UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS APLICADAS A EDUCAÇÃO

DEPARTAMENTO DE CIÊNCIAS EXATAS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

INTEGRAÇÃO DE FERRAMENTAS PARA GESTÃO DE PROJETOS SCRUM

FRANCLIS GALDINO DA SILVA

Orientador: Prof. Msc. Rodrigo de A. Vilar Miranda

RIO TINTO - PB

2015

FRANCLIS GALDINO DA SILVA

INTEGRAÇÃO DE FERRAMENTAS PARA GESTÃO DE PROJETOS SCRUM

Monografia apresentada para obtenção do título de Bacharel à banca examinadora no Curso de Bacharelado em Sistemas de Informação do Centro de Ciências Aplicadas e Educação (CCAE), Campus IV da Universidade Federal da Paraíba.
Orientador: Prof. Msc. Rodrigo de A. Vilar Miranda

RIO TINTO - PB
2015

S586i Silva, Franclis Galdino da.
Integração de ferramentas para gestão de projetos SCRUM. / Franclis Galdino da Silva. – Rio Tinto: [s.n.], 2015.
79 f. : il.-

Orientador (a): Prof. Msc. Rodrigo de A. Vilar Miranda.
Monografia (Graduação) – UFPB/CCAE.

1. Software - desenvolvimento. 2. Scrum - software. 3. Sistemas de Informação.

UFPB/BS-CCAE CDU: 004.41 (043.2)

FRANCLIS GALDINO DA SILVA

INTEGRAÇÃO DE FERRAMENTAS PARA GESTÃO DE PROJETOS SCRUM

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal da Paraíba, Campus IV, como parte dos requisitos necessários para obtenção do grau de BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Assinatura do autor: _____

APROVADO POR:

_____

Orientador: Prof. Msc. Rodrigo de A. Vilar M.
Universidade Federal da Paraíba – Campus IV

_____

Prof. Msc. Yuri de A. Malheiros Barbosa
Universidade Federal da Paraíba – Campus IV

_____

Prof. Msc. Renata Viegas de Figueiredo
Universidade Federal da Paraíba – Campus IV

RIO TINTO - PB
2015

“Sonhos determinam o que você quer. Ações determinam o que você conquista”

Aldo Novak

AGRADECIMENTOS

Primeiramente, agradeço a minha mãe Maria Geisa Galdino, pelo incentivo nos momentos em que estive desanimado, por toda compreensão quando mais precisei, por sempre me apoiar nas minhas escolhas. Enfim, agradeço por todo seu amor, carinho, amizade e proteção.

A minha amada, Daliana Karla, por todo seu amor, carinho e paciência. Por sempre me ajudar a manter minhas metas e por me incentivar a continuar no caminho que eu tracei. Saiba que nos momentos mais difíceis, encontrei em você uma razão para persistir com meus objetivos.

Ao meu orientador Rodrigo Vilar, a quem tive a honra de conhecer, mesmo que há pouco tempo, é uma pessoa digna minha admiração. Especialmente, por sempre se fazer presente quando tive alguma dúvida em relação ao desenvolvimento deste trabalho, além de outras questões acadêmicas e que soube me guiar como ninguém pelas etapas deste trabalho.

A todos os colegas de curso, em especial aos colegas: Renan Soares, Franco Neto, Erickson Silva, Williby Ferreira e Carlo Chianca por compartilharem experiências e aperreios, sem esquecer das boas gargalhadas, sem vocês essa jornada com certeza seria mais difícil.

A todos os professores do Departamento de Ciências Exatas do Campus IV, em especial aos professores: Rafael Magalhães, Rodrigo Rebouças, Yuri Malheiros, Renata Viegas, Yuska Paola, Marcus Carvalho, Adriana Clericuzi, Wagner Emanoel, Tiago Maritan, Jorge Dias, Hermann Hrdlicka e Raoni Kulesza. Obrigado a todos por me guiarem nessa jornada.

A todos colegas do Laboratório Embedded, em especial aos colegas de trabalho André Meireles, Daniel Abella e Vicente Neto.

Por fim, agradeço a todos que não foram citados, mas sabem que participaram dessa conquista e merecem, sempre, minha gratidão.

RESUMO

Com o constante crescimento na adoção de metodologias ágeis em projetos de desenvolvimento de software, especialmente o da metodologia Scrum, também vem crescendo o número de ferramentas que oferecem suporte aos processos, atores, eventos e artefatos da metodologia. Entretanto, muitas dessas ferramentas englobam apenas uma parte do que é proposto por Scrum, sendo necessário gerenciar seus processos e artefatos em mais de uma ferramenta. Além disso, essas ferramentas são utilizadas de forma isolada, ou seja, não há integração entre elas, o que acarreta em diversos problemas, como: duplicação de trabalho, inconsistência de informações, omissão de processos e artefatos da metodologia. Pensando em amenizar esses problemas, parte dessas ferramentas oferecem algum tipo de integração genérica com outras ferramentas. Porém, durante a pesquisa desse trabalho não encontramos nenhuma solução ou referência para integração de ferramentas que ofereçam suporte a Scrum. Portanto, este trabalho tem como objetivo apresentar uma arquitetura com algumas funcionalidades que possam servir como referência para futuros sistemas que desejem integrar ferramentas Scrum, dessa forma, resolvendo ou amenizando os problemas causados pela falta de integração levantados por este trabalho.

Palavras chave: Scrum, Integração, Automação.

ABSTRACT

With the steady growth in the adoption of agile methodologies in software development projects, specially the Scrum methodology, has also increased the number of tools that support the processes, actors, events and artifacts of the methodology. However, many of these tools include only part of what is proposed by Scrum, thus it is necessary to manage its processes and artifacts in more than one tool. In addition, these tools work in isolation, that is, there is no integration between them, which results in various problems, such as duplication of work, information inconsistency, omission of some processes and artifacts of the methodology. In other to resolve these problems, some of these tools provide some sort of generic integration with other tools. However, during the research we did not find any reference to a solution or integration tools that support Scrum. Therefore, this paper aims to present an architecture that serves as a reference for future systems that integrate Scrum tools, thus solving or mitigating the problems caused by lack of integration raised by this work.

Keywords: Scrum, *Integration, Automation.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CARG	*Compound Annual Growth Rate*
FCV	Ferramenta de Controle de Versão
FGP	Ferramenta de Gerenciamento de Projetos
FGT	Ferramenta de Gerenciamento de Testes
FIC	Ferramenta de Integração Contínua
GP	Gerente de Projeto
HTTP	*Hypertext Transfer Protocol*
IC	Integração Contínua
IP	*Internet Protocol*
LDAP	*Lightweight Directory Access Protocol*
ROI	*Return on Investment*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Desenvolver software é uma atividade intrinsecamente complexa. Nos últimos anos, a indústria de desenvolvimento de software teve um alto crescimento no mundo todo, devido ao avanço e disseminação da informática. Dessa forma, as empresas da área aprimoraram suas atividades de desenvolvimento, em busca de melhorias na produção de software. Entretanto, nesse mesmo período, também surgiram diversos problemas ligados a esse tipo de atividade, como: incapacidade de atender à demanda, atraso de cronograma, estouro de orçamento, fracasso na entrega do produto final, baixa qualidade e alta taxa de retrabalho com manutenção (VENTORIN, 2005).

Um dos motivos que gerou esses problemas foi a falta de processos e metodologias para gerenciar times de desenvolvimento, causando confusão entre os desenvolvedores, que não sabem exatamente por onde começar um projeto, muito menos como terminá-lo. Essa confusão impacta diretamente no insucesso dos projetos (NETO, 2010).

Diante disso, desde a década de 60, a comunidade científica de computação pesquisa formas de amenizar esses problemas, tentando melhorar o modo de desenvolver software. Desde então, foram propostos diversos processos, modelos e metodologias para gerenciar o ciclo de vida dos projetos, desde sua concepção até a validação do cliente (FILHO, 2007, p.24).

Dentre essas propostas, surgiu um movimento da Engenharia de Software denominado de Manifesto Ágil (MANIFESTO ÁGIL, 2001), que defende a adoção de processos mais simplificados – os Métodos Ágeis – e sugere um conjunto de conceitos e princípios para o Desenvolvimento Ágil de Software. Segundo o relatório, CHAOS Report 2012 do Standish Group, projetos que utilizam metodologias ágeis são três vezes mais bem-sucedidos que projetos que utilizam metodologias "não ágeis". Além disso, no ano de 2011, as metodologias ágeis eram utilizadas em 9% de todos projetos de TI e adotadas em 29% dos novos projetos, com taxa de 22% *Compound Annual Growth Rate* (CAGR), que neste caso, indica a taxa composta anual de crescimento na adoção das metodologias ágeis.

Dentre as metodologias ágeis, uma tem despertado grande interesse na comunidade de desenvolvimento de software – Scrum – uma metodologia que trata a complexidade no desenvolvimento através de inspeção, adaptação e transparência, para produzir software de maneira flexível e em ambientes que sofrem constantes mudanças. Scrum pode ser aplicada a praticamente qualquer projeto, no entanto é utilizada principalmente no desenvolvimento de software (SCHWABER; SUTHERLAND, 2013).

Apesar de ser uma abordagem relativamente nova, nos últimos anos, a utilização da metodologia Scrum tem crescido. Inclusive, sendo utilizada por empresas mundialmente famosas como: Google, Microsoft, Dell Computer, BBC, Philips, Yahoo, entre outras (MACHADO; MEDINA, 2009). Além disso, recentes pesquisas afirmam que os clientes estão cada vez mais satisfeitos e que em relação a metodologias tradicionais, o uso de Scrum fez com que os atrasos nas entregas diminuíssem (MANN; MAURER, 2005).

Scrum foi criada com o objetivo de otimizar o desenvolvimento de projetos em equipe. Portanto, envolve diversos atores, artefatos e tipos de reunião, além disso, a própria tarefa de desenvolver software também requer alguns processos: controle de versão de código, testes e etc. Dessa maneira, é recomendado usar sistemas informatizados para dar suporte a Scrum e ao desenvolvimento. Por exemplo, para gerenciar os artefatos e o processo Scrum, é recomendado ferramentas para gestão de projetos (Tuleap[1], Jira[2], Redmine[3] e etc.). Algumas dessas ferramentas citadas são específicas para Scrum e outras podem ser adaptadas com o uso de *plugins*. Para o processo de desenvolvimento é recomendado ferramentas para: controle de versão de código-fonte (Gitlab[4], Github[5], BitBucket[6], entre outras), gerenciamento testes (TestLink[7], Tarantula[8], etc.) e integração continua (Jenkins[9], Bamboo[10], Gitlab CI[11]).

Apesar de existir uma grande quantidade de ferramentas que dão suporte a Scrum, muitas delas englobam apenas uma parte do que é proposto pela metodologia, sendo necessário gerenciar alguns dos seus processos, recursos e artefatos em mais de uma ferramenta ou até mesmo de forma manual. Em contrapartida, como forma suavizar esse tipo de problema, algumas dessas ferramentas oferecem soluções genéricas para integração com outras ferramentas. Porém, durante o desenvolvimento desse trabalho, não encontramos nenhuma solução de referência para integração de ferramentas que possam ser utilizadas em projetos que adotem Scrum.

[1] www.tuleap.org
[2] www.atlassian.com/software/jira
[3] www.redmine.org
[4] www.gitlab.com
[5] www.github.com
[6] www.bitbucket.org
[7] www.testlink.org
[8] www.tarantula.fi
[9] www.jenkins-ci.org
[10] www.atlassian.com/software/bamboo
[11] www.gitlab.com/gitlab-ci

1.1 PROBLEMA DE NEGÓCIO E TÉCNICO

Nessa seção serão apresentados os problemas de negócio e técnico abordados neste trabalho.

1.1.1 Problema de negócio

Devido à vasta gama de ferramentas, não integradas, existe muito retrabalho, inconsistência e omissão relacionados ao uso destas ferramentas, o que acaba dificultando a adoção de Scrum com sucesso.

1.1.2 Problema técnico

A maior parte das ferramentas que dão suporte a Scrum possui alguma forma de integração genérica: webhooks, API REST ou SOAP. Assim sendo, é possível fazê-las trabalhar em conjunto e reduzir o retrabalho, a inconsistência e a omissão de informações e processos.

Porém, é difícil encontrar soluções prontas para integrar estas ferramentas. Nesse caso, é sempre necessário destacar algum programador para desenvolver *scripts* ou serviços de integração, o que pode gerar atraso no início dos projetos. Além disso, cada equipe que vai integrar as ferramentas precisa "reinventar a roda", ou seja, não há reuso de soluções de integração.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo desse trabalho é propor uma arquitetura para integração de ferramentas de suporte a metodologia Scrum, e avaliá-la sob o ponto de vista de líderes de equipe de desenvolvimento, no contexto de uma empresa de desenvolvimento de software.

1.2.2 Objetivos Específicos

- Levantar as ferramentas existentes para suporte a processos Scrum;
- Descrever problemas recorrentes da falta de integração de ferramentas;
- Disponibilizar um documento com princípios para integração de ferramentas Scrum.

1.3 METODOLOGIA

- Pesquisa bibliográfica sobre integração, Scrum e ferramentas que lhe dão suporte;

- Analisar e apresentar problemas ocorridos em projetos Scrum causados pela falta de integração;
- Criar um modelo arquitetural para integração de ferramentas para Scrum;
- Apresentar o modelo arquitetural para líderes de equipe e fazer sua avaliação qualitativa através de entrevistas.

1.4 ESTRUTURA DO TRABALHO

Este TCC está organizado em seis capítulos, sendo que no primeiro capítulo tem-se a introdução onde se contextualiza brevemente os temas: problemas relacionados ao desenvolvimento de software, metodologias ágeis e Scrum. Posteriormente são expostos os problemas de negócio e técnico, por fim, os objetivos e a metodologia do trabalho.

O segundo capítulo trata dos conceitos relacionados ao tema deste trabalho, tais como Scrum, ferramentas que dão suporte a Scrum e ao processo de desenvolvimento: ferramentas para gerenciamento de projetos, gerenciamento de testes, controle de versão e integração contínua.

No terceiro capítulo é apresentado o conceito de integração, quais as tecnologias atualmente disponíveis que permitem integração, e como a integração de ferramentas pode automatizar processos do Scrum e amenizar problemas de retrabalho e inconsistência de informações.

No quarto capítulo é apresentado a arquitetura e funcionalidades propostas por este trabalho, como solução para os problemas levantados durante essa pesquisa.

O quinto capítulo apresenta a validação da arquitetura e das funcionalidades propostas por este trabalho, por meio de entrevistas com líderes de equipe de uma empresa de desenvolvimento de software.

O sexto capítulo conclui esse trabalho e propõe possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados alguns dos principais conceitos que fundamentam este trabalho. Inicialmente, os conceitos, papéis, artefatos e atividades da metodologia Scrum. Por fim, uma breve descrição sobre as ferramentas que se adaptam no contexto da pesquisa deste trabalho.

2.1 SCRUM

De acordo com Sutherland (2014), a ideia de Scrum foi apresentada pela primeira vez por Hirotaka Takeuchi e Nonaka Ikudzhiro no artigo *The New Product Development Game* publicado na *Harvard Business Review*, onde os autores observaram o sucesso de projetos utilizando equipes pequenas, sem especialização rígida. Jeff Sutherland usou este trabalho para criar uma metodologia e implantar em sua organização, batizando-a de Scrum, e Ken Schwaber formalizou o processo para utilização na indústria de software publicando o artigo *Scrum Development Process* (SCHWABER, 1997).

Para Schwaber e Sutherland (2011), Scrum é uma estrutura simples que pode ser usada para a organização da equipe e para obter resultados de forma mais eficiente e com maior qualidade, através da análise do trabalho realizado e do ajuste do produto a partir de iterações. A metodologia permite que a equipe escolha as tarefas que devem ser cumpridas, tendo em vista as prioridades do negócio e capacidades técnicas da equipe, bem como a maneira de implementá-las efetivamente. Isso permite criar condições para que a equipe trabalhe com prazer e da forma mais produtiva possível. Por exemplo, a possibilidade de uma escolha independente do âmbito e formas de resolver problemas, sem pressão externa, permite que todos os membros da equipe se sintam “jogadores” ativos e envolvidos no processo.

O uso da metodologia faz com que seja possível identificar e corrigir desvios do resultado desejado nos estágios iniciais de desenvolvimento de software. Scrum centra-se na constante identificação de prioridades, com base nos objetivos de negócios, o que aumenta a utilidade e rentabilidade do projeto em seus estágios iniciais.

Além disso, Scrum foi projetado para assegurar a rápida adaptação as mudanças nos requisitos, permitindo que a equipe adapte rapidamente o produto as necessidades do cliente. Esta adaptação é alcançada através da obtenção de *feedback* sobre os resultados da iteração. Após cada iteração ter um produto que já pode ser usado para mostrar e discutir melhorias, faz

com que a equipe possa realizar os ajustes certos e alterar os requisitos de prioridades. Com isso as chances de existirem grandes mudanças na entrega final do produto diminuem.

2.1.1 *Sprint* – Ciclo de Desenvolvimento

No Scrum, o progresso dos projetos é definido numa série de iterações denominadas de *Sprints* e que tem duração entre 2 e 4 semanas. Uma *Sprint* inicia-se com a *Sprint Planning Meeting* (Reunião de Planejamento da *Sprint*), antes do início da reunião é preciso assegura-se que o *Product Backlog* (Lista de funcionalidades) esteja organizado e com o escopo fechado. Então o *Product Owner* se reúne com o *Scrum Team* e o *Scrum Master*, para priorizar quais itens do *Product Backlog* são mais importantes e que irão agregar maior valor ao produto no final *Sprint*.

A partir da Reunião de Planejamento da *Sprint* é gerado o *Sprint Backlog* que nada mais é do que uma parte menor do *Product Backlog* escolhidos pelo *Scrum Team* para serem produzidos durante a *Sprint*. Com o *Sprint Backlog* definido o *Product Owner* não pode adicionar mais trabalho nessa *Sprint*, porem ele tem autonomia para cancelar a *Sprint*, coisa que raramente ocorre (SCHWABER; SUTHERLAND, 2013).

Durante o andamento da *Sprint* são realizadas *Daily Meetings* (Reuniões Diárias) que duram em média 15 minutos e tem como objetivo assegurar o andamento da *Sprint*. A figura 1 apresenta, resumidamente o ciclo de uma *Sprint*.

Figura 1 - Ciclo da Sprint.
Fonte: www.thiagothamiel.com/2009/07/16/desenvolvimento-agil. [2009]

No final da *Sprint* são realizadas mais duas reuniões que são: *Sprint Review Meeting* e a *Sprint Retrospective Meeting*, que serão descritas na seção 2.1.4.

2.1.2 Papéis Scrum

Segundo Schwaber e Sutherland (2011), os papéis no Scrum são representados de 3 formas: *Product Owner, Scrum Master* e o *Scrum Team*. Cada membro assume uma responsabilidade durante o desenvolvimento do projeto, esses papéis são detalhados a seguir:

Product Owner é o dono do produto, responsável por concentrar as informações vindas dos *Stakeholders* (cliente, usuários, representantes de negócios, etc.) de modo a se obter uma visão única dos requisitos do sistema. Com isso, definir as características do produto numa lista de itens denominada de *Product Backlog*. Além de maximizar o valor do produto e do trabalho do *Scrum Team*. Ele é o único que pode gerenciar o *Product Backlog*, ou seja, é responsável por priorizar as funcionalidades que agreguem mais valor, por deixar claro os itens do *Product Backlog* para que o *Scrum Team* entenda os requisitos no nível necessário e também por aceitar ou recusar o trabalho produzido.

Scrum Master é o responsável por gerenciar o processo Scrum, devendo garantir que ele seja entendido e aplicado na forma correta por todos envolvidos no projeto. Sempre acompanhando o desenvolvimento e removendo os impedimentos, garantindo que o trabalho do *Scrum Team* seja sempre funcional e produtivo. Auxilia tanto o *Product Owner* quanto o *Scrum Team*. Junto ao *Product Owner* ele encontra técnicas para gerenciar efetivamente o *Product Backlog* e gerencia seus interesses mediante ao *Scrum Team* com o propósito de maximizar o ROI. Junto ao *Scrum Team* tem a responsabilidade de remover os obstáculos levantados, facilitar os eventos exigidos ou necessários no Scrum, além de treinar os membros a serem auto gerenciáveis e interdisciplinares.

Scrum Team também denominado de Time de Desenvolvimento, é a equipe auto organizada, auto gerenciada e multifuncional. Composta por no mínimo 5 e no máximo 9 pessoas, tendo em vista que, equipes pequenas como de 3 integrantes, diminuem a interação e resultam num menor ganho de produtividade, podendo encontrar restrições de habilidade durante a *Sprint*. Em contrapartida, equipes com mais de 9 integrantes exigem uma maior coordenação, gerando muita complexidade para o gerenciamento do processo. (SCHWABER, 2004).

2.1.3 Artefatos

Dentre os artefatos do Scrum estão o *Product Backlog* e o *Sprint Backlog* que representam as funcionalidades do projeto, além destes, também podem ser citados o *Burndown Chart* e o *Scrum Board* que são utilizados para indicar o progresso do projeto e *Sprint*. Estes artefatos serão apresentados a seguir:

Product Backlog é uma lista de funcionalidades desejadas para um produto, onde cada funcionalidade pode ser denominada de *user story*. O conteúdo dessa lista é definido e priorizado pelo *Product Owner*. Essa lista inclui tudo o que precisa ser desenvolvido para finalizar o projeto e que possa agregar valor ao negócio, sejam requisitos funcionais ou não. É importante ressaltar que cada item da lista do *Product Backlog* deve ter um valor de negócio associado (*Business Value*), para que se possa medir o ROI e priorizar a realização dos itens (PEREIRA et al, 2007). Na Tabela 1, é demonstrado um exemplo de *Product Backlog* para um sistema de gerenciamento de vagas de emprego.

Como	Quero	Para	Valor de negocio	Estimativa
Gerente de RH	Publicar novas vagas	Encontrar candidatos	50	20
Gerente de RH	Filtrar candidatos	Eliminar candidatos menos promissores	30	15
Candidato	Procurar ofertas de emprego	Candidatar-se a um emprego	50	25
Candidato	Quero filtrar vagas de emprego por categoria	Para encontrar vagas na minha área	40	20

Tabela 1 - Exemplo de Product Backlog.
Fonte: Elaborada pelo autor.

Como exemplificado na Tabela 1, o *Product Backlog* é composto por poucos campos, mas claros o suficiente para entendimento do *Scrum Team* e os demais *Stakeholders*. A primeira coluna, representa quem é o ator da funcionalidade; a segunda, qual a funcionalidade em si; a terceira, a importância da funcionalidade; a quarta, qual o valor de negócio a funcionalidade representa, e a quinta e última coluna, qual o esforço necessário para finalização dessa funcionalidade, para definição do esforço, é feita uma estimava com base na técnica de

Planning Poker[12], em que os membros do *Scrum Team* avaliam a complexidade para implementação da funcionalidade e definem uma determinada pontuação.

Sprint Backlog consiste em uma lista de itens selecionados pelo *Scrum Team* para serem desenvolvidos durante a *Sprint* atual, capaz de transformar-se em um incremento utilizável pelo cliente. Essa lista é formada a partir do *Product Backlog*, onde o *Scrum Team* escolhe os itens baseando-se nas prioridades definidas pelo *Product Owner* e nos pontos de estimativa definidos pelos *Scrum Team*. Os itens selecionados devem ser decompostos em tarefas para que mudanças no progresso possam ser entendidas na *Daily Scrum* (SCHWABER, 2013, p. 19).

Burndown Chart é um modelo gráfico utilizado para representar o esforço restante para um determinado período de tempo. No Scrum esse modelo é utilizado para representar dois gráficos: O *Release Burndown* que representa o progresso atual do projeto e o *Sprint Burndown* que representa o progresso da *Sprint*. A figura 2 demonstra um exemplo do *Burndown Chart*.

