



Universidade Federal do Ceará
Campus Quixadá
Curso de Engenharia de Software

Guyllherme Tabosa Cabral

**IMPLANTAÇÃO DE UMA FERRAMENTA DE INTEGRAÇÃO CONTÍNUA EM
UM NÚCLEO DE PRÁTICAS EM INFORMÁTICA: RELATO DE EXPERIÊNCIA**

Quixadá, Ceará

2014

Lista de ilustrações

Figura 1 – Branch no Sistema de Controle de Versão	11
Figura 2 – Sistema de Controle de Versão Local	11
Figura 3 – Sistema de Controle de Versão Centralizado	12
Figura 4 – Sistema de Controle de Versão Distribuído	13
Figura 5 – Processo Lógico de uma Build	14
Figura 6 – Ambiente de Integração Contínua	16
Figura 7 – Processo de Gerenciamento de Configuração	20
Figura 8 – Processo do NPI	24
Figura 9 – Função NPI	27
Figura 10 – Conhecem Integração Contínua	27
Figura 11 – Utilização em Projetos Pessoais	28
Figura 12 – Conhecimento e Utilização de Ferramentas de Integração Contínua . .	28
Figura 13 – Conhecem Gerência de Configuração	29

Lista de tabelas

Tabela 1 – Conhecimentos em Integração Contínua	26
---	----

Lista de abreviaturas e siglas

NPI	Núcleo de Práticas em Informática
CMMI	Capability Maturity Model Integration
MPS.BR	Melhoria de Processo Brasileiro
GC	Gerência de Configuração
SCV	Sistema de Controle de Versão
SCM	Sistema de Controle de Mudança
IC	Integração Contínua
UFC	Universidade Federal do Ceará
TI	Tecnologia da Informação
SGBD	Sistemas de Gerência de Banco de Dados
CCB	Change Control Board
SBC	Sociedade Brasileira de Computação
GPA	Gestão de Projetos Acadêmicos
MVC	Model View Control
JSP	JavaServer Pages
HTML	HyperText Markup Language
CSS	Cascading Style Sheets

Sumário

1	INTRODUÇÃO	6
2	TRABALHOS RELACIONADOS	8
3	FUNDAMENTAÇÃO TEÓRICA	10
3.1	Gerência de Configuração	10
3.1.1	Plano de Gerenciamento de Configuração	10
3.1.2	Sistema de Controle de Versão	10
3.1.2.1	Sistema de Controle de Versão Local	11
3.1.2.2	Sistema de Controle de Versão Centralizado	12
3.1.2.3	Sistema de Controle de Versão Distribuído	12
3.1.3	Sistema de Controle de Mudança	13
3.1.4	Auditoria de Configuração	14
3.1.5	Ferramentas de Build	14
3.2	Integração Contínua	15
3.2.1	Características de Integração	15
3.2.2	Processo de Integração	16
3.2.3	Benefícios da Integração Contínua	16
3.2.4	Builds Automatizadas	17
3.2.5	Integração Contínua Manual	17
3.2.6	Integração Contínua Automatizada	17
3.2.7	Processo de Escolha da Ferramenta	18
3.3	Processo do NPI	18
3.3.1	Processo de Gerência de Configuração do NPI	19
4	PROCEDIMENTOS METODOLÓGICOS	21
4.1	Analisar as atividades do Núcleo de Práticas	21
4.2	Pesquisar e selecionar a ferramenta de Integração Contínua	21
4.3	Absorção do perfil dos estagiários do Núcleo de Práticas	21
4.4	Implantação da ferramenta de integração contínua	21
4.5	Coleta de Métricas de Código	21
5	DESENVOLVIMENTO/RESULTADOS	23
5.1	Análise das atividades do Núcleo de Práticas em Informática	23
5.2	Pesquisa e Seleção da Ferramenta	24
5.3	Perfil dos Estagiários e conhecimentos sobre Integração Contínua	25
5.4	Processo de Implantação da Ferramenta de Integração Contínua	29

5.4.1	Gestão de Projetos Acadêmicos	29
5.4.2	Implantação	30
5.4.3	Pontos Positivos	30
5.4.4	Pontos Negativos	30
6	DISCUSSÃO	31
7	CONSIDERAÇÕES FINAIS	32
	Referências	33

1 Introdução

O presente trabalho visa implantar no Núcleo de Práticas em Informática da Universidade Federal do Ceará do Campus de Quixadá (NPI) a utilização de uma ferramenta de Integração Contínua (IC), e relatar a experiência obtida em sua implantação.

