mais frequentes, alto padrão de qualidade e flexibilidade nos requisitos aliados a baixos custos de produção. Em consequência deste ambiente que demanda por produtos inovadores e um processo de produção mais simpático às mudanças, os métodos ágeis despontam como uma solução para que as equipes de desenvolvimento obtenham sucesso e sobrevivência no mercado (MATHARU et al., 2015). As *startups*, por exemplo, vêm utilizando dos métodos ágeis para se adaptar melhor à estratégia de negócios (GIARDINO et al., 2014). Os seus objetivos a curto prazo, por sua vez, trazem uma constante pressão para conseguir demonstrar resultados mais rapidamente.

De acordo com o *11º Annual State of Agile Report* (VERSIONONE, 2017), a adoção de métodos ágeis continua crescendo na indústria. O número de grandes organizações que adotam métodos ágeis aumentou de 24% para 26% no último ano. Essa adesão ainda tem muito a crescer, cerca de 95% dos entrevistados revelaram que suas organizações adotavam algum método ágil, mas 60% das esquipes ainda não à praticavam de fato. No entanto, as organizações ainda estão em fase de amadurecimento quanto à aplicação das metodologias ágeis, é o que afirma 80% dos entrevistados. Seus resultados vêm sendo positivos, obtendo números expressivos quanto às taxas de sucesso nos projetos, cerca de 98%, juntamente na transição para os métodos ágeis.

Estas equipes ágeis têm como característica um processo iterativo de desenvolvimento, gerando incrementos funcionais de *software* frequentemente (PRESMAN, 2011a). Ao entregar estas frações funcionais de um produto mais rapidamente, as equipes precisam gerenciar as mudanças no *software*, estar ciente do que funciona, administrar a qualidade, às vezes deixada de lado por conta do ritmo acelerado, além de estar ciente da completude do projeto, atentando-se também ao *feedback* do cliente (PRIKLADNICKI; WILLI; MILANI, 2014).

O uso da Gerência de Configuração de Software (GCS) tem se mostrado crucial para administrar todas estas mudanças nos projetos, seja num todo ou em pontos específicos, como no desenvolvimento de uma funcionalidade. Em diversos momentos, desenvolvedores trabalham em uma ou mais funcionalidades ao mesmo tempo, de modo que, esta pode ser continuamente modificada. Consequentemente o que se tem é um trabalho simultâneo, seja em uma visão micro, por exemplo, no desenvolvimento de um incremento, ou macro, considerando um conjunto de incrementos que constituem uma versão do *software*. Portanto, há uma necessidade de integrar o trabalho destes desenvolvedores e garantir que estas funcionalidades e todos os demais componentes funcionem juntos a cada entrega. A

Integração Contínua (IC) está encarregada de auxiliar nesse processo, evitando que a equipe espere até o final do projeto para integrar suas modificações levando a prováveis problemas de qualidade de *software*, com elevado custo de reparo e gerando atrasos na entrega do projeto (DUVALL; MATYAS; GLOVER, 2007).

A IC é considerada um dos pilares da agilidade. Ainda que o seu processo de implementação seja complicado, seu maior e mais abrangente benefício é a redução destes riscos, mostrando à equipe o que funciona e o que não, além dos problemas que ainda estão pendentes no projeto (FOWLER, 2006), apresentando um *feedback* ao finalizar o desenvolvimento de cada funcionalidade incorporada no *software*.

O ambiente de um laboratório acadêmico busca desenvolver ideias, realizar pesquisas, relacionar a teoria obtida em sala de aula com a prática, de certa maneira, também aproximar o aluno do ambiente profissional da indústria (RODRIGUES; ESTRELA, 2012). Deste modo, o laboratório de computação móvel iMobilis, situado no Instituto de Ciências Exatas e Aplicadas (ICEA), campus da Universidade Federal de Ouro Preto (UFOP), tem implantado em sua cultura diversas práticas de gerenciamento de projetos e equipes, tais como o *BOPE process* (PEREIRA; CARNEIRO; PEREIRA, 2013), adaptações do Scrum e Extreme Programming (XP), além de outras práticas da engenharia de *software*, objetivando esta aproximação além de aperfeiçoar o gerenciamento e a qualidade dos seus projetos, proporcionando também uma melhoria contínua do processo, de modo que, o produto final pretendido, seja desenvolvido dentro do prazo e atendendo os custos estimados (REZENDE, 2005), resultando no sucesso do projeto.

Dentre os projetos realizados, destaca-se o projeto de pesquisa *MobMine*. Promovido pela parceria entre a UFOP e a Vale S.A.[1], uma organização privada multinacional do ramo de mineração. O projeto consiste em uma aplicação para a plataforma Android, que auxilia na criação de soluções e automação de alguns processos internos da organização.

Este projeto tem duração inicial de dois anos e conta com um considerável escopo, portanto, o seu desenvolvimento é realizado por uma equipe composta de dez membros. Estão presentes nesta equipe três professores orientadores e o parceiro financiador do projeto de pesquisa. Os seis membros restantes são alunos que atuam no contexto de desenvolvimento da aplicação, dentre eles está presente o papel de um analista de testes e um desenvolvedor líder, este diferentemente dos demais, não é um aluno e sim um pesquisador recém graduado, responsável por realizar a integração e análise dos incrementos de código fonte que são gerados pelos quatro demais desenvolvedores.

Como consequência do avanço do projeto, a aplicação entra em fase de operação, na qual algumas funcionalidades desenvolvidas anteriormente começam a ser, de fato, utilizadas. Uma vez que, novas funcionalidades e melhorias ainda estão em produção,

[1] <http://www.vale.com/brasil/>

existe uma grande preocupação para que estas não comprometam a versão que está em operação quando forem integradas. Consequentemente, testes são realizados para encontrar o máximo de problemas possíveis que podem ser gerados pela introdução desse incremento. Este processo de produzir, integrar, testar, corrigir e liberar novas versões se repete enquanto a aplicação estiver em desenvolvimento.

Em busca de reduzir os riscos das entregas do *software*, obter um retorno rápido do comportamento destas mudanças no *software* e melhorar a metodologia de desenvolvimento da equipe, a IC se mostrou uma ótima prática a ser implantada pela equipe. Possibilitando deste modo, tratar de maneira aperfeiçoada e constante os problemas enfrentados de integração, como conflitos de código e introdução de *bugs* em versões estáveis. Outro ponto importante é amenizar o esforço e tempo gastos para depurar e encontrar os problemas (FOWLER, 2006), transferindo-os para o desenvolvimento de novas funcionalidades, uma vez que, boa parte dos desenvolvedores são alunos e são esperadas apenas quinze horas de dedicação por semana.

Este trabalho apresenta um relato de experiência ao revisar a literatura, projetar, propor e implantar práticas de IC no contexto do projeto *MobMine*, obtendo um resultado prévio satisfatório, observado pelas respostas da equipe por meio de uma pesquisa realizada em duas fases, antes e depois do início da implantação, além da análise quantitativa sobre o aumento de *Merges* realizados pela equipe no projeto. Os impactos iniciais e a curto prazo, juntamente da avaliação dos desenvolvedores se mostraram positivos. As dificuldades encontradas durante a implementação da prática em um ambiente acadêmico são discutidas, uma vez que, grande parte dos desenvolvedores são alunos em formação profissional que estão estabelecendo o seu primeiro contato com um projeto real, fator diretamente ligado aos desafios encontrados.

O restante do trabalho está dividido como segue. O Capítulo 2 contextualiza e apresenta conceitos importantes abordados. Os trabalhos correlatos também são retratados nesta seção. No Capítulo 3 é apresentada a metodologia de trabalho e o processo de integração de incrementos de *software* utilizado anteriormente pela equipe. As ferramentas utilizadas no desenvolvimento da implantação da prática de IC além da configuração do ambiente também são descritos. Os resultados iniciais são apresentados no Capítulo 4. As conclusões, desafios encontrados e as perspectivas futuras são descritas no Capítulo 5.

2 Revisão bibliográfica

Este capítulo tem como objetivo apresentar os conceitos pertinentes empregados neste trabalho. Tais conceitos são fundamentais para compreensão, apresentação e desenvolvimento da prática de IC, portanto, compõem-se de: GCS, Metodologias Ágeis e IC. Além disto, neste capítulo também são retratados os trabalhos correlacionados.

2.1 Gerência de Configuração de Software

O gerenciamento de configuração de *software* é um conjunto de atividades que buscam administrar as modificações no *software*. Consistem em atividades de supervisão e controle de todos os itens que são parte deste *software*, acompanhando-o em toda sua vida útil (até que seja retirado de operação), no qual se objetiva alcançar a sua qualidade (PRESMAN, 2011b).

Os *softwares* passam por várias mudanças, tanto em fase de desenvolvimento como em fase de uso. Isto faz com que toda a evolução do *software* precise ser controlada, pois cada uma das alterações de um desenvolvedor, seja ela a partir da introdução de novos requisitos, correção de *bugs* ou melhorias, geram novas versões do *software* (SOMMERVILLE, 2011a).

Dentre o conjunto de atividades relacionadas à GCS, este trabalho compreende o controle de versão, construção do sistema e gerenciamento de *releases*, definidos a seguir. O gerenciamento de versões (SOMMERVILLE, 2011a) busca manter as alterações sobre os artefatos do *software* administráveis, de modo a garantir um histórico de mudanças que constituem diferentes versões deste artefato, sem que alguma versão interfira em outras. Desta maneira, uma ferramenta de controle de versão possibilita gerenciar todas as alterações feitas, viabilizando o trabalho de vários desenvolvedores em um mesmo artefato sem que o ato de sobrescrever algo seja um problema, tornando possível reverter para uma versão mais antiga (PRESMAN, 2011b).

A construção do sistema (SOMMERVILLE, 2011a) consiste na montagem, compilação, ligação e testes dos artefatos do *software*, criando um sistema completo e executável, gerando *releases* (versão distribuída ao cliente) ou *builds* de maneira automática ou não. O processo tem um grande nível de complexidade além de ser potencialmente sujeito a erros. Esta atividade também pode interagir com o gerenciamento de versões, de modo que, alguma mudança realizada nos artefatos possa lançar uma construção automática do sistema.

O gerenciamento de *releases* (SOMMERVILLE, 2011a) trata-se da elaboração de uma versão do *software* para ser entregue ao cliente, englobando desde, por exemplo,

estipular a data da entrega até a documentação das mudanças presentes na *releases*, além de manter o acompanhamento destas versões liberadas ao cliente.

2.2 Metodologias Ágeis

Uma alternativa aos métodos tradicionais, no qual é complicado tratar de maneira amigável a volatilidade dos requisitos, os métodos ágeis têm sua origem no Manifesto Ágil (BECK et al., 2001), caracterizado como um marco histórico que concebeu um documento assinado por dezessete renomados desenvolvedores, autores e consultores da área de *software*, denominados como Agile Alliance. Tinham como objetivo discutir práticas de desenvolvimento de *software* de uma maneira mais leve, rápida e centrada em pessoas (PRIKLADNICKI; WILLI; MILANI, 2014).

É especificado como método tradicional, chamado também de metodologias pesadas ou orientadas a documentação, os modelos de processo de *software* que dividem seu desenvolvimento em etapas ou fases pré-definidas, dependentes da conclusão de tarefas anteriores para evoluir no processo, em que ao final, é gerado algum documento, artefato ou versão do *software* (SOMMERVILLE, 2011b). Este modelo dificulta a administração do projeto, pois uma alteração, em algum requisito, faz necessário retornar ao início da fase, como acontece nos modelos baseados em cascata (PRESMAN, 2011a).

Existem diversos métodos ágeis, cada um define suas próprias práticas específicas, porém todos compartilham dos valores e princípios contidos no Manifesto Ágil. Em meio aos principais métodos, o XP é um dos mais utilizados para desenvolvimento de *software* (QURESHI, 2012). Nele os requisitos são expressos como cenários que geram uma série de tarefas, caracterizando um curto intervalo entre as versões do *software*. Dentre as principais práticas do XP está o planejamento incremental, sustentado pelas pequenas versões funcionais que devem ser incluídas em incrementos do *software*, uma vez que, estes estão diretamente ligados à IC, uma prática com intuito de incorporar estes pequenos incrementos ao *software* como um todo (SOMMERVILLE, 2011b).

2.3 Integração Contínua

Esta prática tem como característica a frequente integração das modificações de código fonte realizadas pelos desenvolvedores de uma equipe com a base principal do *software*, pelo menos diariamente. O processo de integração deve contar com uma *build* automatizada, incluindo testes, buscando descobrir erros de integração mais rapidamente. Essa prática, é vista por várias equipes como grande aliada na redução dos problemas e erros de integração de código fonte, o que permite o desenvolvimento de *software* coeso mais rapidamente (FOWLER, 2006).

A IC deve ser vista como uma maneira de acentuar as demais atividades relacionadas à qualidade do *software*, uma vez que, possam estar um pouco de lado ou até mesmo, não ocorrendo no projeto, por exemplo testes de unidade, análise de código, testes de sistema, entre outros. Estas são caracterizadas como atividades capazes de trazer benefícios ou até mesmo fundamentais para o projeto, existindo a possibilidade de serem potencialmente reforçadas a partir de um método contínuo, ou seja, executadas com mais frequência, muitas vezes de maneira automatizada (STÅHL; BOSCH, 2014a).

Não existe um passo a passo a ser seguido para a implementação da IC (FOWLER, 2006). Deste modo, alguns detalhes em como se deve implementá-la ficam em aberto, ocasionando uma disparidade das maneiras como se é realizada a IC nas diferentes equipes de desenvolvimento de *software*, o que resulta em alguns pontos de distinção no processo e nos resultados variando para cada caso (STÅHL; BOSCH, 2014b).

Muitos destes fatores dependem das características da equipe de desenvolvimento e do ambiente em que ela está inserida, entretanto, a partir de constatações em experiências com projetos passados, Fowler (2006) apresenta pontos que devem ser seguidos como elementos básicos ou princípios relacionados à IC, auxiliando a efetivar sua implantação ou até mesmo tornando-a menos árdua. Estes princípios são apresentados a seguir.

A equipe deve **manter um único repositório de código** utilizando-se de ferramentas que fazem parte da GCS que envolvem o controle de versão. Todos os artefatos necessários para realizar uma *build* do *software* em desenvolvimento devem estar neste repositório, que deve estar em um local de conhecimento geral dos desenvolvedores.

Automatizar a *build*. A construção dos artefatos em um sistema completo e executável é um processo complexo e que envolve compilação, manipulação de arquivos e assim por diante. Contudo, espera-se que essa construção seja feita de maneira automática e em um ambiente diferente das IDEs, como por exemplo, um servidor de desenvolvimento que realize *builds* principais do projeto.

Fazer com que a *build* seja auto testável. Aqui entram os testes automatizados com objetivo de verificar boa parte da base do código em busca de problemas. O *feedback* em caso de falha, em qualquer teste específico, deve ser a quebra da *build*, ou seja, interromper o processo de modo que uma versão que não obteve sucesso em algum teste seja construída.

Cada desenvolvedor deve lançar suas modificações todos os dias. Quanto mais recorrente são estes lançamentos com suas devidas modificações, menores serão os blocos de alterações para pesquisar em caso de erros de conflito. Estas modificações devem manter o código perfeitamente executável (sem quebra da *build*). Ao realizar esta prática frequentemente os desenvolvedores conseguem também encontrar mais rapidamente conflitos que podem surgir entre versões.

Cada *commit* (conjunto de modificações) deve atualizar o repositório principal em uma máquina de integração. Um desenvolvedor deve considerar uma alteração no repositório finalizada apenas se a *build* for realizada com sucesso após o lançamento das modificações. Para assegurar tal ação, o desenvolvedor pode utilizar uma abordagem com uma *build* manual ou a partir de um servidor de IC.

Manter a *build* rápida. Como o foco da IC é manter o *feedback* ágil, o tempo de execução da *build* deve ser curto, contudo, o maior gargalo se faz nos testes. Deste modo, uma *build* de dois estágios pode ser considerada, uma vez que, o primeiro estágio objetive executar testes importantes assim que são lançadas as modificações, de modo a balancear a necessidade de procurar *bugs* com agilidade, garantindo uma *build* estável e ainda assim confiável, além de oferecer *feedback* instantâneo. O segundo estágio é realizado quando possível, executando testes secundários sobre este conjunto de alterações, resultando em *builds* posteriores mais completas sobre a *build* principal. Em caso de quebra, sua correção não necessita ser tão urgente como no primeiro estágio, no entanto, é preciso corrigir o quanto antes.

Testar em uma cópia do ambiente de produção. O ambiente de testes deve ser configurado o mais próximo possível de uma cópia exata do ambiente de produção. A virtualização é uma saída para dificuldades em relação à custos, tornando mais acessível esta prática.

Todos podem ver o que está acontecendo. A comunicação é essencial para a IC, logo, é importante que seja fácil aos integrantes da equipe acompanhar o estado do *software*. Um painel de informações, digital ou não, alertas luminosos ou sonoros, entre outros, são indicados para sinalizar os resultados ou estados das últimas *builds*, até mesmo do *software* em geral.

Automatize a implantação do sistema. É importante ter como implantar a aplicação dentro de qualquer ambiente facilmente, ou seja, rodar naturalmente em um ambiente de produção como em qualquer outro. Implantações automáticas ajudam a tornar este processo mais rápido além de reduzir os erros.

Por conseguinte, o presente trabalho buscou utilizar destes princípios. Estes foram abstraídos juntamente com as ideias contidas na discussão sobre os aspectos automatizados da IC (DUVALL; MATYAS; GLOVER, 2007), que utiliza ferramentas que concedem suporte à automação, mesmo que, este processo não necessariamente precise ser, de fato, automatizado para se caracterizar como uma IC, ou seja, utilizando de uma abordagem manual para integração, uma vez que, seja utilizada uma *build* automatizada.

2.4 Trabalhos Correlatos

Existem na literatura diversos trabalhos que retratam casos de implantação e estudos da IC. Estes diferem entre si nas ferramentas utilizadas, tipos de projeto, interpretação, processo de implementação da prática e até mesmo nos resultados obtidos (STÅHL; BOSCH, 2014b).

Miller (2008), expõe uma experiência de aproximadamente cem dias úteis com a IC no desenvolvimento do *Service Factory*, um pequeno projeto executado pelo grupo de padrões e práticas da Microsoft, contando com uma equipe distribuída com cerca de doze pessoas. A natureza e o impacto das quebras de *builds* no projeto foram investigadas mostrando que grande parte são com relação à análise estática de código fonte, testes unitários, compilação e falhas no servidor, no caso, hospedado em uma máquina virtual. Também foi observado que a grande maioria das quebras de *builds* foram corrigidas em um tempo médio de 42 minutos, excluindo *builds* noturnas. O custo (tempo) real de se usar a abordagem de IC em seu projeto, foi pelo menos 40% menor que o custo hipotético não utilizando a prática, além de manter o mesmo nível de qualidade do código. Segundo Miller (2008) mesmo que nítidas, comparado com outros projetos maiores, as vantagens da IC são minimizadas para projetos desta escala.

