Universidade Federal do Ceará

Campus Quixadá

Curso de Engenharia de Software

Guylherme Tabosa Cabral

IMPLANTAÇÃO DE UMA FERRAMENTA DE INTEGRAÇÃO CONTíNUA EM UM NÚCLEO DE PRÁTICAS EM INFORMÁTICA: RELATO DE EXPERIÊNCA

Quixadá, Ceará

2014

Guylherme Tabosa Cabral

IMPLANTAÇÃO DE UMA FERRAMENTA DE INTEGRAÇÃO CONTíNUA EM UM NÚCLEO DE PRÁTICAS EM INFORMÁTICA: RELATO DE EXPERIÊNCA

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Engenharia de Software do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Orientador: Prof Msc. Carlos Diego Andrade de Almeida

Co-Orientador:

Quixadá, Ceará

2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

C117i Cabral, Guylherme Tabosa
Implantação de uma ferramenta de integração contínua em um núcleo de práticas em informática: relato de experiência / Guylherme Tabosa Cabral. – 2014.
42 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2014.
Orientação: Prof. Me. Carlos Diego Andrade de Almeida
Área de concentração: Computação

1. Gerenciamento de configurações de software 2. Integração de dados (Computação) 3. Desenvolvimento ágil de software I. Título.

CDD 005.1

Guylherme Tabosa Cabral

IMPLANTAÇÃO DE UMA FERRAMENTA DE INTEGRAÇÃO CONTíNUA EM UM NÚCLEO DE PRÁTICAS EM INFORMÁTICA: RELATO DE EXPERIÊNCA

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Engenharia de Software do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Área de concentração: Computação

Aprovada em: 19 / novembro/ 2014

BANCA EXAMINADORA

Prof Msc. Carlos Diego Andrade de Almeida
(Orientador)
Universidade Federal do Ceará (UFC)

Prof Msc. Camilo Camilo Almendra
(Membro)
Universidade Federal do Ceará (UFC)

Prof Msc. Régis Pires Magalhães (Membro)
Universidade Federal do Ceará (UFC)

A minha família principalmente meus avós, minha mãe, meu irmão, meu pai e a Mikaely.

Agradecimentos

Agradeço primeiramente a Deus por me dar sabedoria, vontade e disposição para concluir este trabalho, sem ele nada seria possível.

A minha mãe Rejane, pelo exemplo de mulher e amor. Por sempre cuidar de mim mesmo quando não estive perto por todos esses anos de universidade.

A meu avô Nonato, pelo símbolo de pai, modelo de ser humano e seu meu maior exemplo de vida.

A minha avó Rita, por todo o amor, força e cuidado, minha segunda mãe.

A meu irmão Felype por sempre partilhar comigo momentos bons e ruins e cuidar de mim quando precisei.

A meu pai João por ensinar que o trabalho sempre gera resultados, e que o esforço é seu maior aliado para o sucesso.

A minha família pela ajuda e cuidado.

A Mikaely por ter me aturado nos momentos de confusão, medo, raiva, angústia e outros mais, sua presença ao meu lado me fez mais forte e focado nos objetivos.

A meus amigos de faculdade, por todos esses anos de companheirismo, felicidades e ajudas.

A meu orientador Carlos Diego por me guiar na obtenção do melhor resultado possível e disponibilidade e interesse neste projeto.

A UFC e seus funcionários por fornecer as ferramentas para que pudesse evoluir meus conhecimentos. Obrigado a todos.

Resumo

O desenvolvimento ágil está cada dia mais presente no cotidiano das empresas desenvolvedoras de software. A crescente busca por agilidade no desenvolvimento e a competitividade do mercado impactam na existência deste cenário. Portanto, muitas empresas buscam aplicar as metodologias definidas neste ramo de desenvolvimento em seu processos, porém, essa não é uma tarefa simples. A definição e implantação dessas práticas realizadas de maneira não suficientemente adequada podem trazer resultados adversos ao esperado. Portanto este trabalho teve como objetivo implantar a utilização de uma ferramenta de integração contínua em uma fábrica de software, o Núcleo de Práticas em Informática (NPI) da UFC - Campus Quixadá. Uma ferramenta de integração contínua está inserida em uma das práticas definidas pelo Extreme Programming (XP). Para a realização deste objetivo fora analisado o processo vigente executado no NPI, além da experiência do autor que era um membro desta fábrica de software. Somado a esta etapa foi definida a ferramenta de integração contínua a ser implantada de acordo com um conjunto de característica especificadas, o conhecimento dos membros da fábrica acerca de Gerência de configuração e Integração Contínua foi avaliado, e por fim, a implantação, com a definição de pontos positivos e negativos encontrados na implantação.

Palavras-chaves: Integração Contínua. Desenvolvimento Ágil. Gerenciamento de Configuração.

Abstract

The agile development is day by day inserted on the quotidian of software development companies. The growing seek for agility in the development and market competitiveness has impacted in the existence of this scenario. Therefore, many companies seek to apply the defined methodologies in this kind of development in their processes, however, this is not an easy task. The definition and deployment of this practices executed in a way not adequate enough can bring contrary results to the expected. So, this work had as an aim to implant the using of a continuous integration tool on a software factory, the Informatics Practice Center (NPI) at UFC in Quixadá. A continuous integration tool is inserted in one of the practices defined by Extreme Programming (XP). In order to reach this goals, the current process of NPI was analyzed, beyond the author experience thas was a member of this software factory. Added to this stage was defined the continuous integration tool to be implanted according with a set of specified characteristics, the knowledges of the factory's members about Configuration Management and Continuous Integration was evaluated, and finally the deployment, with the definition of positives and negatives points found at the deployment.

Key-words: Continuous Integration. Agile Development. Configuration Management.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Nas últimas décadas, as metodologias ágeis atraíram uma grande atenção por parte das empresas desenvolvedoras de software. Isto deve-se muito à relação das metodologias ágeis com o conjunto de boas práticas pertencentes a ela, que adicionam qualidade, controle e garantia ao produto.

A Integração Contínua é uma dessas práticas, definida como uma ferramenta de gestão de configuração, que auxilia os desenvolvedores e permite que as mudanças que foram realizadas no software sejam imediatamente avaliadas, testadas e verificadas, de modo a prover um *feedback* imediato para correção de possíveis erros de integração. Integrações, estas, que seriam verificadas apenas futuramente, posterior a problemas mais complexos de integração. (DUVALL; MATYAS; GLOVER, 2007).

A metodologia ágil Scrum é utilizada no Núcleo de Práticas em Informática (NPI). T rata-se de um ambiente onde estudantes que estão em fase de conclusão de curso podem estagiar e aprimorar seus conhecimentos adquiridos no decorrer de sua vida acadêmica, além de concluírem seus componentes curriculares obrigatórios. Este surgiu, devido à pouca demanda de empresas de Tecnologia da Informação (TI) na região onde a universidade se encontra e a crescente exigência por profissionais dotados de experiência em desenvolvimento de software (GONÇALVES et al., 2013).

Os projetos lá desenvolvidos têm como objetivo construir soluções que facilitem as atividades do cotidiano da universidade, esta que tem um grande interesse no desenvolvimento destes projetos, pois consegue reduzir custos ao priorizar construções de sistemas internamente (GONÇALVES et al., 2013). Em paralelo, se pode obter um aumento da qualidade dos profissionais formados, além de proporcionar um ambiente real de trabalho que tem como intuito facilitar a entrada dos concludentes no mercado de trabalho.

No NPI existe um modelo de processo definido em que os desenvolvedores devem seguir para o exercício de suas atividades (GONÇALVES et al., 2013). Entretanto, este em momentos não é devidamente seguido, e, em seu processo a utilização de uma ferramenta de integração contínua não está introduzida, podendo ocasionar uma despadronização na maneira como estes desenvolvedores trabalham em seus projetos. Somado-se a isto, o NPI apresenta problemas tais como: "Baixa qualidade da documentação dos sistemas; [...] Falta de uma equipe de manutenção; [...] Rotatividade dos profissionais"(PADUELLI; SANCHES, 2006, p. 4).

A experiência em aplicar ferramentas de gestão de configuração foi abordada por Oliveira e Nelson (2005), eles inseriram a utilização destas com o intuito de evitar a inserção de novos erros oriundos de manutenções realizadas no software. Diferentemente

do trabalho a ser desenvolvido nesse projeto que busca relatar a experiência obtida na implantação de uma ferramenta de integração contínua.

Este trabalho tem como objetivo implantar uma ferramenta de integração contínua no Núcleo de Práticas em Informática da Universidade Federal do Ceará do Campus de Quixadá e como objetivos específicos estudar e analisar ferramentas de integração contínua, e selecionar a que melhor se adapta ao Núcleo de Práticas em Informática; selecionar e implantar uma ferramenta de integração contínua automatizada e por fim coletar relatos e resultados provenientes da implantação da ferramenta.

2 Fundamentação Teórica

Nesta seção será apresentado os conceitos necessários para um completo entendimento deste trabalho. A seção 2.1 aborda os conceitos inerentes à gerência de configuração, juntamente com algumas ferramentas CASE (Computer-Aided Software Engineering), com ênfase em Integração Contínua descrito na seção 2.2 e por fim na seção 2.4 será descrito o processo do NPI e uma ênfase no processo de Gerência de Configuração.