Entende-se Integração Contínua como uma ferramenta de gestão de configuração, que auxilia os desenvolvedores e permite que as mudanças realizadas no software sejam imediatamente avaliadas, testadas, verificadas de modo a prover um *feedback* imediato para correção de possíveis erros de integração que somente seriam verificados futuramente após problemas mais complexos de integração (DUVALL; MATYAS; GLOVER, 2007).

Para esta experiência foi utilizado uma ferramenta de integração contínua automatizada, *Jenkins*, como ambiente de aplicação o Núcleo de Práticas em Informática (NPI) da Universidade Federal do Ceará (UFC) do campus de Quixadá.

O NPI é o local onde estudantes que estão em ano de conclusão de curso podem estagiar e aprimorar seus conhecimentos adquiridos no decorrer do curso além de concluir seus componentes curriculares obrigatórios. Este surgiu devido à pouca demanda de empresas de Tecnologia da Informação (TI) na região onde a universidade se encontra, em Quixadá no Ceará.

Dentro do NPI, os projetos 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, aumenta a qualidade dos profissionais formados por ela, além de proporcionar um ambiente real de trabalho que facilita 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 não é devidamente seguido, e não contempla dentro do processo a utilização de integração contínua, ocasionando 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 explanado por Oliveira e Nelson (2005). Eles inseriram a utilização destas ferramentas para evitar a inserção de novos erros proveniente 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 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 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. 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 como 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 da 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 estava com o líder de projeto e o gerente de configuração e por fim a ferramenta não fornecia suporte ao *redesploy* dos artefatos gerados, o que criou um barreira na equipe acerca da ferramenta.

Uma das principais 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 quebra de build eram violações no sistema de análise de código, seguido por testes automatizados e erros de compilação.

A coleta de métricas através da inspeção contínua é uma grande vantagem incluída dentro da Integração Contínua, pois a sua principal vantagem é aumentar a qualidade do produto e facilitar o processo de manutenção do software. Para isso Moreira et al.

(2010) desenvolveram um framework para extração de métricas de forma automatizada implementado em um ambiente de Integração Contínua, logo ficou evidente importância das inspeções e teste de códigos nas builds como foi descrito anteriormente.

Os testes são fundamentais na qualidade no software , na confiança do produto e em uma build de integração contínua, deste modo Kim, Na e Ryoo (2009) propuseram a criação de um framework automatizado de testes que permitissem facilitar os casos de testes, reuso de componentes, relatórios mais legíveis e integrado com um ambiente de integração contínua.

Implantar uma integração contínua em um ambiente de desenvolvimento ágil foi realizado por Abdul e Fhang (2012), onde estes propuseram as melhores práticas para a realização da implantação, além de relatar as experiências obtida neste processo.

3 Fundamentação Teórica

Nesta seção será apresentado os conceitos necessários para um completo entendimento deste trabalho. A seção 3.1 conceitos inerentes a Gerência de configuração, juntamente com algumas ferramentas CASE e com ênfase em Integração Contínua descrito na seção 3.2 e por fim na seção 3.3 será descrito o processo de gerência de configuração do NPI.

3.1 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).

3.1.1 Plano de Gerenciamento de Configuração

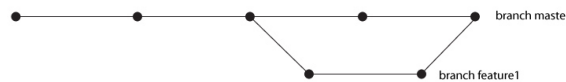
O Plano de Gerenciamento de Configuração (PGC) descreve todas as atividades de configuração e mudança que serão realizadas durante o projeto. Um conjunto de atividades, responsabilidades, ferramentas, recursos e etc. A gerência de configuração tem como objetivo garantir a integridade dos itens de configuração,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

3.1.2 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 sistema 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, visualizar 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 consertar 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, permitindo assim o trabalho em paralelo sobre o mesmo repositório. A figura Figura 1

Figura 1 – Branch no Sistema de Controle de Versão.



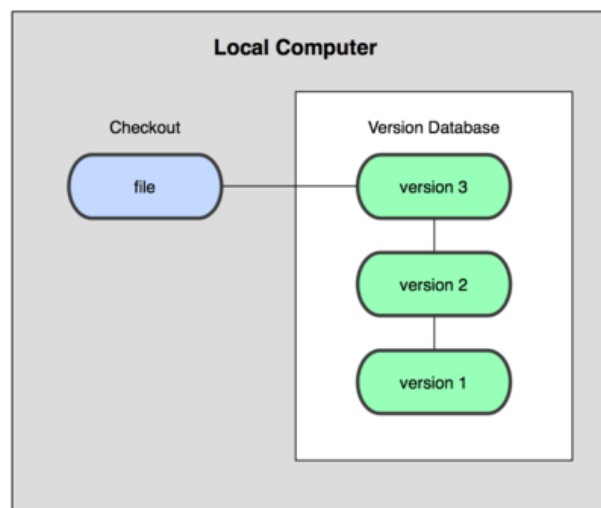
Fonte: Eis (2012).

demonstra a criação de um *branch* paralelo a linha de desenvolvimento principal chamada de *branch feature1* e *branch master* respectivamente, posteriormente as ações realizadas no *branch feature1* são incorporadas ao *branch master*.