Stolberg (2009) realizou um relato de aprendizado e experiência a partir de uma implementação da prática da IC, motivado pela necessidade de executar testes em um contexto de desenvolvimento ágil, migrando dos métodos tradicionais. Inicialmente sua equipe não considerava qualquer estrutura de automação, contando com um processo de *build* manual, maturando assim, a busca pela prática. O esboço e a escolha das ferramentas de IC foram influenciados pelo que funcionaria e se adequaria ao ambiente de desenvolvimento de aplicações Windows .NET C#. Após a implementação, uma análise foi realizada comparando o antes e depois, relacionado aos princípios da IC citados por Fowler (2006) e a utilização destes pela equipe. Com essa experiência, Stolberg (2009) observou que testes em paralelo com desenvolvimento podem superar a desarticulação tradicional que ocorre normalmente entre ambos. A automação dos testes de aceitação se mostrou intimidante, porém atraente, preocupando-o em relação à sua escalabilidade. A dificuldade com a equipe se fez ao convencê-los passar por algumas mudanças, de modo que permitisse realizar a IC. Stolberg (2009) indica que nos seus próximos projetos, um primeiro passo para realizar testes ágeis seria considerar a implantação de um sistema de IC para suportá-los.

Uma apresentação dos artefatos e recursos oferecidos por algumas ferramentas que auxiliam a realização da IC é feita por Geiss (2012) com foco no Jenkins[1] e no desenvolvimento de aplicações Android. O processo de utilização desta ferramenta para

[1] Ferramenta de automação (servidor de Integração Contínua) - <https://jenkins.io/>

projetos Android é descrito, passando pela representação de como funciona a compilação no Android e sua interface com o Jenkins e seus *plugins*, além de mostrar alguns recursos disponíveis da ferramenta que estão relacionados a um projeto deste tipo. Portanto, sua contribuição é fornecer uma descrição para as demais pesquisas e como se constitui basicamente um processo de IC para o desenvolvimento de aplicações Android.

Grande parte das implementações da IC são desenvolvidas em ambientes da indústria de *software* consolidados, envolvendo profissionais de alta qualificação. Entretanto, Hembrink e Stenberg (2013) buscaram o ambiente acadêmico, implementando a prática em uma das diversas equipes que desenvolviam um mesmo projeto estudantil, uma simulação de um projeto real em Java com características ágeis, no curso de Desenvolvimento de Software em Equipes (EDA260) da Universidade de Lund. Esta equipe foi treinada pelos autores ao longo do projeto a fim de desenvolver a IC, realizando adaptações com intuito de adequar a prática ao ambiente do curso, que contou com apoio do Jenkins. Ao final do projeto foi observado por Hembrink e Stenberg (2013) que, dentre as equipes, esta foi a única que utilizou um servidor de IC, automatizando a *build* e compartilhando o *feedback*. Houve também um avanço na cobertura de testes e baixa proporção de *builds* que apresentaram falha, além do aumento na frequência de *releases*, resultado de versões mais gerenciáveis e evolução da qualidade do código. Dentre os problemas e desafios enfrentados, a falta de conhecimento em relação aos testes por parte dos desenvolvedores e a automação dos testes de aceitação, apresentaram maior destaque.

Uma implementação da prática da IC para simplificar o desenvolvimento de um pacote de *software* científico é descrita por Betz e Walker (2013). Este pacote é constituído de programas de simulação molecular amplamente utilizado nas comunidades de química e biologia molecular computacional, compreendendo diversas características que o difere dos outros tipos de *softwares* de mercado comuns, o que ocasionou a adaptação da prática da IC. O processo de IC foi implementado seguindo os princípios de Fowler (2006), utilizando o servidor de integração Cruise Control[2] para conduzir a *build* e os testes automáticos. Após a implementação da IC, Betz e Walker (2013) constataram benefícios também para este tipo de projeto. O processo de depuração se tornou mais simples. Foi percebido um aumento da colaboração entre os diversos grupos de desenvolvedores e o custo financeiro da implementação foi relativamente baixo. Houve também uma dificuldade relatada em encontrar ferramentas para auxiliar a IC, pois a maioria é projetada para lidar com aplicações tradicionais de engenharia de *software*, por conseguinte, a dificuldade de se adaptar algumas destas ferramentas à um ambiente de desenvolvimento de *software* científico de grande complexidade pode desestimular estes projetos à adotarem a prática.

Através de um estudo de caso realizado por Hukkanen (2015) foi investigado a adoção da IC em um cenário empírico de um projeto de *software* Java modular de

[2] Ferramenta de automação (servidor de Integração Contínua) - <http://cruisecontrol.sourceforge.net>

telecomunicações na Nokia Networks. Com isso, foram levantados pelo autor os desafios na adoção desta prática. Uma vez que o projeto é consideravelmente grande e os *stakeholders* estão distribuídos geograficamente, o tamanho do projeto tornou o esforço de integração e teste complexo. O uso eficiente da prática foi prejudicado por problemas relacionados a testes, infraestrutura, gerenciamento de dependências, comunicações e aspectos práticos da IC. Estes desafios interferem uns com os outros e impactam de maneira negativa na produtividade, que está relacionada com o tempo de *build* e seus resultados, depuração e até mesmo o sistema de IC. Hukkanen (2015) observou em seu trabalho que integrar cada incremento de código instantaneamente pode não ser um estado desejado para todos os projetos, além de que, podem não aparecer melhorias imediatas ao adotar um novo sistema de IC. As práticas de gestão do projeto como arquitetura de *software* ou de teste podem necessitar de uma revisão. Hukkanen (2015) também indica que a maneira como os ambientes técnicos são configurados e mantidos parece ser crucial. Quando os problemas não são introduzidos por um determinado desenvolvedor ou se tem pouco controle, a frustação pode fazer com que estes sejam contornados ou ignorados.

Penson et al. (2017) implementaram e testaram uma plataforma de IC com o apoio da ferramenta Jenkins e um conjunto de *plugins* para desenvolvimento C/C++. Foi elaborada uma estrutura de testes especificamente para Arduinos que interage com o Jenkins, cuidando do gerenciamento destes dispositivos conectados a este servidor de *build* via USB. Esta plataforma foi aplicada a um projeto de desenvolvimento de *software* em curso, envolvendo uma equipe de seis desenvolvedores. Após a aplicação foi observado por Penson et al. (2017) que houve um aumento no número de *commits*, erros não detectados anteriormente foram descobertos, os desenvolvedores se sentiram mais confiantes em realizar mudanças, a produtividade e satisfação obtiveram um ganho, além de demonstrar como o processo de IC e a ferramenta Jenkins podem ser adaptados e expandidos.

3 Desenvolvimento

Este capítulo descreve a metodologia de desenvolvimento e o processo de integração utilizado pela equipe em um cenário anterior e as mudanças realizadas, objetivando melhorar o processo e empregar a IC. Por fim, é descrito detalhadamente a implementação da prática juntamente das técnicas, ferramentas e configurações aplicadas.

3.1 O contexto anterior da metodologia geral de desenvolvimento e integração de código

A metodologia de trabalho da equipe se faz da seguinte maneira. Inicialmente os desenvolvedores têm as suas atividades definidas a partir de histórias de usuários, melhorias propostas ou correção de *bugs*. Após a atribuição das atividades aos respectivos desenvolvedores é realizado o Planning Poker, técnica ágil que tem como objetivo estimar o esforço de cada atividade, buscando prever o seu impacto no cronograma, custos e recursos necessários.

O desenvolvimento de *software* é apoiado a partir das seguintes ferramentas. Android Studio[1], uma IDE para construção de aplicações Android. O Git[2], um sistema de controle de versão *open source* instalado em servidor próprio. O GitLab Comunity Edition[3], um gerenciador *open source* de repositórios Git que também oferece opções de gerenciamento de projetos instalado em servidor próprio e provê interface *web*.

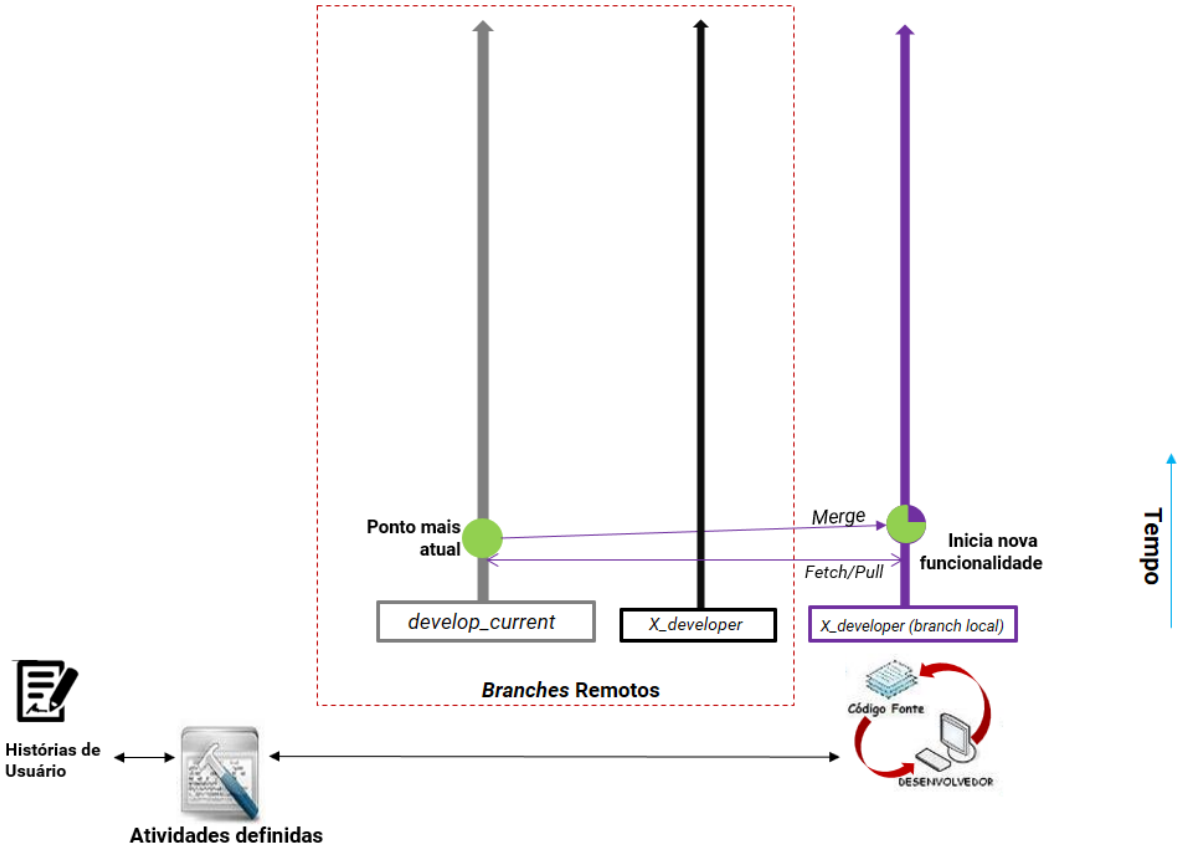
Ao iniciar o desenvolvimento das novas atividades é realizada uma busca no Git (*Fetch/Pull*) procurando por atualizações presentes nos *Branchs*, de preferência no principal, chamado de *develop_current*, alocado remotamente no servidor contendo a base do *software*. Por conseguinte, caso existam alterações, é realizada a junção (*Merge*) deste *Branch* com o *Branch* local do desenvolvedor, alocado em seu computador e que contém suas atividades locais e alterações de código individuais. O *Branch* do desenvolvedor é criado quando ele é incorporado a equipe, constituindo um processo diferente no Git que realiza uma cópia exata (*Clone*) do *Branch* principal, uma vez que o desenvolvedor utiliza somente o mesmo *Branch*, o *Merge* lança as alterações dos outros desenvolvedores que não estavam presentes em seu repositório local, atualizando sua versão local, como ilustrado na Figura 1.

Após concluir sua atividade, o desenvolvedor realiza um *Commit* no seu *Branch* local com o propósito de gravar as suas alterações produzidas no código juntamente com

[1] <https://developer.android.com/studio/index.html>
[2] <https://git-scm.com/>
[3] <https://about.gitlab.com/>

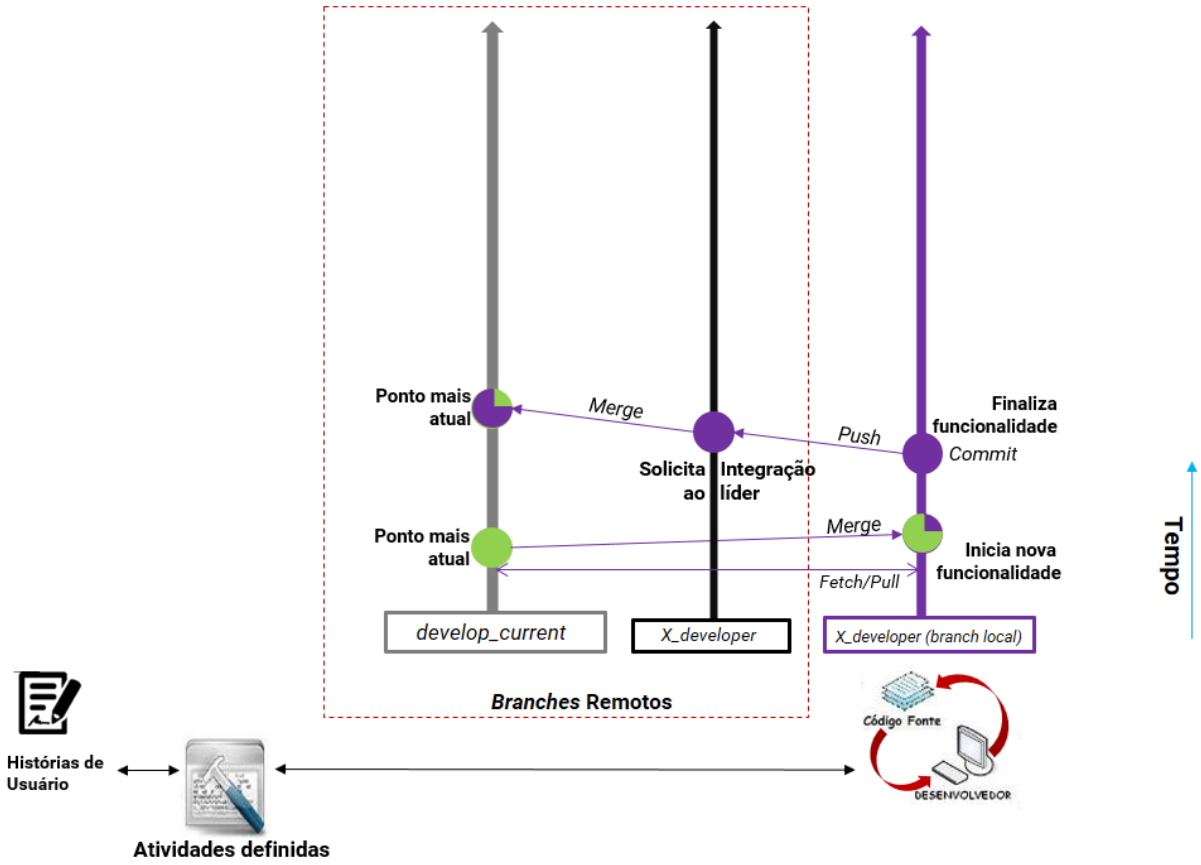
a descrição destas alterações, em seguida, para enviar suas modificações realizadas na versão local para o seu *Branch* remoto, o desenvolvedor executa o comando *Push* no Git. Neste determinado momento o desenvolvedor realiza um pedido informal ao líder para que integre o código do seu *Branch* remoto com o *develop_current*, como ilustrado na Figura 2, dando início ao processo de integração do código.

Figura 1 – Metodologia geral de desenvolvimento: Busca por alterações no *Branch develop_current*.



Fonte: Elaborado pelo autor.

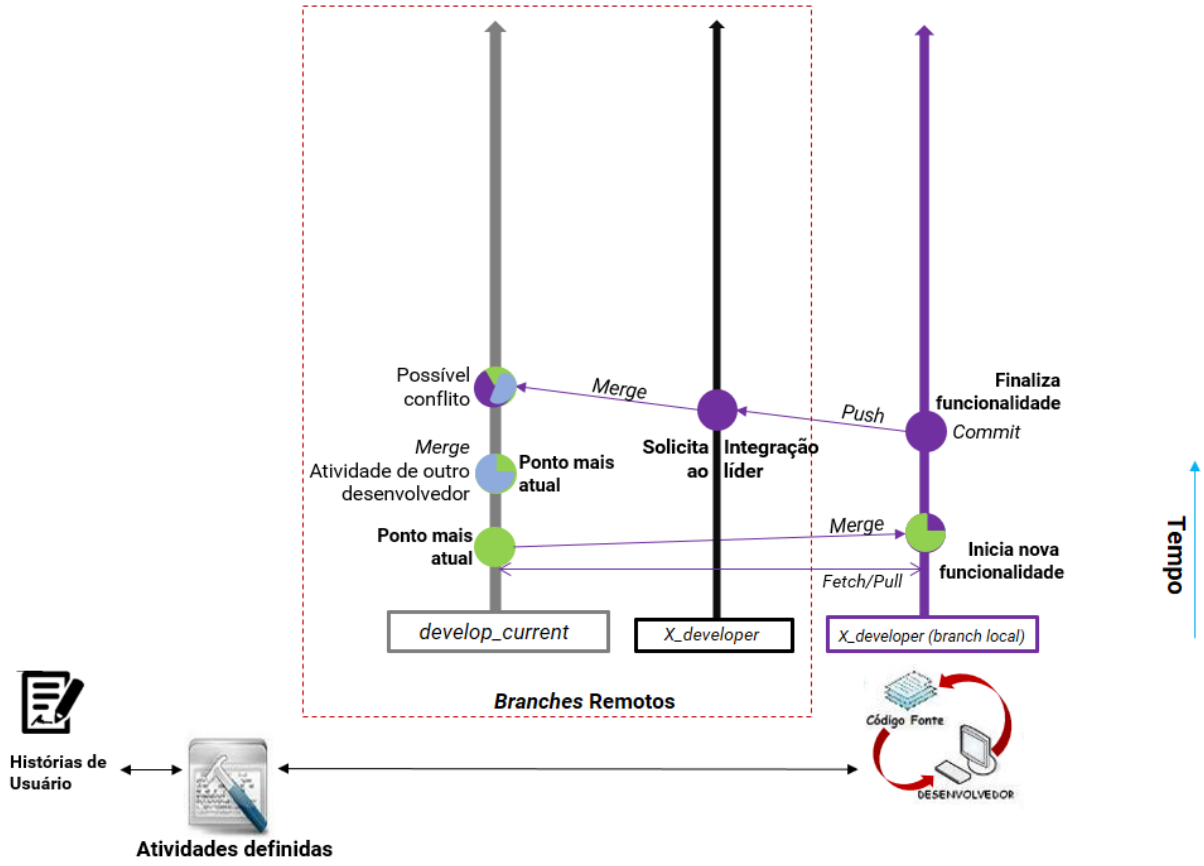
Figura 2 – Metodologia geral de desenvolvimento: Lançamento das alterações



Fonte: Elaborado pelo autor.