2.1 Gerência de Configuração

A gerência de configuração é a área da engenharia de software responsável pela evolução do software. Ela atua durante todo o ciclo de vida do produto de software e, por meio de técnicas, ferramentas e metodologias, visa garantir que as mudanças que irão ocorrer dentro do ciclo de vida do desenvolvimento do software sejam identificadas, avaliadas e comunicada a todos os envolvidos através de ferramentas que auxiliam neste processo de evolução. Portanto "o propósito do processo de Gerência de Configuração é estabelecer e manter a integridade de todos os produtos de trabalho de um processo ou projeto e disponibilizá-la a todos os envolvidos" (SOFTEX, 2013).

2.1.1 Plano de Gerenciamento de Configuração

O Plano de Gerenciamento de Configuração (PGC) descreve todas as atividades de configuração e mudança que serão realizadas durante o projeto. Um conjunto de atividades, responsabilidades, ferramentas, recursos e etc. A gerência de configuração tem como objetivo garantir a integridade dos itens de configuração, que são quaisquer artefatos que estejam sob custódia da gerência de configuração, através do versionamento, da identificação e controle de mudanças e acesso (PRESSMAN, 2010).

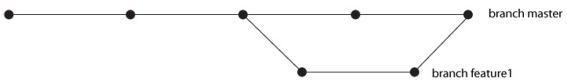
2.1.2 Sistema de Controle de Versão

Um sistema de controle de versão: "[...] combina procedimentos e ferramentas para gerenciar diferentes versões de objetos de configurações que são criadas durante o processo de engenharia de software"(PRESSMAN, 2010, p. 927). Atualmente, o uso de sistemas de controle de versão se tornou comum nas empresas de grande e pequeno porte. Tais ferramentas permitem que se tenha o controle de diferentes versões de arquivos que estão submetidos ao versionamento, recuperação de versões antigas, visualização de alterações realizadas em arquivos e saber por quem, e quando o arquivo foi alterado. Através de

comandos (i.e.,*check-in*,*check-out*) os usuários conseguem se comunicar com o repositório a fim de obter os artefatos ali armazenados (MENEZES, 2011).

Em situações especiais, faz-se necessário que os desenvolvedores trabalhem em uma linha diferente da original chamada de *mainline* geralmente essa situação ocorre quando tem-se como objetivo a correção de *bugs* de versões anteriores do repositório, nesse caso, um *branch*, uma ramificação na linha de desenvolvimento do controle de versão, é criado afim de permitir a realização desta ação, concedendo assim o trabalho em paralelo sobre o mesmo repositório. A Figura 1 demonstra a criação de um *branch* paralelo à linha de

Figura 1: Branch no Sistema de Controle de Versão.



Fonte: Eis (2012).

desenvolvimento principal chamada de branch feature1 e branch master respectivamente. Posteriormente as ações realizadas no *branch feature1* são incorporadas ao *branch master*.

Os sistemas de controle de versão podem possuir três características: Local, Centralizado e Distribuído. Um sistema de controle de versão local armazena todas as informações de um arquivo submetido ao versionamento na máquina, localmente, guardando diferentes versões daquele arquivo. Enquanto um sistema de controle de versão centralizado como o nome diz, possui um único servidor centralizado, como o *subversion* [1], *perforce* [2] este tipo de padrão de SCV mantém em seu único servidor todos os arquivos versionados. Para cada comando de comunicação realizado nos arquivos versionados, uma requisição deverá ser feita, podendo gerar lentidão ou deixar o servidor fora de funcionamento. E por fim os sistemas de controle de versão distribuídos que possuem um servidor central onde os arquivos são submetidos ao versionamento, entretanto, cada desenvolvedor possui em sua máquina de trabalho as versões que estavam no servidor, tornando cada *workstation* um "servidor", portanto, caso ocorra um problema no servidor central, estes podem ser recuperados via *workstation*, mantendo assim, a integridade dos arquivos e evitando ser um ponto único de falha (GIT, 20_) .

2.1.3 Sistema de Controle de Mudança

Todo software sofre mudanças enfrentar essas mudanças é o papel da gerência de configuração, e para isso o gerente de configuração utiliza um sistema de controle de mudança. "O controle de mudança combina procedimentos humanos e ferramentas

[1] http://subversion.apache.org
[2] http://www.perforce.com

automatizadas para proporcionar um mecanismo de controle de mudança" Pressman (2010, p .930). As mudanças devem ser avaliadas com cautela baseando-se, em seu custo benefício. Uma combinação de esforço e *business value*. A mudança tem início quando um "cliente"solicita a mudanças através de um formulário, conhecido como *change request*.

Nesse formulário está descrito os aspectos da mudança. Após a solicitação ser realizada, esta deve ser avaliada, verificando se a mesma já foi solicitada, ou corrigida em caso de *bugs*. Após a mudança ser validada, uma equipe de desenvolvedores avaliam os impactos que esta mudança tem sobre o sistema, verificando custo/benefício e esforço de realização (SOMMERVILLE, 2011). Posterior a esta análise, a mudança será avaliada por um comitê de controle de mudança (CCB), que avaliará o impacto da perspectiva do negócio, que decidirá se esta mudança será revisada, aprovada ou reprovada. Alguns sistemas que fornecem este controle sobre as mudança são: *Redmine* [3]*, GitHub* [4] *Jira* [5]

2.1.4 Auditoria de Configuração

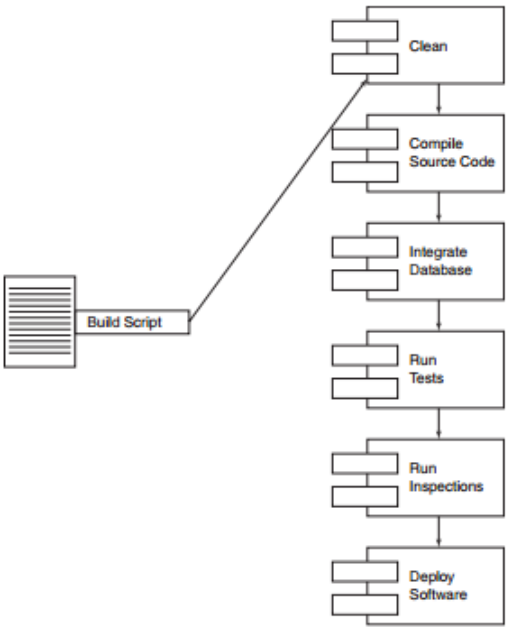
"Uma auditoria de configuração de software complementa a revisão técnica formal ao avaliar um objeto de configuração quanto às características que geralmente não são consideradas durante a revisão" Pressman (2010, p .934). Ela tem como objetivo garantir que mesmo com as mudanças realizadas no software, a qualidade foi mantida. As auditorias se dividem em dois tipos: auditorias funcionais e auditorias físicas, a auditoria física baseia-se em verificar se os itens de configuração estão devidamente atualizados e se as práticas e padrões foram realizados da maneira correta, enquanto a auditoria funcional busca verificar os aspectos lógicos dos itens de configuração.

2.1.5 Ferramentas de Build

As ferramentas de *build* têm como objetivo automatizar processos repetitivos, aumentando a produtividade e facilitando o trabalho do desenvolvedor. Através da definição de uma rotina, ou conjunto de comandos, o desenvolvedor informa a ferramenta que tipo de processo ele deseja automatizar, podendo ser uma compilação, teste de classe, recriação de uma tabela nova no banco de dados, comprimir arquivos css e javascript. Cabe ao desenvolvedor definir o escopo da automatização. Alguns exemplo deste tipo de ferramenta são: *Ant, Grunt, Gulp, Maven.*

[3] http://www.redmine.org
[4] http://www.github.com
[5] https://www.atlassian.com/software/jira

Figura 2: Processo Lógico de uma Build.



Fonte: Duvall, Matyas e Glover (2007).

Na figura Figura 2 um *script* foi definido para realizar as seguintes funções: *clean* no projeto, compilação do código fonte, integração com o banco de dados, execução dos testes, inspeções no código e por fim,o *deploy* da aplicação.

2.2 Integração Contínua

A integração contínua tem como objetivo identificar erros mais rapidamente, permitindo que alterações efetuadas e integradas aos repositórios dos sistemas de controle de versão sejam posteriormente verificadas e, caso erros ocorram, estes sejam notificados imediatamente ao autor da alteração. A melhor definição acerca de integração contínua foi definida por Fowler (2006)

> "[...] uma prática de desenvolvimento de software onde os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por uma *build* automatizada (incluindo testes) para detectar erros de integração o mais rápido possível. Muitos times acham que essa abordagem leva a uma significante redução nos problemas de integração e permite que um time desenvolva software coeso mais rapidamente."Fowler (2006, tradução nossa).

2.2.1 Características de Integração

Os requisitos para utilização de uma ferramenta de integração contínua de acordo com Sepalla (2010) são:

- **Uma conexão com um sistema de controle de versão:**

  A integração contínua necessita desta conexão, pois ela identifica as alterações ocorridas no repositório, e inicia o processo de integração.

- **A definição de uma *build*:**

  A integração contínua possui uma *build* privada que será executada assim que o processo de integração for iniciado, e é esta *build* que definirá quais ações serão realizadas no processo de integração, tais como compilação, testes, análise de código.