3.1.2.1 Sistema de Controle de Versão Local

Um sistema de controle de versão local armazenam todas as informações de um arquivo submetido ao versionamento na máquina local, guardando diferentes versões daquele arquivo localmente como demonstrado na figura Figura 2.

Figura 2 – Sistema de Controle de Versão Local.

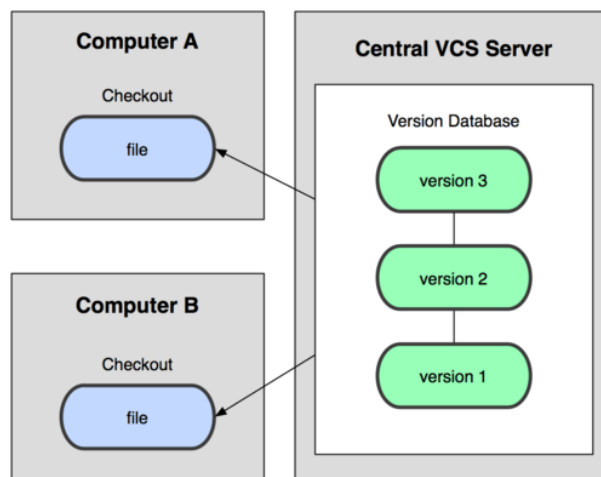


Fonte:(GIT, 20__).

3.1.2.2 Sistema de Controle de Versão Centralizado

Sistema de controle de versão centralizado como o nome diz possuem 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. No exemplo acima dois desenvolvedores

Figura 3 – Sistema de Controle de Versão Centralizado.



Fonte: GIT (20__).

trabalhando em máquinas diferentes realizam a comunicação com o servidor central para obter o artefato de trabalho.

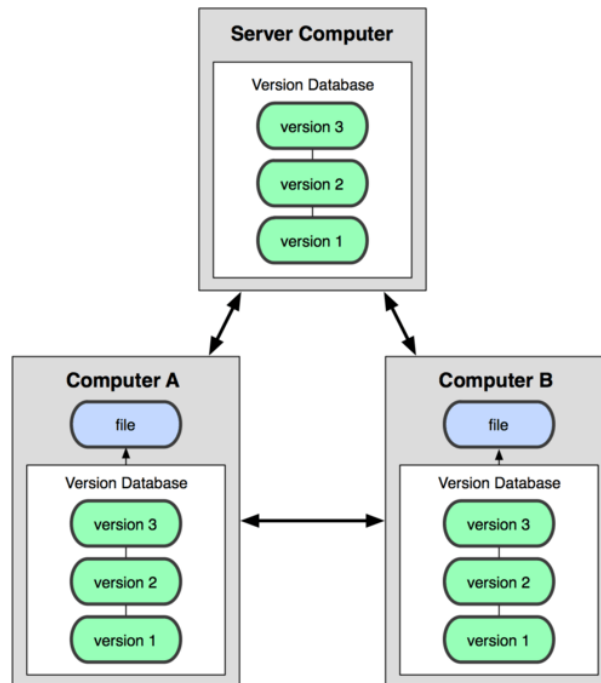
3.1.2.3 Sistema de Controle de Versão Distribuído

Os sistemas de controle de versão distribuído possuem um servidor central onde os arquivos são submetidos a 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 a integridade dos arquivos e evitando ser um ponto único de falha, como mostra a figura Figura 4.

¹ <http://subversion.apache.org>

² <http://www.perforce.com>

Figura 4 – Sistema de Controle de Versão Distribuído.



Fonte: GIT (20__).