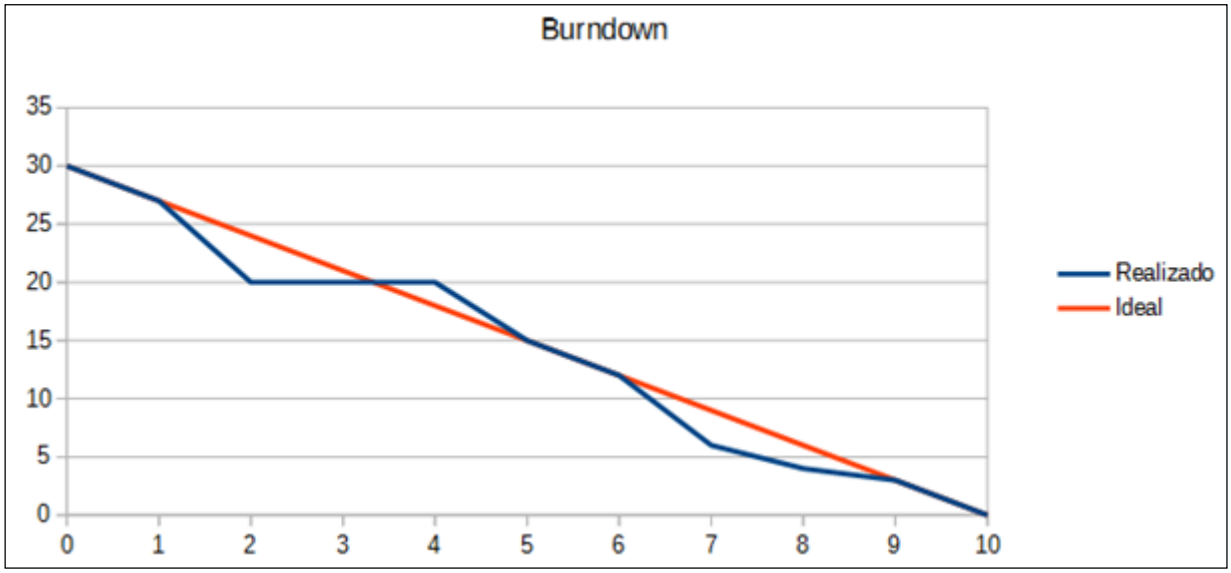

Figura 2 - Burndown Chart.
Fonte: http://www.cesar.edu.br/simplifica-burndown-chart. [2014]

O *Burndown Chart* é composto por quatro elementos básicos, sendo eles:

- Eixo X que representa o tempo (geralmente dias estimados para a *Sprint*)
- Eixo Y que representa o esforço (geralmente horas ou pontos de estimativa)
- Linha vermelha que indica o progresso ideal
- Linha azul que indica o progresso real

[12] www.planningpoker.com

Scrum Board também conhecido como Quadro de Tarefas, é uma representação visual do progresso do *Scrum Team* em relação conclusão de *user stories* de uma *Sprint*. De uma maneira geral, é um quadro onde são exibidas as *user stories* que compõem o *Sprint Backlog*, juntamente com as *tasks* (tarefas) relacionadas a cada *user story*. Geralmente, o *Scrum Board* é criado em um quadro com *post-its* ou em ferramentas especializadas como o Trello[13]. A figura 3, representa um exemplo de *Scrum Board*.

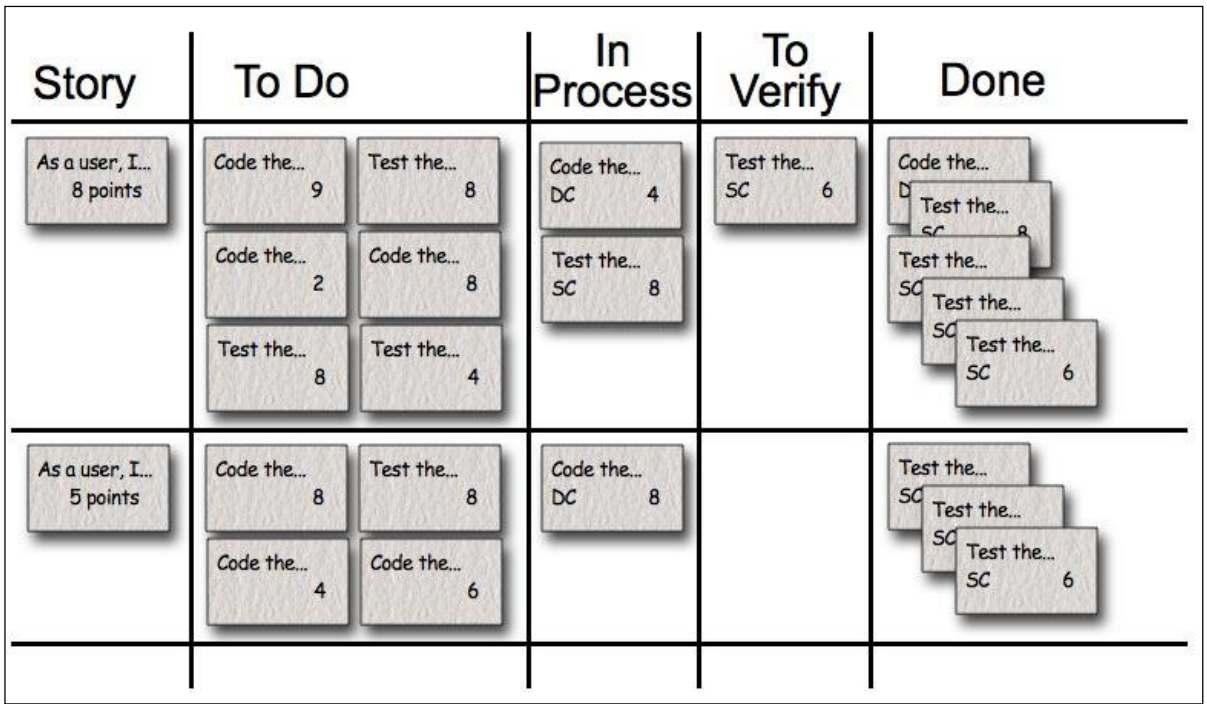

Figura 3 - Exemplo de Scrum Board.
Fonte: https://www.mountaingoatsoftware.com/agile/scrum/task-boards

Como demonstrado na Figura 3, o *Scrum Board* apresenta uma visão global do estado atual do *Sprint Backlog*, permitindo que todos possam verificar o *status* de cada *task*. Onde, cada coluna representa um estado para uma determinada *task*, sendo eles:

- *Story* - esta coluna contém uma lista de todas as *user stories* do *Sprint Backlog* atual;
- *To do* - esta coluna contém todas as *tasks* das *user stories* que ainda não foram iniciadas;
- *In Process* - todas *tasks* que já foram iniciadas;
- *To Verify* - esta coluna indica quais *tasks* estão em processo de revisão;
- *Done* - indica quais *tasks* já foram concluídas

[13] www.trello.com

2.1.4 Eventos Scrum

Segundo Schawaber e Sutherland (2013), existem quatro eventos, também denominados de “reuniões”, que organizam e garantem o andamento do processo em Scrum, sendo eles: *Sprint Planning Meeting*, *Daily Scrum Meeting*, *Sprint Review Meeting* e *Sprint Retrospective Meeting*, estes são descritos abaixo:

Sprint Planning Meeting como descrito na seção anterior é a Reunião de Planejamento da *Sprint*, essa reunião geralmente tem duração de 8 horas para uma *Sprint* de um mês, para *Sprint* com duração menor esse tempo é proporcionalmente menor. Essa reunião é dívida em duas partes:

Na primeira parte, o *Product Owner* apresenta os itens do *Product Backlog* priorizados por ele. Após isso, o *Scrum Team* realiza perguntas para se ter melhor entendimento sobre as funcionalidades, então, avalia a complexidade e escolhe quais itens do *Product Backlog* podem ser entregues no final da *Sprint*, assim, gerando o *Sprint Backlog*.

Na segunda parte, o *Scrum Team* decide como irá produzir as funcionalidades selecionadas para o *Sprint Backlog*, quebrando-as em tarefas menores. Nessa fase o *Product Owner* também deve estar presente para esclarecer possíveis dúvidas que surgirem em relação aos itens do *Sprint Backlog*.

Daily Scrum Meeting trata-se de uma reunião realizada uma vez por dia durante o andamento de cada *Sprint*. Geralmente é realizada pela manhã para ajudar a estabelecer as prioridades das tarefas durante o dia. Tem duração média de 15 minutos e todos os membros presentes devem estar “de pé”, com o intuito que a reunião seja rápida. Os principais objetivos dessa reunião são: analisar o estado atual da Sprint; discutir o foi feito no dia anterior e o que será feito no dia atual; auxiliar e incentivar a comunicação entre a equipe, a fim de identificar dificuldades e impedimentos ao desenvolvimento. Caso existam devem ser tratados e/ou removidos imediatamente pelo Scrum Master.

Sprint Review Meeting é a reunião realizada no final de cada *Sprint*, seu objetivo é analisar o incremento produzido na última *Sprint* e se necessário adaptar o *Product Backlog*. Durante a reunião o *Scrum Team* e os *Stakeholders* discutem o que foi feito na *Sprint*. Nessa reunião o *Scrum Team* deve informar quais incrementos estão “prontos” e quais os problemas ocorreram durante a *Sprint* (caso existam). Após isso, o *Product Owner* deve deixar claro quais incrementos considera “pronto” e quais não considera. Então, discutem sobre o atual progresso do *Product Backlog*, a fim de projetar prováveis datas para a conclusão do projeto.

Sprint Retrospective Meeting é a reunião realizada depois da *Sprint Review* e antes da *Sprint Planning*, e é onde o *Scrum Master* encoraja o *Scrum Team* a revisar o processo de desenvolvimento, tendo como propósito, torná-lo mais eficaz e gratificante para a próxima *Sprint.* O principal objetivo dessa reunião é analisar a última *Sprint*, levando em consideração as pessoas e a relação entre elas, assim como o processo e as ferramentas, a fim de identificar o que ocorreu bem e se feito de outra forma, poderia melhorar ainda mais. (SCHAWABER, 2009).

2.2 FERRAMENTAS PARA SUPORTE A DESENVOLVIMENTO DE SOFTWARE

Nessa seção descreveremos e exemplificaremos alguns tipos de ferramentas que podem ser utilizadas para dar suporte a projetos Scrum juntamente com o desenvolvimento de software. Tento em vista o contexto deste trabalho, alguns critérios foram considerados para seleção das ferramentas, como: a ferramenta deve oferecer suporte a metodologia Scrum ou ao desenvolvimento de software, fornecer algum tipo de integração com outras ferramentas e de preferência que a ferramenta seja *open source* (código aberto). Nas próximas subseções serão apresentadas algumas ferramentas para: gerenciamento de projetos, gerenciamento de testes, controle de versão e integração continua.

2.2.1 Ferramentas para gerenciamento de projetos

De acordo com Araújo e Tinoco (2014), “para que um projeto de software seja bem-sucedido, é necessário que alguns parâmetros sejam analisados, por exemplo, o escopo do software, os riscos, os marcos de referência, além da sistemática a ser seguida”. Um projeto de software geralmente é estruturado em uma sequência de operações para se obter um resultado final. Para isso é preciso estabelecer as etapas (atividades) e gerir recursos (pessoas, materiais, equipamentos e serviços) que serão utilizados durante a execução de cada atividade. Além disso, as metodologias utilizadas para o desenvolvimento de software sugerem vários processos e artefatos. Gerir todos recursos e o processo de desenvolvimento sem o auxílio de ferramentas, torna-se uma tarefa trabalhosa e propensa a erros.

Surge então, a necessidade de ferramentas para gerenciar os projetos, de maneira que se possa: planejar e programar atividades, otimizar os recursos e custos, de modo que, se tenha informações relacionadas e organizadas, para tornar mais fácil e clara a exposição dessas informações aos interessados.

Nas seções 2.2.1.1 e 2.2.1.2 serão apresentadas duas ferramentas que tem como propósito gerenciar projetos, incluindo funcionalidades para gerenciar os artefatos e o processo do Scrum.

2.2.1.1 Tuleap

Tuleap é uma ferramenta *open source* utilizada por milhares de usuários ao redor do mundo, inclusive por empresas de grande porte como Telecom, JTEKT-Toyota, Ericsson, Renault, Orange e Delphi. O propósito da ferramenta é melhorar o gerenciamento de projetos de software conectando todos os membros da equipe de maneira eficiente. Em 2013, recebeu o prêmio *Bossie Awards* 2013 da InforWorld.com como sendo uma das 30 melhores ferramentas *open source* para desenvolvimento de aplicativos (BORCK, 2013).

Tuleap adequa-se a metodologias ágeis, tradicionais, processos híbridos ou personalizados. Oferece suporte para planejamento, *sprints*, tarefas e relatórios. Dentro do ambiente Tuleap é possível visualizar e atualizar os artefatos através da integração direta com controle de versão, revisão de código e ferramentas de integração contínua (Git, Garrit e Jenkins).

A ferramenta também oferece algumas outras funcionalidades interessantes para gerenciamento de projetos, entre elas: opção para criação de wikis referentes a cada projeto, fóruns com tópicos divididos por projeto e *chat* online para facilitar a comunicação entre os membros.

Além disso é uma solução interessante para projetos que utilizem Scrum, pois, oferece opções para criação de *Product/Sprint Backlog* e para priorização de *user stories*. Apresenta também uma implementação do *Scrum Board* para gerenciar os itens do *Product/Sprint Backlog* de forma simples, dando opção para fragmentar as *user stories* em *tasks*. A figura 4 ilustra como o Tuleap apresenta o *Scrum Board*.

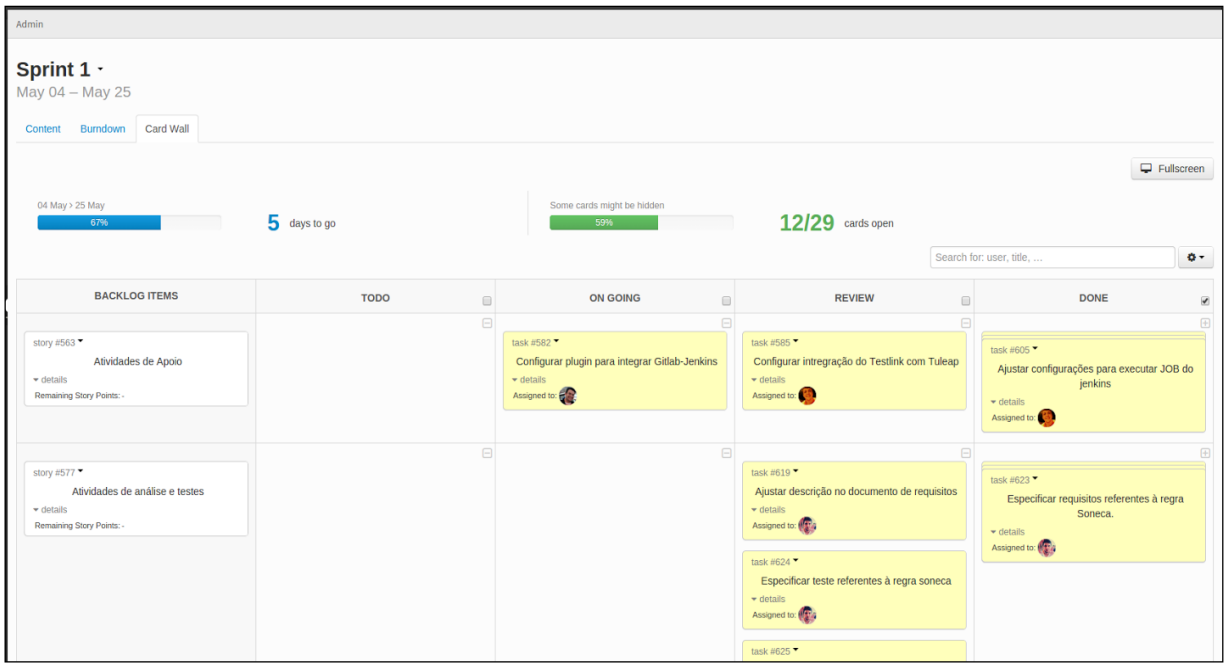

Figura 4 - Scrum Board do Tuleap.
Fonte: *Print screen* da ferramenta Tuleap v.8 [2015]

No *Scrum Board* apresentado pelo Tuleap, as *user stories* podem conter pontos de estimativa que permitem gerar o *Release/Sprint Burndown*. Onde, o *Burndown* é gerado e atualizado automaticamente pelo Tuleap, baseando-se na alteração dos pontos de estimativa definidos nas *user stories.* A figura 5 ilustra como o Tuleap apresenta o *Sprint Burndown*.

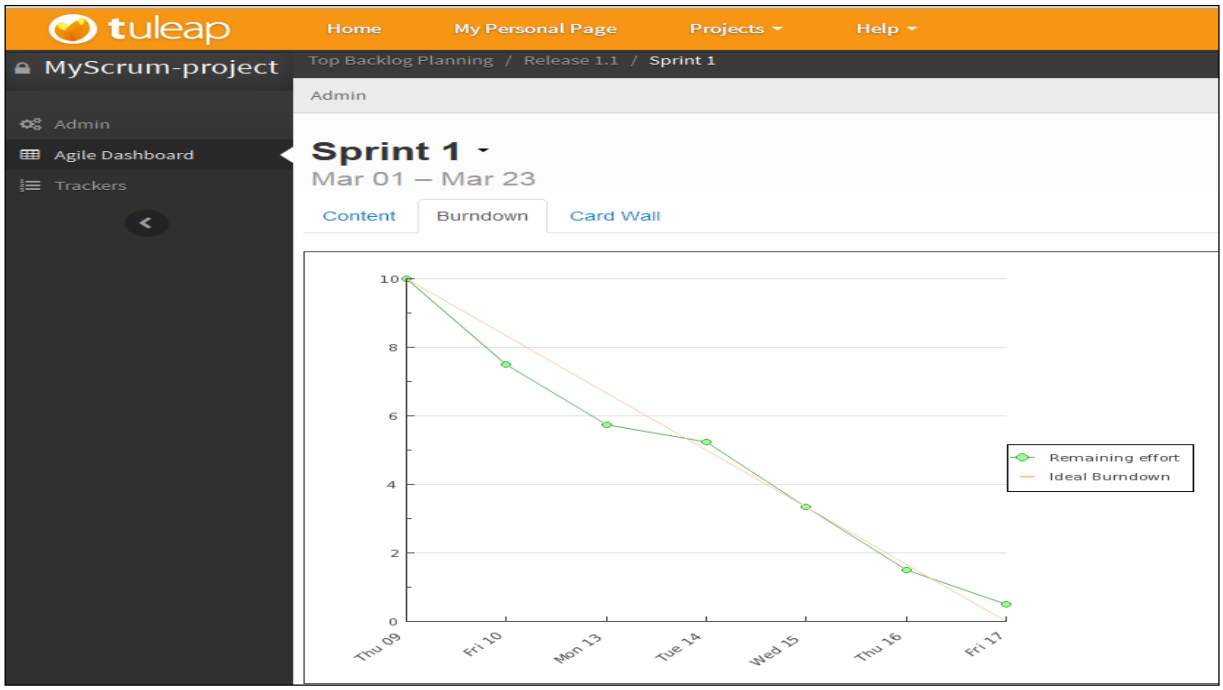

Figura 5 - Sprint Burndown do Tuleap.
Fonte: *Print screen* da ferramenta Tuleap v.8 [2015]

2.2.1.2 Redmine

Assim como o Tuleap, Redmine é uma ferramenta web, *open source* para gerenciamento de projetos e acompanhamento de *bugs*. Possui diversas funções para gerenciamento de projetos, além de, um sistema modular e flexível que permite personalização através de campos definidos pelo usuário.

Além disso, Redmine permite gerenciar vários projetos ao mesmo tempo, com a capacidade de atribuir papéis e compartilhar informações entre vários usuários com diferentes privilégios de acesso, ou seja, o usuário pode ter um papel para cada projeto. A ferramenta também oferece opções para criação de projetos de modo privado ou público, dessa forma, caso o projeto seja criado em modo público é possível compartilhá-lo com pessoas que estão fora do grupo de trabalho.

A ferramenta é equipada com vários módulos que podem ser ativados ou desativados no nível do projeto de acordo com as necessidades dos usuários. Dentre as principais características apresentadas pelo Redmine, essas são as que mais destacam-se:

- Gerenciamento de múltiplos projetos;
- Gerenciamento de usuários com controle de acesso;
- Acompanhamento de *bugs*;
- Calendários e gráficos de Gantt;
- Gerenciamento de documentos e arquivos;
- Notificações por e-mail;
- Gestão de wikis;
- Fóruns de discursão;
- *Time Tracking*;
- *SCM Management*.
- API REST para integração com outras ferramentas.

2.2.2 Ferramentas para controle de versão

Segundo Chacon e Straub (2009), sistemas de controle de versão sugiram para servir a vários propósitos fundamentais. Primeiro de tudo, eles permitem que o desenvolvedor possa armazenar com segurança sucessivas versões do código-fonte. Além de fornecer uma cópia de *backup* do código, este garante que o desenvolvedor possa retroceder para uma versão estável caso algo dê errado. Além disso, sistemas de controle de versão ajudam os membros da equipe

a desenvolver simultaneamente no mesmo código fonte de um projeto, oferecendo maneiras de contornar problemas de conflitos no código fonte.

Por ser uma metodologia, Scrum não informa como deverá ser feito o gerenciamento do código fonte do projeto. Entretanto, como boa prática de desenvolvimento de software, é recomendado que o código fonte seja mantido em sistemas de controle de versão como: CVS, SVN, Git, a fim de evitar problemas de conflitos, perca de arquivos e etc.

Nas próximas subseções serão apresentadas duas ferramentas para controle de versão de código fonte.

2.2.2.1 GitHub

GitHub é o maior serviço web para hospedagem de Projetos de TI da atualidade (CHACON e STRAUB, 2009). Foi concebido para oferecer aos desenvolvedores a chance de compartilhar seus projetos e desenvolvê-los em conjunto com outros desenvolvedores. A ferramenta é desenvolvida com base no sistema de controle de versão Git, mas desde de 2010, oferece suporte ao sistema de controle de versões SVN.

O serviço fornece funcionalidades para criação de wikis, acompanhamento de *bugs*, visualização de arquivos com destaque para a sintaxe da maioria das linguagens de programação, criação ilimitada de repositórios públicos (com código aberto) e para repositórios privados é oferecido diferentes tarifas de pagamento.

Além disso o serviço também oferece o *GitHub Enterprise*[14], uma versão da ferramenta voltada para empresas e que pode ser hospedada em servidores locais, fornecendo suporte para autenticação corporativa por meio de LDAP e com um sistema próprio de *backup*.

2.2.2.2 GitLab

Segundo Voort e Vosamaer (2014), GitLab é um grande projeto *open source* mantido por mais de 600 colaboradores e atualizado mensalmente. Utilizado principalmente para hospedar repositórios e estabelecer estruturas de controle de versão. Atualmente é utilizado por mais de 100.000 organizações ao redor o mundo, incluindo O'Reilly Media, Centro de Pesquisas Jülich, Alibaba, CERN e NASA (DEGELER, 2014).

Possui servidores para gerenciamento de repositórios baseados em Git. É altamente escalável, pois um único servidor GitLab pode lidar com mais de 25.000 usuários, mas também

[14] https://enterprise.github.com/

é possível criar uma configuração de alta disponibilidade com vários servidores ativos, aumentando ainda mais o número de usuários.

Dentre as principais funcionalidades que a ferramenta oferece, abaixo, estão as que mais se destacam:

- Gerenciamento de repositórios baseados em Git;
- Revisões de código e comentários na linha do código;
- Acompanhamento de problemas (*Issues*);
- Criação de wikis;
- Integração contínua e implantação com o GitLab CI;
- Visualização de fluxos de atividades referentes a projetos e pessoas;
- Autenticação via LDAP;
- Integração com outras ferramentas (Slack, Hipchat, Jira, Jenkins);
- Execução em servidor local;
- API REST para integração com outras ferramentas.