- **Um mecanismo de *feedback*:**

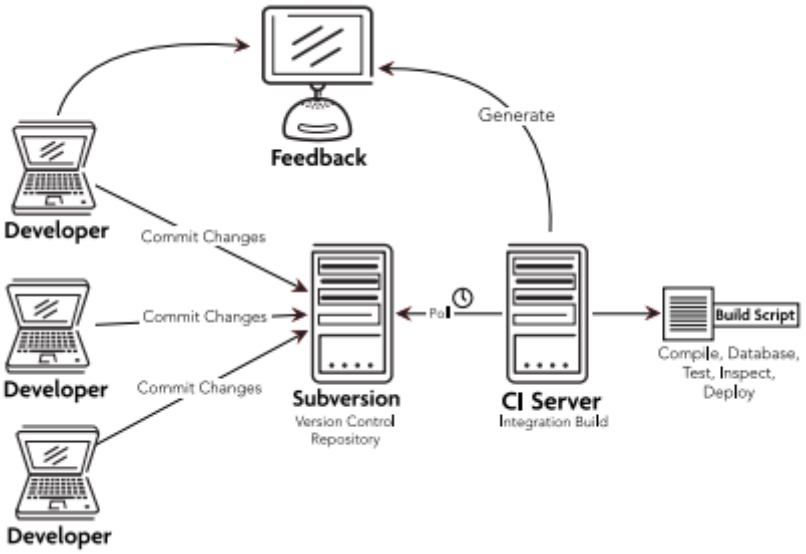
  Um dos principais objetivos da integração contínua consiste em seu *feedback* imediato, sendo assim, um mecanismo deste tipo é essencial para a ferramenta, tais como e-mail, sms.

- **Um processo de integração do código:** O processo de integração consiste em como este será realizado, se manualmente ou através de um servidor de integração contínua.

2.2.2 Processo de Integração

A integração ocorre quando alguma mudança é enviada ao sistema de controle de versão do repositório, que, através de um servidor de integração contínua identifica as mudanças e executa sua *build* privada (MRÁZ, 2013).

Figura 3: Ambiente de Integração Contínua.



Fonte: Duval (20_).

A Figura 3 descreve um ambiente em que um servidor de integração contínua é utilizado. Existem três ambientes de trabalho distintos formado por três desenvolvedores, que obtiveram uma cópia do projeto do repositório do SCV para trabalharem em suas *workstation*. Durante o trabalho, alterações foram efetuadas e foi realizado o *commit* ao repositório central. Após a inserção junto ao repositório, o servidor de integração contínua verifica as alterações e executa uma *build* de integração. Caso exista um problema com a *build* e esta não seja bem sucedida, o responsável pela alteração será informado sobre o ocorrido, assim seu objetivo em diante será a correção da *build*.

2.2.3 Benefícios da Integração Contínua

Esta subseção tem como objetivo destacar os principais objetivos existentes na utilização de uma ferramenta de integração contínua.

As principais vantagens em utilizar um servidor de integração contínua segundo Duvall, Matyas e Glover (2007, p. 29) são:

- **Redução de Riscos**: Através da detecção imediada de código quebrados, ou incorretos, reduz-se riscos atrelados ao produto.
- **Redução de processos manuais repetitivos**: Um conjunto de tarefas são executadas automaticamente pela build privada do servidor de integração contínua.
- **Permitir melhor visibilidade do projeto**: Através da identificação de informações inerentes ao desenvolvimento, como frequência de códigos defeituosos, módulos mais complexos, permitindo maior gerenciamento do projeto.

- **Estabelecer uma maior confiança no produto do time de desenvolvimento**: Através da visualizações de mudanças bem sucedidas, os desenvolvedores sentem maior confiança ao realizarem mudanças.

- **Coleta de métricas a cada *build***: Um dos benefícios que podem ser obtidos através da ferramenta, é a coleta das métricas do código a ser desenvolvido e alterado. Através da utilização de uma ferramenta de análise estática de código é possível realizar uma análise de violações, não conformidades. E por meio desses indicadores, a equipe consegue ter um maior controle no desenvolvimento do software, investindo na melhoria contínua do software.

2.2.4 Integração Contínua Manual

Na integração contínua manual o processo de integração é realizado individualmente, possibilitando que apenas um desenvolvedor realize *check-in* no repositório durante o intervalo de integração Menezes (2011). Este tipo de abordagem permite que apenas uma pessoa realize o *check-in*, assim, as integrações serão contínuas e seguidas, não paralelas. Este tipo de abordagem garante uma maior confiabilidade nas integrações, pois segue um padrão de integração e os itens do repositório possuem maior consistência, garantindo que a estrutura do repositório seja mantida (MENEZES, 2011).

2.2.5 Integração Contínua Automatizada

A integração contínua automatizada é auxiliada pelo uso de um servidor de integração contínua, que obtém do controle de versão as alterações realizadas e executa sua *build* privada com o objetivo de verificar possíveis erros gerados por essas modificações.

> Integração contínua automática possui a vantagem de ser escalável e, deste modo, oferecer maior suporte ao trabalho colaborativo. Com a utilização de Servidores de IC, a responsabilidade de realizar construções da integração é retirada dos desenvolvedores. Portanto, os desenvolvedores podem realizar *check-in* sem a necessidade de conquistar a vez de integrar. Esse fator é fundamental para que os *check-ins* continuem sendo verificados sem a necessidade de um desenvolvedor realizar a construção e identificar problemas, resultando na eliminação do gargalo humano. Menezes (2011, p .54).

2.2.6 Processo de Escolha da Ferramenta

Esta seção define um conjunto de características que auxiliam no processo de escolha de uma ferramenta de integração contínua.

- **Suporte à Linguagem:**

O processo de escolha de um servidor de integração continua deve ser baseado de acordo com o suporte a linguagem, visto que alguns sistemas são construídos para trabalharem com uma linguagem de programação específica.

- **Suporte ao Sistema de Controle de Versão:**

  Como explanado anteriormente, a importância do SCV dentro de um servidor de integração contínua é altíssima, portanto escolher uma ferramenta que integre-se com o repositório é essencial, pois alguns servidores fornecem suporte a SCV mais populares, como *Subversion*, *Git*, entretanto pode não haver suporte ao *Mercurial* por exemplo.

- **Segurança:**

  Garantir que somente pessoas autorizadas devem ter acessos aos artefatos existentes no servidor de integração contínua.

- **Extensibilidade:**

  Capacidade da ferramenta ter funcionalidades adicionadas por meio de *plugins*, ser extensível.

- **Usabilidade:**

  Possuir baixa dificuldade na realização de ações dentro da ferramenta, boa aprendizagem, compreensibilidade.

- **Instalação e Configuração:**

  Facilidade de instalação em diferentes ambientes de operação, tais como sistemas operacionais, hardware através da utilização de recursos. Documentação clara e objetiva do processo de instalação.

2.3 Métricas de Software

Uma métrica de software é uma característica de um determinado sistema de software, desenvolvimento, processo ou documentação, de modo que possa ser medido (SOMMERVILLE, 2011).

As métricas de software são dados quantitativos que informaram o estado de um sistema de software. Através desses dados, é possível um maior controle e tomada de decisão pela parte gerencial de uma organização de software (GUARIZZO, 2008). São exemplos de métricas segundo Merson et al. (2014):

- Cobertura de Testes
- Complexidade Ciclomática

- Débito Técnico

Dentre o conjunto de métricas avaliados, as violações estão inseridas. Violações são não conformidades encontradas em um conjunto de regras definidas.

As violações podem ser classificadas em diferentes graus de severidade:

- Critical: Causam perda de dados, vulnerabilidades de segurança, tornam o sistema inutilizável.
- Major: Erros que impactam uma minoria de usuários do sistema.
- Normal: Erros que afetam uma parte da funcionalidade do sistema.
- Minor: Estão mais relacionados ao estilo do código.

2.4 Processo do NPI

O NPI possui um modelo de processo definido, este processo é baseado nos modelos e metodologias Scrum, MPS.BR e XP. Este modelo define as práticas e o modelo de trabalho dos envolvidos nas atividades do núcleo. Dentro do modelo de processo definido no NPI [6] este trabalho tem como objetivo focar no modelo de processo de gerência de configuração.

O NPI subdivide-se em dois turnos, manhã e tarde, sendo cada turno supervisionado por um professor supervisor diferente. Estes turnos podem ou não estar trabalhando no mesmo projeto, embora o mais comum é que trabalhem em projetos diferentes. As equipes contam com em média oito membros onde comumente destes, dois são alocados para as atividades de requisitos e testes, um para liderança técnica, enquanto o restante da equipe é alocado para as atividades de desenvolvimento, incluindo o líder técnico. O professor supervisor tem como papel o auxílio aos líderes técnicos, acompanhamento do projeto, avaliação dos estagiários, escolha dos projetos a serem desenvolvidos pelas equipes e usualmente realizar o papel de *Product Owner*.