3.1.3 Sistema de Controle de Mudança

Todo software sofre mudanças, lidar com as mudanças é o papel da gerência de configuração, e para isso o gerente de configuração utiliza de um sistemas de controle de mudança. "O controle de mudança combina procedimentos humanos e ferramentas 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 é 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 têm sobre o sistema, verificando custo/benefício e esforço de realização (SOMMERVILLE, 2011). Posterior a essa análise, a mudança será avaliada por um comitê de controle de mudança (CCB) que avaliará o impacto da perspectiva do negócio, o 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

³ <http://www.redmine.org>

⁴ <http://www.github.com>

⁵ <https://www.atlassian.com/software/jira>

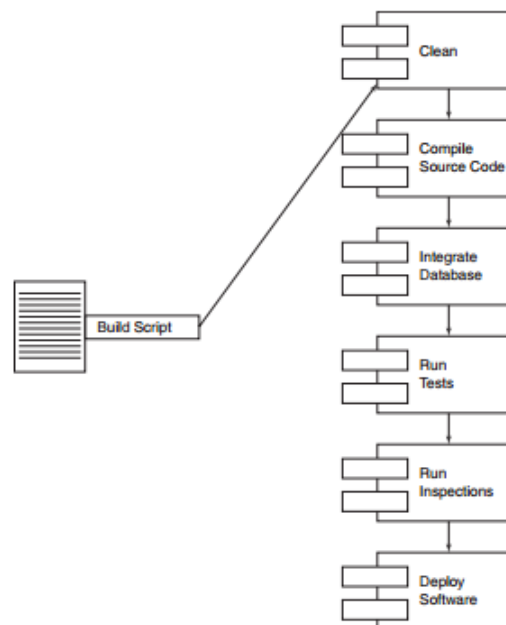
3.1.4 Auditoria de Configuração

"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 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.

3.1.5 Ferramentas de Build

As ferramentas de build tem 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, pode ser desde compilar e testar uma classe, como dropar e criar 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*.

Figura 5 – Processo Lógico de uma Build.



Fonte: Duvall, Matyas e Glover (2007).

Na figura Figura 5 um script foi definido para realizar as seguintes funções, será realizado um clean no projeto, compilará o código fonte, integrará com o banco de dados,

executará testes e inspeções no código e por fim será realizado o *deploy* da aplicação.

3.2 Integração Contínua

A integração contínua tem como objetivo identificar erros o mais rapidamente, permitindo que alterações efetuadas e integradas aos repositórios dos sistemas de controle de versão (SCV) sejam posteriormente verificadas e caso erros ocorram, estes sejam notificados imediatamente ao autor da alteração. A melhor definição a cerca de integração contínua foi definida por Fowler (2000)

"[...] 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 significativa redução nos problemas de integração e permite que um time desenvolva software coeso mais rapidamente."Fowler (2000, tradução nossa).

3.2.1 Características de Integração

Os requisitos para utilizar 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 dá início ao 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 define que ações serão realizadas no processo de integração, tais como compilação, testes, análise de código.

- **Um mecanismo de feedback:**

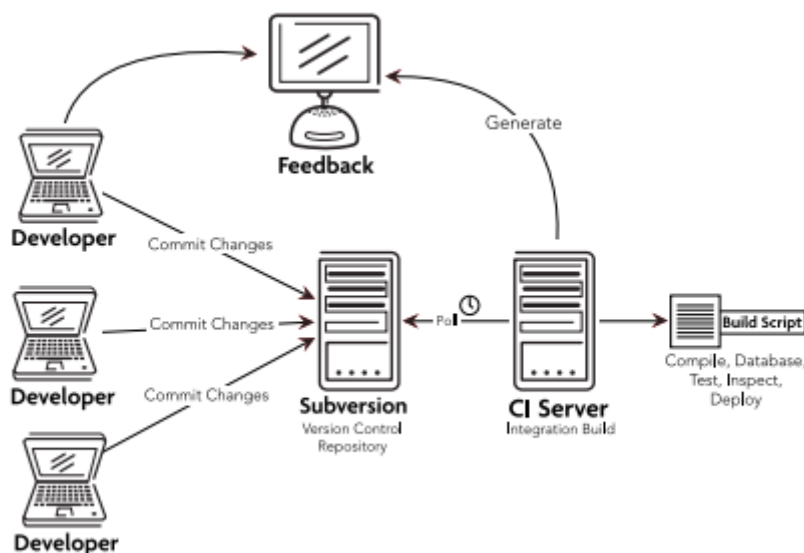
Um dos principais objetivos da integração contínua consiste em seu *feedback* imediato, sendo assim um mecanismo deste tipo é essencial para a ferramenta, tais como e-mail, sms.

- **Um processo de integração do código:** O processo de integração consiste em como esta será realizada, manualmente ou através de um servidor de integração contínua de forma automatizada.

3.2.2 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 a mudança e executa sua build privada (MRÁZ, 2013).

Figura 6 – Ambiente de Integração Contínua.