2.2.3 Ferramentas para gerenciamento de testes

Segundo Caetano (2012), o objetivo geral do gerenciamento de testes é permitir que as equipes possam planejar, desenvolver, implementar e avaliar todas as atividades de testes durante todo o processo de desenvolvimento do software. Incluindo medidas para coordenar todas as ações envolvidas no processo de rastreamento, dependências e relacionamentos entre os ativos de teste e, mais importante, definir, medir e acompanhar os indicadores de qualidade.

O gerenciamento de testes pode ser dividido em diferentes fases: organização, planejamento, criação, execução e apresentação de relatórios. Abaixo estas fases são apresentadas em maior detalhe:

- Organização dos artefatos e recursos de teste é sem dúvida uma parte necessária da gestão de testes, ou seja, é preciso organizar dos itens para testes, juntamente com os artefatos utilizados para realização dos testes. Isso cria uma forma de rastrear as dependências e relações da equipe de desenvolvimento entre os ativos de teste.
- Planejamento de teste é um conjunto completo de tarefas, respondendo a perguntas do tipo: por quê, o quê, onde e quando testar. A razão para a criação de testes é conhecida como motivador dos testes. O que deve ser testado é dividido em muitos *test cases* (casos de teste) para o projeto. Para responder à pergunta onde testar, é definido e

documentado a configuração necessária de hardware e software. A pergunta, quando testar, é respondida de acordo com a evolução do produto produzido.

- Realizar testes implica na execução dos testes, combinando as sequências de casos de teste, com o intuito de identificar problemas no software.
- Relatórios de teste responde à pergunta de como os vários resultados de teste são analisados e relacionadas entre si. De forma que, esses relatórios sejam utilizados para determinar o estado atual do projeto, bem como a qualidade global do sistema.

Nas próximas subseções serão apresentadas duas ferramentas que dão suporte ao gerenciamento de testes.

2.2.3.1 TestLink

De acordo com Fleisher et al (2012), TestLink é uma das mais populares ferramentas *open source* para gerenciamento e execução de testes. Suas funcionalidades incluem especificação de testes, planejamento, elaboração de relatórios e rastreamento de requisitos. A interface é multilíngue e baseada em web, ou seja, pode ser executada em qualquer *browser* que ofereça suporte para JavaScript, XHTML e CSS, o que facilita a manutenção e upgrades da ferramenta, tendo em vista, que instalações do cliente não são mais necessárias.

TestLink gerencia testes independente da tecnologia, ou seja, oferece apoio a várias linguagens de programação (Java, Python) e sistemas de banco de dados (MySQL, PostgreSQL, MS-SQL). Fornece opção para integração com as principais ferramentas para rastreamento de *bugs*, como o Bugzilla, Mantis e Youtrack. Também oferece suporte a autenticação interna e externa por meio do LDAP, incluindo controle de acesso flexível baseado em papéis com vários níveis de permissão. Por ter um repositório centralizado para todos os casos de teste e resultados, facilita a manutenção de vários projetos ao mesmo tempo.

Além disso, TestLink fornece alguns recursos para comunicação, incluindo a capacidade de gerar relatórios em diversos formatos, como, HTML, CSV, MS Word e MS Excel. Os relatórios podem ser gerados com base nos resultados dos planos de teste. Por exemplo, as execuções geram relatórios inteligentes e podem ser visualizados em forma de gráficos. TestLink é propriedade de TeamTest, uma forte comunidade aberta de testadores internacionais e possui uma equipe de desenvolvimento ativa, fornecendo guias de usuário e de desenvolvimento.

Para resumir, TestLink apresenta uma estrutura flexível para o gerenciamento de processos de teste e garante a conformidade com os padrões de processos de teste, conforme especificado pelo IEEE 829 e BCS SIGIST (FLEISHER et al, 2012).

2.2.3.2 Tarantula

Tarantula é a ferramenta *open source* para gerenciamento de testes de software em projetos ágeis. A ferramenta gerencia os requisitos, planejamento, execução e apresentação de relatórios de *test cases*. Oferece aos gerentes de projeto uma visão em tempo real para o status atual dos testes. Permite integração direta com Jira, Bugzilla e Doors, além de fornecer uma API REST para integração com outras ferramentas.

A ferramenta é baseada em testes de projeto, o que significa que o usuário pode selecionar a determinada *release* do projeto que deseja visualizar e testar. Apresenta diversas métricas para o projeto, como: nível de exigência de cobertura, cobertura de testes, casos que falharam e o progresso testado.

Segundo o site[15] da ferramenta, Tarantula pretende ser a melhor ferramenta *open source* para gerenciamento de testes, especialmente em: *agile testing*, gerenciamento de testes, relatórios e usabilidade. Entre suas principais características estão:

- Gerenciamento de casos de teste, execução e requerimentos;
- *Tags* e *SmartTags* para testes;
- Visualização em tempo real dos resultados de testes;
- Histórico de versões dos casos de teste;
- Rastreabilidade de requisitos para *bugs*;
- Exportação de relatórios em PDF;
- Importação de casos de testes de outras ferramentas;
- Integração direta com as ferramentas Jira, Bugzilla e Doors;
- API REST para integração com outras ferramentas;

2.2.4 Ferramentas para integração continua

De acordo com Fowler (2000), Integração Continua "é uma prática de desenvolvimento de software onde membros de uma equipe integram seu trabalho frequentemente. Normalmente cada pessoa integra pelo menos diariamente, levando a vários atos de integração por dia", onde

[15] http://www.tarantula.fi/

essas integrações geram *builds* (testadas), facilitando a detecção de erros de forma mais rápida, já que que cada integração é verificada automaticamente.

Para Smart (2011), Integração Contínua, "em sua forma mais simples, envolve uma ferramenta que monitora o sistema de controle de versão para as mudanças". Dessa forma, sempre que é detectada alterações no código, a ferramenta automaticamente compila e testa o software. Quando detecta algum erro, a ferramenta notifica os responsáveis pelo software para que eles possam corrigir o problema o mais rápido possível.

Além disso, esse tipo de ferramenta ajuda a manter o controle sobre o estado atual do código, monitorando automaticamente a qualidade do código e métricas de cobertura de testes, assim, ajudando a diminuir os custos de manutenção. Combinado com os testes de aceitação automatizados, Integração Contínua, pode atuar como uma ferramenta de comunicação, publicando uma imagem do estado atual do desenvolvimento. E isso pode simplificar e acelerar entrega, ajudando a automatizar o processo de implantação, fazendo com a que a implantação da versão mais recente do software seja gerada basicamente com um clique.

Embora a metodologia Scrum não exija que seja realizada Integração Contínua durante o desenvolvimento do projeto, o conceito de Integração Contínua, encaixa-se muito bem ao contexto de Scrum. Tendo em vista que, ao final de cada *Sprint* precisa se ter uma *build* gerada e testada, para que possa ser apresentada ao *Product Owner* e aos demais *Stakeholders.*

Na próxima subseção será apresentada uma ferramenta que implementa o conceito de Integração Continua.

2.2.4.1 Jenkins CI

Jenkins é uma ferramenta para Integração Continua, assim como a maioria das ferramentas apresentadas neste trabalho, Jenkins também é uma ferramenta *open source*. Em 2010, tornou-se solução de Integração Contínua líder de mercado, com uma fatia de mais de 70% dos usuários. Amplamente reconhecida como uma ferramenta fundamental para desenvolvedores, engenheiros de controle de qualidade, gerentes de projeto, engenheiros de lançamentos, entre outros. Usada por equipes de todos os tamanhos, para projetos com grande variedade de linguagens e tecnologias, incluindo Java, Python, PHP, Ruby entre outras (SMART, 2011).

Além disso, Jenkins é bastante flexível e adaptável em relação a *plugins*, inclusive atualmente existem centenas de *plugins* de código aberto para melhorar ainda mais a Integração Continua. Esses *plugins* dão cobertura a diversas funcionalidades, desde sistemas de controle

de versão, ferramentas de compilação, métricas de qualidade de código, análise estática, testes, integração com sistemas externos e etc.

2.3 RESUMO DO CAPÍTULO

Neste capítulo foi descrito o histórico e conceito da metodologia Scrum, bem como seus princípios, papéis, artefatos e eventos, visando fornecer uma visão teórica da metodologia. Além disso, foram apresentadas ferramentas que dão suporte a metodologia Scrum e ao desenvolvimento de software, como: ferramentas para gerenciamento de projetos, gerenciamento de testes, controle de versão de código e integração contínua.

No próximo capítulo será apresentado o conceito de integração, as principais tecnologias que permitem integração entre ferramentas, e como a falta de integração entre ferramentas geram problemas durante o gerenciamento de projetos Scrum.

3 INTEGRAÇÂO

De acordo com a definição no dicionário português, integração é “a combinação de partes que trabalham isoladamente, formando um conjunto que trabalha como um todo”. O termo integração adequa-se a diversos contextos, entretanto, nesta nesse capítulo iremos descrever o conceito de integração apenas no contexto de integração de ferramentas de software.

3.1 INTEGRAÇÃO DE FERRAMENTAS

Entende-se integração de ferramentas como o desenvolvimento de soluções integradas para processos de automação e de negócios. Seu objetivo é integrar diferentes ferramentas com a intenção de compartilhar informações entre elas, com isso, ter um sistema mais amplo e que possa atender uma quantidade maior de requisitos.

Quando há integração entre ferramentas, o valor da informação é maximizado, e principalmente, evita-se esforços em tarefas de coordenação e organização dessas informações, além de duplicação de trabalho. Desta forma, é possível realocar recursos para realização de outras tarefas.

3.1.1.1 Tecnologias para integração de ferramentas

Atualmente existem diversas tecnologias que permitem a integração de ferramentas, entretanto, nesta seção será descrito apenas as que mais se destacaram durante a pesquisa desse trabalho.

- **REST** (*Representational State Transfer*) é um modelo arquitetural proposto para troca de mensagens entre sistemas utilizando o protocolo HTTP. REST “é capaz de atender requisitos complexos de integração sem agregar complexidade e excesso de acoplamento” (LUSA; KRAEMER, 2014). Tornou-se famoso principalmente pela sua simplicidade, que faz com que a troca de informações entre sistemas não consuma tantos recursos, tornando-se ideal especialmente para sistemas baseados em Web. Além disso, permite a interoperabilidade entre aplicativos escritos em linguagens diferentes e executados em plataformas diferentes o que facilita a integração.
- **SOAP** (*Simple Object Access Protocol*) é um protocolo baseado em XML-RPC. Como seu antecessor, usa XML para codificar os dados e HTTP como o protocolo para transmitir mensagens. SOAP oferece incorporação de segurança baseada em certificados digitais, conexões seguras e criptografia. Além de também permitir a

interoperabilidade entre aplicativos escritos em linguagens diferentes e executados em plataformas diferentes. Boa parte das linguagens de programação atuais têm tem suporte para implementação de SOAP por padrão, além de ser uma das tecnologias mais utilizados para implementar protocolos de Web Services (SNELL et all, 2001).

- **CORBA** (*Common Object Request Broker Architecture*) é um padrão que fornece uma plataforma de desenvolvimento para sistemas distribuídos, facilitando a invocação de método remoto em um paradigma orientado a objetos. Assim como as tecnologias anteriores, CORBA também permite a interoperabilidade entre aplicativos escritos em linguagens diferentes e executados em plataformas diferentes (LINK et al, 2009).

3.1.1.2 Problemas para integrar ferramentas

Apesar da integração de ferramentas propor soluções, integrá-las geralmente não é uma tarefa trivial, levando em consideração que: parte das ferramentas são projetadas para trabalhar de forma isolada, ou seja, não fornecem nenhuma forma de integração com outras ferramentas. Além de que, sistemas de software são desenvolvidos com padrões e tecnologias diferentes, quando oferecem algum tipo de integração geralmente é limitada a uma tecnologia especifica, impossibilitando a integração devido a conflitos de tecnologia.

Um outro problema, igualmente relevante, é a falta de um padrão universal para comunicação entre as ferramentas, ou seja, os padrões existentes implementam a troca de parâmetros de maneiras diferentes, o que dificulta a comunicação e integração entre as ferramentas.

3.2 INTEGRAÇÃO DE FERRAMENTAS PARA GESTÃO DE PROJETOS E AUTOMAÇÃO DE PROCESSOS SCRUM

Como dito anteriormente, Scrum é uma metodologia ágil, entretanto, durante a gestão de projetos Scrum, ainda existem muitos problemas relacionados ao retrabalho, o que acaba prejudicando a “agilidade” da metodologia, mas que podem ser resolvidos ou amenizados com a integração de ferramentas.

Nas próximas subseções será apresentado, como a integração de ferramentas pode ajudar a automatizar processos e a diminuir problemas de retrabalho e inconsistência de informações durante o desenvolvimento de projetos Scrum.

3.2.1 Manter projetos

Um problema comum durante o gerenciamento de projetos Scrum é o retrabalho causado quando é necessário manter (criar, atualizar, deletar) projetos, pois, essa tarefa requer que o mesmo trabalho seja refeito em todas a ferramentas auxiliares. Por exemplo, ao criar um projeto na Ferramenta de Gerenciamento de Projetos (FGP) é necessário criar com os mesmos dados um projeto na Ferramenta de Gerenciamento de Testes (FGT), um repositório na Ferramenta de Controle de Versão (FCV) e *pipeline*(s) na Ferramenta de Integração Continua (FIC). Essa tarefa além de causar retrabalho, pode causar inconsistência, pois, os dados do projeto contidos em uma ferramenta podem ser diferentes dos dados contidos nas outras ferramentas.

Caso exista um sistema que possa integrar essas ferramentas, esses problemas podem ser resolvidos, ou seja, o projeto é mantido nesse sistema, e esse encarregasse de replicar a manutenção do projeto nas ferramentas integradas. Desse modo, o trabalho de manter projetos nas ferramentas é feito uma única vez, e as chances das informações do projeto estarem inconsistentes são menores, visto que, essas informações são “espelhos” das informações contidas no sistema.

3.2.2 Manter usuários

Os mesmos problemas de retrabalho e inconsistência de informações ocorridos na manutenção de projetos, também ocorrem quando é necessário manter usuários em ferramentas não integradas. Visto que, ao manter um usuário em uma ferramenta também é preciso manter esse mesmo usuário nas outras ferramentas. Com um sistema que possa integrar as ferramentas, esses problemas também podem ser solucionados, sendo necessário apenas manter o usuário nesse sistema, e o mesmo fica responsável por replicar as informações do usuário nas ferramentas integradas a ele.

3.2.3 Manter usuários em projetos

Assim como acontece nas tarefas de manutenção de projetos e usuários. Problemas de retrabalho também se repetem quando é necessário manter um usuário em um projeto, isto é, sempre que é preciso associar/remover um usuário a um projeto, é necessário realizar a mesma tarefa em todas as outras ferramentas. Inclusive podendo ocasionar problemas de inconsistência, por exemplo: um usuário é removido de um projeto na FGP, mas continua associado ao repositório do projeto na FCV. Esses problemas também podem ser evitados, caso a manutenção de usuários nos projetos seja feita em um único sistema.

3.2.4 Verificar o estado do projeto

Outro problema identificado durante a pesquisa desse trabalho é quando o *Scrum Master* precisa verificar o estado do projeto através das ferramentas auxiliares. Por exemplo, é necessário que ele consulte a FGP para verificar o andamento da *Sprint* através do *Burndown*; verificar quais *user stories* foram concluídas ou ainda falta concluir, e qual a capacidade do *Scrum Team* em relação à conclusão de *user stories*. Precisa consultar a FGT para verificar a quantidade de testes realizados, quais funcionalidades os testes estão cobrindo e quais foram os *bugs* encontrados. Além de precisar consultar a FIC para verificar a qualidade do código através da análise estática realizada pela ferramenta, e quais *builds* foram geradas durante a *Sprint*.

Com um sistema, que possa recuperar e compilar as informações contidas nessas ferramentas, esse problema pode ser resolvido, pois, o *Scrum Master* teria todas as informações em um único lugar, dando uma visão global do projeto. Dessa forma, facilitando a identificação de possíveis problemas de forma mais rápida e clara.

3.2.5 Alteração nos *status* de *tasks* do *Scrum Board*

Como descrito na seção 2.1.3, o *Scrum Board* apresenta uma visão geral do andamento da *Sprint* baseada na conclusão de *tasks* das *user stories*, porém, para que essa visão esteja de acordo com o andamento da *Sprint* é necessário que os membros do *Scrum Team* atualizem os *status* das *tasks*. Uma boa prática de desenvolvimento de software e que se adéqua bem a ideia do *Scrum Board* é o *Git Workflow*, que recomenda que cada *task* de uma *user story* seja desenvolvida em uma determinada *branch* da FCV. Nesse processo, também é recomendado que ao terminar a implementação da funcionalidade o desenvolvedor crie um *merge request* na FCV, para que o código desenvolvido na *task* seja revisado por um outro desenvolvedor, a fim de verificar possíveis melhorias no código e também para que o conhecimento seja disseminado entre os membros do *Scrum Team*. A figura 6 representa o processo para atualização do *status* das *tasks* no *Scrum Board* baseado no *Git Workflow*.

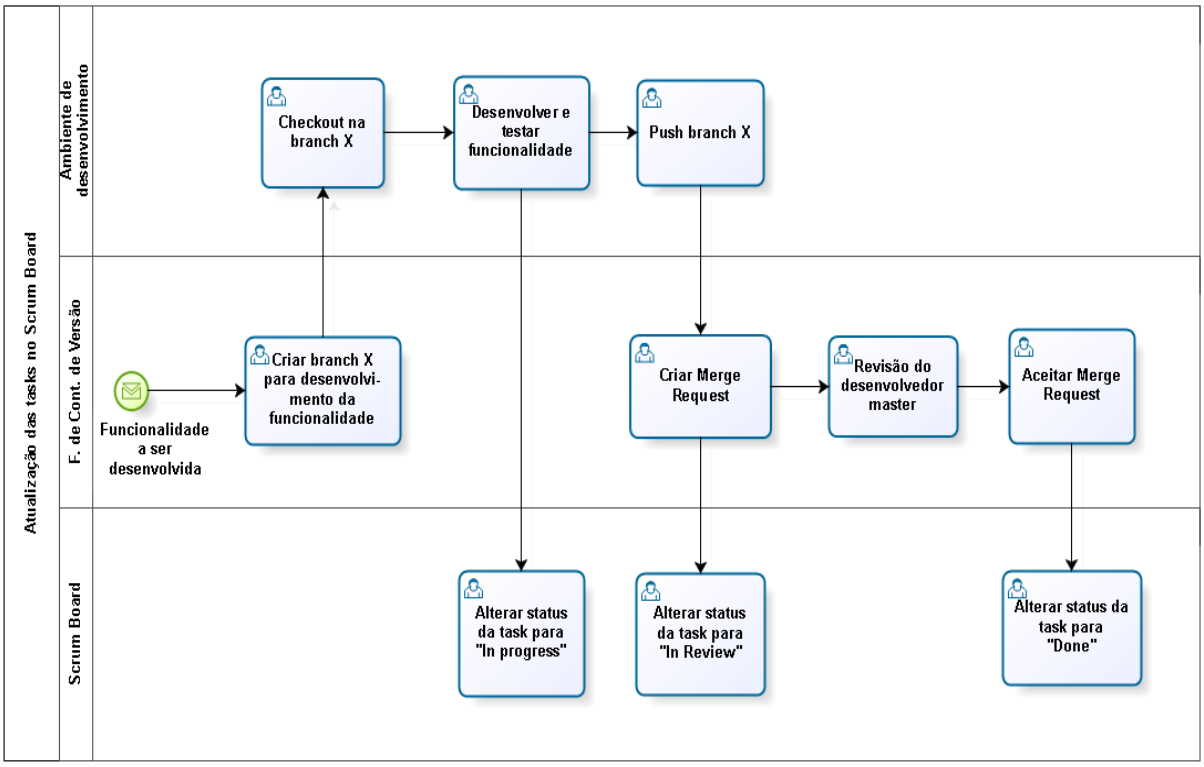

Figura 6 - Processo para atualização do status da tasks no Scrum Board.
Fonte: Elaborada pelo auto

Como apresentado na figura 6, o processo para atualização do *status* das *tasks* no *Scrum Board* inicia-se quando um membro do *Scrum Team* começa a implementar uma funcionalidade, nesse momento ele deve alterar o *status* da *task* relacionada a funcionalidade de "*To Do*" para "*In Progress*", dessa forma, indicando para os demais membros do *Scrum Team* que a *task* já foi inicializada. Ao terminar de implementar e testar a funcionalidade o desenvolvedor deve criar um *merge request* na FCV e nesse momento ele também deve alterar o *status* da *task* de "*In Progress*" para "*In Review*" indicando que a *task* está apta para ser analisada por um outro desenvolvedor. Por fim, caso o *merge request* tenha sido aceito, o desenvolvedor deverá alterar o status da *task* para "*Done*", dessa forma, sinalizando que a funcionalidade desenvolvida na *task* foi concluída e já está presente no código fonte principal do projeto.

Entretanto, um problema comum durante o desenvolvimento de projetos Scrum, é os membros *Scrum Team* esquecerem de atualizar o *status* das *tasks* no *Scrum Board*, o que acaba desencadeando uma série de outros problemas, como: o *Sprint Burndown* ficar inconsistente devido à falta de atualização, assim como, a capacidade do *Scrum Team* pode não ser estimada da forma correta.

Como solução parcial para esse problema as FCV emitem notificações baseadas em eventos (por meio de *webhooks*), por exemplo: foi criada uma *branch*, foi criado um *merge request*, foi adicionado um comentário ao *merge request* ou o *merge request* foi aceito. Dessa forma, é possível automatizar o processo de atualização do *Scrum Board* com base nesses eventos. Sendo necessário apenas que exista um sistema que possa interceptar esses eventos e que faça requisições a FGP para alterar os *status* das *tasks* no *Scrum Board*. A solução para automatização desse processo será apresentada no próximo capítulo.

3.2.6 Execução de *pipelines* da FIC

Como descrito na seção 2.2.4, Integração Contínua também é uma boa pratica para o desenvolvimento de projetos de software e que em conjunto com *Git Workflow* torna-se um processo bem interessante, visto que, a cada *merge request* criado na FCV os desenvolvedores validam o código produzido por meio da execução da *pipeline* de desenvolvimento na FIC. Entretanto, como primeiro critério para aceitação do *merge request*, é que o relatório gerado pela FIC demonstre que código desenvolvido esteja dentro dos parâmetros de qualidade e testes definidos pela equipe. Após a execução da *pipeline* de desenvolvimento, o desenvolvedor deve postar o relatório gerado pela FIC em forma de comentário no *merge request* criado na FCV e só depois disso o *merge request* está apto para ser verificado pelo desenvolvedor *master* do *Scrum Team*. Caso o *merge request* seja aceito, o desenvolvedor precisa executar a *pipeline* de integração de código, com o objetivo de verificar se após a junção do seu código com o código principal do projeto não gerou problemas ou "quebrou" algum teste. Caso a *pipeline* de integração acuse algum problema ou que algum teste falhou, deve ser registrado um *bug* na FGT, para que seja resolvido o quanto antes. A figura 7 representa como esse processo é realizado.