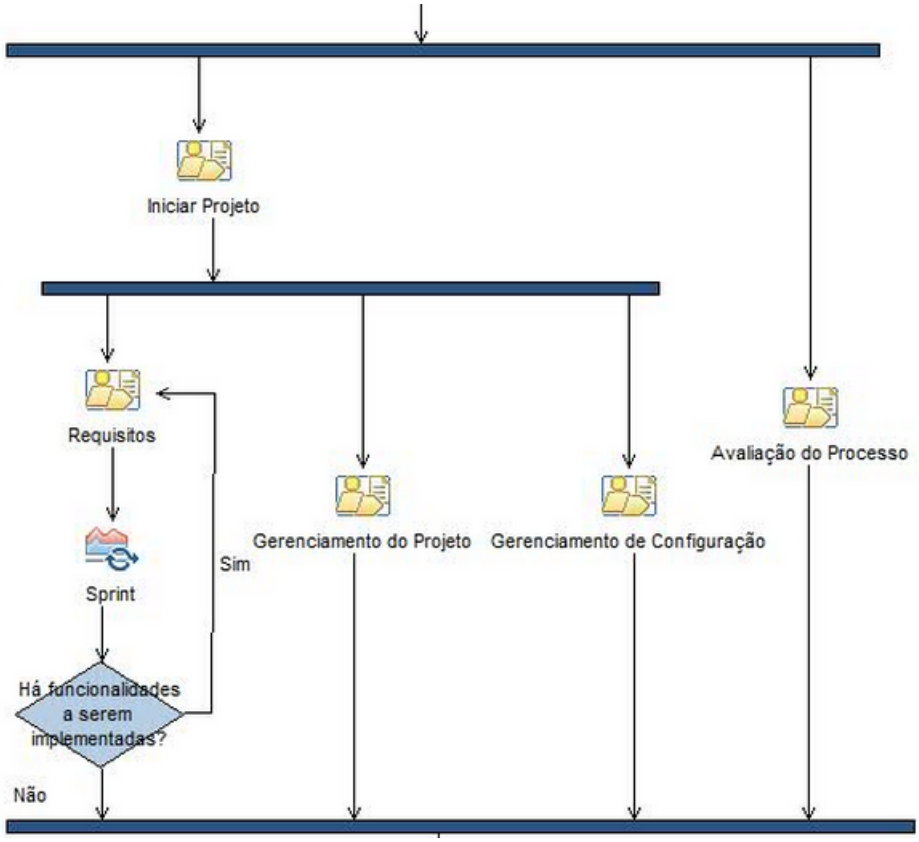
O líder técnico possui papel gerencial bem como de desenvolvimento, suas atribuições partem desde a condução de reuniões, resolução de conflitos,atribuição e definições de tarefas, até o acompanhamento das atividades.

A Figura 4 mostra o processo utilizado no NPI modelado através da ferramenta EPF Composer. Na figura existem duas atividades que ocorrem em paralelo, são elas: Avaliação do Processo e Iniciar Projeto, este que subdivide-se em mais três atividades, a primeira delas a atividade de Requisitos, que posteriormente fornece entrada para um ciclo de *Sprints* que ocorrerá enquanto houver funcionalidades não implementadas,

[6] http://www.npi.quixada.ufc.br/processo/

simultaneamente com a atividade de Requisitos estão de Gerenciamento do Projeto e o Gerenciamento de Configuração.

Figura 4: Processo do NPI.



Fonte: NPI (201_).

2.4.1 Processo de Gerência de Configuração do NPI

O modelo de processo[7] relacionado a gerência de configuração é descrito na Figura 5. Este modelo de processo possui duas atividades que serão descritas abaixo:

- **Criar Plano de Gerenciamento de Configuração:** Esta atividade é realizada pelo líder técnico da equipe envolvida. Esta atividade subdivide-se em quatro etapas são elas:
  - **Identificar Itens de Configuração:** Esta atividade caracteriza-se pela criação, especificação e seleção dos produtos de trabalho, ferramentas, itens que tem objetivo descrever os produtos de trabalho. Exemplos de itens desta atividade são: Requisitos, Diagramas, Testes.
  - **Atribuir Identificadores únicos para os itens de configuração:** Esta atividade possui um nome bem sugestivo tem como intuito atribuir a cada

[7] http://www.npi.quixada.ufc.br/processo/

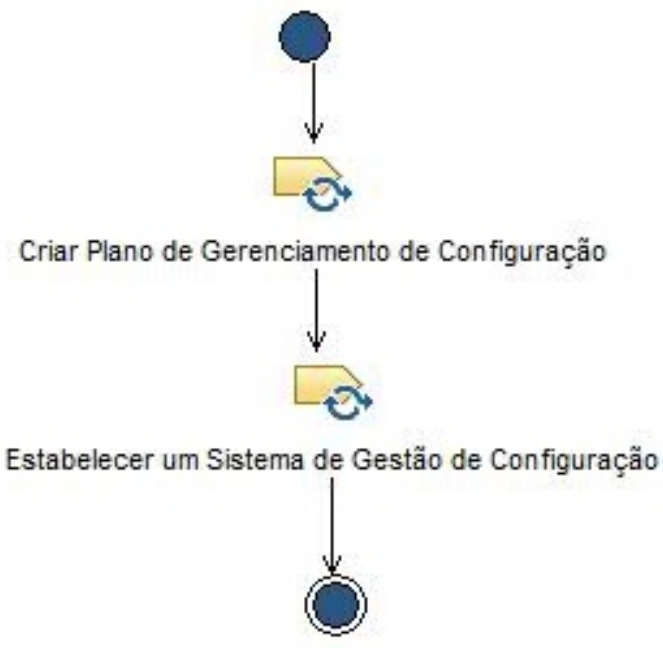
item de configuração um identificador único de modo a facilitar a identificação dentro do projeto. O identificador segue o padrão [$PROJETO$]-[$TIPO$]-EXTRA.EXTENSÃO. Como exemplo um artefato possuiria o seguinte identificador: [GPA]-[REQ]-Especificacao.doc

  - **Identificar o responsável por cada item de configuração:** Esta atividade tem como objetivo atribuir a cada item de configuração um responsável, permitindo assim, uma maior facilidade na identificação do responsável de um determinado item de configuração.
  - **Criar Plano de Gerenciamento de Configuração:** Esta atividade tem como objetivo a elaboração do PGC explicado na subseção 2.1.1 por meio dos dados obtidos com as tarefas anteriores. O plano define os responsáveis pelas atividades de Gerência de Configuração, ferramentas e ambientes a serem utilizados e todos os itens de configuração identificados.

- **Estabelecer um sistema de Gestão de Configuração:** Esta atividade tem como requisito que o plano de gerenciamento de configuração esteja concluído, e possui apenas uma etapa:

  - **Estabelecer um sistema de Gestão de Configuração:** Esta atividade tem como objetivo definir as ferramentas de acesso, ambiente de armazenamento e métodos para criação e alteração dos itens de configuração (NPI, 201_).

Figura 5: Processo de Gerenciamento de Configuração.



Fonte: NPI (201_).

3 Trabalhos Relacionados

Nesta seção será descrito trabalhos que influenciaram os conceitos envolvidos neste trabalho, além de demonstrar pontos comuns e distintos entre si e o proposto.

O trabalho de Pereira, Arruda e Gomes (2013) descreve a implantação de uma ferramenta de integração contínua em um departamento de desenvolvimento e pesquisa, o Sedna, de uma empresa de engenharia, em um ambiente de MPS.BR nível F. Onde os principais clientes do Sedna são voltados a área de óleo e gás.

Durante o período de implantação, o Sedna fornecia manutenção a três sistemas, onde um tratava-se de uma aplicação web, deste modo o *deploy* da aplicação para todos os seus usuários era de responsabilidade do Sedna. Tal ação tornava-se bastante custosa devido ao grande número de web sites que deveriam ser atualizados.

Dentro do Sedna algumas ferramentas de gerência de configuração já eram utilizadas, tais como o Atlassian Jira, Subversion (SVN) e o Atlassian Confluence. Ainda com a utilização destas ferramentas a equipe possuía grandes dificuldades no tempo de realização do *deploy*, pois esta atividade consumia uma grande parte do tempo da equipe, tempo de aprendizagem e realização do *deploy* da aplicação, principalmente para novos membros da equipe, e um *feedback* atrasado para problemas básicos de commits errôneos.

Os autores descrevem que os pontos positivos da integração estão a utilização de uma ferramenta de integração contínua da mesma empresa que fornecia o sistema de gerenciamento de projetos, a ferramenta utilizada foi o Atlassian Bamboo e a utilização de ferramentas de automação do processo de build, o que facilitava o trabalho da integração contínua. Bem como o autor destaca as experiências negativas da implantação, que estão na ausência de uma máquina com requisitos mínimos exigidos para a utilização, a ausência de treinamento da equipe, onde os conhecimentos de IC estavam com o líder de projetos e o gerente de configuração, e por fim a ferramenta não fornecia suporte ao *redeploy* dos artefatos gerados, o que gerou uma barreira na equipe acerca da ferramenta.

Uma das principiais vantagens da utilização da integração contínua é o seu *feedback* imediato acerca de problemas de integração, e, entender e interpretar as principais causas dos problemas de integração foi realizado por Miller (2008). Quando este percebeu que em um projeto da Microsoft, o Service Factory, a maior causa de falha na build eram violações no sistema de análise de código seguido por testes automatizados e erros de compilação.

A obtenção de métricas através da coleta contínua de métricas é uma grande vantagem incluída dentro da integração contínua, pois a sua principal vantagem é o aumento da qualidade do produto ao facilitar o processo de manutenção do software. Para isso Moreira et al. (2010) desenvolveram um *framework* para a extração de métricas

automatizadas, implementado em um ambiente de integração contínua, logo, evidenciou-se a importância das inspeções e teste de códigos nas *builds*, como foi descrito anteriormente.

Os testes são fundamentais na qualidade do software, e na confiança do produto em uma *build* de integração contínua. Assim Kim, Na e Ryoo (2009) propuseram a criação de um *framework* automatizado de testes que permitisse facilitar a construção dos casos de testes, reuso de componentes, relatórios mais legíveis e integrado ao ambiente de integração contínua.