Fonte: Duval (20__).

A Figura 6 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 realizadas e commitadas 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.

3.2.3 Benefícios da 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 imediata de código quebrados, ou incorretos, reduz-se risco 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.

3.2.4 Builds Automatizadas

Builds são rotinas de execução definidas com o objetivo de reduzir processos repetitivos. Durante o processo de desenvolvimento de um software muitas ações tendem a serem repetidas por parte dos desenvolvedores, utilizar o tempo para a realização de atividades que poderiam ser automatizadas, de forma manual, reduz a produtividade e preocupações com melhorias devido ao tempo "apertado". Somando-se a isso, uma build garante que tudo que está nela definido será executado, evitando assim, que determinada ação seja esquecida, ou caso um novo membro entre na equipe uma explicação do que ele deve fazer, ou não esquecer de fazer, não faz-se necessário.

3.2.5 Integração Contínua Manual

Na IC 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*, as integrações serão contínuas e seguidas, não paralelas. Este tipo de abordagem garante uma maior confiabilidade das integrações, pois segue um padrão de integração e os itens do repositório possuem maior consistência, e a garantia da estrutura do repositório é mantida (MENEZES, 2011).

3.2.6 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.

IC 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).

3.2.7 Processo de Escolha da Ferramenta

- **Suporte a Linguagem:**

O processo de escolha de um servidor de integração contínua 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 servidor 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 existente 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 ambiente de operação, tais como sistemas operacionais, hardware através da utilização de recursos. Documentação clara e objetiva do processo de instalação.

3.3 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.

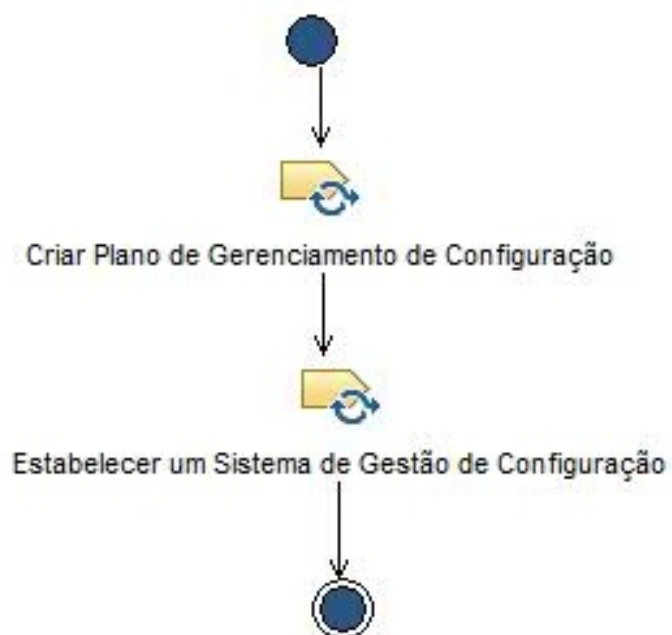
⁶ <http://www.npi.quixada.ufc.br/processo/>

3.3.1 Processo de Gerência de Configuração do NPI

O modelo de processo relacionado a gerência de configuração é descrito na figura Figura 7. Este modelo de processo possui duas atividade que serão descritas abaixo:

- **Criar Plano de Gerenciamento de Configuração:** Esta atividade é realizada pelo líder técnico da equipe envolvida. Esta atividade subdividi-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 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 3.1.1 por meio dos dados obtido 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 7 – Processo de Gerenciamento de Configuração.



Fonte: NPI (201__).

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, para tal foi analisado o processo¹ 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 3.2.7.

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 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. Ao inicio será realizado um treinamento para explanação do funcionamento da ferramenta, bem como a utilização desta impactará nas atividades dos estagiários.

4.5 Coleta de Métricas de Código

Esta etapa consiste na coleta contínua de métricas através da inspeção contínua de código do software em desenvolvimento através da integração contínua. Para isso fora

¹ www.npi.quixada.ufc.br/processo/

utilizado uma ferramenta de análise estática de código, o Sonarqube ² integrado ao Jenkins em um processo *post-build*, que servirá de insumo para melhorias no código, afim de aumentar sua qualidade e diminuir esforços futuros de manutenção.

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.

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*.