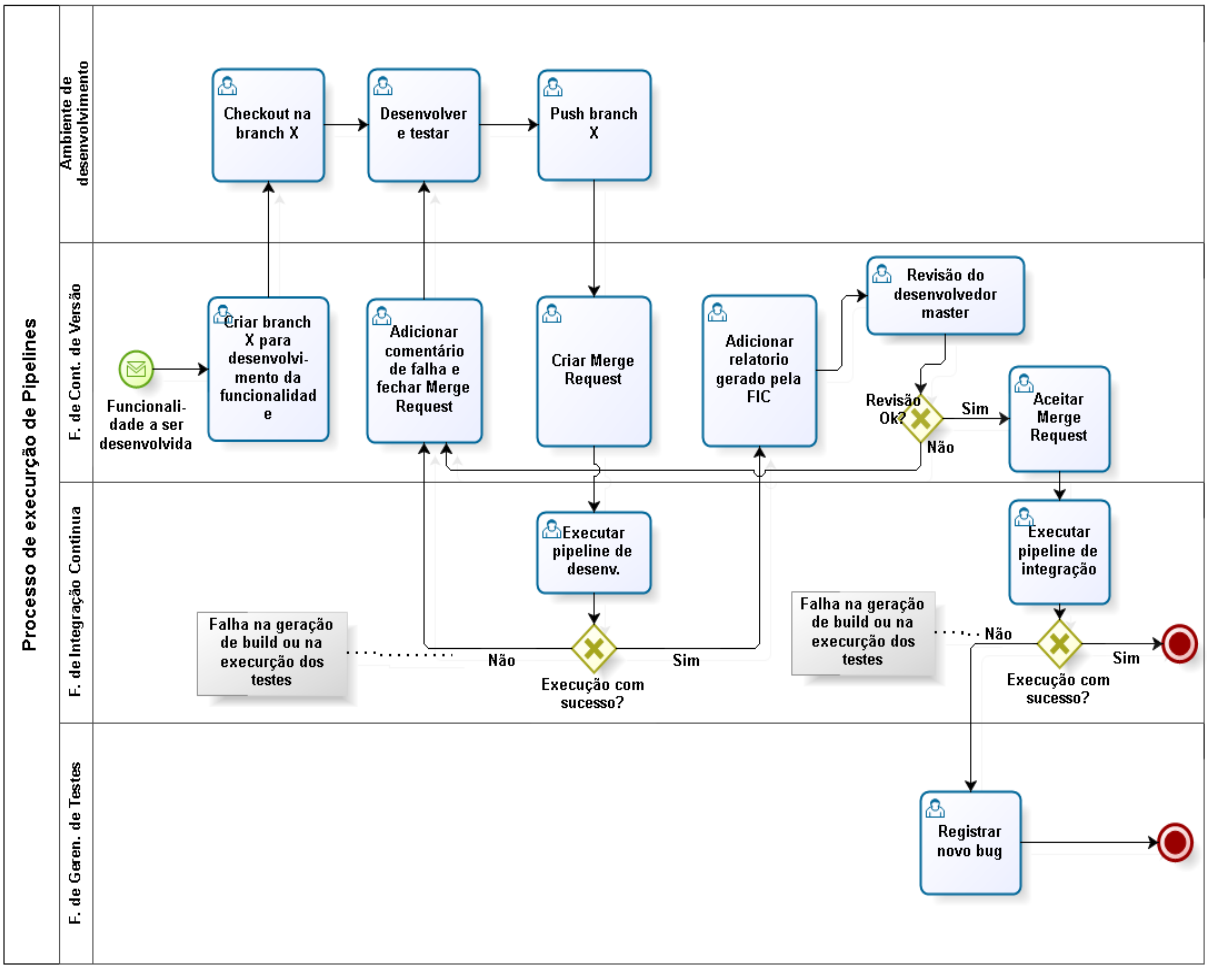

Figura 7 - Processo da execução manual de pipelines.
Fonte: Elaborada pelo autor

Como podemos observar na figura 7, todo o processo é realizado pelo desenvolvedor de forma manual, o que faz com que ele fique ocioso enquanto espera pelos resultados gerados pela FIC. Como mencionado anteriormente, as FCV geram notificações baseadas em eventos do tipo: foi criado um *merge request* ou o *merge request* foi aceito. Além disso, a FIC também gera notificações ao terminar a execução de alguma *pipeline*. Ou seja, é possível interceptar as notificações dessas ferramentas e usá-las para automatizar alguns desses processos. Porém é necessário que exista um sistema que receba uma notificação de uma ferramenta e realize alguma requisição na outra. A solução para automatização desse processo será descrita no capítulo a seguir.

3.3 RESUMO DO CAPÍTULO

Neste capítulo foi apresentado o conceito de integração, bem como as principais tecnologias que permitem integração entre ferramentas. Após isso, foi descrito e exemplificado

como a falta de integração entre ferramentas pode gerar problemas de retrabalho, inconsistência de informações e omissão de processos/artefatos durante o gerenciamento de projetos Scrum.

No próximo capítulo será apresentada a arquitetura e as funcionalidades propostas por este trabalho como forma de solucionar/amenizar os problemas causados pela falta de integração de ferramentas

4 SOLUÇÃO PROPOSTA

Neste capítulo, apresentaremos a arquitetura proposta, que tem como objetivo integrar ferramentas que dão suporte a metodologia Scrum, reduzindo o retrabalho, a inconsistência e a omissão de informações causadas pela falta de integração.

4.1 ARQUITETURA PROPOSTA

De uma maneira geral, a arquitetura proposta por este trabalho é dada da seguinte forma: o sistema é formado por um cliente Web (*front-end*), responsável por fazer as requisições do sistema, um servidor (*back-end*) responsável por receber e tratar as requisições feitas pelo *front-end*, e caso seja necessário, repassá-las para as ferramentas integradas ao sistema.

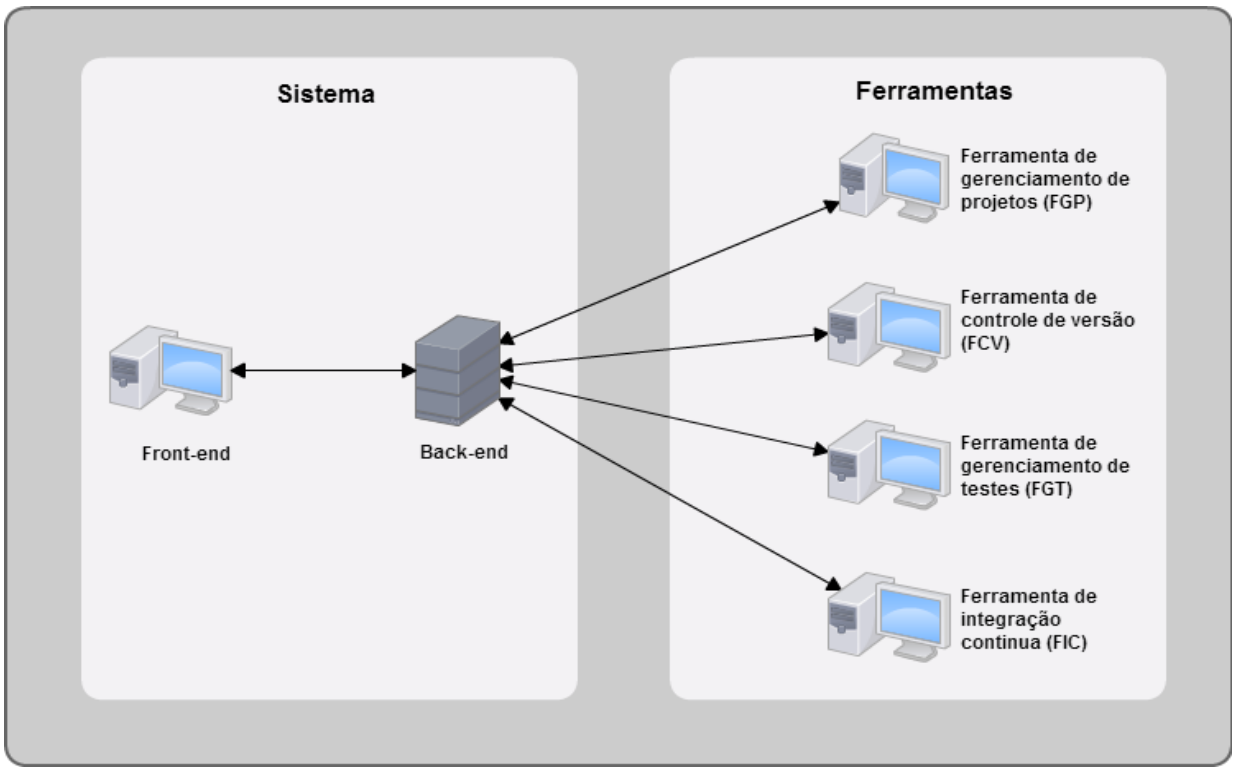

Figura 8 - Visão geral da arquitetura proposta.
Fonte: Elaborada pelo autor.

Como demonstrado na figura 8, percebe-se que as ferramentas não interagem entre si, ou seja, uma ferramenta não se comunica com as outras, apenas com o *back-end* do sistema. Assim, é evitado o acoplamento entre as ferramentas, facilitando a troca de uma ferramenta por outra. Por exemplo, suponhamos que ao final de uma *Sprint*, a equipe de desenvolvimento deve cadastrar uma *build* na Ferramenta de Gerenciamento de Projetos (FGP) como resultado de uma

Sprint. Após isso, a mesma *build* deve ser cadastrada na ferramenta de gerenciamento de testes (FGT) para que a equipe de qualidade possa testa-la. Neste caso, a arquitetura propõe que ao invés da FGP cadastrar a *build* diretamente na FGT, ela deve disparar uma requisição para o sistema (por meio de *webhooks* ou outra tecnologia de integração), e o sistema deve encarregar-se de cadastrar a *build* na FGT. Dessa forma, caso o Gerente de Projetos deseje usar como FGP o Tuleap ao invés de Redmine, será necessário apenas implementar os métodos de integração proposto pela arquitetura.

Além disso, a arquitetura proposta tende a resolver problemas de integração entre as ferramentas, levando em consideração que, as ferramentas podem implementar maneiras diferentes de integração, o que pode impossibilitar a integração direta entre elas. Por exemplo, a ferramenta Tuleap disponibiliza integração via SOAP, já a ferramenta Gitlab via REST, assim, a integração entre ambas tornar-se inviável (veja a Figura 9).

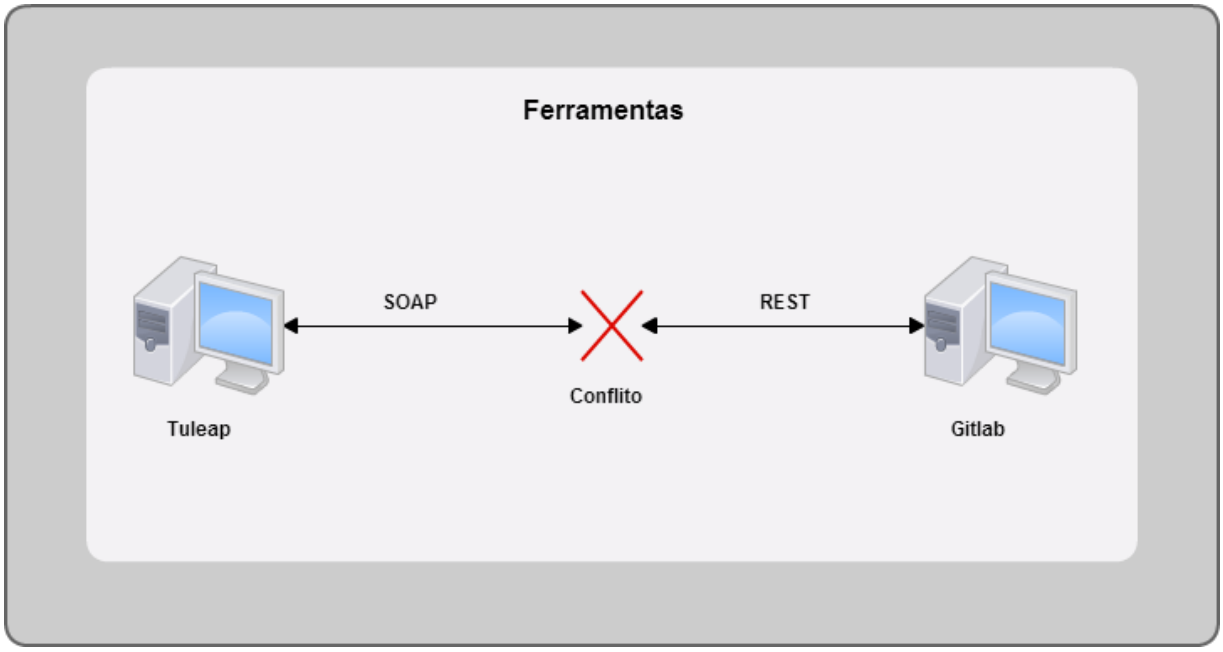

Figura 9 - Problema de integração (Conflito de tecnologia).
Fonte: Elaborada pelo autor.

Na figura 9, podemos observar que a integração entre as ferramentas Tuleap e Gitlab é impossibilitada devido ao conflito de tecnologias. Dessa forma, sendo necessário um intermediário entre ambas, que neste caso é o sistema proposto, que servirá como “tradutor” entre ambas.

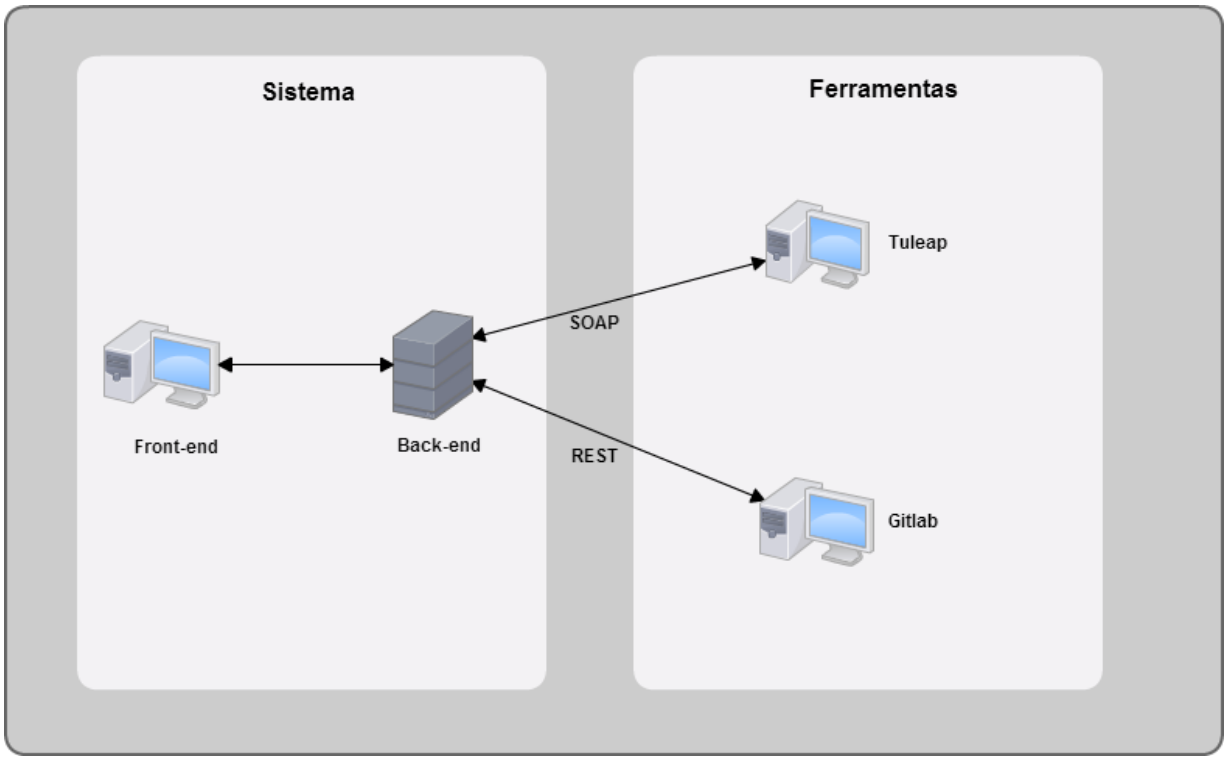

Figura 10 - Solução para o problema conflitos de tecnologia.
Fonte: Elaborada pelo autor.

Como demonstrado na figura 10, o sistema proposto servirá como um adaptador entre as ferramentas. Dessa forma, permitindo a integração entre ferramentas que disponibilize integração através de tecnologias diferentes, como é o caso de Tuleap e Gitlab.

4.2 FUNCIONALIDADES PROPOSTAS

Nas próximas subseções serão apresentadas algumas funcionalidades que complementam a arquitetura proposta e que tem objetivo de reduzir problemas de retrabalho, inconsistência e omissão de informações e processos em projetos que utilizam a metodologia Scrum.

4.2.1 Instalação e configuração de ferramentas

Um dos primeiros problemas enfrentados pelas equipes Scrum é a instalação e configuração do ambiente. Principalmente no início de projetos, quando não há um ambiente já instalado e configurado, as equipes precisam pesquisar por maneiras de montar esse ambiente, o que geralmente demora horas ou até mesmo dias. E quando não há sucesso, pode levar até a omissão de artefatos ou processos pospostos pelo Scrum, visto de que, gerenciar o artefato ou processo de maneira manual torna-se uma tarefa trabalhosa e propensa a erros.

Por esse motivo, propomos que a instalação e configuração das ferramentas que dão suporte a Scrum seja realizada pelo sistema. A sequência de passos para a instalação das ferramentas é representada da seguinte forma (Figura 11).

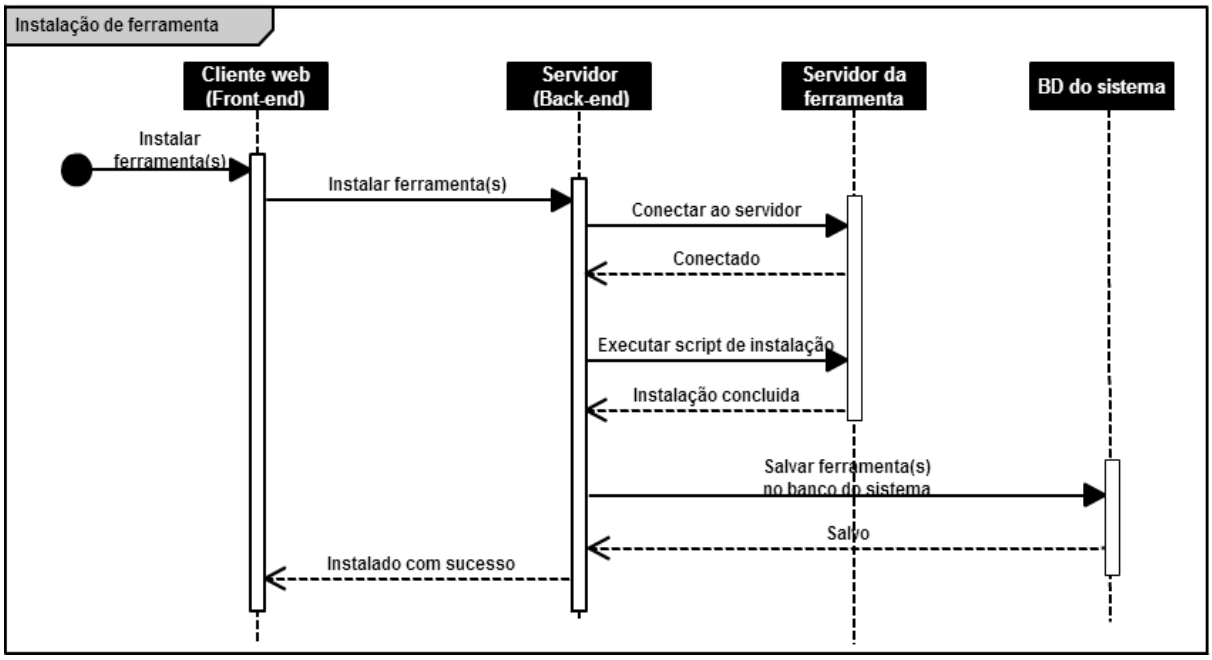

Figura 11 - [Diagrama de Sequência] Instalação de ferramentas.
Fonte: Elaborado pelo autor.

Como demonstrado na figura 11, o usuário requisita a instalação da ferramenta pelo *front-end*, passando alguns dados como: IP e dados de autenticação na máquina onde será realizada a instalação, além do tipo da ferramenta que deseja instalar (FGP, FCV, FGT ou FIC). Então, o *front-end* repassa requisição ao *back-end* do sistema, e esse último encarrega-se de se conectar à máquina onde será instalada a ferramenta e de executar o *script* de instalação de acordo com o tipo de ferramenta informado. Por fim, os dados referentes a ferramenta são salvos no banco de dados do sistema e o usuário é informado que a ferramenta foi instalada.

4.2.2 Manter projetos

Como mencionado na seção 3.2.1, manter projetos em ferramentas sem integração, causam problemas de retrabalho e inconsistência de informações. Como solução para esses problemas a arquitetura propõe que a manutenção de projetos seja feita em um único sistema,

onde, esse sistema encarrega-se de replicar a manutenção nas ferramentas integradas a ele. A figura 12 representa o diagrama de sequência para criação de um projeto pelo sistema.

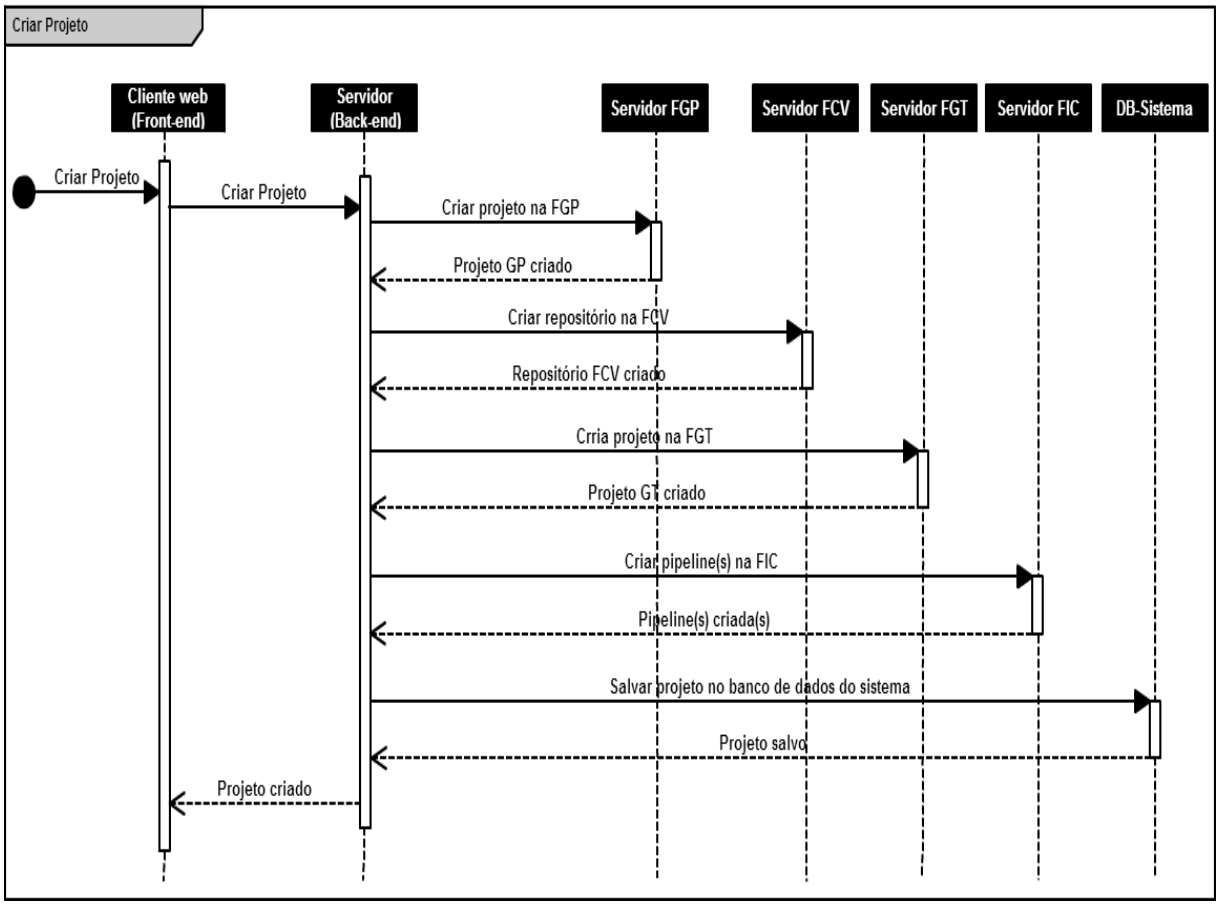

Figura 12 - [Diagrama de Sequência] Criar projeto.
Fonte: Elaborado pelo autor.