Nesse processo de junção dos *Branches* surgem muitos problemas de conflito de código, uma vez que os desenvolvedores em raras ocasiões checavam se haviam alterações no *Branch develop_current* antes de enviar suas mudanças para o seu *Branch* remoto. Deste modo, modificações realizadas na mesma parte do código, possivelmente alteradas recentemente por outro desenvolvedor, podem ser lançadas utilizando de uma versão mais antiga em seu *Branch* local, ignorando as alterações recentes. Como é possível observar na Figura 3, as buscas por alterações se davam apenas ao iniciar a atividade e raramente ao lançar suas modificações, o que poderia ocasionar diversos conflitos.

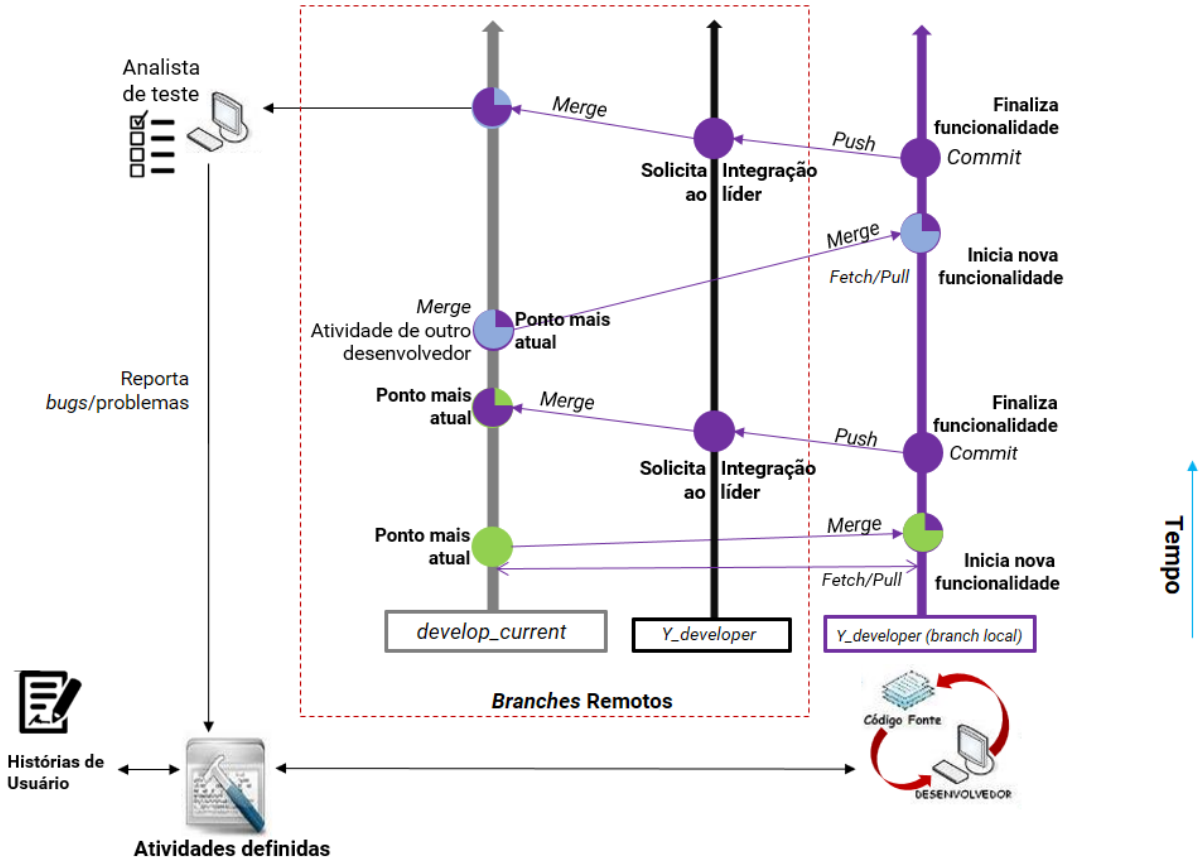
Figura 3 – Metodologia geral de desenvolvimento: Possível cenário de conflito



Fonte: Elaborado pelo autor.

Essa tarefa de integração requer esforço e tempo, pois ao encontrar conflitos entre as versões, o líder da equipe realiza uma auditoria com os autores das respectivas alterações dos trechos de código, comparando-as e analisando suas consequências. No final da semana, mais especificamente nas sextas-feiras, são construídas versões contendo as alterações que surgiram durante a semana. Estas versões são apuradas pelo analista de teste em busca de *bugs* e inconsistências. Durante todo esse processo de integração descrito, *bugs* podem surgir e também passarem despercebidos ao aceitar ou rejeitar conflitos, que por sua vez, provavelmente apenas serão descobertos e reportados junto aos resultados do analista de testes, ilustrado na Figura 4.

Figura 4 – Metodologia geral de desenvolvimento: Análise da versão



Fonte: Elaborado pelo autor.

3.2 Implantando a IC

Visto que o projeto se encontra em pleno desenvolvimento e por se tratar de um projeto real, foram necessários alguns cuidados ao tentar implantar a prática. Pontos como a cultura de desenvolvimento da equipe, as ferramentas utilizadas e suas práticas de testes, foram observados e tratados de maneira que seu impacto fosse anteriormente mensurado e que não prejudicasse o andamento do projeto.

3.2.1 Planejamento

De acordo com as características do projeto, foram realizadas pesquisas e estudos comparativos de ferramentas que melhor se adequariam ao ambiente de desenvolvimento de uma aplicação Android. Mesmo as ferramentas que já eram utilizadas pela equipe como

o Android Studio, Git e Gradle[4] (automação da *build*), foram comparadas com outras respectivas ferramentas de mesma natureza como a IDE Eclipse[5], o sistema de controle de versão Subversion[6] e o Ant[7] para automação de *build*. Porém, em diversas características, as ferramentas utilizadas atualmente se mostraram ótimas, além de manterem uma boa harmonia de funcionamento e conhecimento pela equipe. O Android Studio, por exemplo, é atualmente a IDE oficial para desenvolvimento Android e utiliza nativamente o Gradle para realizar a *build* da aplicação.

Como servidor de IC foi escolhido o Jenkins após comparações com outros servidores como CruiseControl[8], Travis CI[9], TeamCity[10]. Foram consideradas características como popularidade, situando o Jenkins como o servidor utilizado por quase dois a cada três entrevistados (MAPLE; SHELAJEV, 2016), dispor de licença *open source*, compatibilidade com o Git e o ambiente Android, além de ter grande extensibilidade (POLKHOVSKIY, 2016), o que proporciona adicionar *plugins* para utilizar, por exemplo, a ferramenta de automação de *build* Gradle.

A partir da definição das ferramentas foi construído um ambiente de simulação de integração para um projeto fictício, utilizando de características presentes no projeto *MobMine*. Este ambiente simulado teve como objetivo obter um primeiro contato com o Jenkins, os conceitos da IC e analisar o funcionamento deste sistema.

Após tomar conhecimento sobre o contexto anterior da metodologia empregada pela equipe, foi realizada uma análise, descrita a seguir, considerando os conceitos e os princípios da IC apresentados por Fowler (2006). Em alguns pontos a equipe se mostrou perto de obter uma certa vantagem por já empregá-los, mesmo que em alguns casos não corretamente ou da melhor maneira, mas que acabou tornando um pouco mais fácil a abstração pela equipe. Outros pontos simplesmente não se aplicam ao contexto, como a necessidade de cada desenvolvedor lançar suas modificações todos os dias, uma vez que, não é uma regra os desenvolvedores dedicarem todos os dias ao projeto. Dado a sua carga horária esperada de apenas quinze horas por semana, há dias em que os desenvolvedores não produzem alterações no *software*, portanto, este princípio não pôde ser aplicado, mas foi realizada uma adaptação, uma vez que, os desenvolvedores lançavam suas modificações apenas quando uma funcionalidade era finalizada.

A equipe já mantinha um único repositório de código, contando com o sistema de versionamento Git, porém este não era utilizado da melhor maneira. *Commits* e *Merges* eram realizados apenas ao finalizar uma funcionalidade antes de enviar as modificações

[4] <https://gradle.org/>
[5] <https://eclipse.org/>
[6] <https://subversion.apache.org/>
[7] <http://ant.apache.org/>
[8] <http://cruisecontrol.sourceforge.net/>
[9] <https://travis-ci.org/>
[10] <https://www.jetbrains.com/teamcity/>

para o *Branch* remoto do desenvolvedor, conduzindo-se antemão ao princípio de realizar *Commits* a cada pequena modificação e *Merges* com pequenos conjuntos destas alterações, tornando mais fácil a depuração e o próprio acompanhamento das modificações. A busca por atualizações no *Branch* principal *develop_current* era pouco aplicada, muito pelo esquecimento dos desenvolvedores.

A *build* era automatizada a partir do Gradle presente no Android Studio, porém é essencial que essa *build* fosse realizada em um ambiente diferente do de produção dos desenvolvedores, como por exemplo, um servidor de integração, evitando dependências de configurações locais. A equipe possui uma suíte de testes de aceitação automatizados, porém estes eram executados somente após a separação de uma *release* pelo líder e rodavam em um ambiente de produção local na máquina do analista de testes, o mesmo utilizado para a criação destes testes.

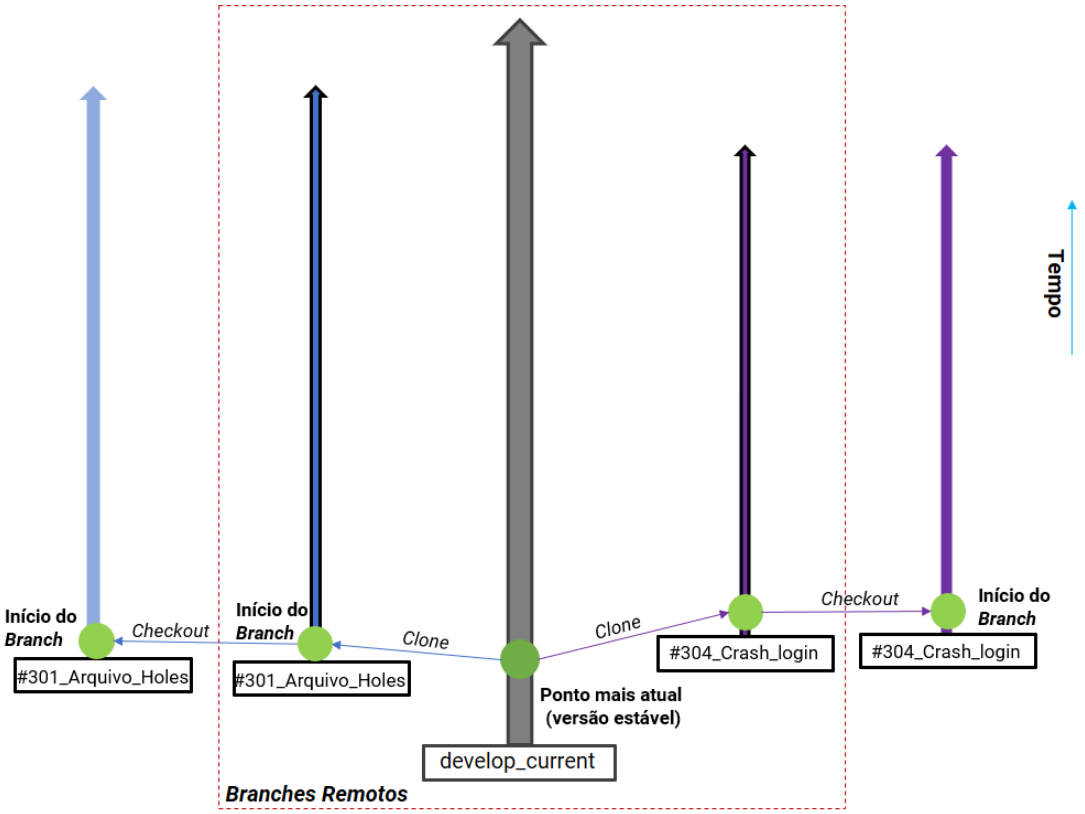
Os testes e o desenvolvimento contam com a utilização de *tablets* com configurações similares ou idênticas aos utilizados no ambiente de operação. Este é um ponto positivo ao executar os testes pois é possível contar com o *hardware* alvo.

3.2.2 Melhorias propostas empregadas

Com base na análise anterior e objetivando empregar os demais princípios e conceitos da IC por Fowler (2006), foram propostas à equipe algumas melhorias quanto a sua metodologia de desenvolvimento. As propostas empregadas são descritas a seguir.

Um novo conceito de *Branches* temporários foi proposto e empregado pela equipe. Estes *Branches* são relacionados a cada atividade (*Issue*) gerada no GitLab, geralmente atividades que envolvem alterações de código fonte, caracterizadas com as tags "Funcionalidade", "Melhoria" ou "*Bug*". Os nomes destes *Branches* seguem o padrão código da *Issue* juntamente do seu nome. Esse *Branch* pode ser criado facilmente pelo próprio GitLab por meio da opção de criar um *Branch* para determinada *Issue*, a partir do *Branch* principal *develop_current*. Ao criar o novo *Branch*, o Git realiza uma cópia exata (*Clone*) do *develop_current* para o *Branch* remoto da respectiva atividade, posteriormente copiado (*Checkout*) para um *Branch* local pelo desenvolvedor, conforme ilustrado na Figura 5.

Figura 5 – Metodologia e disposição dos *Branchs*: *Branchs* temporários

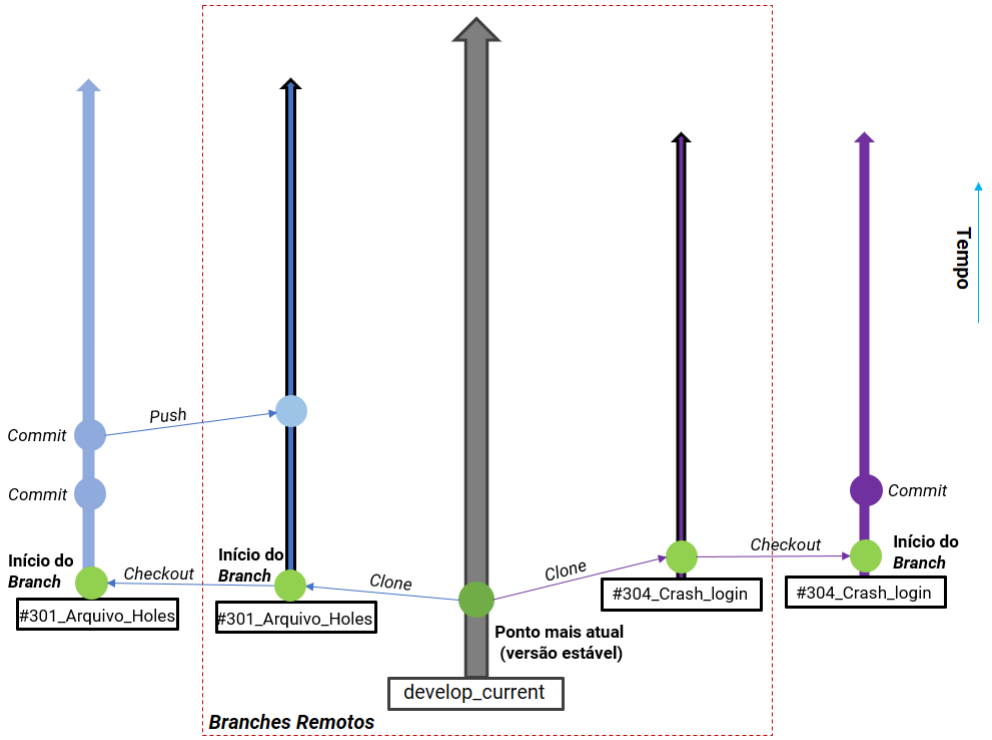


Fonte: Elaborado pelo autor.

Geralmente, cada desenvolvedor trabalha em apenas uma atividade o que mantém cerca de oito *Branches* ativos. Isto é possível devido a exclusão dos respectivos *Branches* assim que a atividade é validada, visto na Figura 8. O *Branch* denominado *develop_current* é visto como um ponto central para que todos lancem suas mudanças caracterizando a integração com o código da aplicação. Deste modo, esse *Branch* será monitorado pelo servidor de IC.

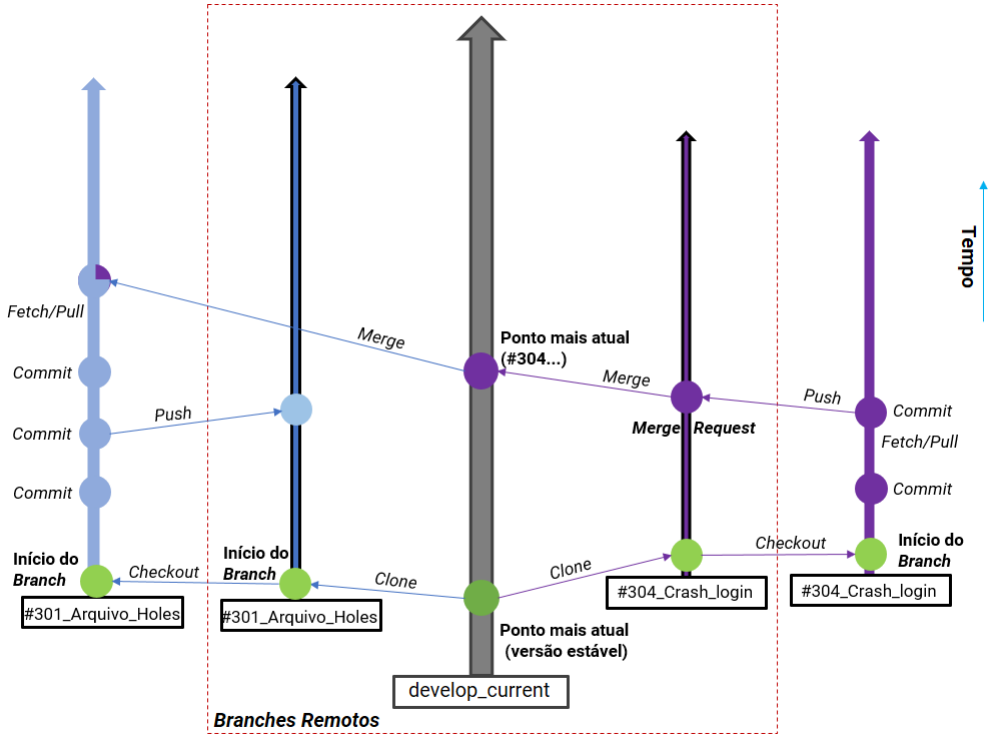
Os desenvolvedores também foram instruídos a realizar *Commits* a cada pequeno conjunto de mudanças em uma classe ou método do *software* Figura 6, além de lançar essas modifcações também para o seu *Branch* remoto de modo a evitar problemas como, por exemplo, a perda destas mudanças devido ao mal funcionamento do *hardware*. Foi introduzido também conceito de *Merge Request*, no qual o desenvolvedor com intenção de lançar modificações no *Branch develop_current* abre uma requisição de *Merge* no GitLab e desta maneira os demais desenvolvedores podem avaliar estas mudanças para somente depois ser aceito o *Merge*. Outro ponto reforçado foi buscar alterações no *Branch develop_current*, pelo menos sempre antes de solicitar um *Merge Request*, e aplicá-las, quando houverem, ao *Branch* da *Issue* em questão, como ilustrado na Figura 7.