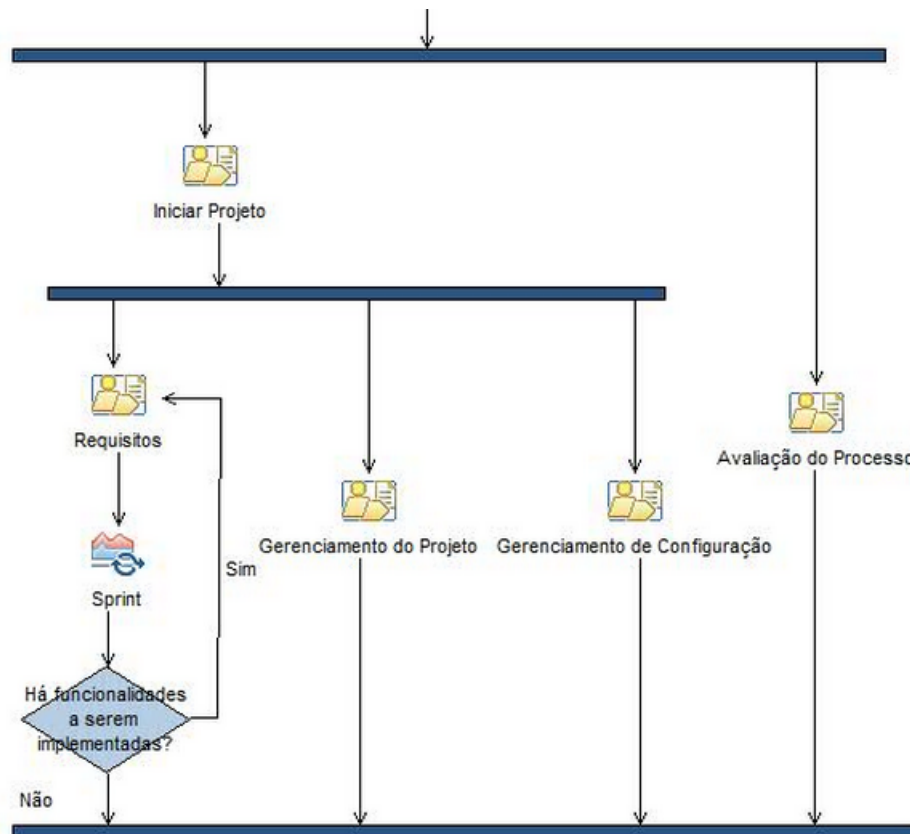
O NPI subdivide-se em dois turnos, manhã e tarde, sendo cada turno supervisionado por um professor orientador diferente. Estes turnos podem ou não estar trabalhando no mesmo projeto, embora o mais comum é que trabalhem em projetos diferentes. As equipes contam com em média oito membros onde comumente 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 8 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 subdividi-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, simultaneamente com a atividade de Requisitos estão de Gerenciamento do Projeto e o Gerenciamento de Configuração.

¹ www.npi.quixada.ufc.br/processo/

Figura 8 – Processo do NPI.



Fonte: NPI (201__).

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.

- **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, notificadoros 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.

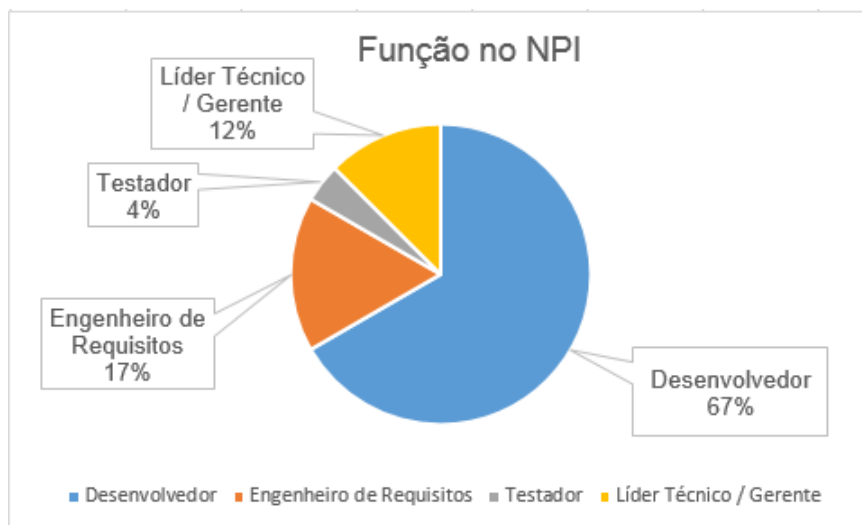
Tabela 1 – Conhecimento em Integração Contínua.

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
Você sabe o que é Gerência de Configuração?	Desconheço ou nunca utilizei nenhuma delas
	Sim
	Não

Fonte: Elaborado pelo autor.

Ao todo, vinte e três estagiários participaram da pesquisa, 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.