O processo para criar um projeto pelo sistema, inicia-se quando o usuário faz a requisição pelo *front-end* informando algumas informações referentes ao projeto, então o *front-end* faz a requisição ao *back-end* passando as informações do projeto e esse executa a tarefa de replicar essa requisição nas ferramentas integradas a ele. Por fim o sistema salva as informações do projeto em seu banco de dados e informa ao usuário que o projeto foi criado em todas as ferramentas.

4.2.3 Manter usuários

Como solução aos problemas descritos na seção 3.2.2, a arquitetura também recomenda que a manutenção de usuários seja feita a partir de um único sistema, desse modo, é evitado o retrabalho e as chances de inconsistência nos dados do usuário são menores. A figura 13 demonstra o diagrama de sequência para a criação de um usuário através da arquitetura proposta.

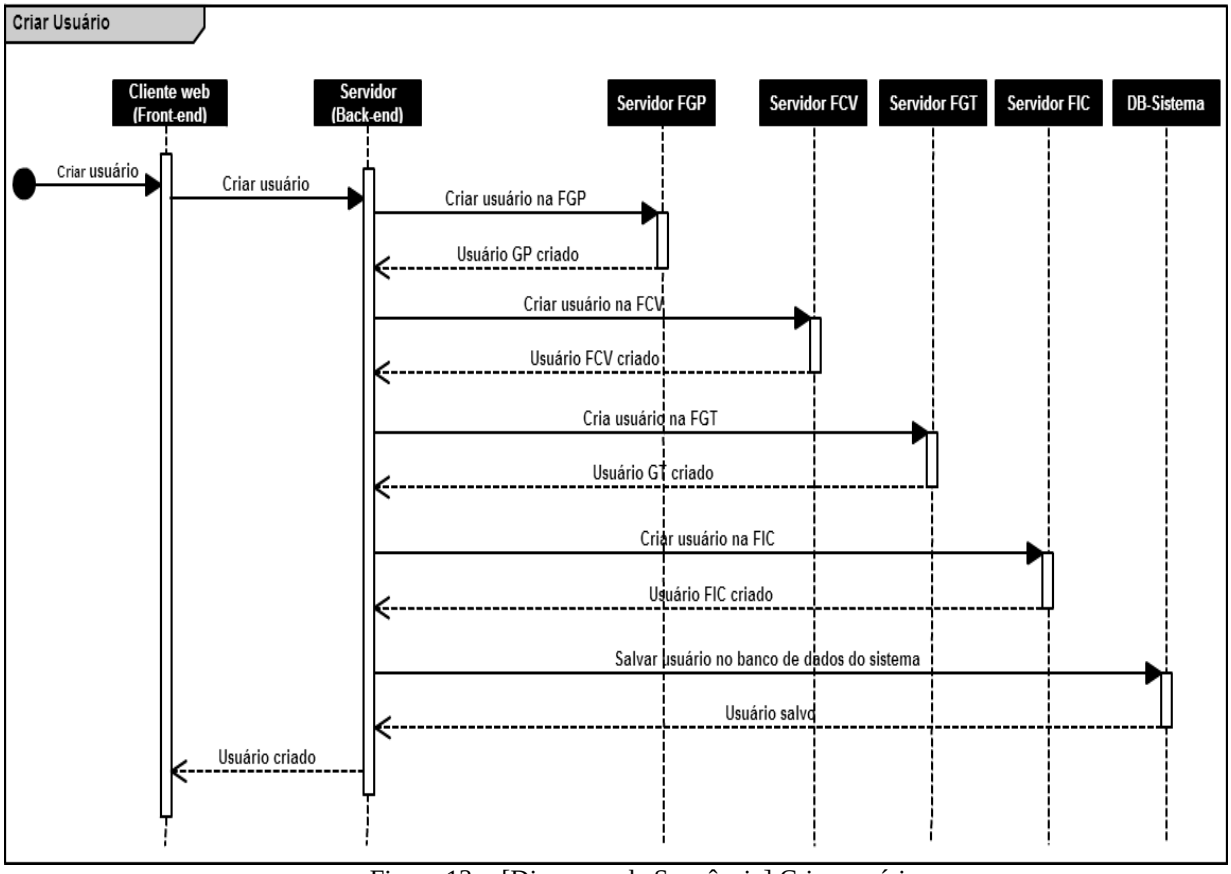

Figura 13 - [Diagrama de Sequência] Criar usuário.
Fonte: Elaborada pelo autor.

4.2.4 Manter usuários em projetos

Pensando em reduzir os problemas de retrabalho e inconsistência de informações descritos na seção 3.2.3. A arquitetura também propõe que a tarefa de manter usuários em projetos seja realizada através do sistema que implementar essa arquitetura (Observe a figura 14).

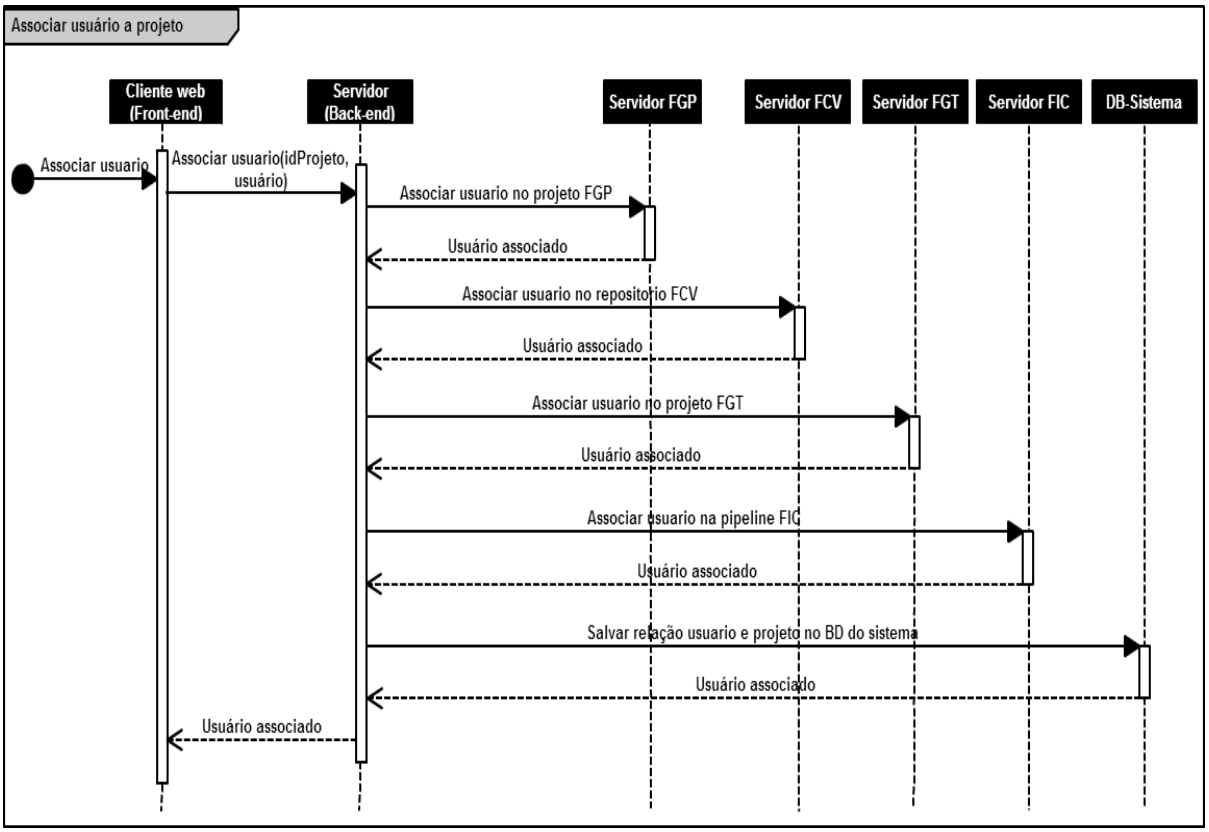

Figura 14 - [Diagrama de Sequência] Associar usuário a projeto nas ferramentas.
Fonte: Elaborada pelo autor.

A figura 14 ilustra a funcionalidade de associar um usuário a um projeto, o processo para isso inicia-se quando um usuário do sistema faz uma requisição no *front-end*, informando um outro determinado usuário e em qual projeto se deseja associar este usuário. Então, o *front-end* repassa a requisição ao *back-end*, e esse encarrega-se de fazer requisições para associar o usuário a projetos das ferramentas integradas ao sistema. Após isso, o *back-end* salva a relação entre usuário e projeto no banco de dados do sistema. Por fim, o usuário que que fez a requisição é informado que a associação foi realizada.

4.2.5 Verificar estado do projeto com base nas ferramentas integradas

Como descrito na seção 3.2.4, existe um problema de retrabalho quando necessário verificar o estado do projeto através das ferramentas auxiliares. Pois é preciso acessar mais de uma ferramenta para se ter informações sobre s*prints, user stories, Burndown*, cobertura de testes, *bugs*, qualidade do código, *builds* geradas entre outras informações. Como solução a esse problema, propomos uma funcionalidade que recupera essas informações nas ferramentas integradas ao sistema proposto. Dessa forma, é preciso acessar um único sistema para verificar essas informações, assim, se tem uma visão geral do projeto o que pode facilitar a identificação de possíveis problemas de forma mais rápida e clara.

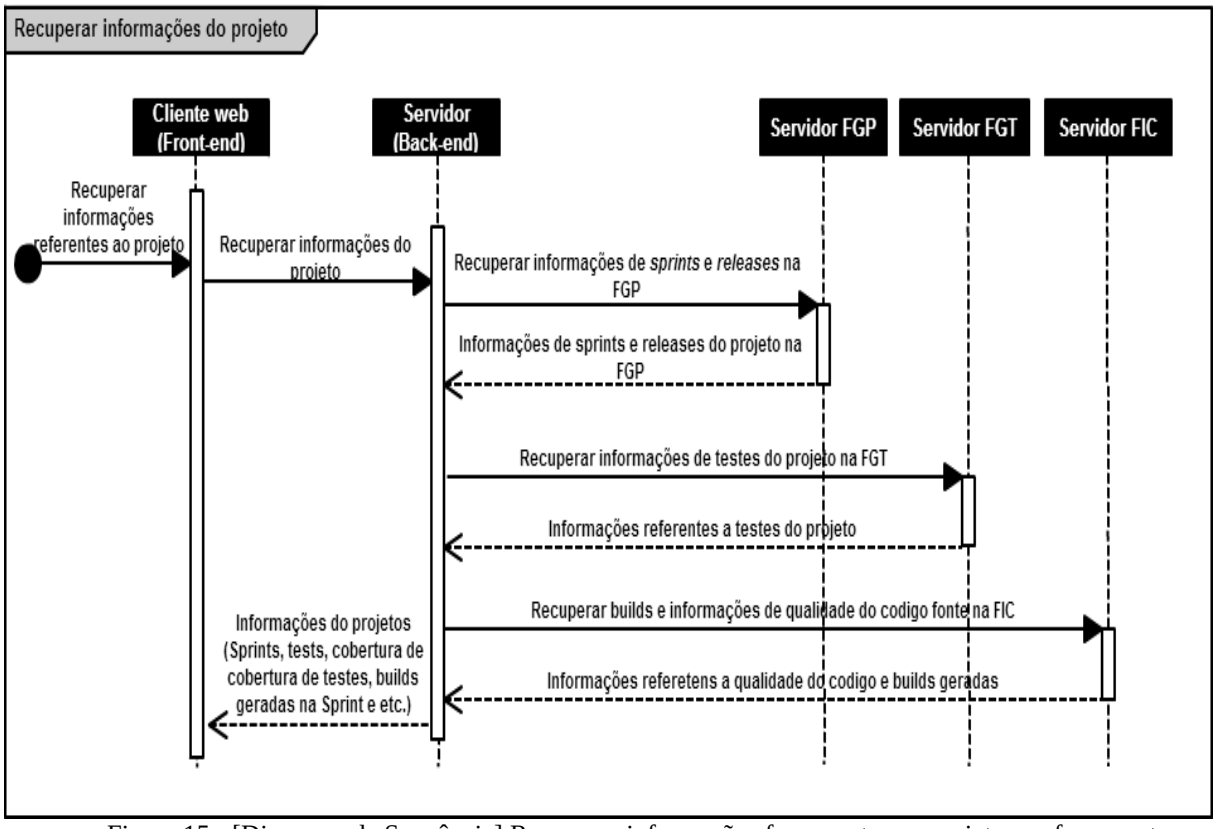

Figura 15 - [Diagrama de Sequência] Recuperar informações ferramentas ao projeto nas ferramentas.
Fonte: Elaborada pelo autor.

Como demonstrado na figura 15, o usuário faz uma requisição no *front-end*, a fim de recuperar informações referentes a um determinado projeto, então, a requisição é repassada para o *back-end*, e esse encarregasse de recuperar as informações sobre s*prints*, *user stories*, testes, *bugs*, *builds* e etc., nas ferramentas integradas ao sistema. Por fim, as informações são processadas pelo *back-end* e apresentadas ao usuário no *front-end*. Vale ressaltar que essas informações são recuperadas diretamente das ferramentas integradas e não são mantidas no banco de dados do sistema proposto. Dessa forma, as informações apresentadas serão sempre as mais atualizadas, o que evita problemas de inconsistência de informações.

4.2.6 Atualização automática do status das tasks no *Scrum Board* baseado em eventos gerados pela FCV

Como descrito na seção 3.2.5, o fato dos desenvolvedores esquecerem de atualizar os *status* das *tasks* no *Scrum Board* geram alguns problemas, como: o *Sprint Burndown* ficar inconsistente e a capacidade do time pode ser estimada erroneamente. Sabendo que as FCV geram notificações baseadas em eventos do tipo: foi criada uma *branch*, foi criado um *merge request* ou foi aceito um *merge request*. Propormos uma funcionalidade que intercepte essas

notificações e faça requisições na FGP para alterar os *status* das *tasks* de acordo com a notificação recebida. A figura 16 exemplifica como o processo para atualização das tasks do Scrum Board é realizado.

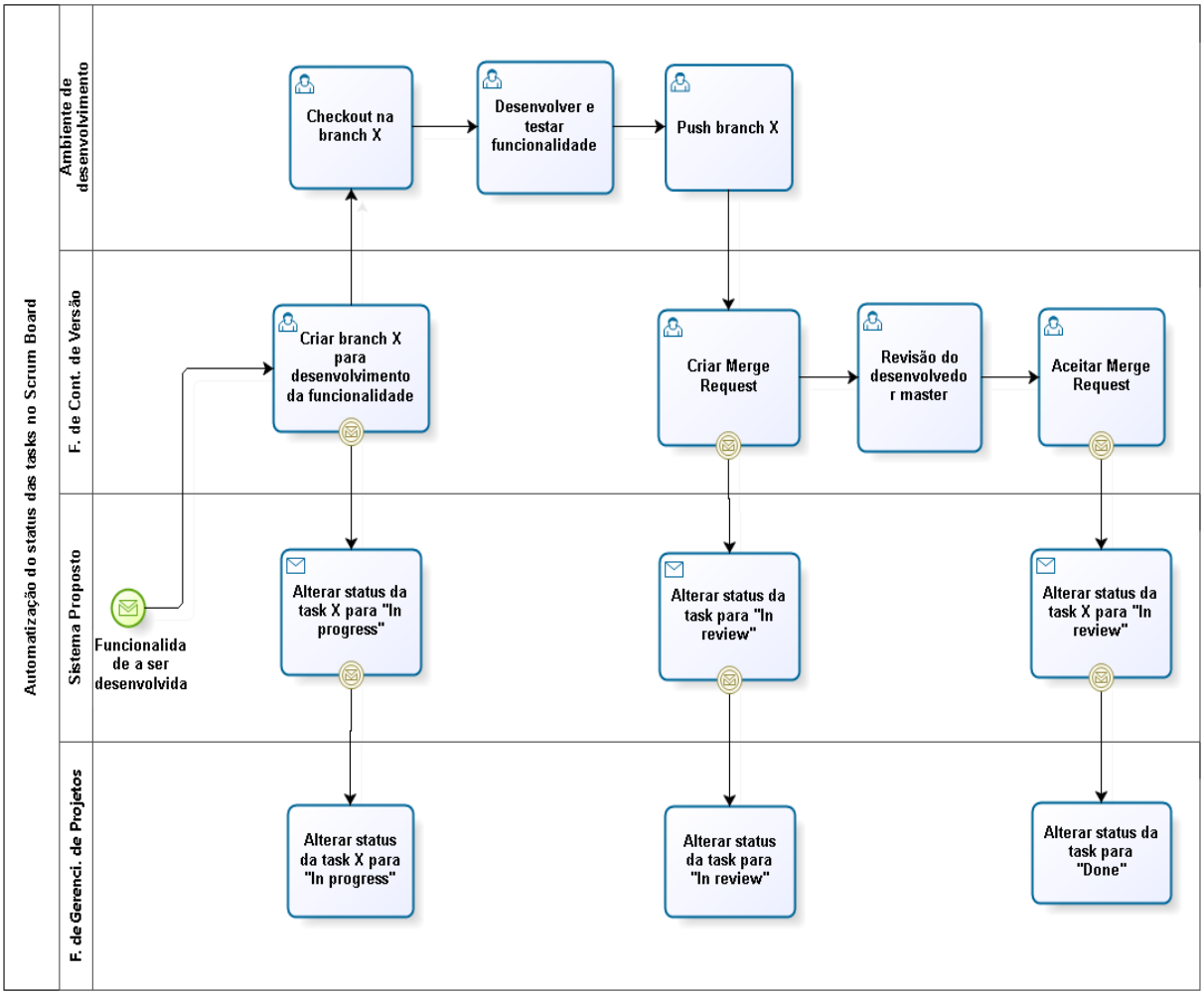

Figura 16 - Atualização automática do status das tasks no Scrum Board.
Fonte: Elaborada pelo autor.

Como demonstrado na figura 16, ao criar uma *branch* na FCV é gerada uma notificação sobre a criação da *branch*, então, o sistema proposto intercepta essa notificação e faz uma requisição a FGP para que o *status* da *task* referente a *branch* seja alterado de "To Do" para "*In Progress*". Posteriormente, no momento em que é criado um *merge request* é gerada uma outra notificação pela FCV, o sistema intercepta a notificação e faz outra requisição a FGP para alterar o status da task para *"In Review"*. Por fim, quando o *merge request* é revisado e aceito por um membro do *Scrum Team*, a FCV gera mais uma notificação e novamente o sistema proposto intercepta essa notificação e faz uma requisição a FGP para alterar o status da *task* para "*Done*". Dessa forma, é evitado problemas causados pela falta de atualização do *Scrum*

Board, e os membros do *Scrum Team* não precisam mais se preocupar com a atualização do *status* das *tasks*, já que isso é feito de forma automatizada pelo sistema proposto.

4.2.7 Automatização na execução de *pipelines* da FIC baseados em eventos gerados pela FCV

Como descrito na seção 3.2.6, Integração Continua é uma boa pratica a ser utilizada durante o desenvolvimento do projeto, levando em consideração que, os desenvolvedores validam o código produzido por meio de execuções de *pipelines* na FIC. Por outro lado, existe um *overhead* durante a execução das *pipelines* que faz com os desenvolvedores fiquem ociosos enquanto aguardam a finalização da execução das *pipelines*. Sabendo que as FCV emitem notificações baseadas em eventos do tipo: foi criado um *merge request* ou foi aceito um *merge request*, e que as FIC também geram notificações do tipo: foi finalizada a execução de uma *pipeline*. Propomos uma funcionalidade que intercepta essas notificações, com o intuito de automatizar o processo na execução de *pipelines*. Dessa forma, os desenvolvedores não precisam mais executar a *pipelines* de forma manual e nem ficar esperando a finalização da execução das *pipelines* para poderem postar os resultados gerados pela FIC no *merge request* criado na FCV. A figura 17, apresenta como é a automatização desse processo.

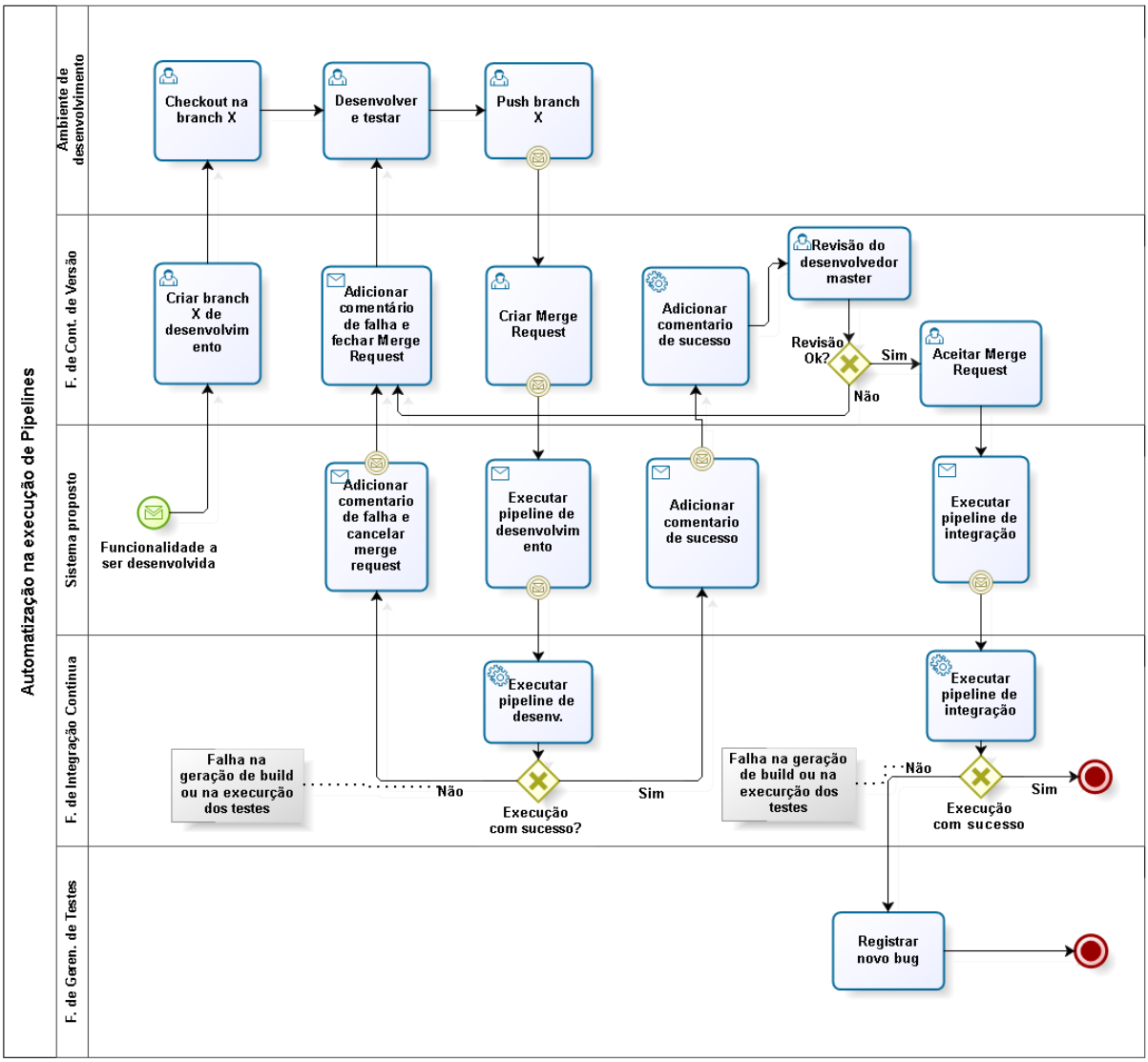

Figura 17 - Automatização na execução de pipelines.
Fonte: Elaborada pelo autor

Como demonstrado na figura 17, quando o desenvolvedor cria um *merge request*, a FCV emite uma notificação informando que foi criado um *merge request*, então, o sistema proposto intercepta essa notificação e faz uma requisição na FIC para executar a *pipeline* de desenvolvimento, no final da execução a FIC gera um relatório, e emite uma notificação informando que a execução da *pipeline* terminou, logo, o sistema intercepta o evento da FIC e caso a execução tenha falhado, é feita uma requisição para a FCV para adicionar um comentário de falha fechar o *merge request*, por outro lado, caso a execução da *pipeline* tenha sucesso é adicionado um comentário no *merge request* com o relatório gerado pela FIC. Após isso, o *merge request* é revisado por um outro desenvolvedor do *Scrum Team*, e caso esteja dentro dos paramentos definido pela equipe o *merge request* é aceito, nesse momento, é gerada uma outra notificação pela FCV informando que o *merge request* foi aceito, e novamente o sistema

proposto intercepta a notificação e requisita a FIC que execute a *pipeline* de integração, com o intuito de identificar se após o *merge request* ter sido aceito não foi gerado algum problema no código principal do software.