Figura 6 – Metodologia e disposição dos *Branchs*: *Commits*



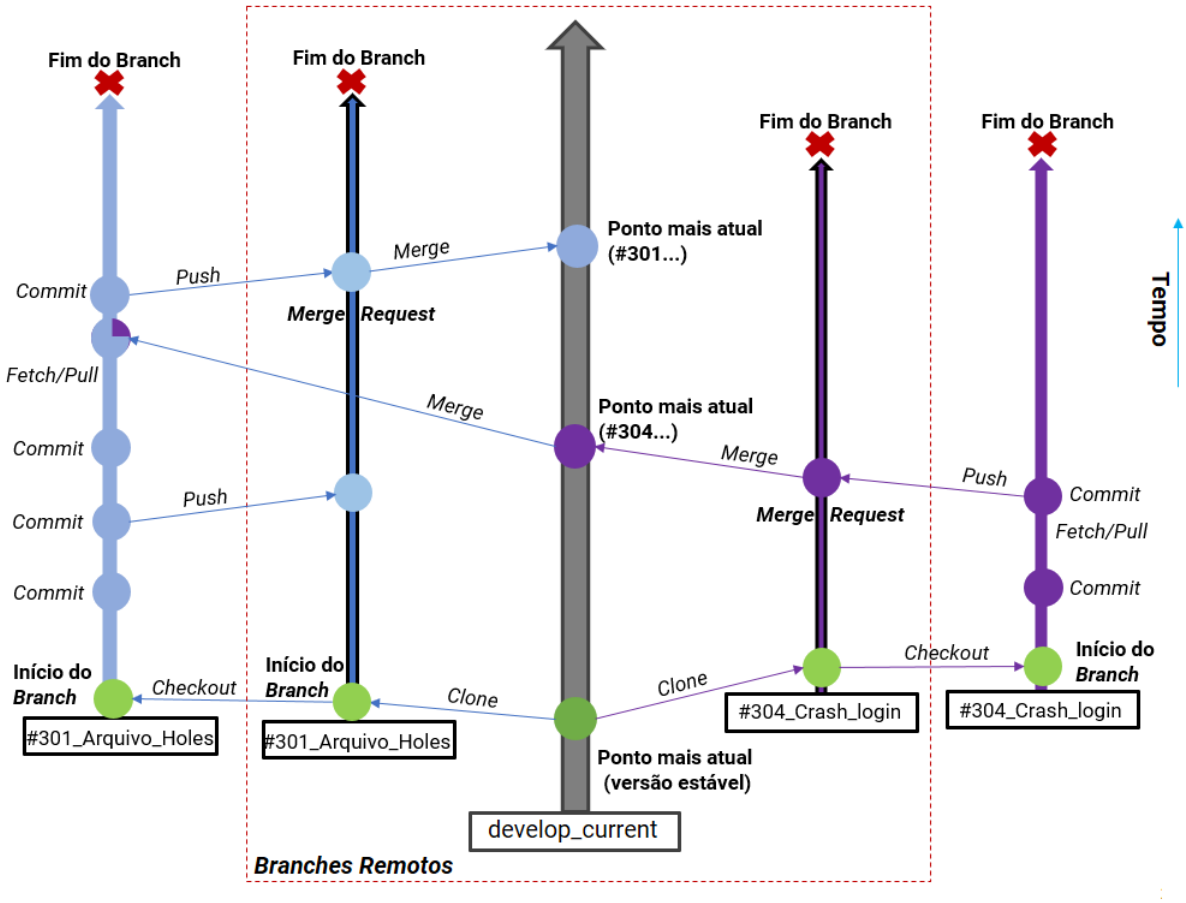
Fonte: Elaborado pelo autor.

Figura 7 – Metodologia e disposição dos *Branchs*: *Merge Request*



Fonte: Elaborado pelo autor.

Figura 8 – Metodologia e disposição dos *Branchs*: Fim dos *Branchs*



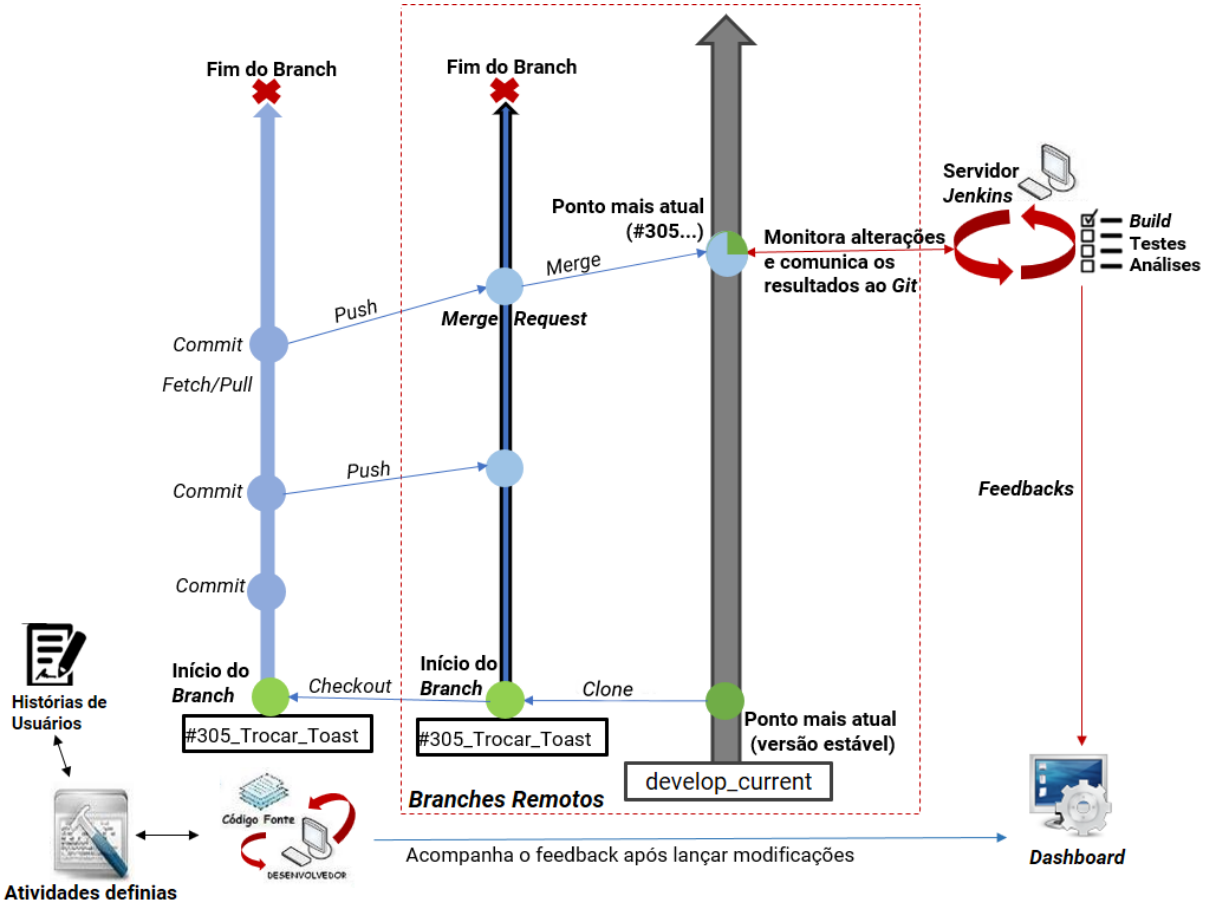
Fonte: Elaborado pelo autor.

O servidor Jenkins foi implantado com o objetivo de alavancar os princípios da IC a serem seguidos pela equipe. Em vista disso, a automação e execução da *build* são realizadas no servidor, um ambiente diferente dos ambientes de desenvolvimento locais. Os testes de aceitação rodam durante o processo de *build* que é iniciado a partir das modificações lançadas no *Branch develop_current* e disponibiliza, ao final, um *feedback* do estado do sistema e as mudanças realizadas. Com a introdução do Jenkins, a metodologia geral de desenvolvimento da equipe passou por modificações, ilustradas na Figura 9.

Ao finalizar o processo de *build*, são gerados *logs* contendo informações detalhadas do processo e dos resultados de compilação, testes, análises estáticas, entre outros. Neste *log*, em caso de quebra da *build*, é possível encontrar os problemas, erros ou inconformidades após a mudança. Além do *log* um *dashboard* apresenta o *feedback* destes resultados. A equipe deve então ficar atenta para estes resultados, que apresentam também informações do membro responsável pela mudança e quando ela ocorreu. Esse desenvolvedor deve então providenciar a correção do problema o quanto antes. Ao persistir o problema ocasionando o atraso na solução, o *Branch develop_current* deve ser revertido para um estado anterior

estável e testar, se houverem, as modificações de outros desenvolvedores, evitando que o projeto sofra uma estagnação. A responsabilidade com a *build*, consequentemente, deve ser de cada desenvolvedor, o qual não pode validar sua atividade até obter o *feedback* da introdução das suas mudanças disponibilizado pelo Jenkins. Os testes, funcionamento do Jenkins e o processo de *build* são retratados na seção a seguir.

Figura 9 – Metodologia de desenvolvimento com o servidor Jenkins



Fonte: Elaborado pelo autor.

3.2.3 Configuração do Ambiente

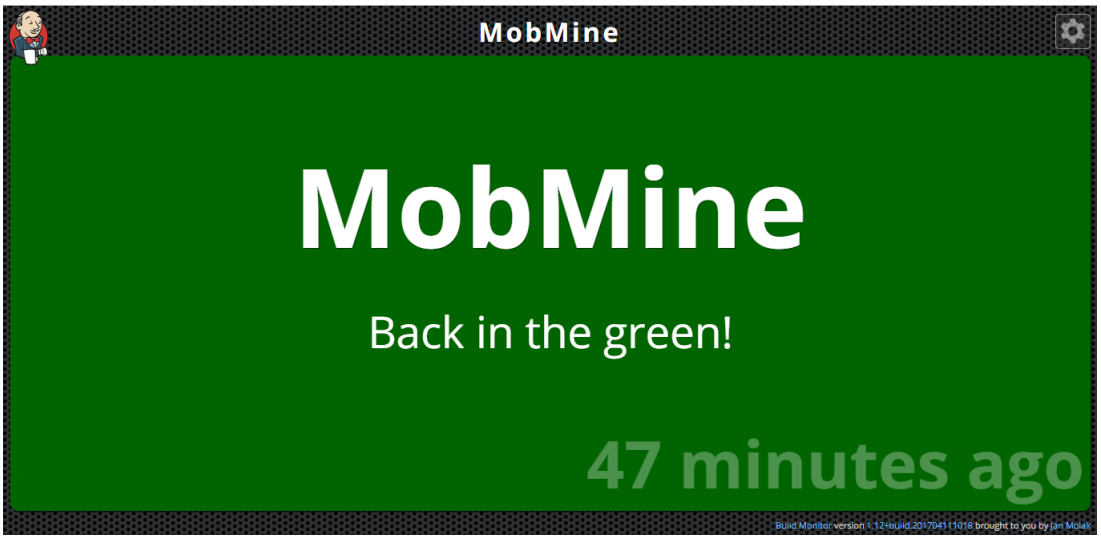
O servidor Jenkins foi instalado em uma máquina com processador Intel Core i5, 4GBs de RAM e sistema operacional Ubuntu16.04/Linux. O Jenkins é um *software* modularizado, o que torna possível estender suas funcionalidades por meio de diversos *plugins* disponíveis na comunidade. Consequentemente, é necessário configurar o Jenkins de acordo com as características de cada projeto utilizando-se também dos *plugins* adequados, desta maneira, para utilizar o Jenkins em um contexto diferente, é necessário realizar a instalação dos plugins e configura-lo de modo a suportar, por exemplo, a linguagem de programação do projeto e ferramentas.

O *MobMine* por se tratar de um projeto de desenvolvimento para a plataforma Android, é necessário a instalação do Android SDK, contendo as ferramentas de desenvolvimento para esse sistema operacional. O caminho da instalação precisa ser referenciado no Jenkins que utiliza destas ferramentas durante o processo de *build*. Da mesma maneira é necessário também o pacote Java Development Kit (JDK) e a instalação do Git, porém estes também podem ser instalados automaticamente dentro das configurações do Jenkins.

Como o Jenkins precisa monitorar o repositório de código fonte e enviar alguns *feedbacks* ao GitLab, é imprescindível a instalação do seu respectivo *plugin*. Este deve ser configurado para que seja possível a efetiva comunicação entre Jenkins e o projeto. Para que o Jenkins possa compilar o projeto utilizando o Gradle, responsável por executar os testes e configurações de compilação da aplicação, também se faz necessário o seu respectivo *plugin*.

Plugins de caráter gerencial também foram instalados, como é o caso do Build Monitor Plugin[11], que fornece uma visão em grande escala do status do projeto, visto que, se estável, é representando pela cor verde, observado na Figura 10, e se houver falha pela cor vermelha, observado na Figura 11. É ideal como um dispositivo de *feedback* visual para ser exibido em uma tela que esteja facilmente visível no laboratório pelos desenvolvedores. O Global Build Stats Plugin[12], é focado em estatísticas de compilação e permite coletar e exibir estatísticas globais de resultados de compilação. É uma ferramenta que informa a tendência global de *builds* no Jenkins ao longo do tempo, como por exemplo, quantos estão acontecendo diariamente e semanalmente, o tempo de duração, entre outros.

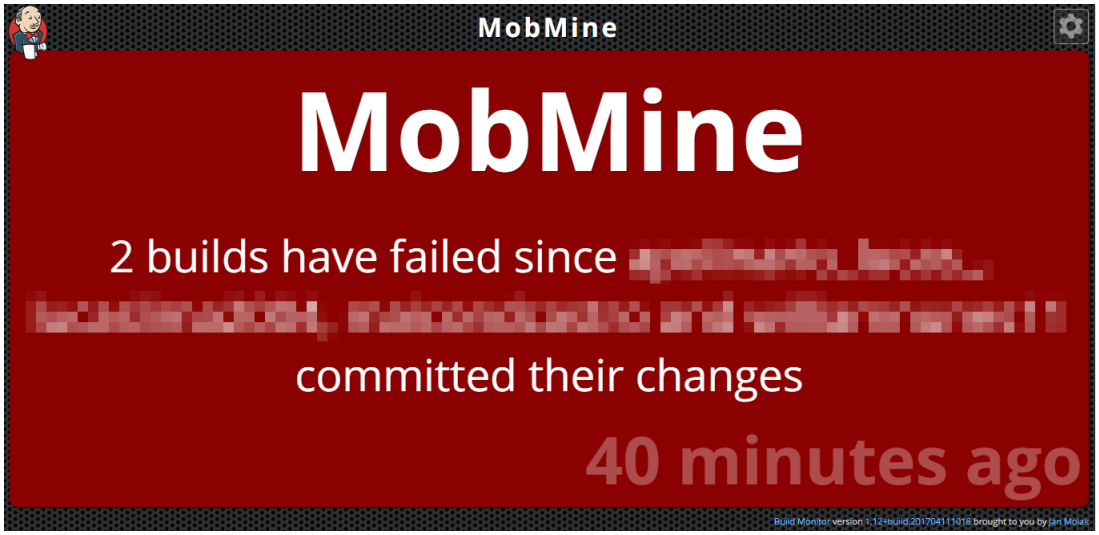
Figura 10 – Layout do monitor quando *build* estável.



Fonte: Tela do Jenkins, Build Monitor Plugin, Capturada pelo autor.

[11] <https://wiki.jenkins.io/display/JENKINS/Build+Monitor+Plugin>
[12] <https://wiki.jenkins.io/display/JENKINS/Global+Build+Stats+Plugin>

Figura 11 – Layout do monitor quando há quebra da *build*.



Fonte: Tela do Jenkins, Build Monitor Plugin, Capturada pelo autor.

Visando a análise estática de código, foi instalado o servidor *open source* SonarQube[13], que provê um relatório das vulnerabilidades, *bugs* e *code smells* presentes no *software*. Um *plugin*[14] disponível no Jenkins permite a integração de ambos.

O processo de *build* é realizado da seguinte maneira. Após o GitLab reportar ao Jenkins um evento de merge no *Branch develop_current*, este realiza uma cópia (*Clone/Checkout*) do *develop_current* para sua área de trabalho, contendo os arquivos do repositório além das informações de controle de versionamento. De posse desses arquivos e informações o Jenkins invoca o Gradle que inicia a compilação dos artefatos da aplicação, executando e configurando testes, como por exemplo, testes unitários, se presentes.

Caso a compilação do Gradle falhe, o processo é finalizado caracterizando a quebra da *build* e modificando o *status* do projeto na *dashboard* do Jenkins. Em caso de sucesso, são gerados dois arquivos de extensão Android Package (APK), um deles é referente ao aplicativo em si, que vai rodar no sistema Android e ser utilizado pelo usuário, já o outro arquivo é referente aos testes de aceitação desenvolvidos pelo analista de teste.

Seguindo o processo, um *script* é invocado para rodar estes testes de aceitação. Neste *script* é inicializada a interface Android Debug Bridge (ADB), uma ferramenta do pacote Android SDK, que permite a comunicação com uma instância de emulador ou dispositivo Android físico conectado ao computador, no cenário do projeto, um *tablet* Samsung Galaxy Tab A. Com o ADB é possível ter controle sobre o sistema Android presente no *tablet* a partir de linha de comando. Após reconhecer os dispositivos conectados, o *script* instala os dois APKs e inicia a execução dos cenários de teste, apresentando ao

[13] <https://www.sonarqube.org/>
[14] <https://wiki.jenkins.io/display/JENKINS/SonarQube+plugin>

final quantos deles obtiveram sucesso e quantos falharam. Em caso de cenários com falha o processo de *build* é quebrado.

Ao obter sucesso em todos os testes, a próxima etapa executa uma análise estática do código realizada pelo Sonar Qube. Caso não exista nenhuma inconformidade durante a análise, como por exemplo, problemas de comunicação entre Jenkins e o Sonar Qube, o processo é finalizado com sucesso e o *status* do projeto é atualizado na *dashboard* do Jenkins.

As *builds* realizadas são organizadas em suas respectivas seções dentro do projeto no Jenkins, cada seção contém todos os detalhes da *build*, tais como o evento responsável por iniciar sua execução, a data e hora, as modificações que foram aplicadas ao repositório juntamente da descrição dos respectivos *Commits*, o *log* de todo o processo, os artefatos gerados (APKs, documentos), os arquivos do projeto e um atalho para os resultados da análise do Sonar Qube.