Implantar uma integração contínua em um ambiente de desenvolvimento ágil foi realizado por Abdul e Fhang (2012). Estes propuseram um conjunto de boas práticas e coletaram experiências acerca desta implantação. Estes afirmam que os engenheiros podem não ser tão facilmente convencidos de aceitar a integração contínua, independentemente se esta é implementada *top-down*, quando parte de uma hierarquia mais elevada da organização para as camadas inferiores, ou *bottom-up*, quando parte dos desenvolvedores até os níveis mais altos da hierarquia, e que em ambas as abordagens existem prós e contras.

4 Procedimentos Metodológicos

4.1 Analisar as atividades do Núcleo de Práticas

Esta atividade visa identificar como as atividades ocorrem dentro do Núcleo de Práticas em Informática, com o objetivo de gerar informações que servirão de complemento para a escolha da ferramenta a ser implantada de acordo com o contexto que será detectado nesta ação. Para tal foi analisado o processo[1] existente modelado pela ferramenta *EPF Composer*. Além da experiência do autor, pois este era um estagiário da organização com experiência de 8 meses nas atividades lá realizadas.

4.2 Pesquisar e selecionar a ferramenta de Integração Contínua

Esta atividade consiste em colher informações e selecionar a ferramenta que melhor se adapta a realidade existente no Núcleo de Práticas em Informática. Para a escolha da ferramenta foi preciso definir um conjunto de requisitos que a ferramenta deveria possuir e suprir, de modo a filtrar a ferramenta escolhida dentre as diversas existentes. As definições foram descritas subseção 2.2.6.

4.3 Absorção do perfil dos estagiários do Núcleo de Práticas

Esta atividade tem como objetivo entender e identificar os conhecimentos dos estagiários do núcleo acerca de integração contínua, experiência de uso em projetos pessoais, conhecimento na ferramenta e como esta funciona. Para assim elaborar soluções a serem utilizadas na implantação da ferramenta como: acompanhamento, treinamento, etc. Para tal fora realizado um questionário online de escopo fechado que tinha como objetivo extrair o conhecimentos dos estagiários sobre integração contínua, seu uso e gerência de configuração de software.

4.4 Implantação da ferramenta de integração contínua

A implantação será realizada no núcleo de práticas e a utilização da ferramenta será aplicada a um projeto piloto que estará em desenvolvimento no momento da aplicação, de modo a avaliar o impacto que a ferramenta causará. Ao inicio será realizado um treinamento para explanação do funcionamento da ferramenta, bem como a utilização desta impactará nas atividades dos estagiários para assim, ser implantada.

[1] www.npi.quixada.ufc.br/processo/

4.5 Coleta de Métricas do Código

Esta etapa consiste na coleta contínua de métricas através da inspeção contínua de código do software em desenvolvimento utilizando a integração contínua. Para isso fora utilizado uma ferramenta de análise estática de código, o Sonarqube [2] integrado ao Jenkins em um processo *post-build*, onde o código será verificado e analisado com intuito de encontrar não conformidade que servirá de insumo para melhorias no código. Aumentando assim, sua qualidade, controle de acompanhamento do projeto e diminuição de esforços de retrabalho e futuras manutenções .

4.6 Análise dos dados coletados

Esta atividade tem como objetivo analisar os dados das violações fornecidos pela ferramenta SonarQube, de modo a representar graficamente os dados quantitativos coletados do código.

[2] www.sonarqube.org/

5 Desenvolvimento/Resultados

Esta seção tem como objetivo apresentar as etapas para a elaboração deste trabalho. A seção é composta de cinco subseções. A seção 5.1 retrata a primeira fase do trabalho onde é explanado as atividades do NPI. A seção 5.2 descreve como a ferramenta de integração contínua foi escolhida e sob quais critérios, enquanto que a seção 5.3 detalha qual é o perfil dos estagiários do NPI no que concede à integração contínua. A seção 5.4 descreverá o processo de implantação da ferramenta, seus pontos positivos e negativos.

5.1 Análise das atividades do Núcleo de Práticas em Informática

A análise de como as atividades eram executadas dentro do NPI foi primeiramente analisada de acordo com o processo definido, disponível no site do NPI [1], o qual regula como as atividades ocorrem. As atividades e o processo baseia-se no SCRUM e nas metodologias ágeis como equipes de pequeno número de componentes *Sprint Planning*, *Product Backlog*, *Sprint Review*.

5.2 Pesquisa e Seleção da Ferramenta

O processo de escolha da ferramenta de integração contínua teve como primeiro critério estar em acordo com a realidade das atividades executadas no NPI. Consequentemente o primeiro ponto a ser considerado foi o suporte que a ferramenta deveria prover as linguagens utilizadas no NPI, a linguagem Java. Outro aspecto considerado está relacionado com o custo de aquisição, esta não poderia ser paga ou deveria possuir uma versão *free* que atendesse a demanda das atividades. Extensibilidade, devido as constantes mudanças de tecnologias utilizadas, fornecer suporte a diferentes linguagens, ferramentas, faz-se essencial; usabilidade, pois o NPI não conta com um Gerente de Configuração, sendo assim esta tarefa de manter uma integração contínua deve ser facilitada ao máximo por meio de sua usabilidade, tais como, *inteligibilidade*, *apreensibilidade*; possuir segurança adequada, definição de usuário, papéis.

Deste modo a ferramenta escolhida foi o Jenkins , anteriormente conhecido como Hudson, é uma ferramenta de integração contínua *open source*, que fornece suporte a projetos de diferentes linguagens e tecnologias ,.NET, Ruby, Grails, PHP, bem como Java, linguagem base de sua construção. (SMART, 2011), e como esta preenche os requisitos definidos será descrito abaixo.

[1] www.npi.quixada.ufc.br/processo/

- **Suporte a Linguagem:**

  O Jenkins permite suporte a uma grande gama de linguagens, tais como Java, PHP, Rails, Grails, Python, entre outras.

- **Suporte ao Sistema de Controle de Versão:** O Jenkins consegue integrar nativamente com os principais sistemas de controle de versão tais como: *CVS*, *SVN*, *Mercurial*, e o *Git* através da utilização de plugin.

- **Segurança:** A segurança do Jenkins é habilitada através de permissões e papéis, onde a base de dados de usuários pode ser pela base interna do Jenkins, LDAP, usuários do sistema operacional e também através do usuário vinculado ao GitHub.

- **Extensibilidade:** Jenkins é extremamente flexível e adaptável, permitindo assim oferecer uma melhor atuação para diferentes propósitos, através das centenas de plugins disponíveis. Plugins este que oferecem tudo desde sistemas de controle de versão, ferramentas de build, ferramentas de análise estática de código, notificadores de build, alterações de UI, integração com sistemas externos (*Jira, Redmine*) (SMART, 2011).

- **Usabilidade:** "Primeiramente, Jenkins é fácil de usar. A interface é simples e intuitiva, e o Jenkins como um todo possui uma curva de aprendizado baixa" (SMART, 2011, p .3).

- **Instalação e Configuração:**

  Facilidade de instalação, diferentes ambiente de operação, tais como sistemas operacionais, utilização de recursos. Documentação clara e objetiva do processo de instalação informando dependência existentes.

5.3 Perfil dos Estagiários e conhecimentos sobre Integração Contínua

Entender os conhecimentos dos estagiários do NPI acerca do entendimento, funcionalidade e como esta mudaria suas rotinas de trabalho foi essencial para um entendimento e aperfeiçoamento do processo de implantação da ferramenta.

Para tal, fora realizado um questionário fechado, distribuído de maneira eletrônica para todos os estagiários do NPI. Embora todos não tenham respondido, uma boa amostra foi obtida em confronto com o número total de estagiários em atividade. O referido questionário será apresentado abaixo.

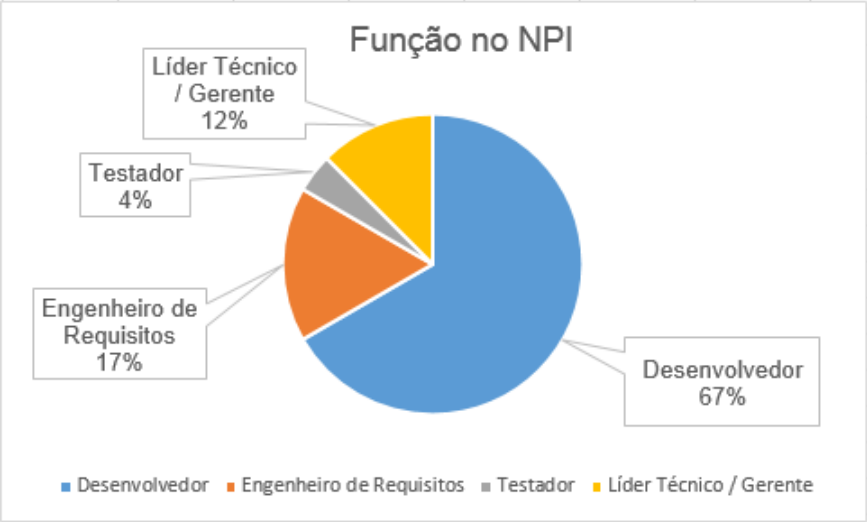
Tabela 1: Conhecimento em Integração Contínua.

Perguntas	**Opções de Respostas**
Qual a sua função no NPI?	Testador Engenheiro de Requisitos Testador Líder Técnico / Gerente
Você sabe o que é Integração Contínua?	Sim Não
Você já utilizou Integração Contínua em algum projeto?	Sim Não
Você conhece ou utilizou alguma destas ferramentas de Integração Contínua?	Atlassian Bamboo Apache Continuum CruiseControl Jenkins / Hudson Outra Desconheço ou nunca utilizei nenhuma delas
Você sabe o que é Gerência de Configuração?	Sim Não

Fonte: Elaborado pelo autor.