4.3 RESUMO DO CAPÍTULO

Neste capítulo foi apresentada a arquitetura e as funcionalidades propostas por este trabalho como forma de solucionar/amenizar os problemas causados pela falta de integração de ferramentas em projetos que utilizam a metodologia Scrum.

No próximo capítulo será apresentada a validação da arquitetura proposta por esse trabalho, com base em entrevistas e questionários com líderes de equipe de uma empresa de desenvolvimento de software.

5 VALIDAÇÃO DA ARQUITETURA PROPOSTA

Neste capítulo, será apresentado a validação da arquitetura e das funcionalidades propostas por este trabalho. Para isso foi feito uma pesquisa de ambiente, com o intuito de apresentar o contexto onde a pesquisa foi realizada. Após isso, será apresentada uma validação qualitativa realizada por meio de entrevistas e questionários(Apêndice A) com líderes de equipe do Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded), que se trata de um laboratório de pesquisa e desenvolvimento para vários projetos, com aproximadamente 200 funcionários e que possui histórico vasto de parcerias com empresas tecnológicas de grande porte, tais como: ASUS, Sony, Samsung, AOC, LG, Petrobrás, Toshiba, Compal, DL, Ford entre outras. Onde, cada parceria gera uma ou mais equipes dentro do laboratório. Essas equipes são responsáveis por produzir software para diversas plataformas (*desktop*, *web, mobile* e etc.), e todas as equipes utilizam Scrum como metodologia de desenvolvimento de software.

Vale ressaltar que já está em fase de desenvolvimento um sistema que implementa a arquitetura e as funcionalidades propostas por este trabalho, mas por questão do escopo da pesquisa, e também por limitações relacionadas ao tempo para desenvolvimento, o sistema não foi apresentado neste trabalho. Portanto, a validação deste trabalho será com base na arquitetura e nas funcionalidades propostas.

5.1 CONTEXTO DA PESQUISA

Nessa seção serão apresentados os líderes de equipes, detalhando há quanto tempo eles trabalham com Scrum, que papéis eles já assumiram, quais ferramentas para gestão de projetos, controle de versão, gerenciamento de testes e integração contínua suas equipes utilizam e quais práticas de Scrum e desenvolvimento ágil eles adotam.

5.1.1 Os entrevistados

As entrevistas foram feitas com 18 líderes que coordenam equipes com no mínimo 4 e no máximo 36 pessoas (média de 10,95 pessoas por equipe) e que trabalham com Scrum a no mínimo 2 e no máximo 100 meses (média de 33,1 meses). É importante destacar que os entrevistados assumiram diferentes papéis durante o período em que trabalharam com Scrum. Na figura 18 são apresentados os papéis que os entrevistados assumem ou já assumiram.

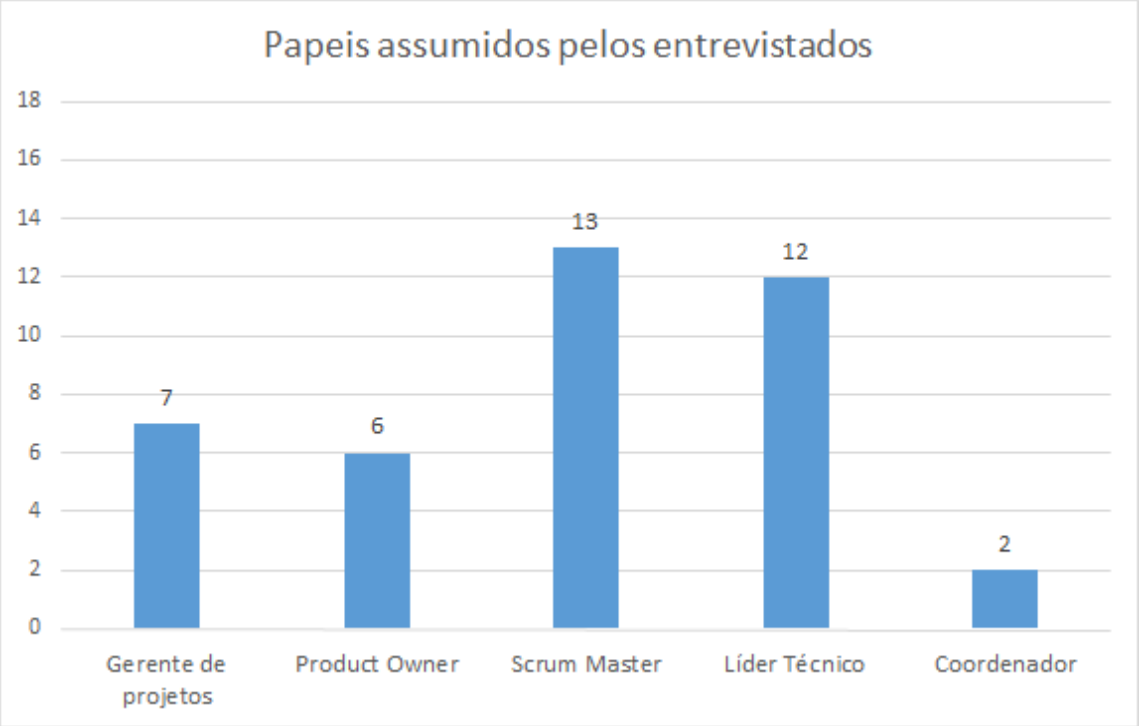

Figura 18 - Papéis assumidos pelos entrevistados.
Fonte: Elaborada pelo autor.

Podemos observar na figura acima, que a maioria dos entrevistados já assumiram os papéis de *Scrum Master* ou *Product Owner*. Dessa forma, é possível afirmar que certamente eles têm experiência com a metodologia Scrum e com o processo de desenvolvimento de software.

5.1.2 Ferramentas utilizadas pelas as equipes dos entrevistados

Por meio das entrevistas, foi feito um levantamento sobre quais as ferramentas para gestão de projetos, controle de versão, gerenciamento de testes e integração contínua as equipes coordenadas pelos líderes utilizam durante o desenvolvimento de projetos. A seguir serão apresentadas as ferramentas mais utilizadas pelas equipes com base nas repostas dos entrevistados:

Ferramentas para Gestão de Projetos: As ferramentas de gestão de projetos utilizadas pelas equipes para gerenciar o processo, os eventos e os artefatos produzidos durante o desenvolvimento. Vale ressaltar, que neste caso, as equipes utilizam mais de uma ferramenta para a gestão de projetos. Na figura 19 são apresentadas as ferramentas para gestão de projetos utilizadas pelas equipes.

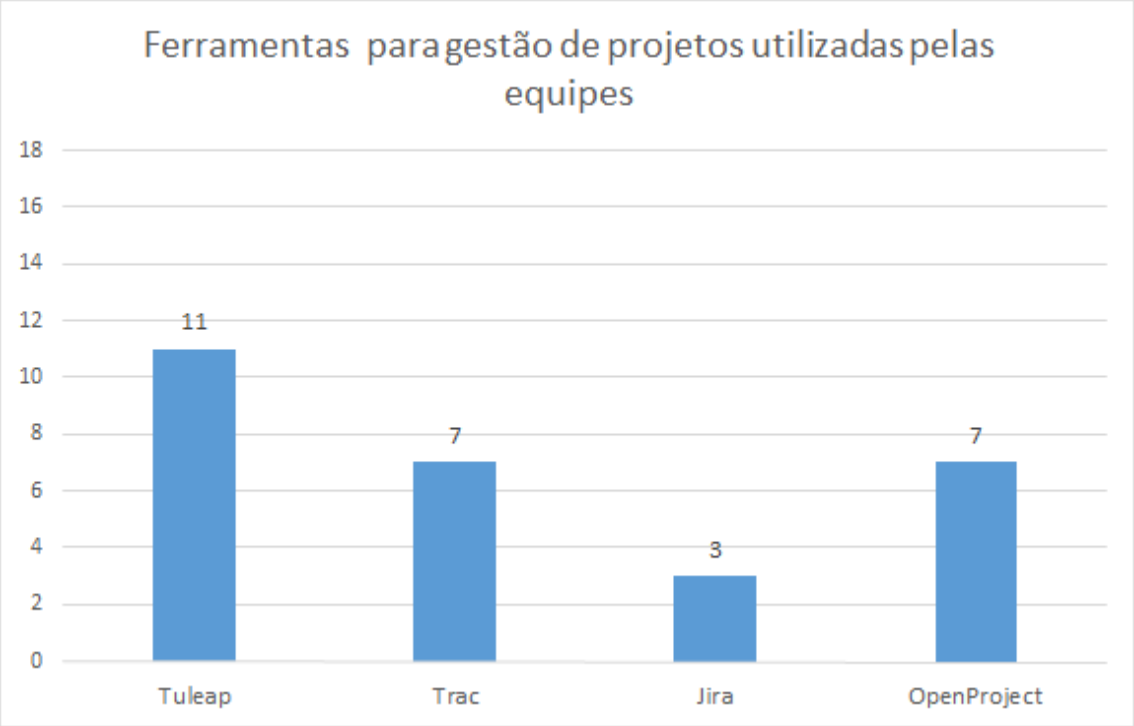

Figura 19 - Ferramentas para gestão de projetos utilizadas pelas equipes.
Fonte: Elaborada pelo autor.

Ferramentas para Controle de Versão: As ferramentas de controle de versão utilizadas pelas equipes. Também é importante ressaltar que há equipes que utilizam mais de uma ferramenta para o controle de versão. A figura 20 apresenta as ferramentas para controle de versão utilizadas pelas equipes.

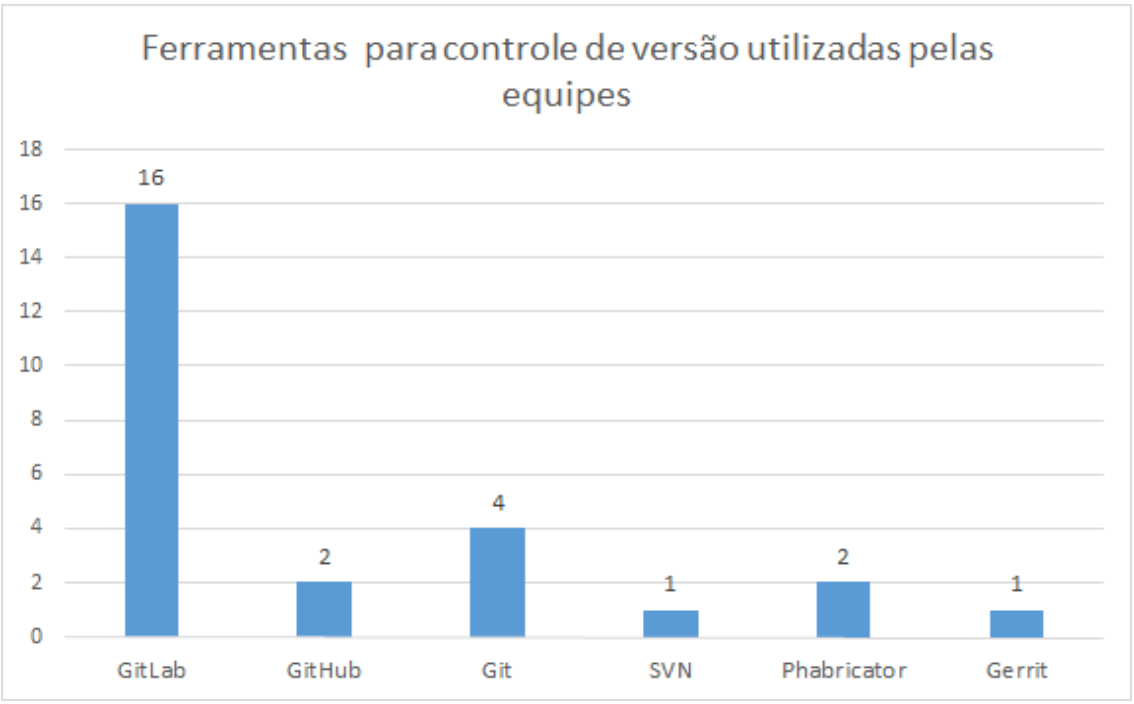

Figura 20 - Ferramentas para controle de versão utilizadas pelas equipes.
Fonte: Elaborada pelo autor.

Ferramentas para Gerenciamento de Testes: As ferramentas de gerenciamento de testes utilizadas pelas equipes para gerir *test cases*, *bugs* e executar de testes. A figura 21 apresenta as ferramentas para gerenciamento de testes utilizadas pelas equipes.

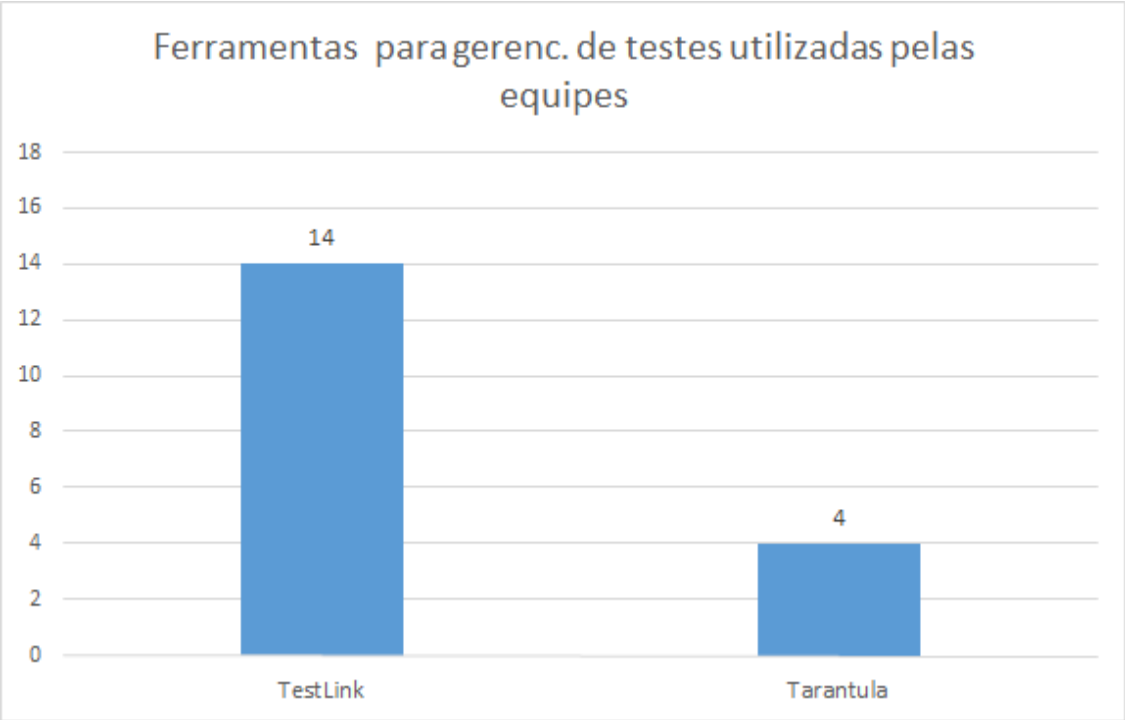

Figura 21 - Ferramentas para gerenciamento de testes utilizadas pelas equipes
Fonte: Elaborada pelo autor.

Ferramenta para Integração Contínua: As ferramentas de integração contínua utilizam pelas equipes, para gerenciamento de *builds*, execução de testes automatizados e análise estática. Na figura 22 são apresentadas as ferramentas para integração contínua utilizadas pelas equipes.

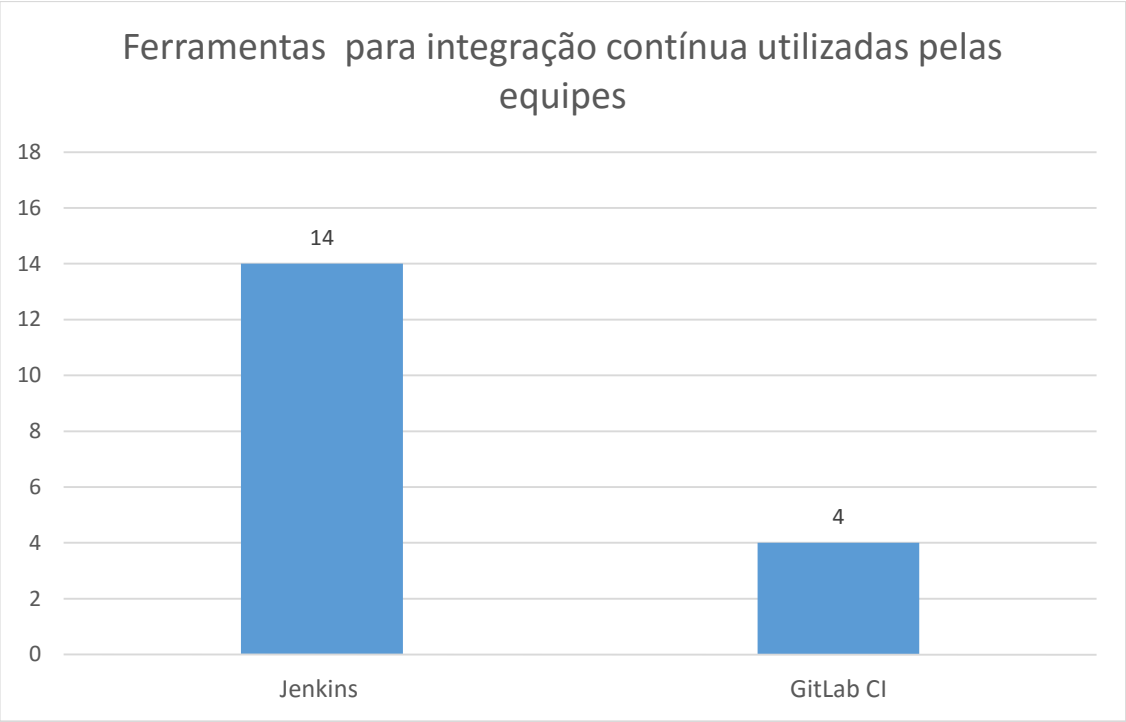

Figura 22 - Ferramentas para integração contínua utilizadas pelas equipes.
Fonte: Elaborada pelo autor.

Levando em consideração a quantidade de equipes que utilizam determinada ferramenta, sugerimos que as ferramentas integradas inicialmente ao sistema que implementar a arquitetura proposta sejam: Tuleap ou OpenProject para gerenciamento de projetos, GitLab para controle de versão, Jenkins para integração contínua e TestLink para gerenciamento de testes, dessa forma, o sistema poderá atender uma maior quantidade de equipes na empresa onde essa pesquisa foi realizada.

5.1.3 As práticas de Scrum e do Desenvolvimento Ágil adotadas pelas equipes

Durante as entrevistas foi questionado sobre quais práticas de Scrum e do Desenvolvimento Ágil as equipes adotavam durante o processo de desenvolvimento e, a partir disso, identificou-se quais as práticas mais adotadas pelas equipes. Dessa forma, sendo possível disponibilizar informações que possam servir como base para sugestões de futuras funcionalidades para a arquitetura, a seguir, as principais práticas adotadas pelas equipes:

Eventos: Eventos recomendados pelo Scrum adotados pelas equipes durante o processo de desenvolvimento. A figura 23 apresenta o percentual dos eventos adotados.

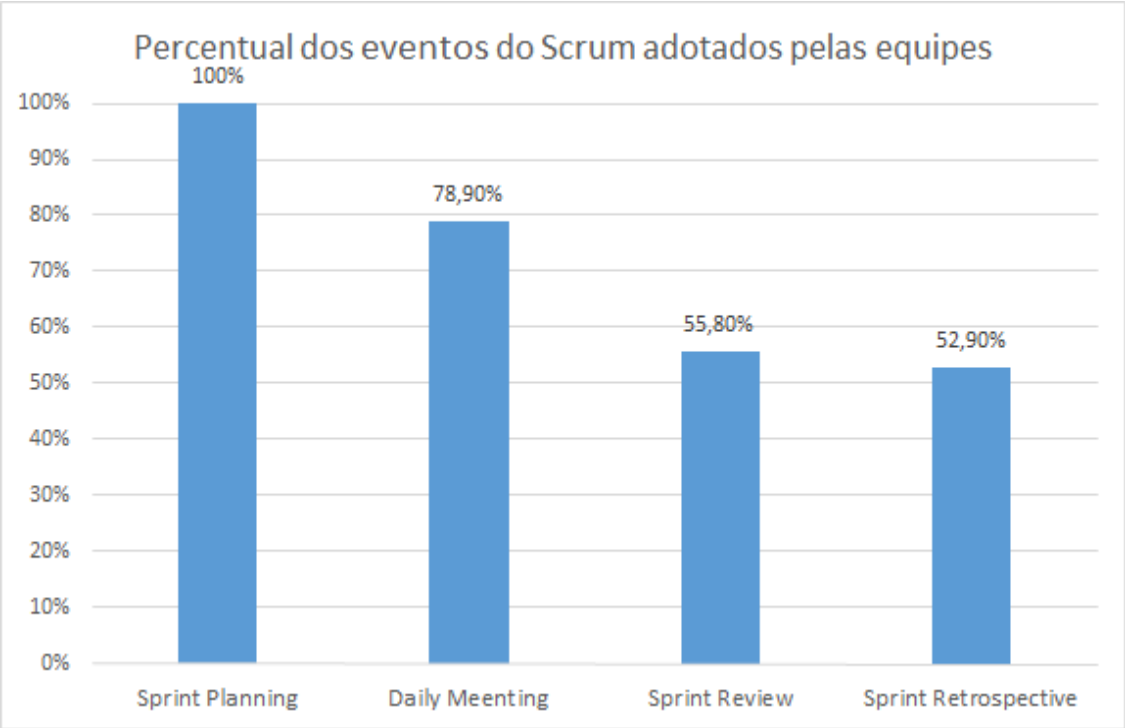

Figura 23 - Percentual dos eventos do Scrum adotados pelas equipes.
Fonte: Elaborada pelo autor.

Com base na figura 23 podemos observar que nem todas as equipes adotam os principais eventos recomendados pelo Scrum, pois apesar de todas equipes adotarem o *Sprint Planning*, apenas a 52,9% das equipes adotam os eventos que ocorrem no final da Sprint (*Sprint Review e Sprint Retrospective*).

Artefatos: Artefatos recomendados por Scrum adotados pelas equipes durante o processo de desenvolvimento. Na figura 24 é apresentado o percentual dos artefatos adotados.