4 Resultados

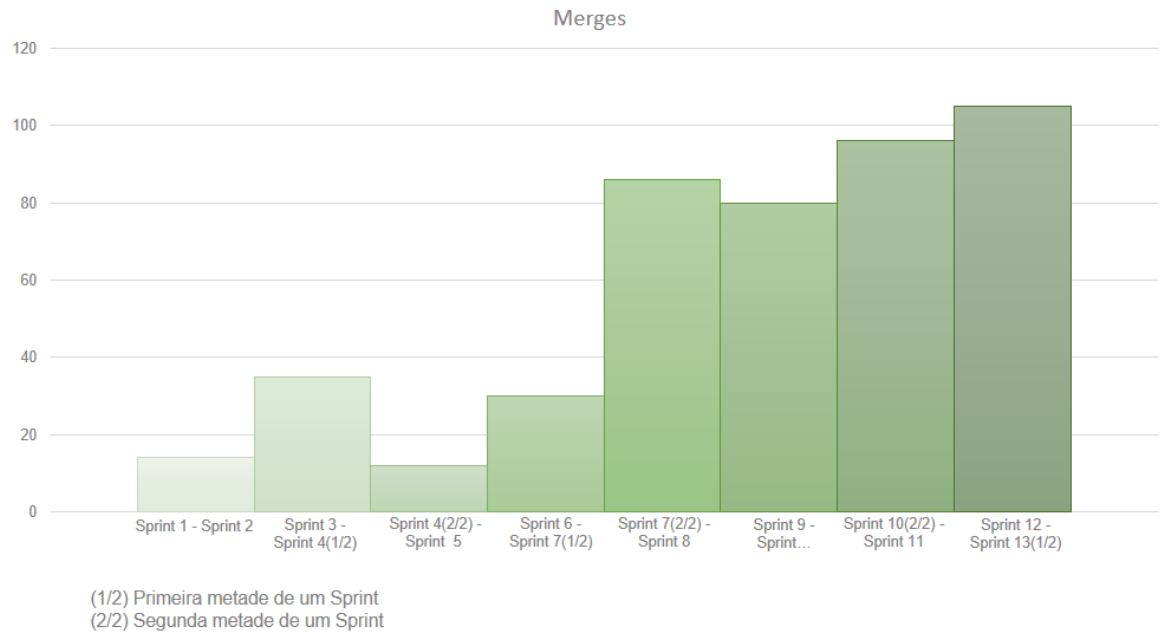
A introdução da IC na equipe teve o seu início na *sprint* doze, considerando o acumulado de *sprints* do projeto, cada uma com duração média de quatro semanas. Contou com a apresentação da proposta de melhorias no processo e da metodologia de *Branches* utilizada pela equipe, na qual ocorreram as primeiras mudanças. O servidor de IC Jenkins teve suas atividades iniciadas, de fato, no início da *sprint* treze. Os dados e opiniões coletados correspondem ao período desde a sua introdução até os dias atuais, caracterizando o decorrer de uma *sprint* e meia, cerca de sete semanas, apresentando resultados promissores apresentados a seguir.

Além dos dados coletados a partir do histórico de registros, gerados frequentemente durante cada *sprint*, e dos dados do projeto disponíveis no GitLab, um questionário dividido em duas etapas foi aplicado aos desenvolvedores integrantes da equipe. A primeira etapa foi aplicada pouco antes do início da introdução da prática e a segunda em meados da *sprint* treze. O objetivo com este questionário foi coletar as opiniões e a percepção dos desenvolvedores quanto aos impactos da implementação da IC na cultura de trabalho, abordando questões relacionadas também à avaliação própria dos desenvolvedores quanto o conhecimento e aplicação da metodologia de trabalho.

Com a adoção das melhorias relacionadas ao sistema de controle de versão, foi observado que a quantidade de *Commits* realizados pela equipe se manteve acima da média, caracterizando cerca de 16,6% do total de *Commits* realizados desde o inicio da utilização do Git no projeto, uma vez que a média é de 12% considerando uma *sprint* e meia. A quantidade de *Merges* demonstrou evolução neste período, caracterizando cerca de 22,5% do total realizados, ficando acima da média de, também, 12% a cada uma *sprint* e meia, apresentado na Figura 12. Um fator importante é a *sprint* treze acontecer em um período de final de semestre, o que implica na redução da produtividade e dedicação de alguns desenvolvedores ao projeto *MobMine*, apresentando como justificativa a grande carga de provas e trabalhos acadêmicos neste período.

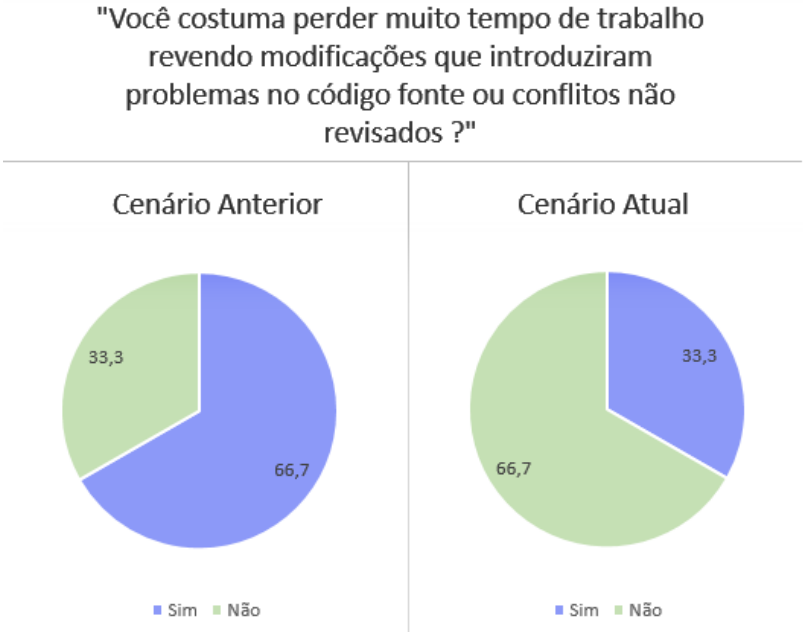
Os resultados desta evolução foram percebidos similarmente pelos desenvolvedores, que responderam esbarrar menos com problemas de conflito de *Merge*, além da redução do tempo para resolvê-los revendo as modificações que introduziram estes problemas, como visto na relação da Figura 13. Eles também procuram mais por atualizações no *Branch develop_current*, pelo menos, sempre antes de enviar uma solicitação de *Merge* para o *develop_current*. Este fator pode estar também relacionado à pequena melhoria no hábito em seguir a metodologia de desenvolvimento, revelado pela opinião dos próprios desenvolvedores quando argumentados a respeito no questionário.

Figura 12 – Atividade do projeto em quantidade de *Merges* relacionando o período de análise (*sprint* doze até metade da *sprint* treze) aos períodos anteriores



Fonte: Dados do Projeto.

Figura 13 – Respostas com relação ao tempo gasto em soluções de conflitos e problemas de integração



Fonte: Dados dos Questionários.

Dentre os desenvolvedores, cinco dos seis perceberam progressos iniciais com a implementação da IC. Os pontos destacados foram a automação e as funcionalidades sendo relacionadas aos *Branches*, tornando-as mais modulares e facilitando a detecção e depuração dos *bugs*, que podem surgir em partes anteriormente estáveis do *software*. Argumentados sobre a opinião quanto ao potencial futuro da adoção da prática pela equipe, quatro dos seis avaliaram que esta pode ser de grande importância no decorrer do desenvolvimento demonstrando também grande nível de entusiasmo quanto a continuidade e aperfeiçoamento da prática. Com relação aos aprimoramentos que poderiam ser alavancados para com a IC, foram relacionados pelos desenvolvedores, a cobertura e novos tipos de testes, a geração de relatórios de testes automatizados mais organizados e amadurecimento da equipe com relação em seguir os princípios adotados.

Após a implantação do servidor Jenkins foram realizadas algumas *builds* disparadas com a realização de integração de código no *develop_current* e também de maneira manual. O tempo de duração médio de uma *build* completa atingido foi de cinquenta e cinco minutos, realizando apenas de quatro a seis *builds* durante toda semana. Cerca de nove minutos do processo são dedicados à análise estática realizado pelo SonarQube, que pode ser amenizado com um *hardware* mais potente. Cerca de quarenta minutos foram dedicados aos testes de aceitação.

A primeira *build* não contou com a presença dos testes de aceitação, ocorrendo com sucesso. Com a introdução dos testes ao processo, o que se teve foi uma sucessão de *builds* quebradas devido à instabilidade de alguns cenários de teste além de artefatos e arquivos necessários que estavam disponíveis apenas no ambiente do analista de teste. Três quebras de *build* resultaram da detecção de *bugs*, duas delas por meio dos testes e uma no processo de compilação, estes foram resolvidos rapidamente com baixo custo de dedicação. Demais quebras, em sua grande parte, foram resultadas por problemas de conexão com a rede ou com o *tablet*. Depois de várias *builds*, apenas nos dois ultimos dias do periodo de análise de *builds* de duas semanas, com início a partir da instalação do servidor Jenkins na *sprint* treze, foi obtido estabilidade nas *builds* caracterizando uma sequência de *builds* com sucesso.

5 Conclusão

Este trabalho implantou a IC no projeto de pesquisa acadêmico *MobMine*, com resultado inicial satisfatório observado pelas respostas da equipe de desenvolvimento por meio do questionário realizado e considerando a evolução do número de *Merges* observados no projeto, demonstrando a curto prazo, impactos positivos, embora desde a maturação da ideia em implantar a IC no projeto MobMine, inúmeros desafios tenham surgido durante o curso deste trabalho. A inexperiência anterior do autor em relação à IC, o aprendizado em conjunto com a equipe e orientador surgindo à medida em que se evoluíam, caracterizaram fatores de acréscimo no nível de complexidade da realização do trabalho.

O nível disciplinar mediano da equipe em seguir o processo regularmente, auto avaliado pelos próprios desenvolvedores no questionário, foi um grande desafio, principalmente ao propor modificações em uma rotina que era confortável e familiar para a maioria dos desenvolvedores. A falta de experiência dos desenvolvedores em projetos dessa magnitude pode ser o reflexo dessa dificuldade em seguir adequadamente tais metodologias. Essa mudança na cultura do processo de integração e controle de versão vem acontecendo aos poucos, a animação da equipe para com a IC é um fator que favorece a continuidade desta evolução. Inicialmente o líder de desenvolvimento ainda continua avaliando e sendo o único com autonomia para aceitar as solicitações de integração (*Merge Request*) com o *Branch develop_current*, porém com o amadurecimento da aplicação da IC pela equipe, a aprovação da solicitação tende à ser do próprio desenvolvedor que a fez, conforme avaliação dos demais.

A configuração do ambiente apresentou contratempos em alguns pontos. A utilização de um Android Virtual Device (AVD), foi a primeira opção para executar a aplicação e seus testes de aceitação durante a *build*. O AVD demostrou ser a solução mais fácil e rápida, uma vez que, o Jenkins disponibiliza um *plugin* que auxilia inteiramente na sua utilização. Porém problemas no gerenciamento da tecnologia de Hardware Acceleration do Android no Linux e na execução do *plugin* do Jenkins ao realizar a comunicação entre um dispositivo virtual emulado com o ADB consumiram considerável tempo para serem solucionados. Por fim as limitações de performance do servidor, não obtendo um bom desempenho ao executar o dispositivo virtual e rodar a aplicação com os testes, inviabilizaram esta opção. Como consequência, a saída foi utilizar um dos dispositivos físicos disponíveis para os desenvolvedores, o que favorece o princípio de executar a aplicação no seu ambiente alvo, mas torna a utilização do dispositivo limitada apenas para este fim.

O elevado tempo de duração da *build*, se caracterizou como um ponto negativo e um desafio, o que pode gerar um cenário de *builds* em fila aguardando para serem executadas

pelo Jenkins ocasionando atrasos no *feedback* ao lançar alguma mudança no repositório, porém não provoca grandes efeitos colaterais neste projeto, dado as características do projeto como o tamanho da equipe e dedicação esperadas dos desenvolvedores. Em um contexto futuro com a adição de cenários de testes em conjunto da evolução do *software*, pode ser considerado a adoção de *builds* noturnas (FOWLER, 2006) que realizem testes completos em toda extensão do *software*, deixando com que apenas um seleto conjunto de testes, geralmente os relacionados com funcionalidades essenciais no *software*, execute no ato da integração.

Todos estes desafios encontrados também influenciaram no curto período de tempo de análise de resultados a partir da implantação do servidor Jenkins, diminuindo a abrangência temporal dos dados referentes aos impactos da utilização da IC no projeto. O estudo destes dados terá prosseguimento em busca de acompanhar os impactos a longo prazo.

Com a implantação da prática de IC, o próximo passo será aprimorar o processo de modo a conseguir um elevado nível de maturidade da equipe para realizar a implantação da Entrega Contínua, com o objetivo de diminuir as lacunas de tempo entre as liberações de versões para o ambiente de operação (HUMBLE; FARLEY, 2014), garantindo, em um cenário ideal, que versões possam ser entregues a qualquer instante de maneira confiável.

Referências

BECK, K. et al. Manifesto for agile software development. 2001. Citado na página 19.

BETZ, R. M.; WALKER, R. C. Implementing continuous integration software in an established computational chemistry software package. In: *Software Engineering for Computational Science and Engineering (SE-CSE), 2013 5th International Workshop on*. San Francisco, CA, USA: IEEE Computer Society, 2013. p. 68–74. Citado na página 23.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. *Continuous integration: improving software quality and reducing risk*. Upper Saddle River, NJ: Addison-Wesley Professional, 2007. Citado 2 vezes nas páginas 16 e 21.

FOWLER, M. Continuous integration. 2006. Disponível em: <https://martinfowler.com/articles/continuousIntegration.html>. Acesso em: 23 set. 2016. Citado 9 vezes nas páginas 16, 17, 19, 20, 22, 23, 30, 31 e 43.

GEISS, M. Continuous integration and testing for android. *Berlin Institute of Technology (TU-Berlin)*, 2012. Citado na página 22.

GIARDINO, C. et al. What do we know about software development in startups? *IEEE software*, IEEE Computer Society, v. 31, n. 5, p. 28–32, 2014. Citado na página 15.

HEMBRINK, J.; STENBERG, P. Continuous integration with jenkins. *Coaching of Programming Teams (EDA 270), Faculty of Engineering, Lund University, LTH*, 2013. Citado na página 23.

HUKKANEN, L. *Adopting Continuous Integration - A Case Study*. Dissertação (Mestrado) — Aalto University, School of Science, Espoo, 2015. Citado 2 vezes nas páginas 23 e 24.

HUMBLE, J.; FARLEY, D. Entrega contínua: Como entregar software. Bookman Editora, 2014. Citado na página 43.

MAPLE, S.; SHELAJEV, O. *Java Tools and Technologies Landscape Report 2016*. 2016. Disponível em: <https://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016>. Acesso em: dez. 2016. Citado na página 30.

MATHARU, G. S. et al. Empirical study of agile software development methodologies: A comparative analysis. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 40, n. 1, p. 1–6, 2015. Citado na página 15.

MILLER, A. A hundred days of continuous integration. In: *Agile Conference, 2008. AGILE'08*. Toronto, ON, Canada: IEEE Computer Society, 2008. p. 289–293. Citado na página 22.

PENSON, W. et al. Continuous integration platform for arduino embedded software. In: *Electrical and Computer Engineering (CCECE), 2017 IEEE 30th Canadian Conference on*. Windsor, ON, Canada: IEEE Computer Society, 2017. p. 1–4. Citado na página 24.

PEREIRA, I. M.; CARNEIRO, T. G. de S.; PEREIRA, R. R. Developing innovative software in brazilian public universities: tailoring agile processes to the reality of research and development laboratories. *Proceedings of the 4th Annual Conference on Software Engineering and Applications (SEA 2013)*, Thailand, Bangkok, 2013. Citado na página 16.

POLKHOVSKIY, D. *Comparison between Continuous Integration tools*. Dissertação (Mestrado) — Tampere University of Technology, Tampere, 2016. Citado na página 30.

PRESMAN, R. S. *Engenharia de Software: Uma abordagem Profissional*. 7. ed. São Paulo: Pearson Makron Books, 2011. 52-106 p. Citado 2 vezes nas páginas 15 e 19.

PRESMAN, R. S. *Engenharia de Software: Uma abordagem Profissional*. 7. ed. São Paulo: Pearson Makron Books, 2011. 514-537 p. Citado na página 18.

PRIKLADNICKI, R.; WILLI, R.; MILANI, F. *Métodos ágeis para desenvolvimento de software*. Porto Alegre: Bookman Editora, 2014. 3-15 p. Citado 2 vezes nas páginas 15 e 19.

QURESHI, M. R. J. Agile software development methodology for medium and large projects. *IET software*, IET, v. 6, n. 4, p. 358–363, 2012. Citado na página 19.

REZENDE, D. A. *Engenharia de software e sistemas de informação*. 3. ed. [S.l.]: Brasport, 2005. 1-19 p. Citado na página 16.

RODRIGUES, N. N.; ESTRELA, N. V. Simple way: Ensino e aprendizagem de engenharia de software aplicada através de ambiente e projetos reais. *Anais do VIII Simpósio Brasileiro de Sistemas de Informação*, São Paulo, 2012. Citado na página 16.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson Education, 2011. 475-492 p. Citado na página 18.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson Education, 2011. 18-53 p. Citado na página 19.

STÅHL, D.; BOSCH, J. Continuous integration flows. In: *Continuous software engineering*. Switzerland: Springer, 2014. p. 107–115. Citado na página 20.

STÅHL, D.; BOSCH, J. Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, Elsevier, v. 87, p. 48–59, 2014. Citado 2 vezes nas páginas 20 e 22.

STOLBERG, S. Enabling agile testing through continuous integration. In: *Agile Conference, 2009. AGILE'09*. Chicago, IL, USA: IEEE Computer Society, 2009. p. 369–374. Citado na página 22.

VERSIONONE. *11th Annual State of Agile Report*. 2017. Disponível em: <https://explore.versionone.com/state-of-agile>. Acesso em: fev. 2017. Citado na página 15.

Apêndices

APÊNDICE A – Tutorial de Instalação e Configuração do Servidor Jenkins

Este tutorial tem o objetivo de ilustrar e apresentar o processo de instalação e configuração do servidor Jenkins e seus *plugins*, considerando as características do projeto *MobMine*, uma aplicação Android, citado neste trabalho. A instalação é direcionada para o sistema operacional Ubuntu 16.04 LTS.