Ao todo, vinte e três estagiários participaram da pesquisa de um total de trinta e quatro, cerca de 67% dos estagiários, quase o NPI em sua totalidade, as devidas respostas serão exibidas abaixo na ordem em que as perguntas foram apresentadas aos questionados. A elaboração deste questionário teve como objetivo gerar dados quantitativos de modo a entender o perfil dos estagiários do NPI, facilitando assim o processo de implantação da ferramenta de integração contínua. Onde esses dados geraram conhecimentos para apresentação e explicação as equipes de forma mais proveitosa e focada nas dificuldades.

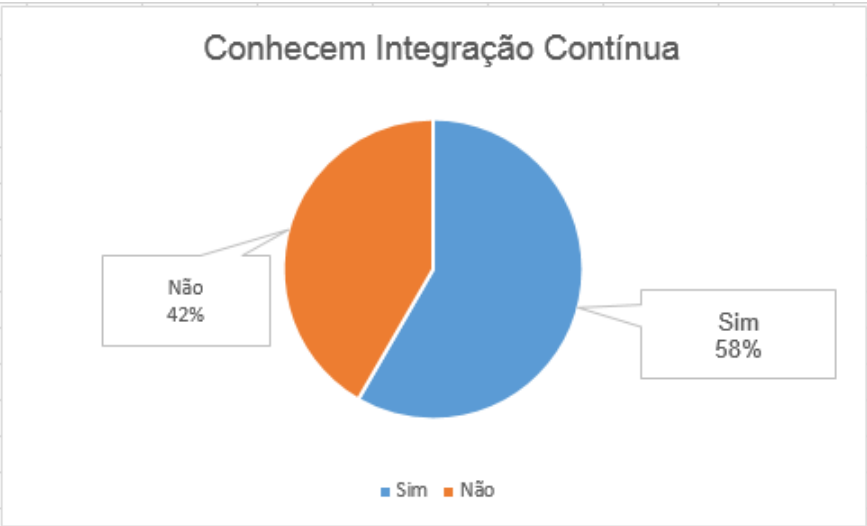
Figura 6: Função no NPI.



Fonte: Elaborado pelo autor.

A Figura 6 demonstra que a grande maioria dos estagiários do NPI estão alocados para atividades exclusivas de desenvolvimento, posteriormente atividade de engenharia de requisitos, líderes técnicos e gerentes. Este resultado obtido por meio das respostas ajudou a elucidar os conhecimentos e os tipos de conhecimentos predominante nos estagiários do núcleo.

Figura 7: Conhecem Integração Contínua.

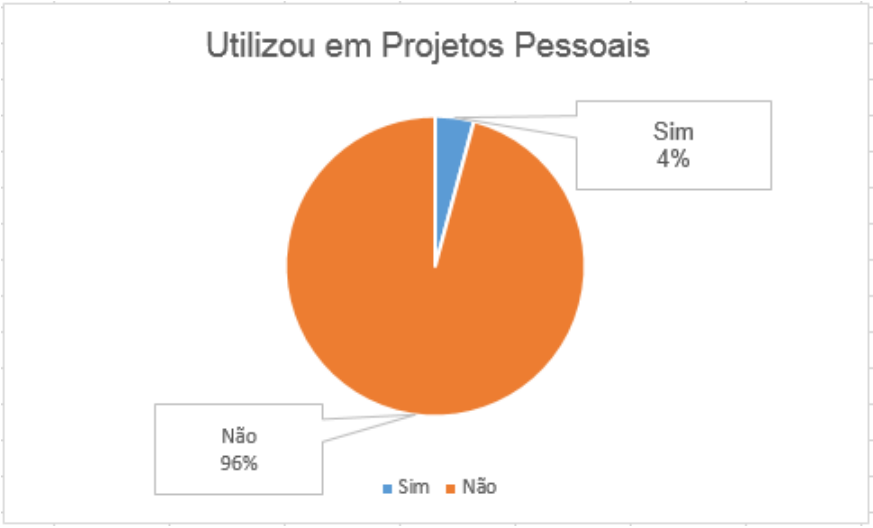


Fonte: Elaborado pelo autor.

Como descrito na Figura 7, a maioria dos questionados conheciam o que era uma ferramenta de integração contínua, e como esta funcionava, embora esta não tenha sido uma superioridade notável, facilitou o processo de implantação em razão dos conhecimentos prévios dos estagiários a cerca do assunto, permitindo assim uma menor rejeição na

implantação devido ao conhecimentos dos benefícios que este tipo de ferramenta causaria ao projeto.

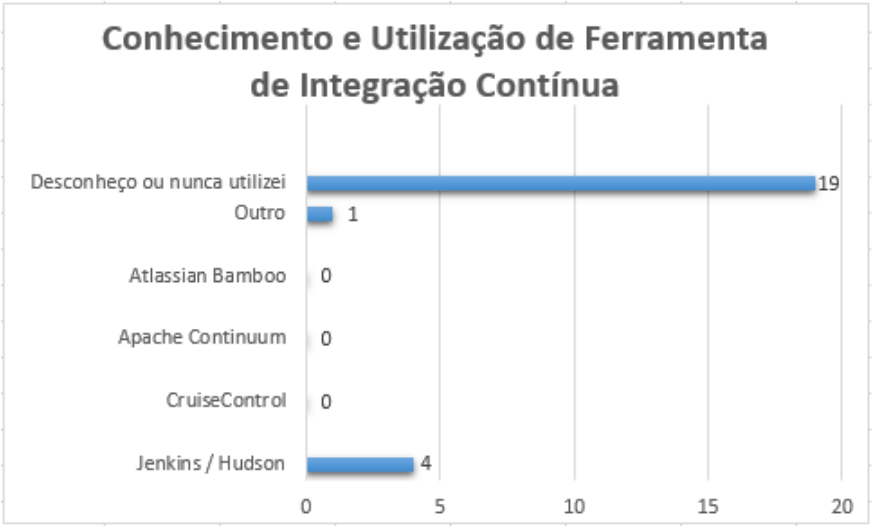
Figura 8: Utilização em Projetos Pessoais.



Fonte: Elaborado pelo autor.

Embora a maioria dos estagiários conheça a ferramenta, pouco mais de 4% dos questionados utilizaram a integração contínua de forma prática, isto é, enfrentaram o impacto de sua utilização. Seja através do *feedback* imediato fornecido pela ferramenta, ou pela alteração de seus processos de trabalho.

Figura 9: Conhecimento e Utilização de Ferramentas de Integração Contínua.

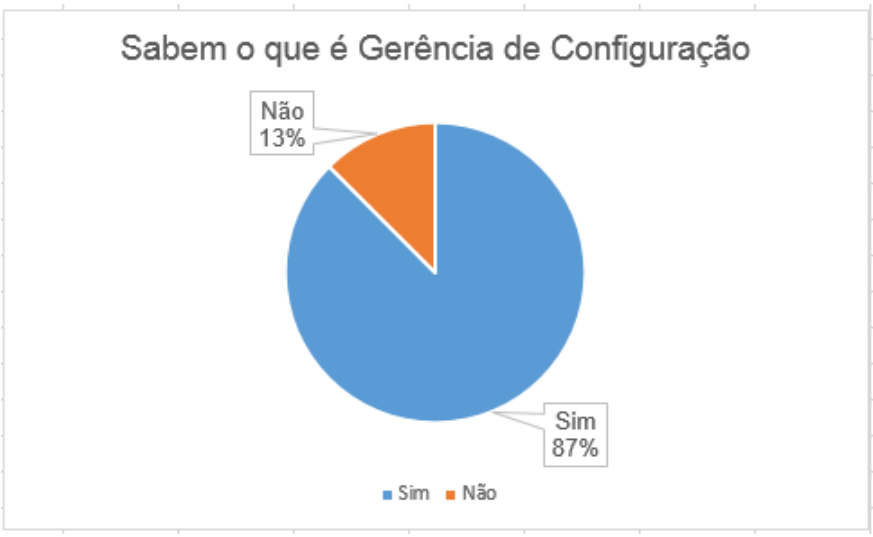


Fonte: Elaborado pelo autor.

O gráfico da Figura 9 contrasta com o gráfico anterior, onde a maioria desconhece ou nunca utilizou nenhuma ferramenta, e dentre a única ferramenta citada, o Jenkins

/ Hudson, enquanto um questionado citou outra ferramenta mas não especificou qual seria esta. De todo modo a familiarização de alguns questionados com a ferramenta facilitará o processo de aceitação desta por parte dos membros, e gerará uma unificação de conhecimento, pois todos os membros irão trabalhar e conhecer apenas uma ferramenta, no caso o Jenkins.

Figura 10: Conhecem Gerência de Configuração.



Fonte: Elaborado pelo autor.

A Figura 10 destaca que cerca de 87% dos estagiários sabiam o que é uma atividade de gerência de configuração, sendo assim, supões-se que estes sabiam quais são as atribuições, responsabilidade e deveres exercidos por esta atividade.