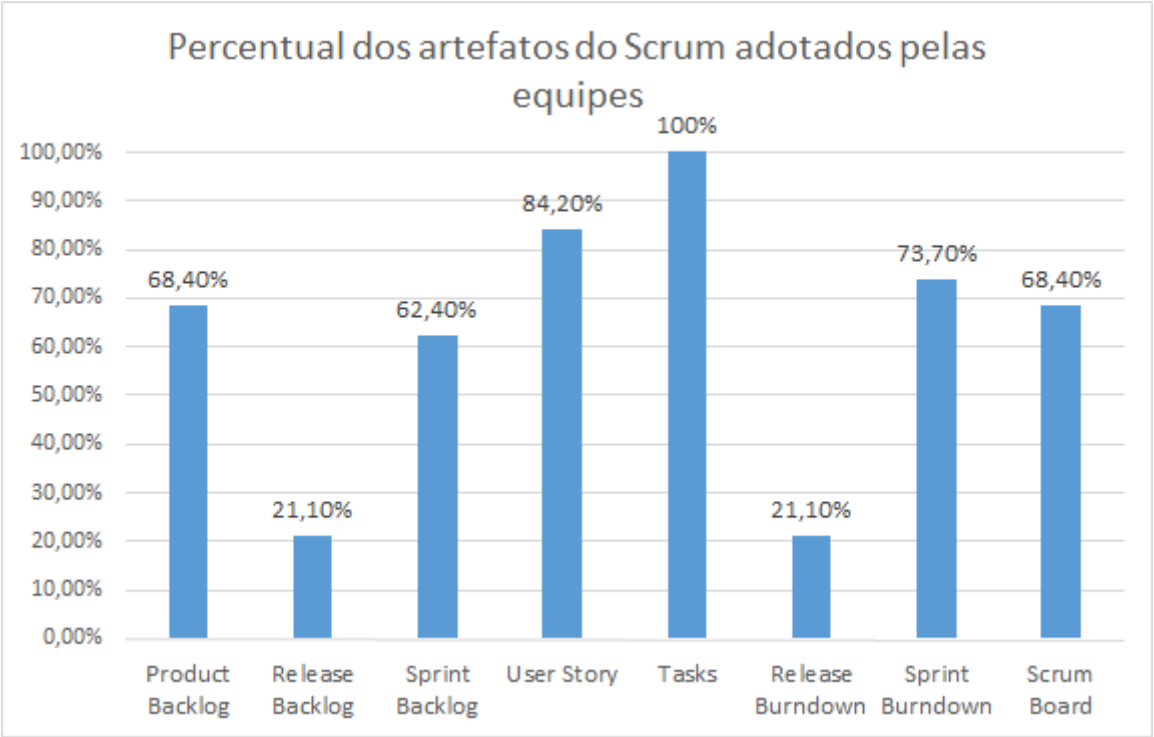

Figura 24 - Percentual dos artefatos do Scrum adotados pelas equipes.

Fonte: Elaborada pelo autor.

Práticas de desenvolvimento ágil: As práticas de desenvolvimento ágil adotadas pelas equipes dos entrevistados. A figura 25 apresenta o percentual das práticas de desenvolvimento ágil adotadas.

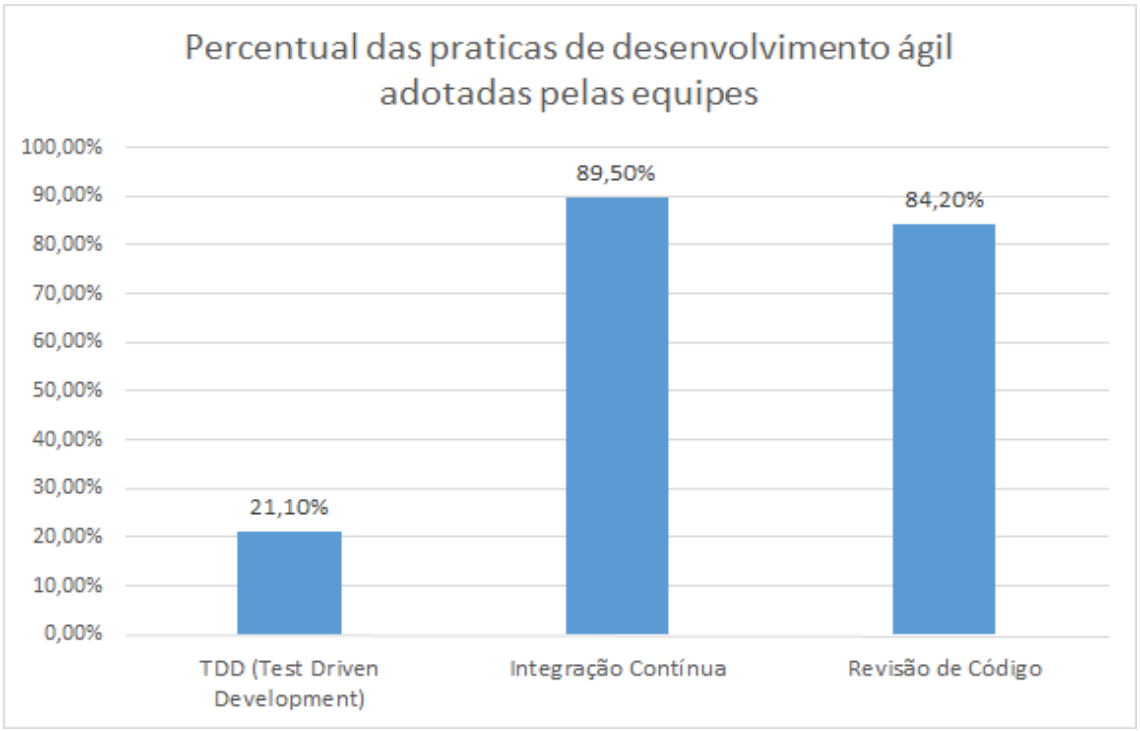

Figura 25 - Percentual das práticas de desenvolvimento ágil adotadas pelas equipes.

Fonte: Elaborada pelo autor.

Podemos observar na figura 25, que a grande maioria das equipes adota a prática de integração contínua, dessa forma, podemos afirmar que a funcionalidade para automação na execução de *pipelines*, poderá ser utilizada por mais de 85% das equipes.

5.1.4 Integração de ferramentas utilizadas pelas equipes

Durante a validação deste trabalho também foi questionado se as equipes faziam uso de algum tipo de integração entre as ferramentas. Com base nas respostas, observamos que 27.5% utilizam alguma forma de integração e 72.5% não utilizam. No grupo que alega fazer uso de integração de ferramentas, 82.3% fazem integração entre a FIC (Jenkins ou GitLab CI) e FCV (GitLab) e 17.7% fazem integração entre a FGP (Taiga) e FCV (GitLab).

É importante ressaltar que as formas de integração utilizadas pelas equipes são feitas com o uso de *plugins* disponíveis para determinadas ferramentas ou por meio de *scripts* desenvolvidos pelas equipes. Todavia, nenhum dos entrevistados alegou fazer integração entre todas as ferramentas utilizadas pelas equipes, nem ter reutilizado soluções de integração oriundas de outro projeto.

5.2 VALIDAÇÃO DA ARQUITETURA E DAS FUNCIONALIDADES PROPOSTAS

Nessa seção será apresentado como os entrevistados avaliaram a utilidade da arquitetura e das funcionalidades propostas por este trabalho.

5.2.1 Utilidade da funcionalidade instalar ferramentas

Como mencionado na seção 4.2.1, existe um grande problema quando as equipes precisam montar o ambiente no início dos projetos, pois é preciso instalar e configurar todas ferramentas necessárias. Por isso, umas das primeiras funcionalidades propostas para a arquitetura foi a de instalar ferramentas de forma automatizada. Essa funcionalidade foi muito bem avaliada pelos entrevistados, inclusive gerando comentários como: “...funcionalidade indispensável para o sistema...” e “...ajudaria muito a nossa vida…”. A figura 26 apresenta o percentual de utilidade da funcionalidade instalar ferramentas.

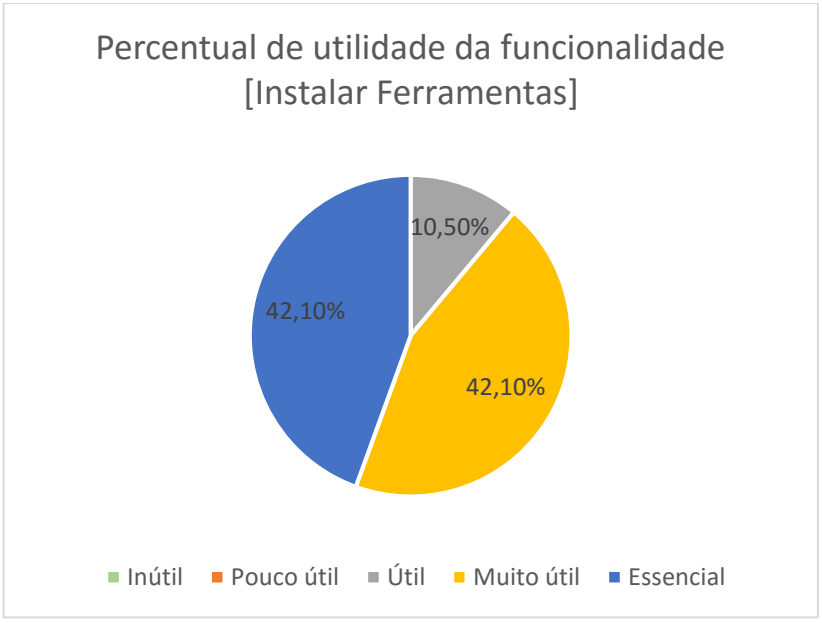

Figura 26 - Percentual de utilidade da funcionalidade instalar ferramentas.
Fonte: Elaborada pelo autor.

Como ilustrado na figura acima, o percentual de utilidade da funcionalidade concentra-se entre “muito útil” e “essencial”, destacando a importância da funcionalidade para um sistema, e que certamente ajudaria as equipes, principalmente no início dos projetos.

5.2.2 Utilidade da funcionalidade manter projetos

Como podemos observar na figura 27, a funcionalidade de manter projetos também foi muito bem avaliada, pois, quase a metade dos entrevistados consideram a funcionalidade como essencial para o sistema.

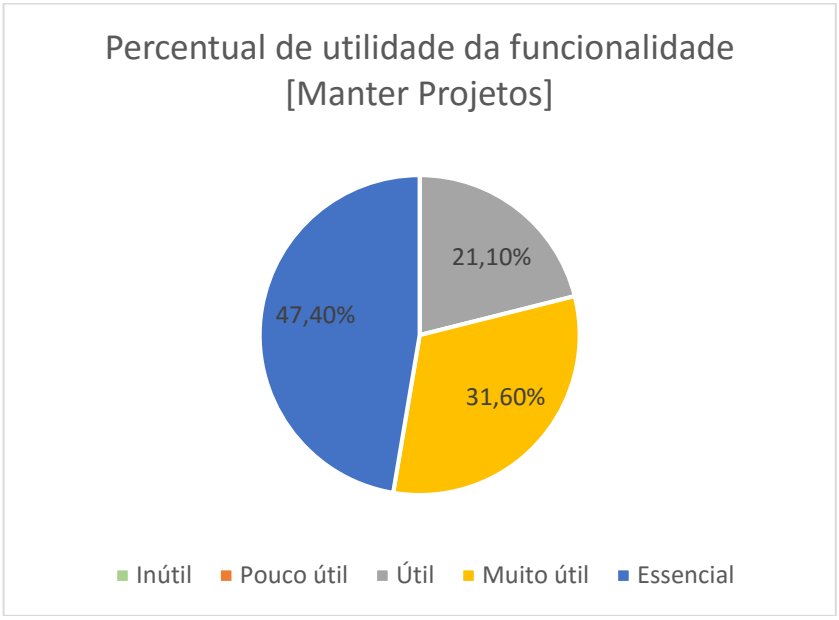

Figura 27 - Percentual de utilidade da funcionalidade manter projetos.
Fonte: Elaborada pelo autor.

5.2.3 Utilidade da funcionalidade manter usuários

Como mencionado na seção 3.2.2, existe muito retrabalho quando é necessário manter usuários, pois é preciso manter o usuário em todas as ferramentas. Podemos observar na figura 28, a funcionalidade de manter usuários de forma automatizada foi bem avaliada, pois, apesar de 10,5% dos entrevistados a considerarem como "pouco útil", a maior parte dos entrevistados a consideraram como "útil", "muito útil" e "essencial".

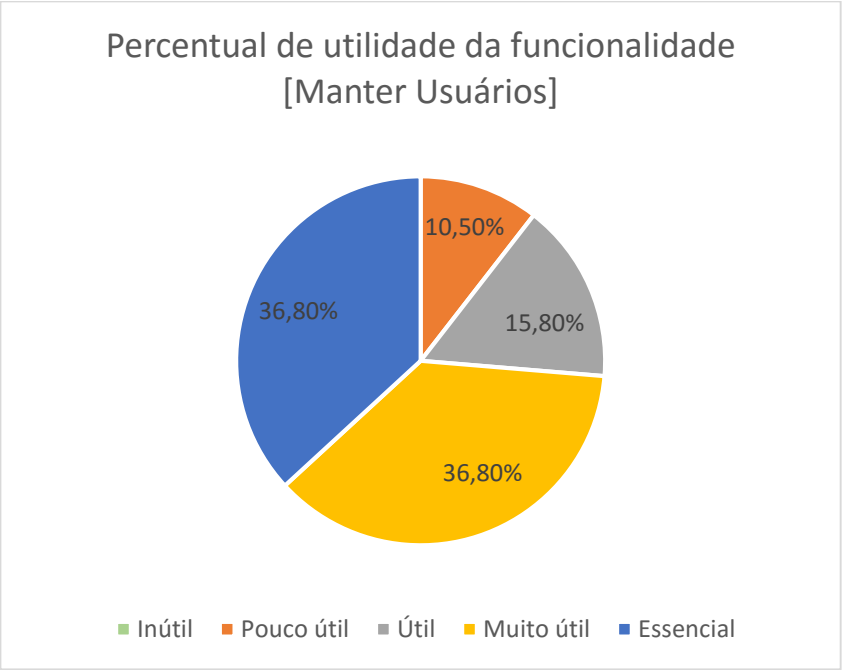

Figura 28 - Percentual de utilidade da funcionalidade manter usuários.
Fonte: Elaborada pelo autor.

5.2.4 Utilidade da funcionalidade verificar estado do projeto

Como podemos observar na figura 29, a funcionalidade de verificar estado do projeto também foi muito bem avaliada, pois, quase 65% dos entrevistados a consideraram como muito útil para o sistema, pois com essa funcionalidade os gerentes de projetos não precisam acessar todas as ferramentas para se ter uma visão geral dos projetos.

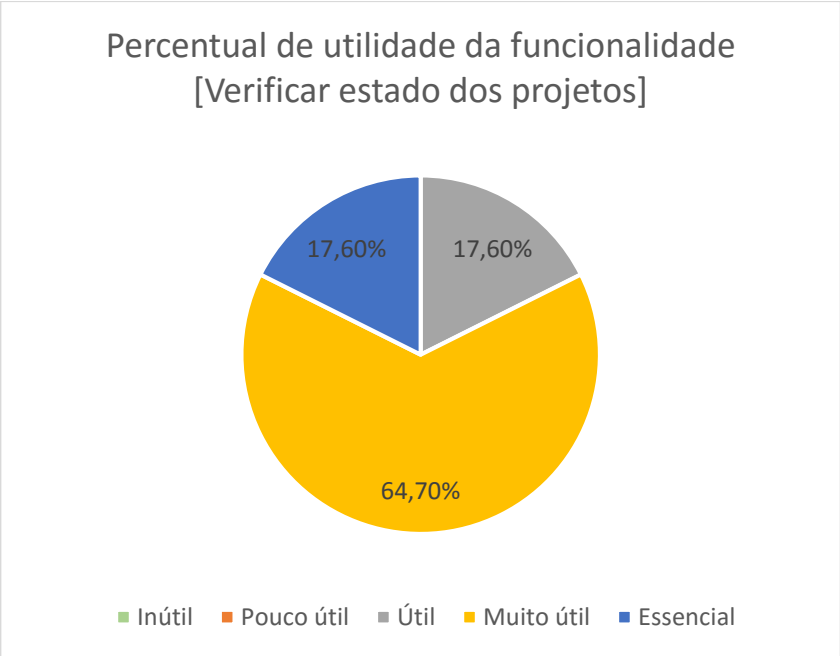

Figura 29 - Percentual de utilidade da funcionalidade verificar estado dos projetos.
Fonte: Elaborada pelo autor.

5.2.5 Utilidade da funcionalidade para automação do *Scrum Board*

Podemos perceber na figura 30, que a funcionalidade para automação do *Scrum Board*, não foi tão bem avaliada como as funcionalidades anteriores, no entanto, não foi considera como "inútil", já que quase 90% dos entrevistados a consideraram no mínimo como "útil".

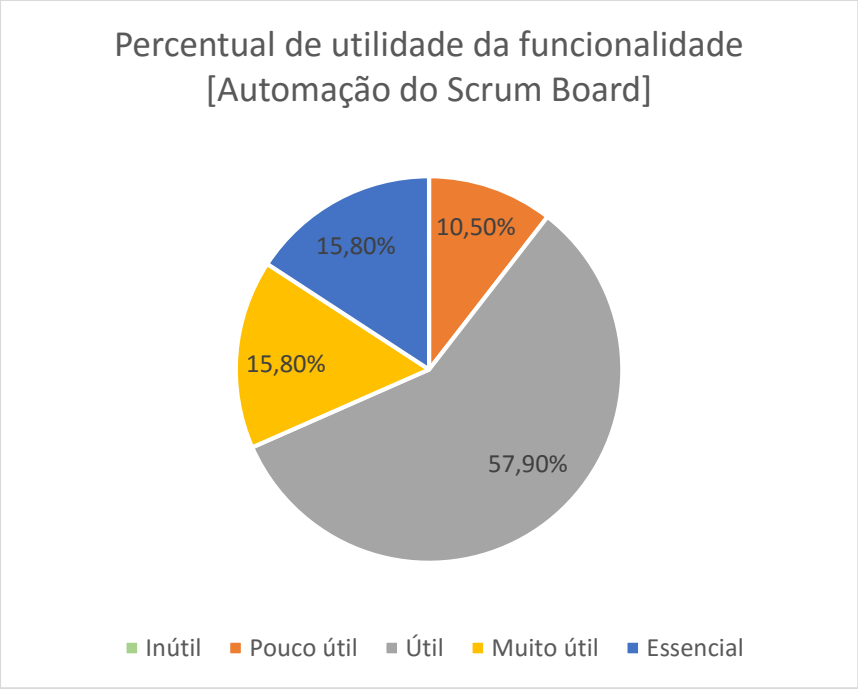

Figura 30 - Percentual de utilidade da funcionalidade automação do Scrum Board.
Fonte: Elaborada pelo autor.

5.2.6 Utilidade da funcionalidade automação da execução de *pipelines*

Como podemos observar na figura 31, a utilidade da funcionalidade automação da execução de *pipelines* foi avaliada como "essencial" por mais da metade dos entrevistados. É importante dizer, que apenas os 2 entrevistados que afirmaram não utilizar integração contínua consideraram a utilidade da funcionalidade como "pouco útil".

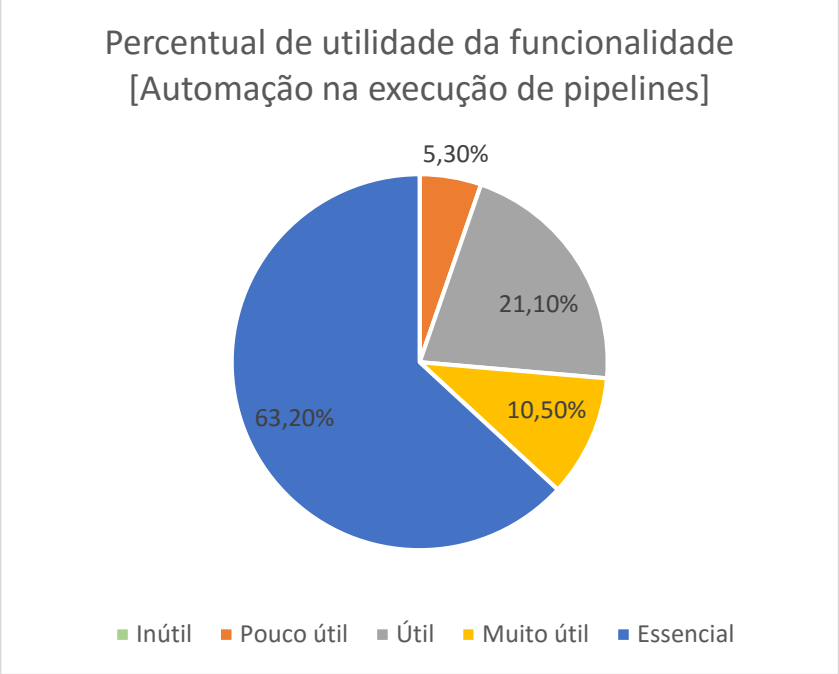

Figura 31 - Percentual de utilidade da funcionalidade automação da execução de pipelines.
Fonte: Elaborada pelo autor.

5.2.7 Resumo da utilidade das funcionalidades propostas pela arquitetura

Para melhor visualizar a utilidade das funcionalidades, um gráfico de caixa (*box plot*[16]) também foi construído com base nas respostas dos entrevistados. Esse gráfico é apresentado na figura 32.

[16] Os gráficos de *box plot* são uma forma de representar resumidamente a dispersão de um conjunto de amostras.

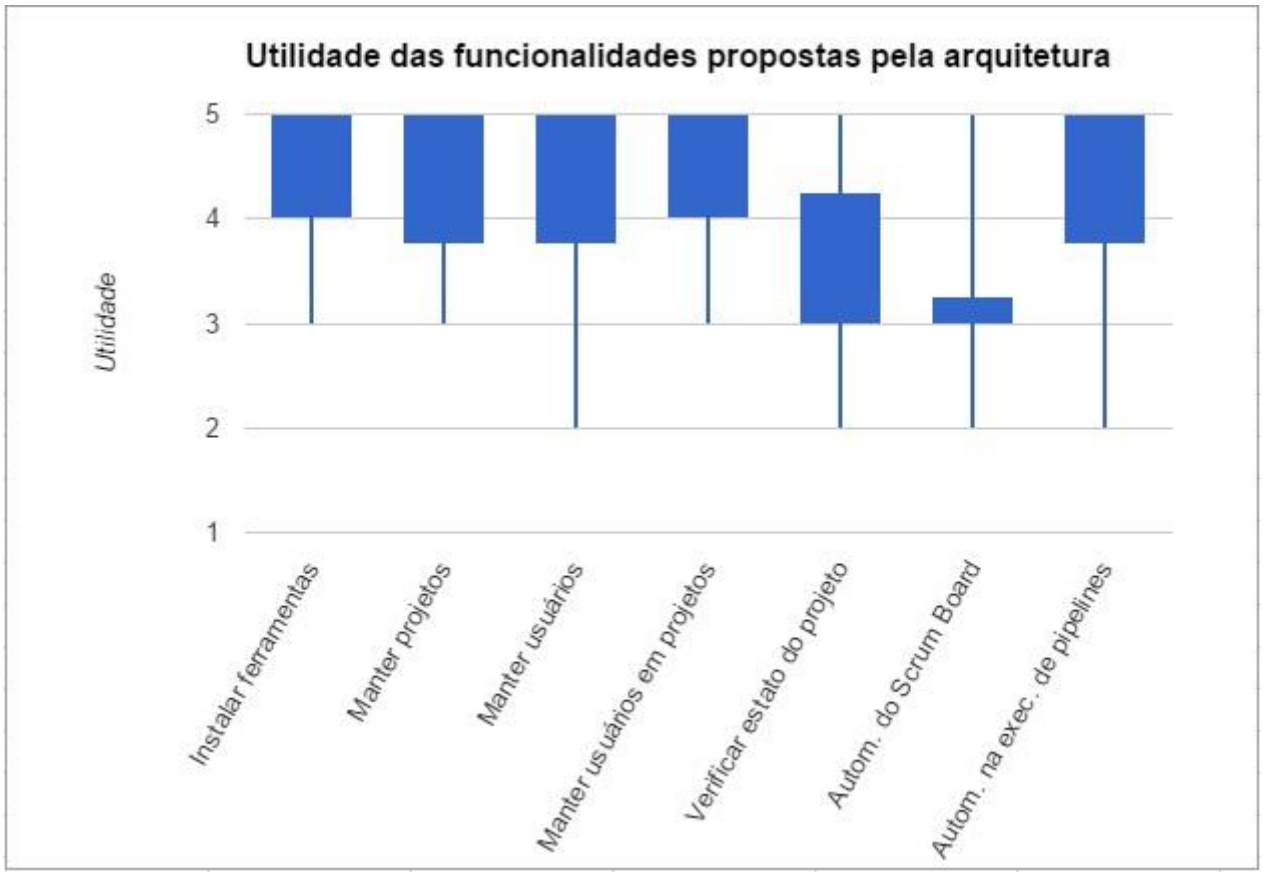

Figura 32 - *Box plot* das funcionalidades propostas.
Fonte: Elaborada pelo autor.