A.1 Instalação do Java

Inicialmete é necessário a instalação do Java, futuramente exigido no processo de *build* do Jenkins. Utilizando o terminal do Ubuntu (acessado por CTRL+ALT+T), os seguintes comandos, precedidos de "$", são recomendados para a instalação do pacote completo JDK:

- Atualize o índice de pacotes: *$ sudo apt-get update*
- Instale o Java: *$ sudo apt-get install default-jdk*
- Verifique a versão: *$ sudo java –version*
- Confira se a variável de ambiente está configurada: *$ echo $JAVA_HOME*
- Caso a saída no terminal seja vazia:
  - Descubra o local da instalação do Java: *$ sudo update-alternatives –config java*
  - Copie o local informado.
  - Abra arquivo de variáveis de ambiente: *$ sudo nano /etc/environment*
  - Adicione ao final do arquivo a linha: JAVA_HOME="*aqui o local de instalação Java copiado anteriormente*"
  - Recarregue o arquivo: *$ source /etc/environment*
  - Confira se a variável de ambiente está configurada: *$ echo $JAVA_HOME*

A.2 Instalação do Jenkins e Configurações Gerais

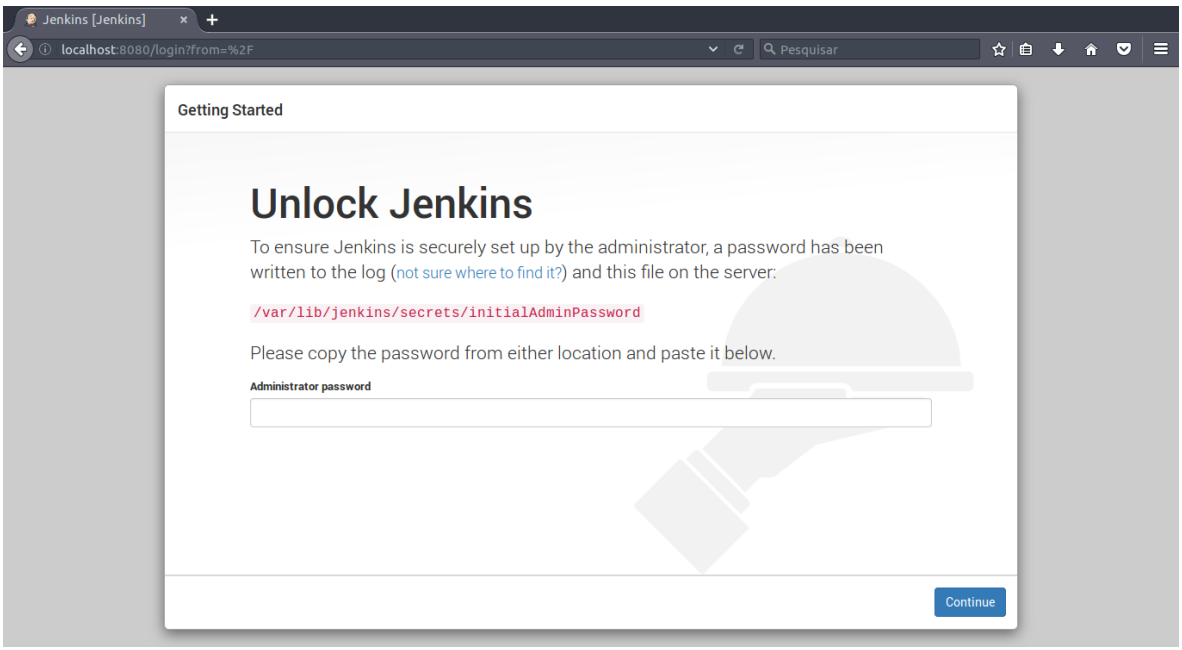
Neste trabalho foi utilizado o repositório de versões estáveis (LTS), acesse a página do Jenkins[1] para mais informações sobre e como utilizar o repositório de versões recentes.

[1] <https://jenkins.io/download/>

São recomendados para a instalação do Jenkins os seguintes comandos precedidos de "$" ou "#":

- Para utilizar o repositório do Jenkins, adicione a chave ao terminal: *$ wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -*
- Ao receber a saída "OK" no terminal, adicione o repositório: *$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'*
- Atualize o índice de pacotes: *$ sudo apt-get update*
- Instale o Jenkins: *$ sudo apt-get install Jenkins*
- Caso tenha problemas em executar o Jenkins na porta 8080 (padrão da instalação), troque para uma porta preferencial: *$ sudo gedit /etc/default/jenkins*
  Ajuste a linha "HTTP_PORT=8080"
- Inicie o Jenkins: *$ sudo /etc/init.d/jenkins start*
- Se tudo estiver de acordo, a mensagem "OK" será apresentada no terminal. O Jenkins está agora em execução na porta padrão 8080 (<http://localhost:8080/>) ou na porta configurada preferencialmente.
- Uma tela para configuração e desbloqueio do Jenkins será mostrada ao acessar a porta pelo navegador:

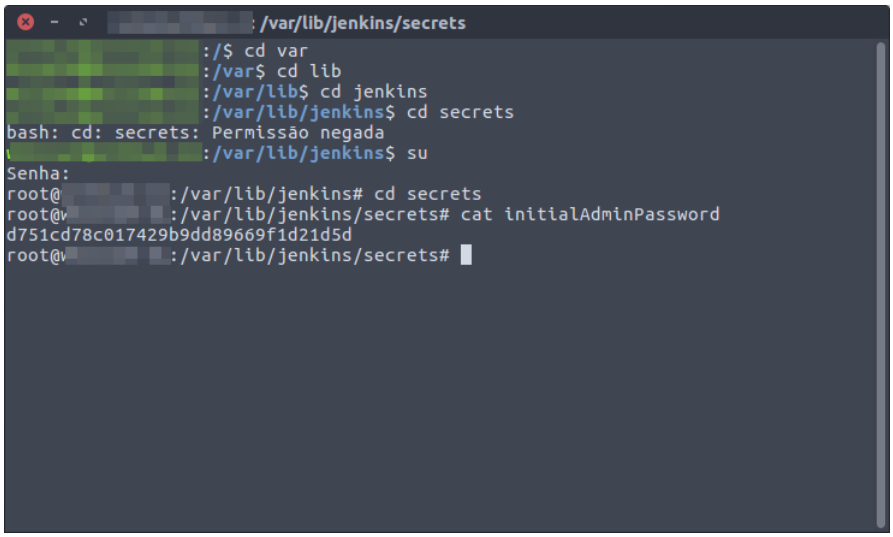
Figura 14 – *Unlock Jenkins*



Fonte: Capturado pelo autor.

- Para liberar a utilização do Jenkins, acesse como usuário *root*: *# cd /var/lib/jenkins/secrets/*
- Execute: *# cat initialAdminPassword*
- Copie a chave impressa no terminal e cole na caixa de texto (*Administrator password*, veja na Figura 14) presente no navegador.

Figura 15 – Terminal: Acesso *root*



Fonte: Capturado pelo autor.

- Ao continuar, uma nova tela é apresentada em que duas opções são exibidas para prosseguir:

Figura 16 – Tela Jenkins: Opção de instalação

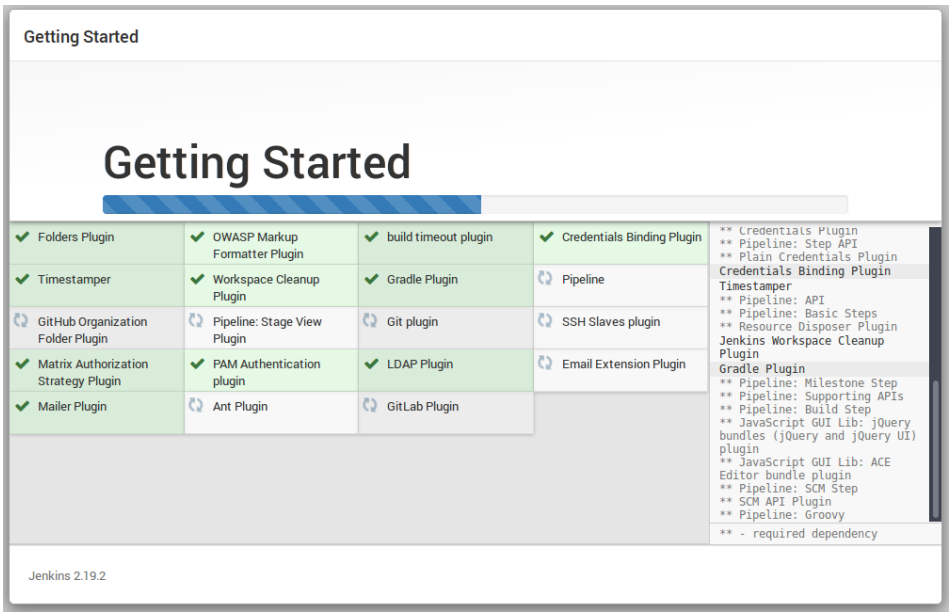
Fonte: Capturado pelo autor.

- Recomenda-se a opção "*Install suggested plugins*", que instala os plugins sugeridos pelo Jenkins. Para usuários que já tenham pré-definidos os *plugins* utilizados, a

outra opção possibilita configurar inicialmente quais deles serão instalados antes mesmo de iniciar o servidor. É possível facilmente rever essas escolhas posteriormente desinstalando e instalando *plugins* na configuração geral do Jenkins.

- Uma tela de carregamento informa a completude do processo:

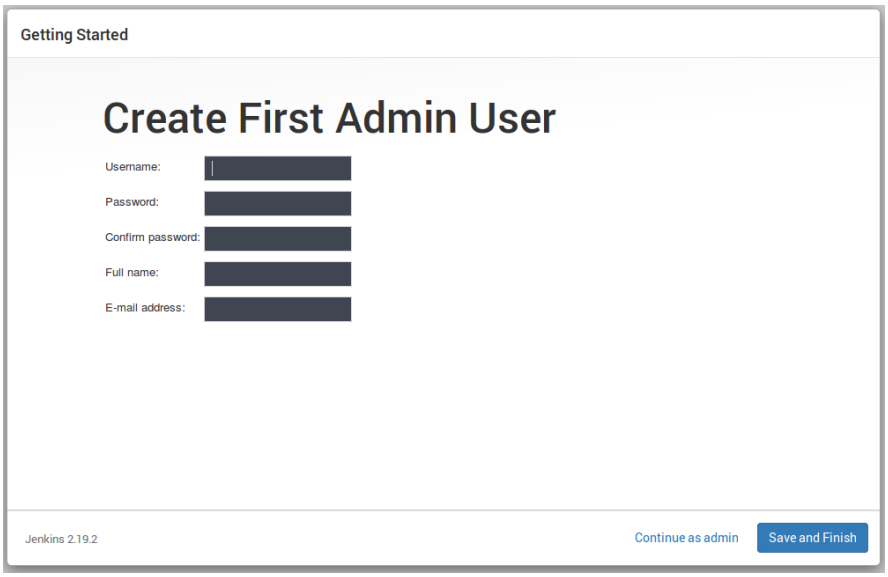
Figura 17 – Tela Jenkins: *Download* e instalação de *plugins* padrões



Fonte: Capturado pelo autor.

- Ao final é exibida a tela de criação e identificação do usuário administrador, mas é possível também continuar apenas como "*admin*" (padrão), porém pouco seguro.

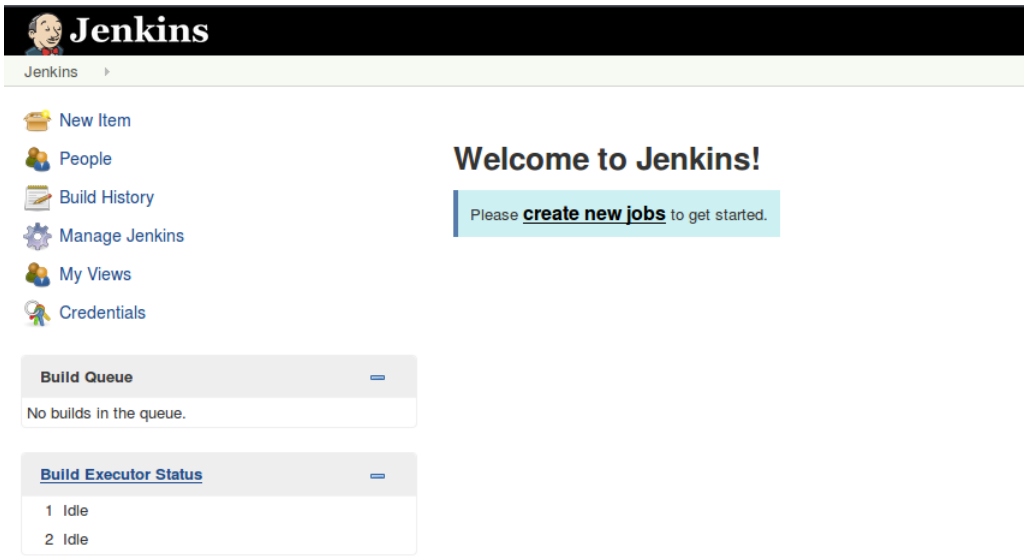
Figura 18 – Tela Jenkins: Configuração de usuário administrador



Fonte: Capturado pelo autor.

- Finalizando o processo de configuração inicial é informado que o Jenkins está pronto. Consequentemente agora é possível iniciar suas atividades ao clicar no botão presente na tela e ser redirecionado para a tela inicial:

Figura 19 – Tela inicial do Jenkins



Fonte: Capturado pelo autor.

A.2.1 Configurações básicas do Jenkins para um projeto Android

É necessário configurar o Jenkins de acordo com as características de cada projeto. Desta maneira, para utiliza-lo é fundamental realizar a instalação dos *plugins* e configurar a comunicação com as ferramentas utilizadas pela equipe: Git, Gradle, GitLab e, por se tratar de uma aplicação Android, o Software Development Kit (SDK). Para realizar a *build* é necessário que a versão correta do JDK, SDK, entre outros, estejam devidamente instalados, configurados e funcionais na máquina que o Jenkins está instalado. Outro fator importante é garantir que as variáveis de ambiente estejam bem configuradas no Jenkins e no Ubuntu. Uma vez que o JDK foi anteriormente instalado, faltam o SDK e Git.

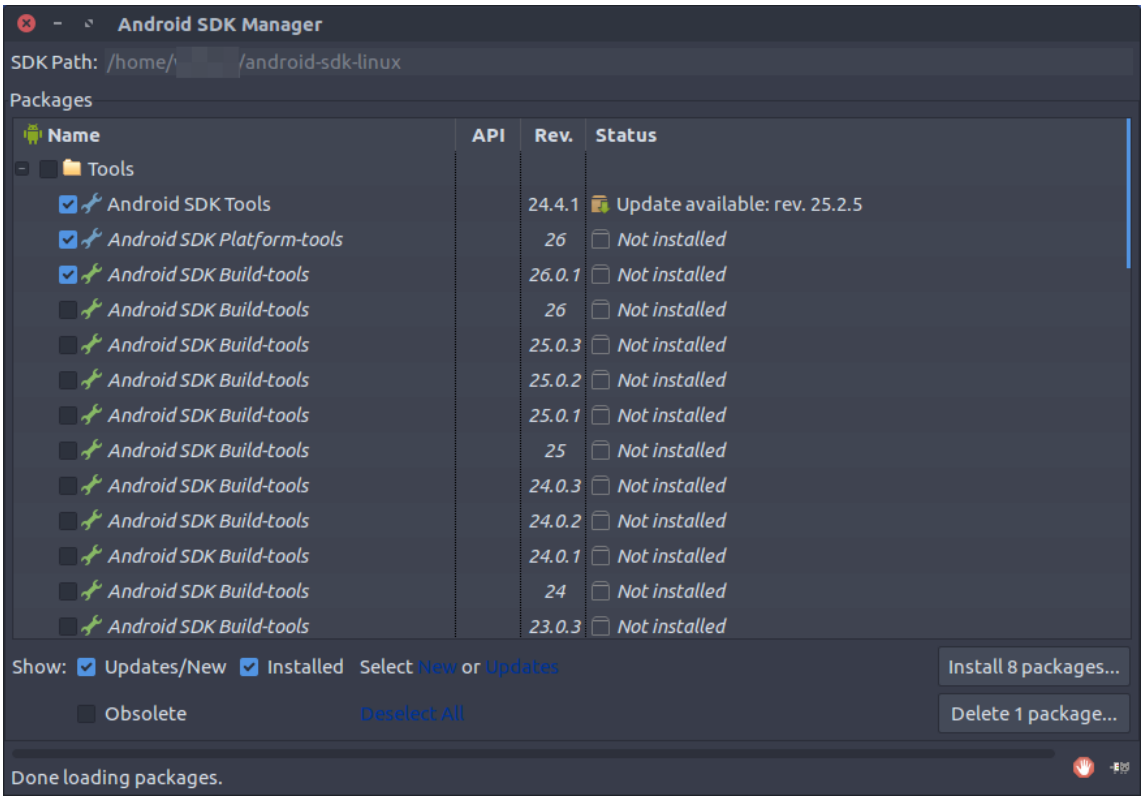
A.2.1.1 Instalação do SDK

Utilizando o terminal do Ubuntu, os seguintes comandos, precedidos de "$", são recomendados para a instalação do pacote completo SDK:

- Realize o download do arquivo: *$ wget http://dl.google.com/android/android-sdk_r24.4.1-linux.tgz*
- Descompacte o arquivo: *$ tar -xvf android-sdk_r24.4.1-linux.tgz*

- Abra o arquivo de variáveis de ambiente: *$ sudo nano /etc/environment*
- Adicione ao final do arquivo a linha: "ANDROID_HOME=/$HOME/android-sdk-linux"
- Adicione ao final do arquivo a linha: "PATH=${PATH}:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools"
- Recarregue o arquivo: *$ source /etc/environment*
- Abra o Android SDK Manager: *$ android*
  A partir do Android SDK Manager realize as atualizações e instalações das ferramentas necessárias de acordo com o projeto.

Figura 20 – Tela do Android SDK Manager



Fonte: Capturado pelo autor.

