FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JULIANA DA SILVA ZAFALÃO TAVARES

ELABORAÇÃO DE UM FRAMEWORK PARA INTEGRAÇÃO DE SOFTWARE

MARÍLIA
2012

JULIANA DA SILVA ZAFALÃO TAVARES

ELABORAÇÃO DE UM FRAMEWORK PARA INTEGRAÇÃO DE SOFTWARE

Trabalho de Curso apresentado ao Curso de Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito para obtenção do grau de Bacharel em Ciência da Computação.

Orientador
Profº. Ms. FÁBIO LÚCIO MEIRA

MARÍLIA
2012

Tavares, Juliana da Silva Zafalão

Elaboração de um Framework para Integração de Software / Juliana da Silva Zafalão Tavares; Orientador: Profº. Ms. Fábio Lúcio Meira, Marília, SP [s,n], 2012.
57f.

Trabalho de Curso (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário de Marília – UNIVEM, Marília, 2012.

1. Framework 2. Engenharia de Software 3. Integração Contínua

CDD: 005.1

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM SISTEMAS DE INFORMAÇÂO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Juliana da Silva Zafalão Tavares

ELABORAÇÃO DE UM FRAMEWORK PARA INTEGRAÇÃO DE SOFTWARE

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Sistemas de Informação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Sistemas de Informação.

Nota: 5.0 (CINCO)

Orientador: Fabio Lucio Meira

1°. Examinador: Elvis Fusco

2°. Examinador: Mauricio Duarte

Marília, 11 de dezembro de 2012.

Dedico este trabalho primeiramente a Deus por seu amor e cuidado comigo. A minha família, pelo amor e apoio em todos os momentos, em especial a duas importantes pessoas que não estão entre nós, minha mãe, Julia Sueli e minha avó Maria, que estarão eternamente em meus pensamentos.

AGRADECIMENTO

Novamente agradeço a Deus em primeiro lugar por jamais ter me deixado perder as forças e ânimo em continuar o curso apesar de todas as dificuldades.

Agradeço ao meu pai Paulo, minha tia Vera, meu tio Rubens, meus irmãos Luis Américo, José Victor, Julio César e Mariana, à esposa do meu Pai, Lucilene e a minha cunhada Naylla por sempre estarem ao meu lado nos momentos difíceis, e por terem me dado tanta força para que eu pudesse realizar este trabalho. Mesmo morando em São Paulo e eles em Promissão todos estão sempre no meu coração e no pensamento. A eles que devo toda força e coragem para terminar esse curso de TCC que por dois anos ficou parado e nesse último ano retomado e graças a Deus será concluído como sempre sonhei.

Agradeço a minha mãe por todo amor e carinho que teve quando estava ao meu lado, e que mesmo que hoje ela seja a minha estrela no céu, em nenhum momento deixou de estar no meu coração.

Agradeço a minha avó Maria, que tanto desejou que eu estivesse em uma universidade e que infelizmente não pode estar presente para prestigiar o meu sucesso.

Agradeço ao meu professor e orientador Fábio Lúcio Meira, que apesar de estar concluindo esse trabalho de TCC à distância, teve toda paciência e compreensão para que esse trabalho fosse realizado.

Agradeço à empresa INDRA, na qual hoje trabalho na cidade de São Paulo e que por muitas vezes pode estar me liberando em dias importantes de trabalho para que eu pudesse estar vindo até Marília e dar continuidade neste curso que foi tão sonhado.

Agradeço aos meus dois primeiros chefes Sérgio Fernandes Lima e Evandro Mauricio de Souza, que me deram oportunidade para mostrar o meu trabalho e seguir a carreira profissional que eu tanto almejei e ter muito sucesso como venho tendo.

Agradeço aos vários meus amigos de faculdade que sempre estiveram ao meu lado tanto nos momentos difíceis e nos momentos alegres passamos durante o período de faculdade.

Agradeço aos amigos conquistados em São Paulo, que desde quando cheguei sempre estiveram me apoiando para a realização deste trabalho.

Agradeço aos meus amigos de Promissão e Marília que sempre me apoiaram para a realização da conclusão do deste curso de Ciência da Computação.

Agradeço aos amores que passaram na minha vida ao longo desses sete anos de curso.

Obrigada!

TAVARES, Juliana da Silva Zafalão Tavares. **Elaboração de um Framework para Integração de Software,** 2012. 57f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2012.