Observando o gráfico exposto na figura acima, é possível observar que, a utilidade da maioria das funcionalidades propostas se concentra entre “muito útil” e “essencial” (apresentado no gráfico como 4 e 5 respectivamente), assim, solidificando a importância dessas funcionalidades para um futuro sistema que implementar a arquitetura proposta.

5.2.8 Utilidade da arquitetura proposta

Durante as entrevistas também foi questionado como os líderes de equipe consideraram a utilidade da arquitetura proposta. A figura 33 apresenta como os entrevistados avaliaram a utilidade da arquitetura proposta.

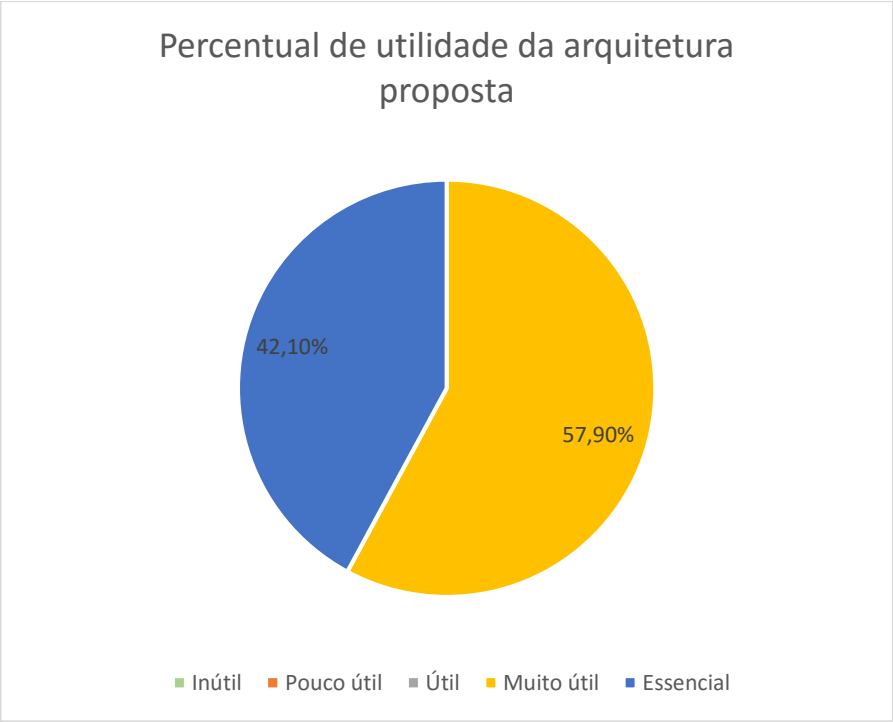

Figura 33 - Percentual de utilidade da arquitetura proposta.
Fonte: Elaborada pelo autor.

Observando a figura 33, podemos concluir que a arquitetura proposta por este trabalho pode ser de grande utilidade, não somente por servir como referência para futuros sistemas, mas também, por oferecer funcionalidades que tem o propósito de resolver problemas enfrentados no cotidiano das equipes que utilizam ferramentas para suporte a Scrum. Levando em consideração que, com a integração de ferramentas oferecida pela arquitetura, problemas como retrabalho e inconsistência de informações podem ser resolvidos.

5.2.9 Pontos fortes e fracos identificados na arquitetura

Durante as entrevistas foi questionado aos líderes quais os pontos fortes e fracos em relação a arquitetura e funcionalidades eles poderiam citar, a seguir os principais pontos levantados pelos entrevistados:

Pontos fortes:

- Possibilidade de integração com diferentes ferramentas, adaptando-se ao contexto de cada equipe;
- Redução do retrabalho, as tarefas precisam ser feitas apenas uma única vez, o que acaba também reduzindo problemas de inconsistência de dados;
- Facilidade de instalação e configuração do ambiente, o sistema provê de forma fácil essa tarefa;

- Independência das ferramentas, não há acoplamento entre as ferramentas, o que facilita a troca de uma ferramenta por outra;
- Identidade das ferramentas, o sistema não substitui as funcionalidades das ferramentas, apenas as complementa.

Pontos Fracos:

- As ferramentas utilizadas pelas equipes podem não fornecer métodos de integração;
- A integração com novas ferramentas pode ser complexa;
- O sistema precisa ter um bom esquema de tolerância a falhas para lidar com problemas na execução de operações (serviço fora do ar, atraso nas respostas);
- Cada ferramenta geralmente mantém seus esquemas de integração com outras ferramentas (garantindo o correto funcionamento), logo, o sistema deve garantir a consistência da integração em casos de atualização de versão das ferramentas.

5.2.10 Sugestões para a arquitetura

Durante a validação, os entrevistados levantaram algumas sugestões e melhorias para a arquitetura proposta, a seguir as sugestões mais relevantes:

- ***OpenStack***: Em vez de usar servidores dedicados para as ferramentas, por que não usar o *OpenStack?* Que é um sistema de *cloud computing* que disponibiliza diversos serviços, entre eles, o de maquinas virtuais, que neste caso, poderiam ser utilizadas como servidores para as ferramentas, reduzindo custos e facilitando o gerenciamento de *backups*.
- ***QA Warnings***: Levando em consideração que o sistema é integrado as ferramentas, por que não analisar a qualidade dos artefatos produzidos nelas? Por exemplo: Alertar que existem *releases* definidas sem *sprints* planejadas, *requirements* sem descrição, *user stories* sem pontos de estimativa, *user stories* sem DoD (*Definition of Done*), *user stories* sem pontuação ou com a pontuação muito alta, *sprints* finalizadas sem *builds* geradas e etc.
- **Integração fácil:** O mais importante para a arquitetura é que a integração com diferentes ferramentas seja de maneira fácil, levando em consideração que as equipes utilizam diferentes ferramentas, é importante que o sistema seja flexível o suficiente para integrar as ferramentas que as equipes já utilizam.
- **Geração de relatórios**: O sistema poderia ter funcionalidades para geração de relatórios a respeito da saúde dos projetos, indicando como o está o *burndown* da *release* e das *sprints*, como está a cobertura de testes e qual a quantidade de *bugs.*

- **Instalação simultânea das ferramentas**: Em vez de instalar ferramenta por ferramenta, por que não instalar a suíte completa de ferramentas que serão integradas ao sistema? Dessa forma o sistema ficaria ainda mais automatizado.
    - **Integração com mais de uma ferramenta do mesmo tipo**. A arquitetura oferece suporte integração com uma ferramenta de cada tipo, mas em algumas equipes é utilizada mais de uma ferramenta do mesmo tipo no projeto, por exemplo: uma equipe utiliza o Tuleap, mas devido a solicitação do cliente a equipe também utiliza o Trac para gerenciar o mesmo projeto. Por esse motivo, foi sugerido que a arquitetura oferece suporte para mais de uma ferramenta do mesmo tipo.

5.3 RESUMO DO CAPÍTULO

Neste capítulo foi apresentada a validação da arquitetura e as funcionalidades propostas por este trabalho, que foi realizada por meio de entrevistas e questionários com líderes de equipe do Laboratório Embedded.

Seguem, no próximo capítulo, as considerações finais deste trabalho e sugestões para trabalhos futuros.

6 CONSIDERAÇÕES FINAIS

Como pudemos observar no decorrer deste trabalho, a falta de integração entre ferramentas ocasiona muitos problemas em projetos que utilizam a metodologia Scrum, mas que podem ser completamente ou parcialmente resolvidos com um sistema que integre as ferramentas que oferecem suporte a Scrum.

Portanto, o presente trabalho teve como objetivo expor a existência desses problemas, e propor uma arquitetura que sirva como referência para sistemas que desejem integrar ferramentas para suporte a Scrum.

Com o propósito de investigar a utilidade da arquitetura proposta no capítulo 4, foram realizadas entrevistas com líderes de equipes de uma empresa de desenvolvimento de software. Para isso, foi feito um levantamento sobre o ambiente em que os entrevistados trabalham, quais as ferramentas para suporte a Scrum as equipes coordenadas pelos líderes utilizam e quais práticas do Scrum adotam. Com disso, fornecer dados que possam ser utilizados para sugestões de novas funcionalidades para a arquitetura.

Após a validação, foi possível observar que as funcionalidades apresentadas, foram avaliadas de forma satisfatória, pois, tendem a resolver os problemas expostos por este trabalho. Sendo assim, considera-se que os objetivos apresentados no Capítulo 1 deste trabalho foram alcançados e que a arquitetura proposta, de modo geral, é de grande utilidade. No entanto, é importante ressaltar a necessidade de se implementar um sistema com base na arquitetura e de realizar experimentos e investigações a respeito dos problemas levantados e verificar se a arquitetura, de fato, os resolve.

Por fim, é possível perceber a existência de uma contribuição científica e tecnológica apresentada por este trabalho, uma vez que essa arquitetura pode trazer grandes benefícios para diversas equipes de desenvolvimento de software que utilizam ferramentas para dar suporte a metodologia Scrum.

6.1 TRABALHOS FUTUROS

Como sugestão de trabalhos e atividades a serem desenvolvidas posteriormente para dar continuidade a pesquisa apresentada neste trabalho, são recomendados os seguintes pontos:

- **Implementação de um sistema com base na arquitetura proposta.** Este trabalho apresentou uma arquitetura para integração de ferramentas e possíveis funcionalidades. Com base na validação apresentada, é possível identificar que a arquitetura é útil e que

certamente ajudaria a resolver ou amenizar os problemas levantados. Porém, se faz necessário a criação de um sistema que possa implementar a arquitetura e as funcionalidades propostas por esse trabalho.

- **Realizar experimentos com o sistema que implementar a arquitetura.** O foco da validação realizada neste trabalho foi na utilidade da arquitetura e das funcionalidades propostas. Por isso, é importante que sejam realizados experimentos a fim de identificar se os problemas levantados por este trabalho serão resolvidos ou amenizados com o sistema que implementar a arquitetura proposta.
- **Avaliar os pontos fortes/fracos da arquitetura e as sugestões levantadas pelos entrevistados**. Esse trabalho propôs uma arquitetura e um conjunto de funcionalidades para integração de ferramentas, no entanto, durante a validação foram levantados alguns pontos e sugestões que podem ser avaliadas e que possam servir como referência para trabalhos futuros.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, M. A. P.; TINOCO, A. S. **Ferramentas para Gestão de Projetos**. 2014. Revista Engenharia de Software Magazine ed. 45 Disponível em: <http://www.devmedia.com.br/ferramentas-para-gestao-de-projetos-revista-engenharia-de-software-magazine-45/23563>. Acesso em: setembro 2014.

BORCK, J. R. **Bossie Awards 2013: The best open source application development tools.** 2013. Disponível em: <http://www.infoworld.com/article/2606775/open-source-software/119849-Bossie-Awards-2013-The-best-open-source-application-development-tools.html>. Acesso em: maio 2015.

CAETANO, C. **Gestão de Testes: Ferramentas Open Source e melhores práticas na gestão de testes**. 2012. Revista Engenharia de Software Magazine ed. 3. Disponível em: <http://goo.gl/doJncx>. Acesso em: setembro 2015.

CARVALHO, G. et al. **A Abordagem Sistêmica na Gestão de Projetos de Software.** 2009. 7 f. Universidade Federal do Pernambuco, Recife, 2007. Disponível em: <http://www.uspleste.usp.br/jporto/woses2009/anais/01-WOSES-2009.pdf>. Acesso em: abril 2015.

CHACON, S.; STRAUB, B. **Pro Git.** 2009. Apress. 2ª ed. 2009. Disponível em: <https://progit2.s3.amazonaws.com/en/2015-09-08-d73ca/progit-en.816.pdf> Acesso em: setembro 2015.

CHIAVENATO, I. **Introdução à teoria Geral da Administração**. Rio de Janeiro, 2000.

DEGELER, A. **How GitHub rival GitLab is building a business with just 0.1% paying customers.** 2014. Disponível em: < http://thenextweb.com/insider/2014/06/04/github-rival-gitlab-building-business-just-0-1-paying-customers/>. Acesso em: maio 2015

FAGUNDES, P. B. **Framework Para Comparação e Analise de Métodos Ágeis**. 2005. Universidade Federal de Santa Catarina, Florianópolis, 2005. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/101860/220977.pdf?sequence=1&isAllowed=y>. Acesso em: abril 2015.

FARIAS, A. C. **Ferramentas CASE: Suporte, Adoção e Integração.** 2001. Universidade Federal de Pernambuco, Recife, 2001. Disponível em: <http://www.di.ufpe.br/~acf/publications/CASETools-report.pdf>. Acesso em: abril 2015.

FILHO, C. F. **História da Computação: O caminho do pensamento e da tecnologia.** 2007. EDIPUCRS, 2007, p.24. Disponível em: <http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf>. Acesso em: outubro 2015.

FLEISHER, D. et al. **Evaluation of Open Source Tools for Test Management and Test Automation.** 2012. School of Business, International Business Information Management. Course WWI2009I.

FOWLER, M. **Continuous Integration**. 2000. Disponível em: <http://martinfowler.com/articles/continuousIntegration.html>. Acesso em: agosto 2015.

FOWLER, M. **The New Methodology**. 2001. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em: abril 2015.

KAWAGUCHI, K. **7 Ways to Optimize Jenkins/Hudson White Paper.** 2011. Disponível em: <http://di388e0fcqllf.cloudfront.net/whitepapers/7WaysToOptimizeJenkins.pdf>. Acesso em: junho 2015.

KNIBERG, Henrik. **Scrum e XP direto das Trincheiras**. 2011. InfoQ. Disponível em: < http://www.infoq.com/br/minibooks/scrum-xp-from-the-trenches>. Acesso em: abril 2015.

LEITÃO, M. V. **Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo.** 2010. 72 f. Trabalho de Conclusão de Curso (Engenharia de Computação). Universidade Federal de Pernambuco, Recife, 2010. [Orientador: Prof. Cristine Martins Gomes de Gusmão]. Dísponivel em: <http://tcc.ecomp.poli.br/20101/TCC_final_Michele .pdf> Acesso em: abril 2015.

LINK, E.; ALEXANDRE, E. B. P.; WOLF, J. L.; STRZYKALSKI. **Uma Introdução ao CORBA.** 2009. Disponível em: < http://www.giulianoprado.xpg.com.br/sd/arquivos/artigo_corba.pdf>. Acesso em: dezembro 2015.

LUSA, D. A. e KRAEMER, R. P. **Conhecendo o modelo arquitetural REST**. 2014. Revista Engenharia de Software Magazine ed. 58.

MACHADO, M; MEDINA, S. G. **SCRUM – Método Ágil: uma mudança cultural na Gestão de Projetos de Desenvolvimento de Software.** 2009. Disponível em: < http://www.uniesp.edu.br/fagu/revista/downloads/edicao12009/Artigo_5_Prof_Marcos.pdf>. Acesso em: outubro 2015.

MANIFESTO ÁGIL. 2001. Disponível em: <http://www. manifestoagil.com.br/>. Acesso em: outubro de 2015.

MANN, C.; MAURER, F. **A case study on the impact of scrum on overtime and customer satisfaction**. 2005. In: AGILE DEVELOPMENT CONFERENCE. IEEE, 2005. p. 70-79.

MENDES, F. F. et al. **Análise de Ferramentas para Apoio à Gerência de Projetos e Gerência de Requisitos de Software.** 2011. Universidade Federal de Goais. Disponível em: <https://www.cercomp.ufg.br/up/18/o/Artigo_Wamps.pdf>. Acesso em: maio 2015.

NETO, P. A. S. **Introdução à Engenharia de Software.** 2010. Disponível em: <http://www.ufpi.br/subsiteFiles/pasn/arquivos/files/IntroducaoEngenhariaDeSoftware.pdf>. Acesso em: outubro 2015.

PEREIRA, P. et al. **Entendendo Scrum para Gerenciar Projetos de Forma Ágil.** 2007. Mundo PM, 2007. Disponível em: <http://www.siq.com.br/DOCS/EntendendoScrumparaGerenciarProjetosdeFormaAgil.pdf>. Acesso em: abril 2015.

SCHWABER K.; SUTHERLAND J. **Guia do Scrum.** 2013. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: abril 2015.

SCHWABER, K. **Agile Project Management with Scrum**. Microsoft Press, 2004.

SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. 2002.

SCHWABER, K. **Scrum development process. In: Business Object Design and Implementation**. Springer London, 1997. p. 117-134

SNELL, J.; TIDWELL, D.; KULCHENKO, P. **Programming Web services with SOAP.** O'Reilly Media, Inc. 2001.

SMART, J. F. ***Jenkins: the definitive guide*.** " O'Reilly Media, Inc.". 2011.

SOARES, M. S. **Metodologias ágeis extreme programming e scrum para o desenvolvimento de software**. Revista Eletrônica de Sistemas de Informação ISSN 1677-3071 doi: 10.5329/RESI, v. 3, n. 1, 2004.

STEFFEN, J. B. **Scrum: Basicamente SCRUM.** 2011. Disponível em: <https://www.ibm.com/developerworks/community/blogs/rationalbrasil/entry/scrum_basicamente14>. Acesso em: abril 2015.

STANDISH GROUP. **CHAOS MANIFESTO 2012**. Boston: The Standish Group International, 2012. Disponível em: <http://pt.slideshare.net/Placosta/chaos-report-2012>. Acesso em: outubro 2015.

STANDISH GROUP. **CHAOS MANIFESTO 2013**. Boston: The Standish Group International, 2013. Disponível em: <http://www.versionone.com/assets/img/files/ChaosManifesto2013.pdf>. Acesso em: abril 2015.

SUTHERLAND, J. **Scrum - a arte de faze o dobro de trabalho na metade do tempo.** 2014. TEXTO EDITORES LTDA. 2014. Disponível em: <https://goo.gl/DYcY1s> Acesso em: outubro 2015.

VENTORIN, A. J. **Principais problemas relacionados ao desenvolvimento de sistemas.** 2005. Revista Universo Acadêmico, 1a Edição, 2005. Disponível em: <http://saomateus.multivix.edu.br/wp-content/uploads/2013/03/Revista_Mundo_Acad_1.pdf>. Acesso em: outubro 2015.

VOORT, J. V. D.; VOSAMAER, J. **GitLab Feature Highlight: Groups.** 2014. Disponível em: <https://about.gitlab.com/2014/06/30/feature-highlight-groups/>. Acesso em: maio 2015.

APÊNDICES

Segue a relação de apêndices utilizados para execução deste estudo:

APÊNDICE A – Questionário aplicado durante a validação da arquitetura

APÊNDICE A

Questionário sobre integração de ferramentas Scrum

*Obrigatório

Nome

Há quanto tempo você trabalha com Scrum (em meses)? *

Que papeis de liderança você já assumiu? *

- [ ] Coordenador
- [ ] Gerente de projetos
- [ ] Scrum master
- [ ] Product owner
- [ ] Líder técnico
- [ ] Outro:

Qual o tamanho do seu time (número de membros)? *

Que tipo de software seu time desenvolve?

- [ ] Mobile
- [ ] Web
- [ ] Desktop
- [ ] Outro:

Há quanto tempo o time está montado na configuração atual (em meses)?

Qual(is) ferramenta(s) para gerenciamento de projetos sua equipe utiliza?

- [ ] Tuleap
- [ ] Trac
- [ ] Jira
- [ ] Redmine
- [ ] Outro:

Qual(is) ferramenta(s) para gerenciamento de testes sua equipe utiliza?

- [ ] Testlink
- [ ] Tarantula

- [ ] Bugzila
- [ ] Testopia
- [ ] Outro:

Qual(is) ferramenta(s) para controle de versão do código fonte sua equipe utiliza?

- [ ] GitLab
- [ ] SVN
- [ ] GitHub
- [ ] Mercurial
- [ ] Git
- [ ] Outro:

Qual(is) ferramenta(s) para integração contínua sua equipe utiliza?

- [ ] Jenkins
- [ ] Bamboo
- [ ] GitLab CI
- [ ] Apache Continuum
- [ ] Outro:

Existe integração automática de ferramentas para suporte ao processo de desenvolvimento no seu time? Qual?

Quais das seguintes práticas de Scrum/desenvolvimento ágil/gestão de projetos o seu time adota?

- [ ] Product Backlog
- [ ] User stories
- [ ] User Stories - Acceptance criteria ou Definition of Done
- [ ] User Stories - Story points
- [ ] User Stories - Remaning points
- [ ] Definição de Releases com agenda
- [ ] Definição de Sprints com agenda
- [ ] Release backlog
- [ ] Sprint Planning
- [ ] Rejeição de Stories mal escritas
- [ ] Criação de tasks
- [ ] Daily meeting
- [ ] Atualização do Scrum Board
- [ ] Sprint Burndown
- [ ] TDD

- [ ] Revisão de código
- [ ] Integração contínua
- [ ] Garantia de Qualidade dos artefatos, Ferramenta
- [ ] Controle de bugs
- [ ] Sprint Review
- [ ] Sprint Retrospective
- [ ] Uso do Histórico para calcular Capacity e Velocity do time
- [ ] Release Burndown
- [ ] Acompanhamento de Key Performance Indicators (KPI)

Qual a utilidade de cada uma das seguintes funcionalidades do Turmalina no seu projeto?

	Inútil	Pouco útil	Útil	Muito útil	Essencial
Instalar e configurar ferramentas	○	○	○	○	○
Integração com LDAP	○	○	○	○	○
Manter projetos	○	○	○	○	○
Manter usuários	○	○	○	○	○
Manter requisitos	○	○	○	○	○
Automatização do Cardwall	○	○	○	○	○
Execução automática de pipelines (jobs)	○	○	○	○	○
KPI Reports	○	○	○	○	○
QA warnings releases	○	○	○	○	○
QA warnings backlog	○	○	○	○	○
QA warnings sprint	○	○	○	○	○
QA warnings user story	○	○	○	○	○
QA warnings task	○	○	○	○	○

Sobre a arquitetura proposta para o Turmalina, que pontos fracos e fortes você pode citar?

Essa arquitetura poderia ser adotada na integração de ferramentas do seu time? Como?

De uma maneira geral, como você avalia a utilidade do Turmalina?

1 2 3 4 5

Inútil ◯ ◯ ◯ ◯ ◯ Essencial

Sugestões

Enviar

Nunca envie senhas pelo Formulários Google.

Powered by

Este formulário foi criado em Departamento de Ciências Exatas (DCX/UFPB).

Denunciar abuso - Termos de Serviço - Termos Adicionais