5.4 Processo de Implantação da Ferramenta de Integração Contínua

Esta atividade tem como objetivo explicar como o processo ocorreu, apresentando o contexto do projeto, pontos negativos e positivos da implantação e aspectos a serem melhorados. A distribuição do conteúdo se dará da seguinte forma: subseção 5.4.1 apresentará o projeto piloto onde a ferramenta foi implantada. O projeto GPA (Gestão de Projetos Acadêmicos) foi escolhido devido estar em construção no momento da aplicação deste trabalho e devido a indisponibilidade de aplicação e treinamento para os membros dos outros projetos.

5.4.1 Gestão de Projetos Acadêmicos

O projeto GPA módulo de pesquisa tem como objetivo facilitar o processo de criação, submissão, aceitação e divulgação dos projetos da UFC do campus de Quixadá.

Antes do desenvolvimento do sistema, este processo era totalmente manual. Enquanto este trabalho estava sendo desenvolvido o software do GPA era construído. Abaixo será descrito características deste sistema.

- **Back-end:** A linguagem base da construção do sistema é o Java, com a utilização do Spring Framework. Este framework utiliza-se do padrão arquitetural MVC (Model View Control) além da rapidez de execução e segurança através da utilização do módulo de segurança Spring Security. A utilização do framework começou junto com a construção do sistema, sendo necessário treinamento aos membros das equipes, pois tratava-se de uma tecnologia nova no NPI.
- **Front-end:** Para a criação da aplicação front-end fora utilizado JSP (JavaServer Pages), HTML (HyperText Markup Language), CSS (Cascading Style Sheets) Javascript e o Bootstrap como framework front-end pois esta garante um padrão de interface na aplicação.
- **Build:** Para uma ferramenta de gestão de dependência e ferramenta de build, fora utilizado o Apache Maven, pois este garante que todos os membros do projetos tenham os mesmo itens de configuração corretos do projeto.
- **ORM - (Object-Relational Mapping):** O Hibernate é uma framework para realização do mapeamento objeto relacional com o objetivo de abstrair a persistência dos dados.
- **Gerenciamento de Projeto:** O Redmine foi a ferramenta utilizada para o gerenciamento do projeto durante a construção do sistema.

5.4.2 Pontos Positivos

- **Máquina com boas configurações:**A implantação ocorreu em uma máquina com boas configurações para hospedagem. A execução da ferramenta não sofreu engasgos ou problemas de lentidão devido a alta performance do hardware.
- **Boa receptividade da equipe:** Ao serem explanados de como a ferramenta atuava e quais eram seus benefícios para o projeto e produto, os membros da equipe não reagiram de maneira contrária à utilização da ferramenta, mesmo que esta alterasse seu processo de trabalho.
- **Fácil entendimento:** A ferramenta forneceu uma interface amigável que permitia o bom entendimento das ações que estavam sendo executadas, facilitando a interpretação dos dados por ela gerados. Alguns membros comentaram que comparada a outras ferramenta do gênero como o CruiseControl, o Jenkins possuía uma interface mais bonita e era mais simples de utilizar.

- **Coleta contínua de métricas:** Através da integração do SonarQube como processo *post-build* do Jenkins, o resultados das análises serviram de insumo para melhoria do código através de refatoração. Assim, uma alteração no processo do NPI ocorreu, de modo a contemplar uma atividades de correção de *issues* fornecidas pelo SonarQube ao escopo da *Sprint* a fim de melhorar a qualidade do sistema, reduzir esforços e futuras manutenções. Além de permitir uma maior controle e acompanhamento do projeto, além de criar um entendimento da equipe de como construir o software daquele ponto em diante.

- **Comunicação:** Com a utilização da integração contínua percebeu-se que a comunicação da equipe melhorou, pois as informações de falha de *build* foram discutidas assim que informadas aos responsáveis, que priorizavam sua correção, mesmo em horário fora de expediente. A falha era comunicada pessoalmente aos encontros entre aulas, e caso não fosse possível via meio virtual, redes sociais e e-mail.

5.4.3 Pontos Negativos

- **Ausência de um Gerente de Configuração:** A ausência de um gerente de configuração com experiência na equipe foi sentida durante problemas com o repositório, gerando gastos no tempo da equipe e ociosidade na ferramenta. Embora em algum casos, o autor tenha ficado informalmente responsável por essas atividade perante equipe e projeto.

- **Problemas estruturais:** A execução da ferramenta em uma máquina de um dos laboratórios do NPI constantemente era desligada por algum funcionário da universidade, ou esta era desligada devido as constantes quedas de energia ocorridas, sendo necessário a reinicialização manual.

- **Ausência de testes automatizados:** A ausência de testes automatizados no projeto inviabilizou a valoração de um dos principais benefícios da integração contínua, que se trata da execução de testes em sua *build* privada para verificação de problemas de integração, pois os testes embora fossem documentados e os casos de testes existissem, a forma como era testado era manual, através de teste de exploração.

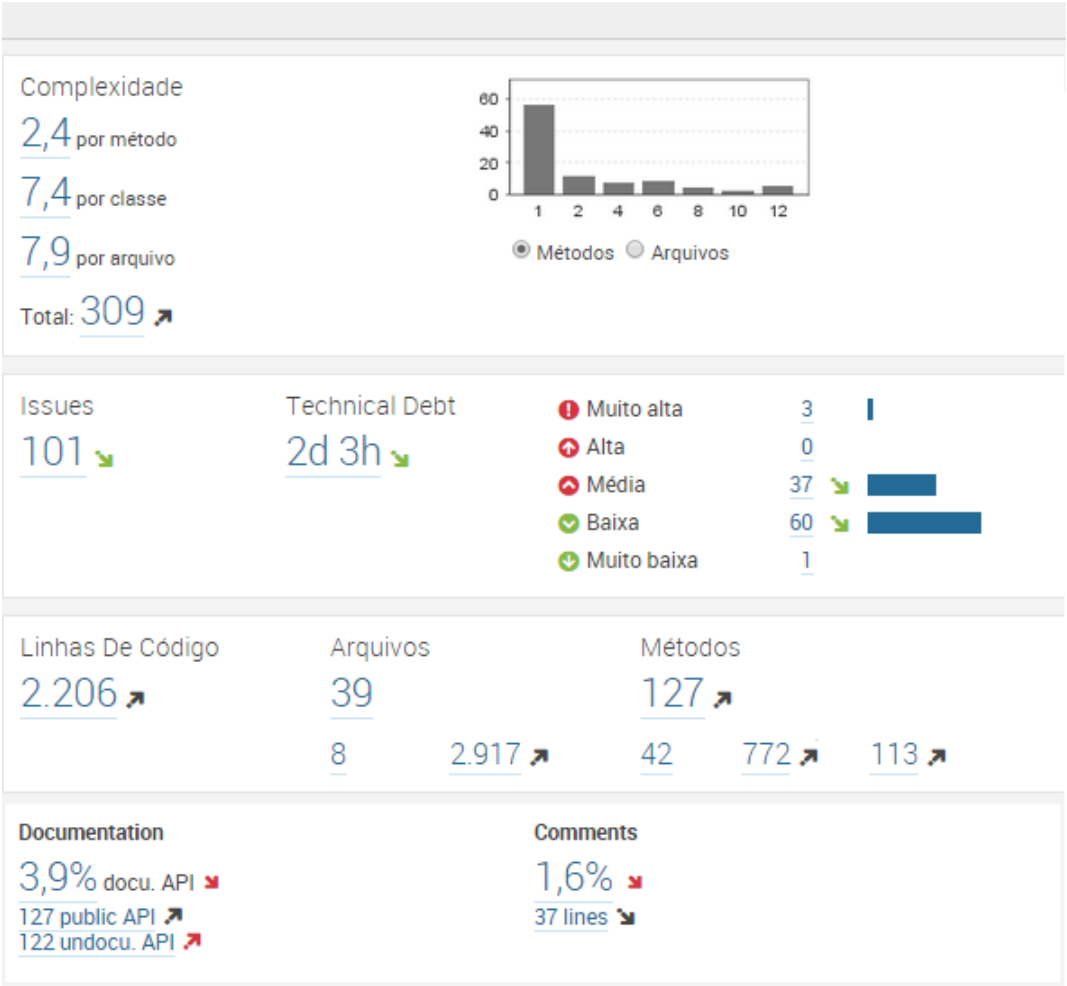
- **Identificação de build em *Branches* distintos:** Ao longo que o projeto trabalhava com o desenvolvimento em vários *branches*, a ferramenta SonarQube não fornecia uma identificação individual para cada, assim tornava-se difícil obter informações mais precisas sobre análise aplicada a cada *branch*.

5.5 Métricas do Código

A coleta das métricas do código foi realizado pelo SonarQube, onde este era um processo de *post-build* da *build* privada do Jenkins. Embora das muitas informações fornecidas pela ferramenta, apenas algumas foram bastantes focadas, como as *issues* e linha de código duplicadas.

Após cada coleta, um histórico foi armazenado na base de dados da ferramenta, onde a última análise do dia foi mantida. O histórico das métricas do código foi mantido no painel da ferramenta, onde todos os membros da equipe podiam visualizar.

Figura 11: Painel do SonarQube.



Fonte: Painel do SonarQube.