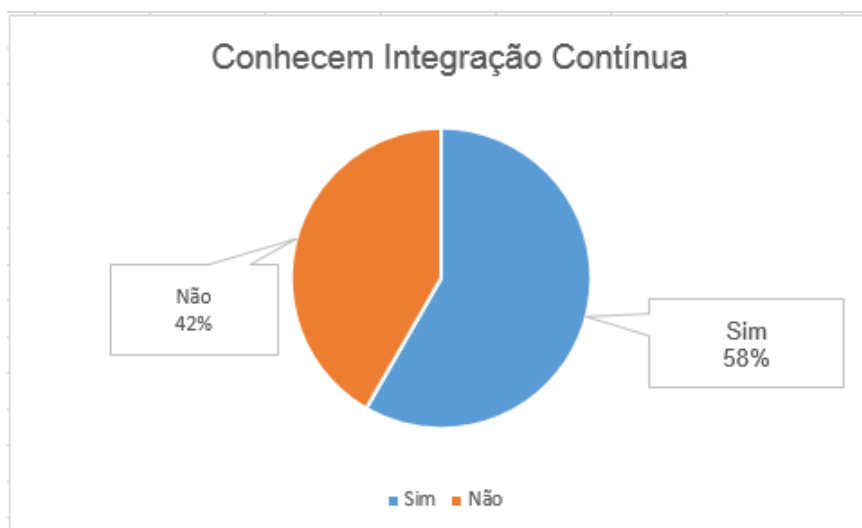
Figura 9 – Função no NPI.



Fonte: Elaborado pelo autor.

A Figura 9 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 10 – Conhecem Integração Contínua.

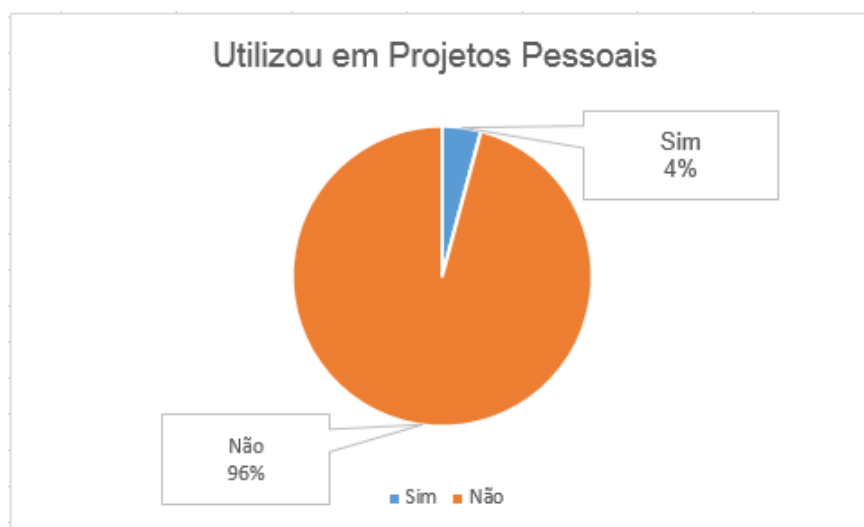


Fonte: Elaborado pelo autor.

Como descrito na Figura 10, 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.

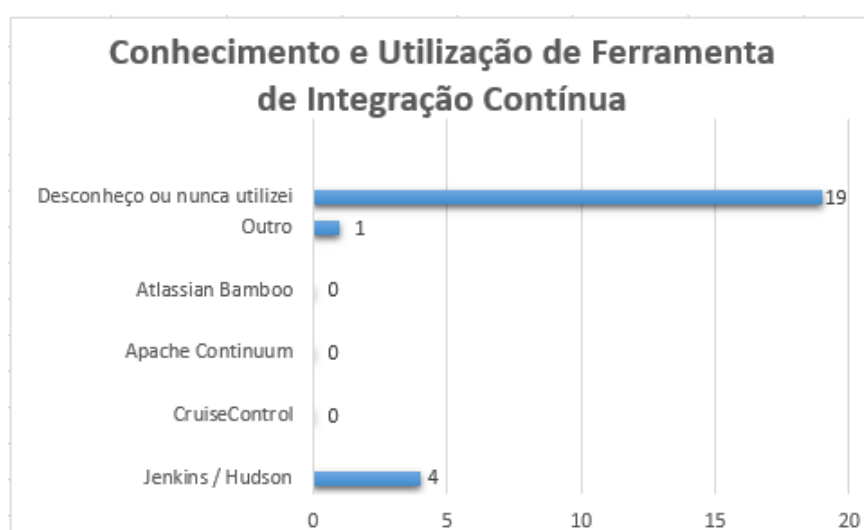
Figura 11 – Utilização em Projetos Pessoais.