A.2.1.2 Instalação do Git

O Git pode ser instalado automaticamente pelo Jenkins a partir de suas configurações. Porém com o objetivo de manter a ferramenta instalada na máquina sem dependências em relação ao Jenkins, utilizando o terminal do Ubuntu, os seguintes comandos, precedidos de "$", são recomendados para a instalação do Git:

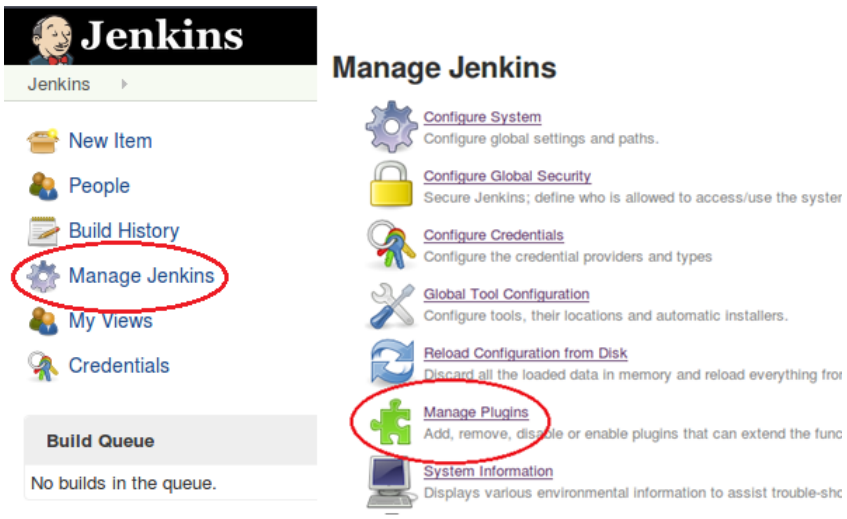
- Atualize o índice de pacotes: *$ sudo apt-get update*

- Instale o Git: *$ sudo apt-get install git*

A.2.1.3 Gerenciamento e instalação de *plugins*

O Jenkins possui um espaço em seu *software* especialmente dedicado aos *plugins*. Para gerenciá-los recomenda-se seguir os seguintes passos:

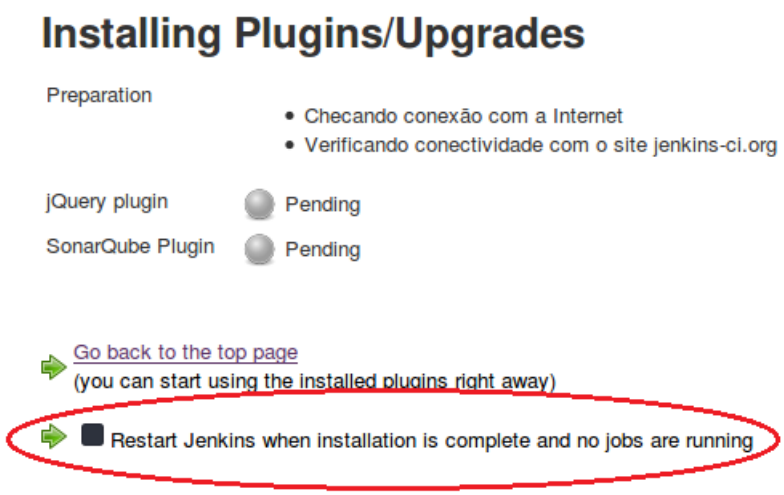
Figura 21 – Tela Jenkins: Passos para encontrar as configurações de *plugins*



Fonte: Capturado pelo autor.

- Acesse "*Manage Jenkins*" no menu à esquerda e após acesse "*Manage Plugins*". Atualize todos que lhe forem recomendados. É possível encontrar no rodapé da página um *link* chamado "*All*" logo após de "*Select*", clique-o e depois clique em "*Download now and install after restart*". Após estas ações é apresentada a tela referente ao donwload das atualizações, ilustrada na Figura 22:

Figura 22 – Tela Jenkins: Atualizar *plugins*



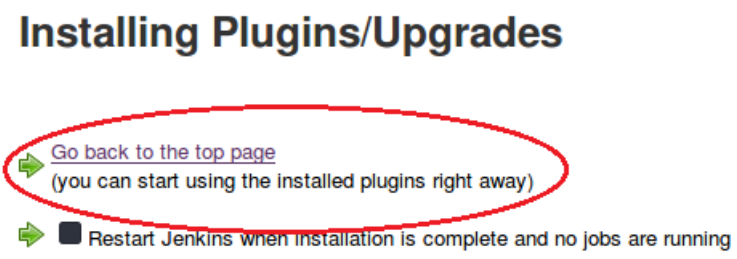
Fonte: Capturado pelo autor.

- Marque a opção "*Restart Jenkins when installation is complete and no jobs are running*", desta maneira o Jenkins vai reiniciar automaticamente após o download e instalar as atualizações.
- Ao final do processo uma tela informa que o Jenkins está sendo reiniciado:

Figura 23 – Tela Jenkins: Processo de reinicialização

Fonte: Capturado pelo autor.

Figura 24 – Tela Jenkins: Redirecionamento

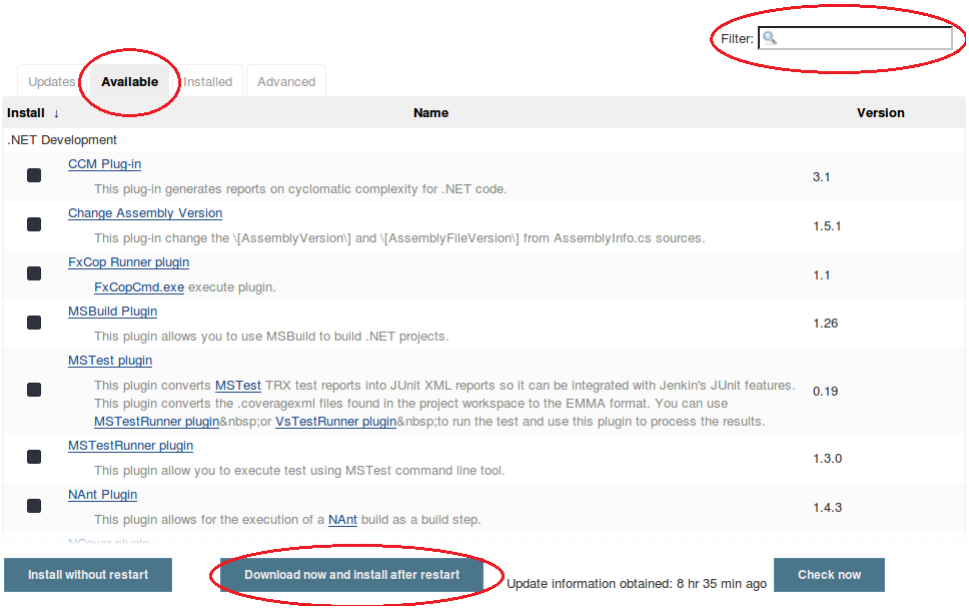


Fonte: Capturado pelo autor.

- Logo após, uma nova tela é apresentada, apenas clique em "*Go back to the top page*", redirecionando-o para a tela inicial. Navegue novamente, como demonstrado na Figura 21 até o gerenciamento de *plugins* para, de fato, instalar os novos *plugins*.

- Siga até a aba "*Avaliable*" e utilize a barra superior no canto direito para filtrar o nome do *plugin* desejado, encontrando-o de maneira mais rápida. Uma vez que o projeto utiliza o repositório Git, a ferramenta de gerencimento GitLab e o Gradle para automação e gerenciamento da *build*, os respectivos *plugins* devem ser instalados. Alguns destes podem vir instalados na configuração padrão do Jenkins, caso não encontre-os, confira na aba "*Installed*".

Figura 25 – Tela Jenkins: Instalação de *plugins*



Fonte: Capturado pelo autor.

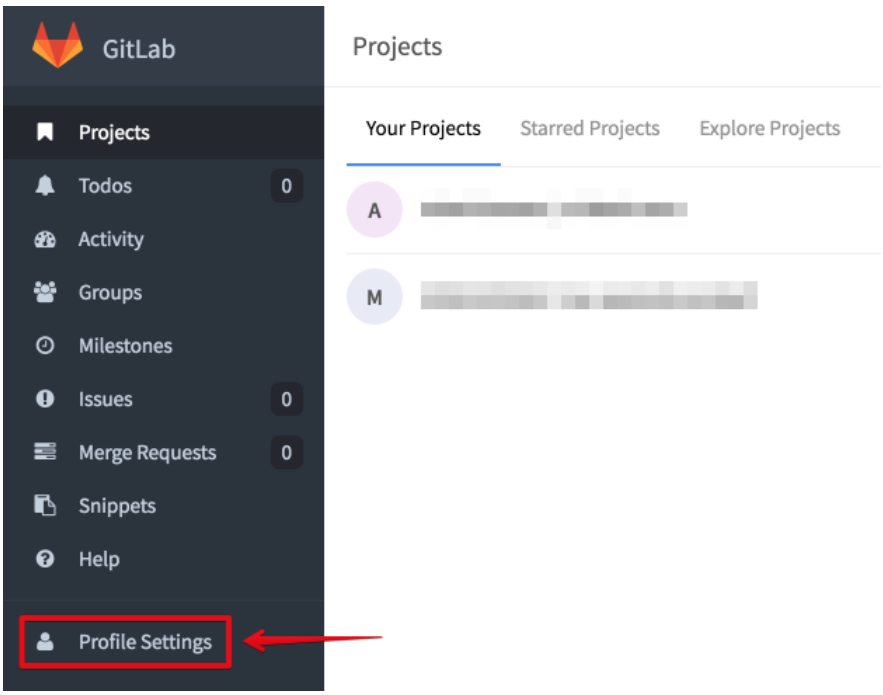
- Selecionados os plugins, clique em "*Download now and install after restart*". Em seguida a tela de *download* de *plugins* é apresentada. Repita o processo visto anteriormente (marque a opção "*Restart Jenkins when installation is complete and no jobs are running*"). Após instalar os *plugins* com sucesso é possível encontra-los na aba "*Installed*" em "*Manage Plugins*".

A.2.1.4 Configuração das ferramentas e *plugins* no Jenkins

Antes de criar e configurar um *Job*, é necessário configurar o Jenkins com o objetivo de estabelecer a comunicação correta do Jenkins com um repositório externo, por exemplo, ou as ferramentas instaladas na máquina em que o Jenkins se encontra.

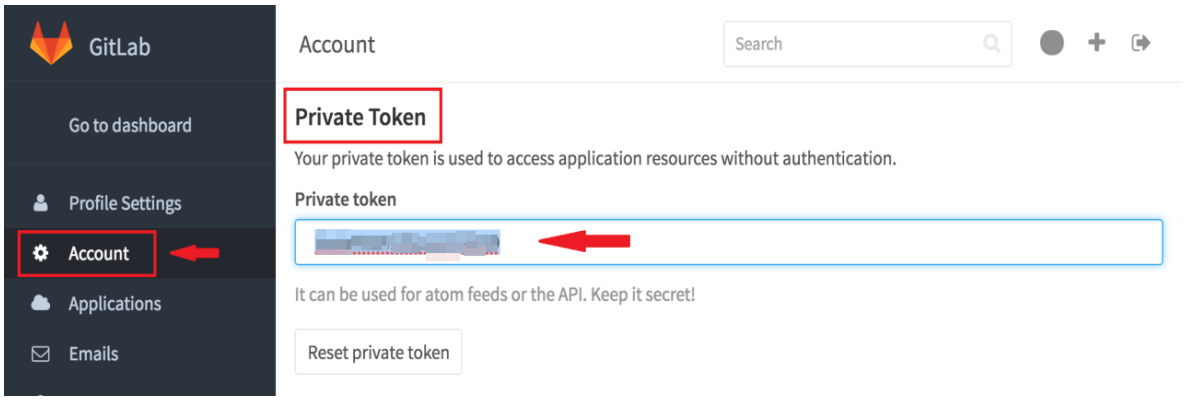
- Para obter conexão com o GitLab é necessário uma chave que se encontra no prório GitLab. Acesse a página do GitLab, no menu lateral, clique na opção "*Profile Settings*" e em seguida na opção "*Account*". Copie o código disponível na aba "*Private Token*"

Figura 26 – Tela GitLab: Opções do usuário



Fonte: Capturado pelo autor.

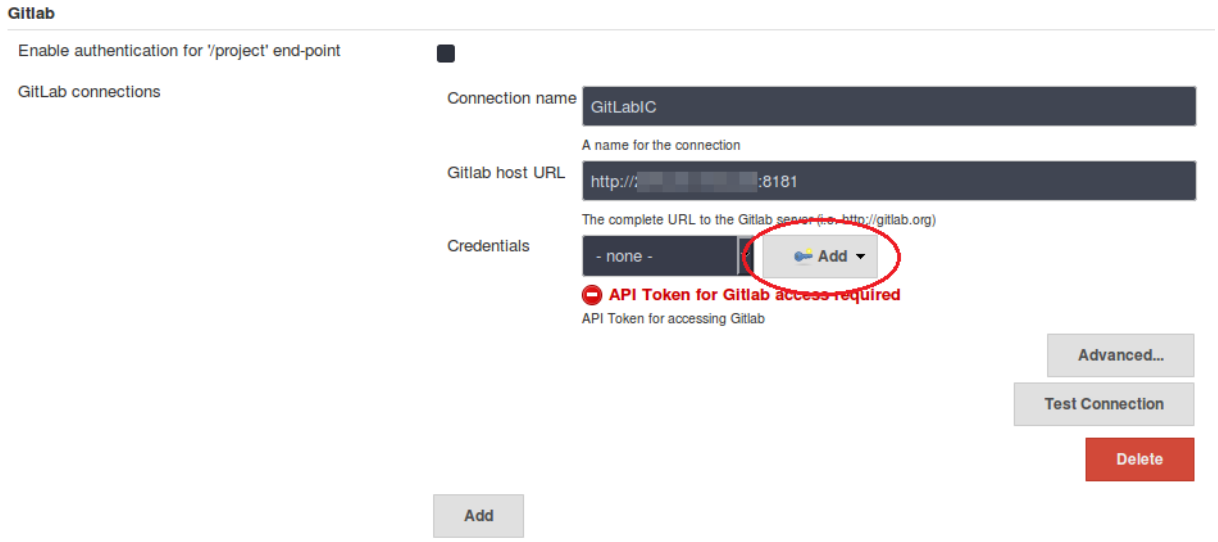
Figura 27 – Tela GitLab: *Private Token*



Fonte: Capturado pelo autor.

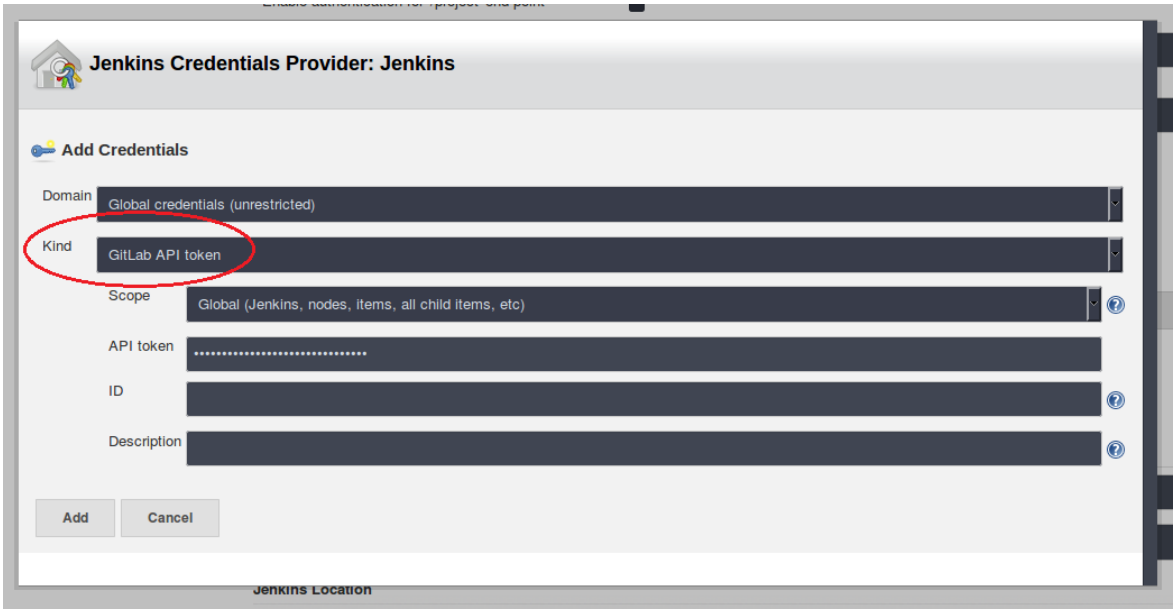
- Agora no Jenkins, navegue até "*Manage Jenkins*" no menu à esquerda e após clique em "*Configure System*". Encontre o menu "*GitLab connections*". Informe um nome para a conexão, o *link* do servidor GitLab e adicione a chave criando uma credencial ao clicar em "*Add*", depois "*Jenkins*". Apresentada a nova janela, na opção "*Kind*", selecione "*GitLab API token*" e cole o código. Por fim clique em "*Add*".

Figura 28 – Tela Jenkins: Menu "*GitLab connections*"



Fonte: Capturado pelo autor.

Figura 29 – Tela Jenkins: Criando uma credencial para estabelecer conexão com o GitLab

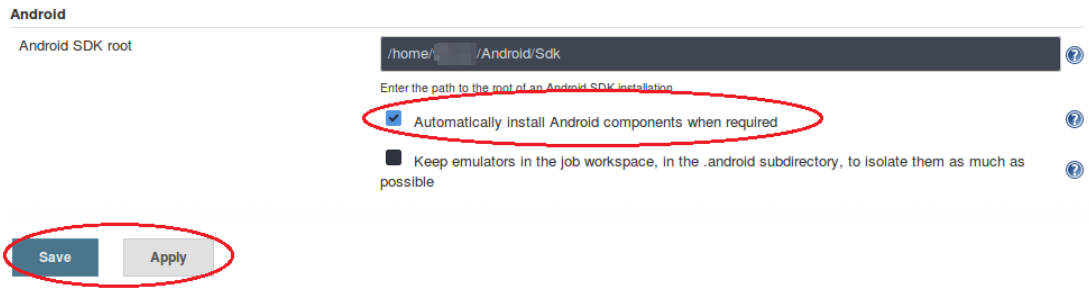


Fonte: Capturado pelo autor.

- Para testar a conexão, clique no botão "*Test Connection*", isso vai imprimir ao lado uma mensagem de "*Error*" ou "*Success*".
- Prosseguindo com a configuração, o Jenkins precisa conhecer o caminho da instalação do SDK para realizar a *build*. Encontre o menu "*Android*". Informe o caminho da

pasta SDK na máquina que hospeda o Jenkins e marque a opção "*Automatically install Android components when required*".

Figura 30 – Tela Jenkins: Configuração do local da instalação do SDK

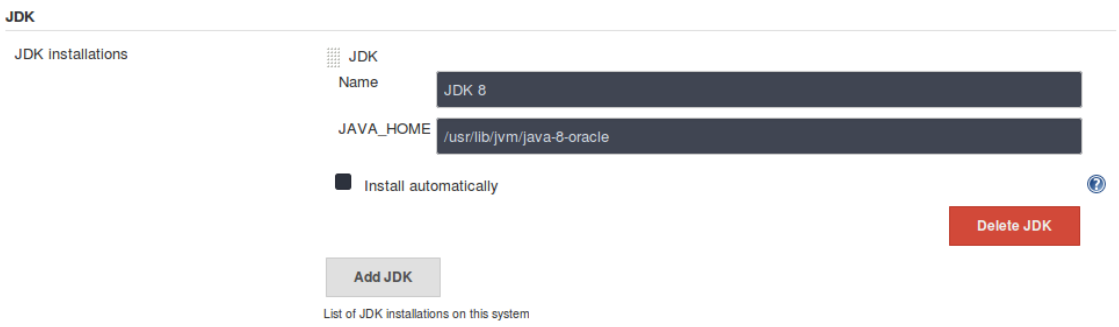


Fonte: Capturado pelo autor.

- Ainda nesta mesma seção, encontre a aba "*Global properties*" e marque a opção "*Environment variables*". Insira na caixa de texto referente a "*Name*" a entrada "*ANDROID_HOME*" e informe o caminho em que está instalado o SDK na máquina que hospeda o Jenkins. Para finalizar e aplicar as mudanças clique em "*Apply*" e em seguida "*Save*".