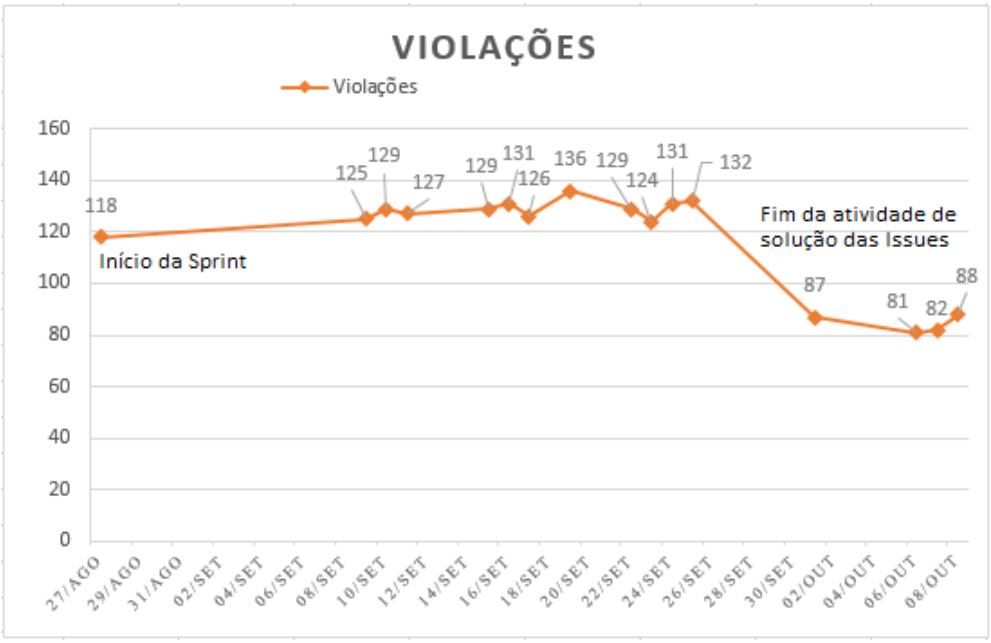
5.5.1 Violações de código

A ferramenta também forneceu o histórico das violações existente a cada análise realizada. O histórico das violações/*issues* está mostrado na Figura 12.

Neste percebe-se que, após a atribuição de um desenvolvedor para a atividade de solucionar as *issues* existentes para o escopo da *Sprint*, esta teve uma queda considerável,

as violações tinham uma média em torno de 129 violações, e após este o processo de solução das *issues*, esta teve uma queda para 87 e posteriormente, 81 violações. Uma queda de 38%, mesmo quando possivelmente os outros membros da equipe continuaram a inserir código com violações.

Figura 12: Histórico de Violações.



Fonte: Elaborado pelo autor.

6 Considerações Finais

Com o mercado cada vez mais competitivo para consumir software, construir software com uma maior qualidade é essencial para a sobrevivência no mercado de trabalho. Por isso, muitas empresas estão investindo na utilização de ferramentas de integração contínua no desenvolvimento de seus produtos. Entretanto, realizar uma implantação de uma ferramenta desta natureza, requer cuidados, pois caso esta não seja suficientemente bem planejada, poderá trazer resultados adversos ao esperado, pois uma implantação forçada ou mal sucedida pode gerar resistência da equipe, bem como diminuir a produtividade da equipe, burocratizar o processo, gerando assim um mal estar entre os membros do time e perca na qualidade do produto.

O trabalho proposto visou implantar uma ferramenta de integração contínua no Núcleo de Práticas em Informática da Universidade Federal do Ceará do Campus de Quixadá. O processo de implantação se iniciou a partir da extração de como os softwares eram desenvolvidos no NPI, obtido através da experiência do autor, e da análise do processo vigente. Somado-se a isto, fora pesquisada e selecionada uma ferramenta de integração contínua adaptada ao desenvolvimento praticado no NPI. Depois de escolhida a ferramenta, esta foi devidamente implantada e assim, conseguiu-se obter pontos positivos e negativos da implantação. Verificou-se que a utilização da coleta contínua das métricas foi essencial na melhoria da qualidade do produto.

O trabalho realizado por Pereira, Arruda e Gomes (2013) serviu como sabe para este trabalho, pois ajudou com conhecimentos sobre integração contínua este trabalho diverge nos pontos positivos e negativos identificados, pois era aplicado em um contexto de ferramentas semelhantes, mas em um organização com um nível de maturidade mais elevado, e converge na abordagem utilizada, e na condução do trabalho.

As limitações deste trabalho são a implantação de uma ferramenta de integração contínua em um ambiente de pequeno porte, onde seu impacto não pode ser validado em um ambiente maior de desenvolvimento. Além de uma validação científica mais criteriosa.

Como trabalhos futuros, sugerimos um acompanhamento de como a implantação desta ferramenta alterará o processo de desenvolvimento, e a coleta de dados que confirmem a melhoria dos sistema a longo prazo além da validação de uma integração contínua em um ambiente de projeto com testes automatizados.

Este trabalho permitiu a criação de uma cultura de integração contínua no NPI, possibilitando que os envolvidos visualizassem o impacto que esta ferramenta exerce no desenvolvimento do software e relatar as lições aprendidas que poderão ser utilizadas em futuras implantações.

Referências

ABDUL, F. A.; FHANG, M. C. S. Implementing continuous integration towards rapid application development. *International Conference on Innovation , Management and Technology Research*, p. 118 – 123, 2012. Citado na página 26.

DUVAL, P. M. Continuous integration: Patterns and anti-patterns. *DZone Refcardz*, n. 1, p. 1–6, 20_. Citado na página 19.

DUVALL, P.; MATYAS, S.; GLOVER, A. *Continuous Integration: Improving Software Quality and Reducing Risk*. [S.l.]: Addison-Wesley, 2007. (A Martin Fowler signature book). ISBN 9780321336385. Citado 3 vezes nas páginas 12, 17 e 19.

EIS, D. Janeiro 2012. Disponível em: <http://tableless.com.br/introducao-das-premissas-dos-controles-de-versao/>. Acesso em: 27.3.2014. Citado na página 15.

FOWLER, M. Maio 2006. Disponível em: <http://martinfowler.com/articles/continuousIntegration.html>. Acesso em: 27.3.2014. Citado na página 17.

GIT. 20_. Disponível em: <http://git-scm.com/book/pt-br/Primeiros-passos-Sobre-Controle-de-Vers~ao>. Acesso em: 27.3.2014. Citado na página 15.

GONÇALVES, E. J. T. et al. Núcleo de práticas em informática: Contribuindo para a formação em sistemas de informação através do desenvolvimento de projetos de software. *XXXIII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (WEI*, Maceió, n. 1, p. 601–610, 2013. Citado na página 12.

GUARIZZO, K. *Métricas de Software*. 38 p. — Faculdade de Jaguariúna, Jaguariúna, 2008. Citado na página 21.

KIM, E. H.; NA, J. C.; RYOO, S. M. Test automation framework for implementing continuous integration. *Sixth International Conference on Information Technology: ew Generations*, n. 6, p. 784–789, 2009. Citado na página 26.

MENEZES, G. G. L. de. *OURIÇO:UMA ABORDAGEM PARA MANUTENÇÃO DA CONSISTÊNCIA EM REPOSITÓRIOS DE GERÊNCIA DE CONFIGURAÇÃO*. Niterói: [s.n.], 2011. Citado 2 vezes nas páginas 15 e 20.

MERSON, P. et al. Continuous inspection: A pattern for keeping your code healthy and aligned to the architecture. p. 13, 2014. Citado na página 21.

MILLER, A. A hundred days of continuous integration. *Agile Conference*, p. 289 – 293, 2008. Citado na página 25.

MOREIRA, G. de S. P. et al. Software product measurement and analysis in a continuous integration environment. *Seventh International Conference on Information Technology*, n. 7, p. 1177 – 1182, 2010. Citado na página 25.

MRÁZ, M. *Use of continuous integration in web application development*. Dissertação (Mestrado) — Masarykova Univerzita Fakulta Informatiky, Brno, 2013. Citado na página 18.

NPI. 201_. Disponível em: <http://www.npi.quixada.ufc.br/processo/>. Acesso em: 20.4.2014. Citado 2 vezes nas páginas 23 e 24.

OLIVEIRA, P. A. de; NELSON, M. A. V. Integração de ferramentas para minimizar erros provenientes de efeitos colaterais inseridos durante a manutenção. *Workshop de Manutenção de Software Moderna*, n. 1, 2005. Citado na página 12.

PADUELLI, M. M.; SANCHES, R. Problemas em manutenção de software caracterização e evolução. *III Workshop de manutenção moderna*, 2006. Citado na página 12.

PEREIRA, J. A. D.; ARRUDA, R. F. V. de; GOMES, A. M. Lessons learned during the implementantion of a continuous integration software in mps.br level f environment. *IX Workshop Anual do MPS*, 2013. Citado 2 vezes nas páginas 25 e 39.

PRESSMAN, R. S. *Engenharia de Software*. [S.l.]: Pearson Education, 2010. 1056 p. Citado 2 vezes nas páginas 14 e 16.

SEPALLA, A. *Improving software quality with Continuous Integration*. Dissertação (Mestrado) — Aalto University, Esbo, Maio 2010. Citado na página 18.

SMART, J. F. *Jenkins The Definitive Guide*. [S.l.]: O'Reilly Media, 2011. Citado 2 vezes nas páginas 29 e 30.

SOFTEX. *Guia de Implementacao - Parte 2: Fundamentacao para Implementacao do Nivel F do MR-MPS-SW:2012*. [S.l.], 2013. Disponível em: <http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_de_Implementacao_Parte_2_2013.pdf>. Acesso em: 17.4.2014. Citado na página 14.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. [S.l.]: Pearson Education, 2011. 529 p. Citado 2 vezes nas páginas 16 e 21.