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 12 – Conhecimento e Utilização de Ferramentas de Integração Contínua.

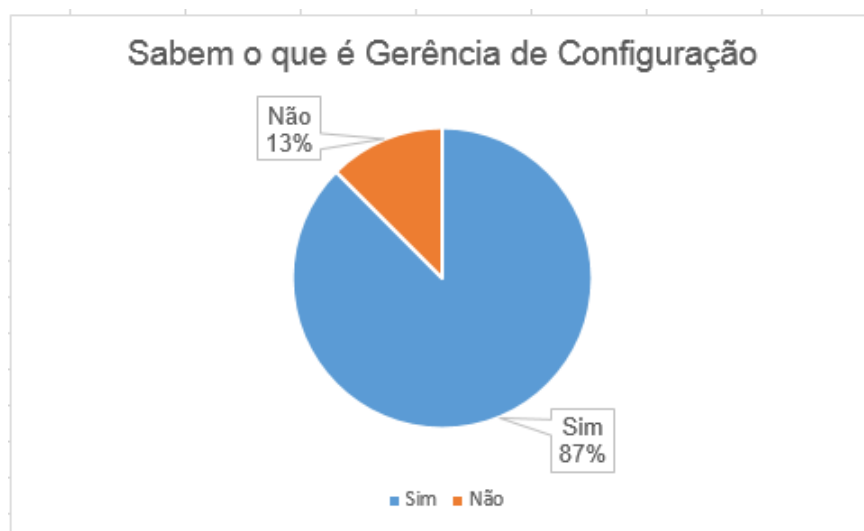


Fonte: Elaborado pelo autor.

O gráfico da Figura 12 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 13 – Conhecem Gerência de Configuração.



Fonte: Elaborado pelo autor.

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.

5.4.1 Gestão de Projetos Acadêmicos

O projeto GPA (Gestão de Projetos Acadêmico) 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 Implantação

5.4.3 Pontos Positivos

5.4.4 Pontos Negativos

6 Discussão

7 Considerações Finais

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.
- DUVAL, P. M. Continuous integration: Patterns and anti-patterns. *DZone Refcardz*, n. 1, p. 1–6, 20__.
- 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.
- EIS, D. Janeiro 2012. Disponível em: <<http://tableless.com.br/introducao-das-premissas-dos-controles-de-versao/>>. Acesso em: 27.3.2014.
- FOWLER, M. Maio 2000. Disponível em: <<http://martinfowler.com/articles/continuousIntegration.html>>. Acesso em: 27.3.2014.
- GIT. 20__. Disponível em: <<http://git-scm.com/book/pt-br/Primeiros-passos-Sobre-Controle-de-Vers~ao>>. Acesso em: 27.3.2014.
- 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.
- 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.
- MENEZES, G. G. L. de. *OURIÇO: UMA ABORDAGEM PARA MANUTENÇÃO DA CONSISTÊNCIA EM REPOSITÓRIOS DE GERÊNCIA DE CONFIGURAÇÃO*. Niterói: [s.n.], 2011.
- MILLER, A. A hundred days of continuous integration. *Agile Conference*, p. 289 – 293, 2008.
- 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.
- MRÁZ, M. *Use of continuous integration in web application development*. Dissertação (Mestrado) — Masarykova Univerzita Fakulta Informatiky, Brno, 2013.
- NPI: Npi. 201__. Disponível em: <<http://www.npi.quixada.ufc.br/processo/>>. Acesso em: 20.4.2014.
- 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.

- PADUELLI, M. M.; SANCHES, R. Problemas em manutenção de software caracterização e evolução. *III Workshop de manutenção moderna*, 2006.
- PEREIRA, J. A. D.; ARRUDA, R. F. V. de; GOMES, A. M. Lessons learned during the implementantion of a continuous integraton software in mps.br level f environment. *IX Workshop Anual do MPS*, 2013.
- PRESSMAN, R. S. *Engenharia de Software*. [S.l.]: Pearson Education, 2010. 1056 p.
- SEPALLA, A. *Improving software quality with Continuous Integration*. Dissertação (Mestrado) — Aalto University, Esbo, Maio 2010.
- SMART, J. F. *Jenkins The Definitive Guide*. [S.l.]: O'Reilly Media, 2011.
- 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.
- SOMMERVILLE, I. *Engenharia de Software*. 9. ed. [S.l.]: Pearson Education, 2011. 529 p.