- Para seguir com a configuração das ferramentas navegue novamente para "*Manage Jenkins*" no menu à esquerda e dessa vez acesse "*Global Tool Configuration*". Encontre o menu "*JDK*", clique em "*Add JDK*", desmarque a opção "*Install automatically*". Informe um nome que identifique a instalação JDK e o local da instalação do Java na caixa de texto "*JAVA_HOME*".

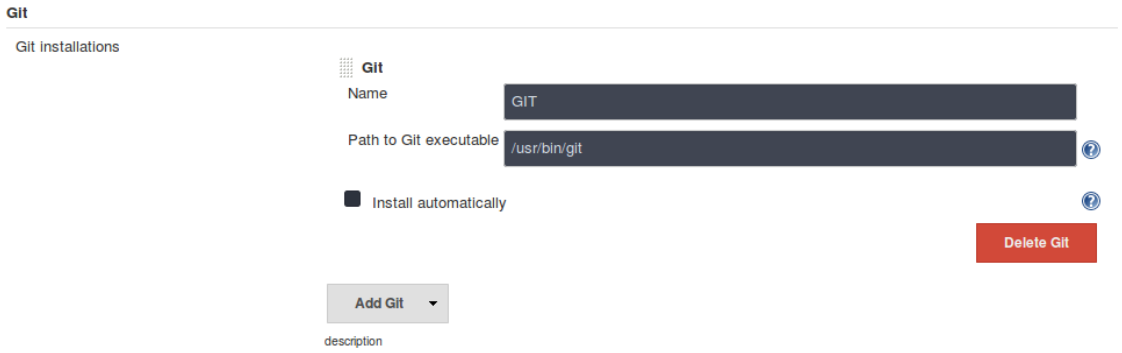
Figura 31 – Tela Jenkins: Configuração do local da instalação do JDK



Fonte: Capturado pelo autor.

- Ainda na mesma seção, encontre o menu "*Git*" e clique em "*Add Git*", logo após clique na opção "*Git*". Informe um nome para a instalação e o caminho da instalação do Git. Para encontra-lo execute o comando no terminal: *$ whereis git*.

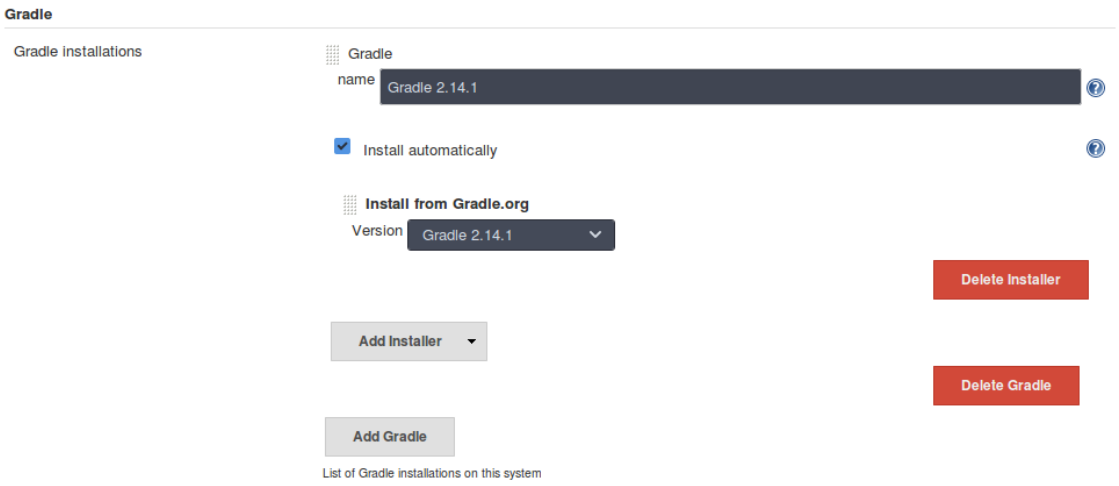
Figura 32 – Tela Jenkins: Configuração do local da instalação do Git



Fonte: Capturado pelo autor.

Ainda na mesma seção, encontre o menu "*Gradle*", clique em "*Add Gradle*" e informe um nome para a instalação, dessa vez marque a opção "*Install automatically*" e navegue até a versão usada no projeto. Para finalizar e aplicar as mudanças clique em "*Apply*" e em seguida "*Save*".

Figura 33 – Tela Jenkins: Configuração do local da instalação do Gradle



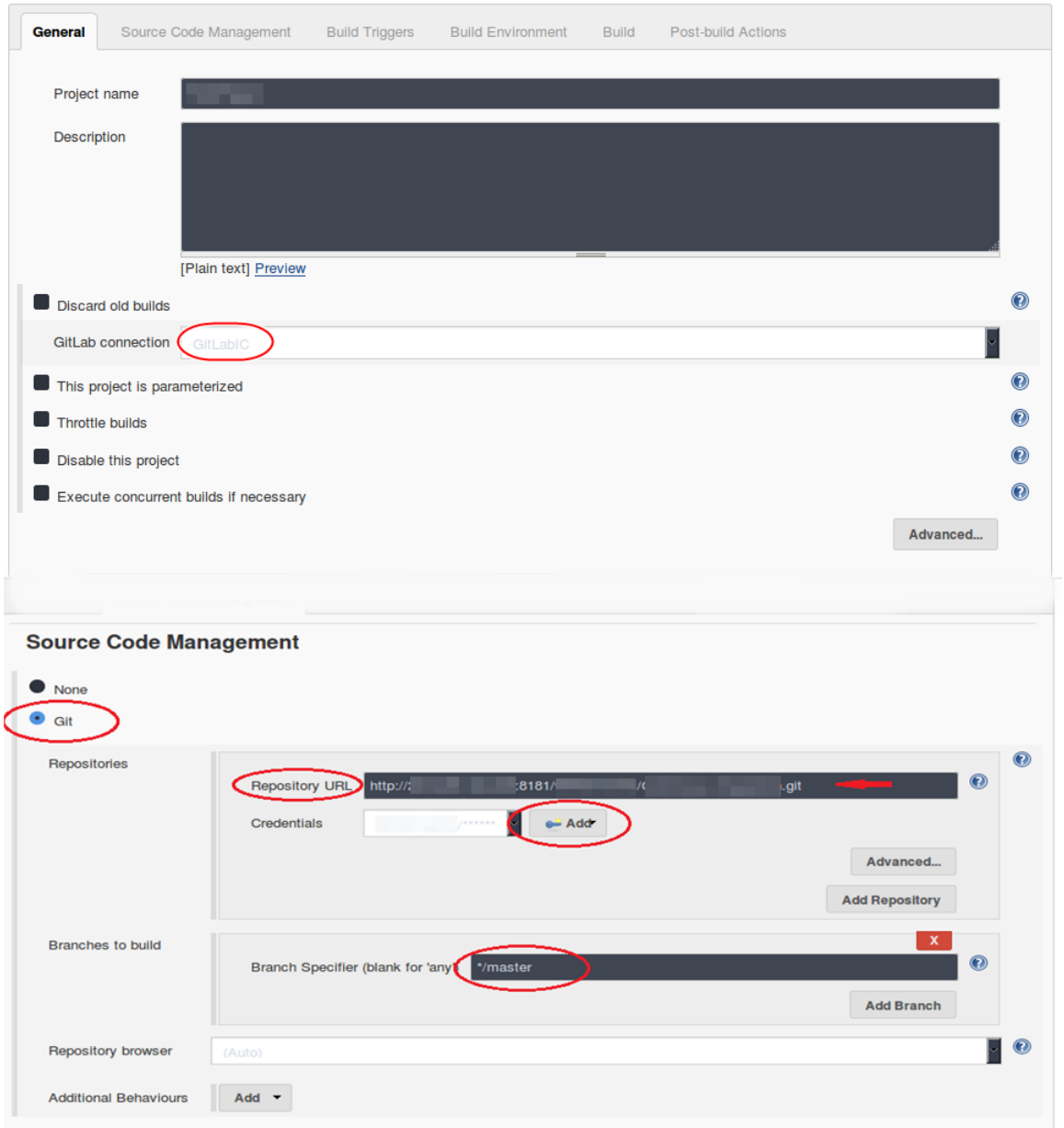
Fonte: Capturado pelo autor.

A.2.1.5 Configuração de um *Job* no Jenkins

É possível agora criar o *Job* referente ao projeto. Na tela inicial do Jenkins clique em "*New Job*", quando uma nova tela for apresentada, informe um nome para o projeto e selecione a opção "*Freestyle project*" e clique em no botão "*OK*" na parte inferior da tela. Após o redirecionamento, uma nova tela é apresentada contendo as configurações do projeto. Os seguintes passos são recomendados para a configuração do *Job* e a realização de *builds*.

- Na caixa de texto "*GitLab connection*" verifique se está presente e selecionada a opção com o nome que foi escolhido anteriormente para a conexão com o GitLab.
- Navegue até "*Source Code Management*", clique na opção "*Git*" e será mostrado varias opções.

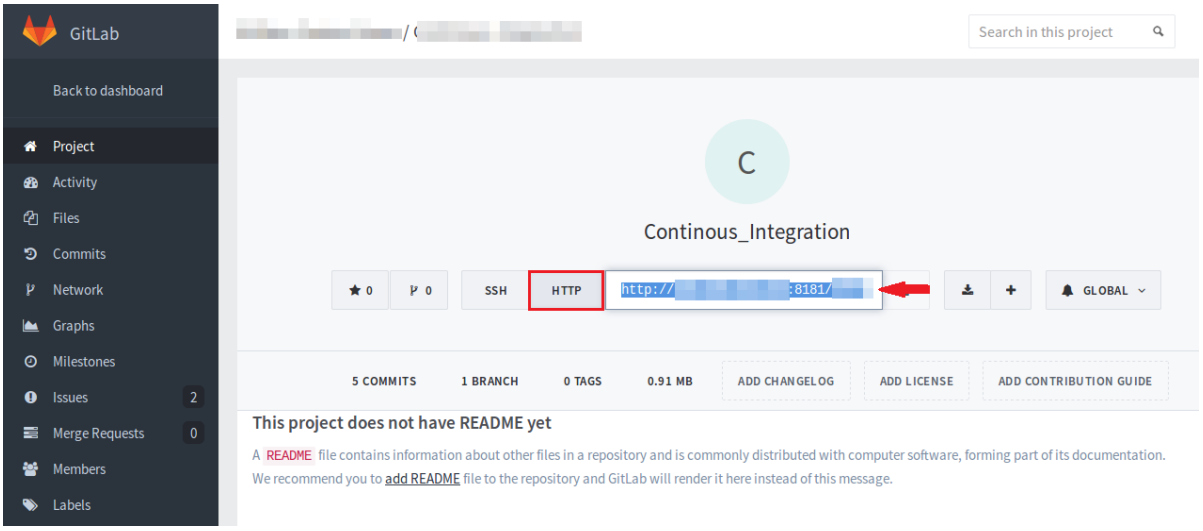
Figura 34 – Tela Jenkins: Configuração *Job* I



Fonte: Capturado pelo autor.

- Informe na opção "*Repository URL*" o endereço referente ao seu repositório, disponível na página principal do seu projeto no GitLab. Na opção "*Credentials*" clique em "*Add*" e depois "Jenkins". Em seguida, ao ser exibida a nova janela, na opção "*Kind*", deixe "*Username with password*" e em '*Username*" informe o seu nome de usuário do GitLab e em "*Password*" sua senha. Por fim clique em "*Add*".

Figura 35 – Tela principal do projeto no GitLab



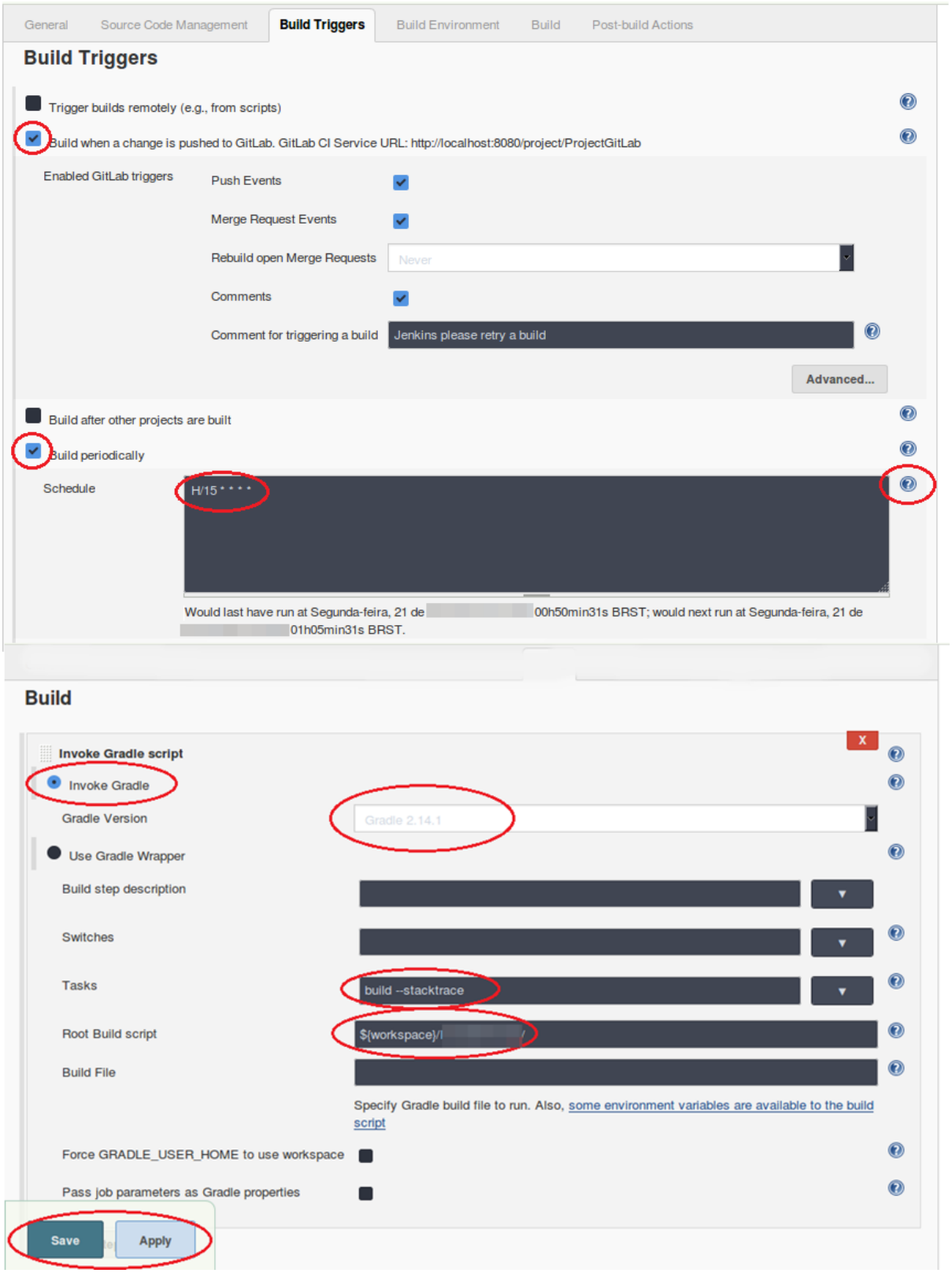
Fonte: Capturado pelo autor.

- Na aba "*Build Triggers*" a definição segue as caracteristicas do projeto ou a maneira, predefinida pela equipe, de como a *build* será executada pelo Jenkins. Nesse caso foram selecionadas as opções "*Build when a change is pushed to GitLab*", esta opção dispara uma *build* quando uma mudança é acionada no repositório, e "*Build periodically*" que estima um certo tempo para realizar uma *build* de acordo com o tempo definido no campo "*Schedule*" seguindo a gramática estipulada pelo Jenkins, visível ao clicar na interrogação ao lado da caixa de texto, veja na Figura 36.

- Na aba "*Build*", clique em "*Add build step*" e selecione a opção "*Invoke Gradle script*" e marque "*Invoke Gradle*" selecionando a versão configurada anteriormente. Para realizar uma tarefa no Gradle e acompanhar o processo de execução da *build*, registrando em terminal próprio o *log*, útil para verificar erros no código, informe "*build –stacktrace*" na opção "*Tasks*".

- Clique no botão "*Advanced*" e na opção "*Root Build script*". Informe o diretório raiz no qual se encontra o arquivo Gradle ("*build.gradle*") no projeto. Como o Jenkins cria uma pasta de nome "*workspace*", a pasta deve ser informada da seguinte maneira. Caso o nome do *Job* seja diferente do nome do projeto: ${workspace}/"PastaDoProjeto". Caso os nomes sejam iguais informe apenas: ${workspace}. Finalize clicando em "*Apply*" e em seguida "*Save*".

Figura 36 – Tela Jenkins: Configuração *Job* II

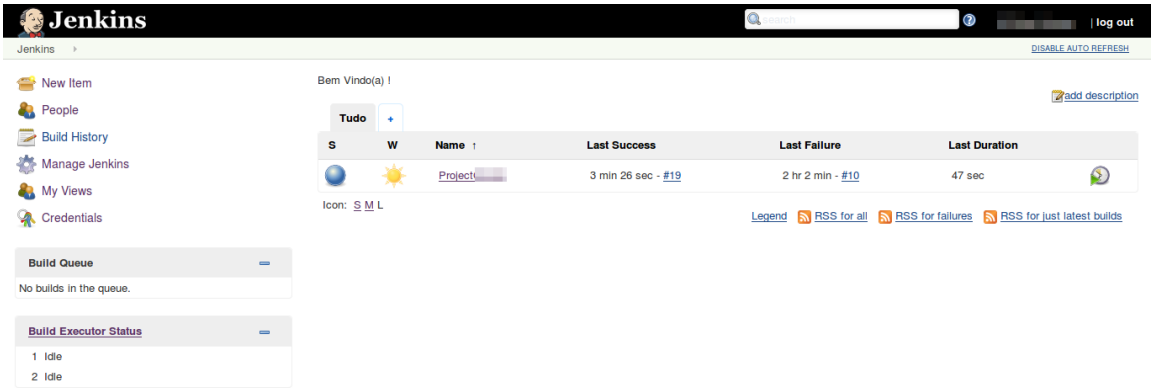


Fonte: Capturado pelo autor.

Com isso o *Job* está criado e com uma congifuração básica. Para realizar uma *build*

manual clique no botão "*Build Now*" na página principal do projeto e o Jenkins dará inicio ao processo de *build* tentando clonar seu repositório. Em caso de falha (verifique se é um erro no processo de *build* que remete a algum problema de configuração ou se tem origem no código fonte e etc.) a esfera tem sua cor alterada para vermelho, em caso de sucesso (sem erros) azul e processo de *build* não realizado, interrompido pelo usuário ou *Job* desabilitado, cinza.

Figura 37 – Tela Jenkins: *Job* ativo



Fonte: Capturado pelo autor.