RESUMO

Associado à arte de desenvolvimento de software e o constante monitoramento de seus processos estão as métricas de software, que auxiliam as equipes de desenvolvimento a trabalharem de forma organizada, coesa e buscar um único ideal, a melhoria de seus produtos de forma contínua. Com base nos processo de desenvolvimento de software esse trabalho atua em cima do objetivo de elaborar um *Framework* de desenvolvimento de software com base nos princípios da Integração Contínua, demonstrando o quão eficaz pode se tornar as aplicações e determinando as métricas desenvolvimento software, e principalmente seguindo as métricas de qualidade de código e os benefícios que elas podem proporcionar nesse processo.

Palavra Chave: Framework. Engenharia de Software. Integração Contínua.

TAVARES, Juliana da Silva Zafalão Tavares. **Elaboração de um Framework para Integração de Software,** 2012. 57f. Trabalho de Curso (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2012.

ABSTRACT

Associate art software development and constant monitoring of their processes are the software metrics that help development teams to work in an organized, cohesive and seek a single ideal, the improvement of its products continuously. Based on the process of software development work that acts upon the goal of developing a software development framework based on the principles of Continuous Integration, demonstrating how effective can become the applications and determining the metrics software development, and especially following metrics for code quality and the benefits they can provide in this process.

Keywords: Framework. Software Engineering. Integration Continuous.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

INTRODUÇÃO

Mesmo com a evolução dos computadores, das técnicas e ferramentas nos últimos anos, a produção de softwares confiáveis, com o mínimo de erros possíveis e entregues dentro do prazo, sem extrapolar o custo estipulado ainda é uma tarefa muito difícil. Com base nestas necessidades surgiram os métodos e práticas ágeis no desenvolvimento de software, visando melhorar o processo e simplificar o trabalho da equipe. Neste sentido, as equipes de desenvolvimento são formadas proporcionalmente ao tamanho do software, o que infere grandes quantidades de atribuições e atividades a serem executadas. Deste modo, gerenciar o desenvolvimento do software torna-se uma tarefa difícil e dispendiosa, o que requer processo contínuo de integração de atividades, visando obter melhor qualidade e resultados com o desenvolvimento do software.

Partindo desta premissa, de que a integração contínua é uma prática de desenvolvimento de software onde os membros de uma equipe integram seu trabalho frequentemente, geralmente cada membro integra pelo menos uma vez ao dia seus componentes desenvolvidos – podendo haver múltiplas integrações por dia. Cada integração é verificada por um *build* automatizado (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, além de permitir que um time desenvolva um software coeso mais rapidamente. (FOWLER 2006).

Este trabalho foi desenvolvido tendo como principal objetivo elaborar um Framework de desenvolvimento de software com embasamento nos estudos da Integração Contínua falando sobre seus princípios e características. Com base nisso, pretende-se obter o embasamento suficiente para adotar se não todas, mas as melhores práticas estudadas em um ambiente real de desenvolvimento.

A metodologia a ser utilizada é o estudo de conceitos de Integração de Software, definir quais são as principais atividades, produtos de trabalho e papéis envolvidos na Integração de Software, propor uma metodologia de customização da tarefa de Integração através da combinação dos elementos anteriores.

Este trabalho está dividido em quatro capítulos, organizados da seguinte forma: o Capítulo 1 apresenta o estudo detalhada da prática de integração contínua, o Capítulo 2 apresenta o papel do integrador em um processo de desenvolvimento de software, o Capítulo

3 o uso de framework em um processo de desenvolvimento de software, o Capítulo 4 apresenta o estudo de caso proposto neste trabalho e o Capítulo 5 apresenta a conclusão do trabalho.

CAPÍTULO 1 – INTEGRAÇÃO CONTÍNUA

Quando se trata de trabalho em equipe, normalmente subentende em divisão de tarefas entre pessoas. “Porém, trabalho em equipe não é um problema de divisão e conquista apenas. É um problema de divisão, conquista e integração” (ANDRES et. al., 2004).

Segundo Fowler (2006), Integração Contínua é uma prática de desenvolvimento de software em que os membros de uma equipe integram o seu trabalho frequentemente. Normalmente, cada pessoa integra pelo menos uma vez por dia, levando as múltiplas integrações em um único dia. Cada integração é verificada por um *build* automatizado (incluindo a execução dos testes). Dessa forma, os erros de integração podem ser detectados o mais rápido possível. Muitas equipes perceberam que essa abordagem leva a uma redução significativa nos problemas de integração. Além disso, permite que a equipe desenvolva sistemas coesos mais rapidamente.

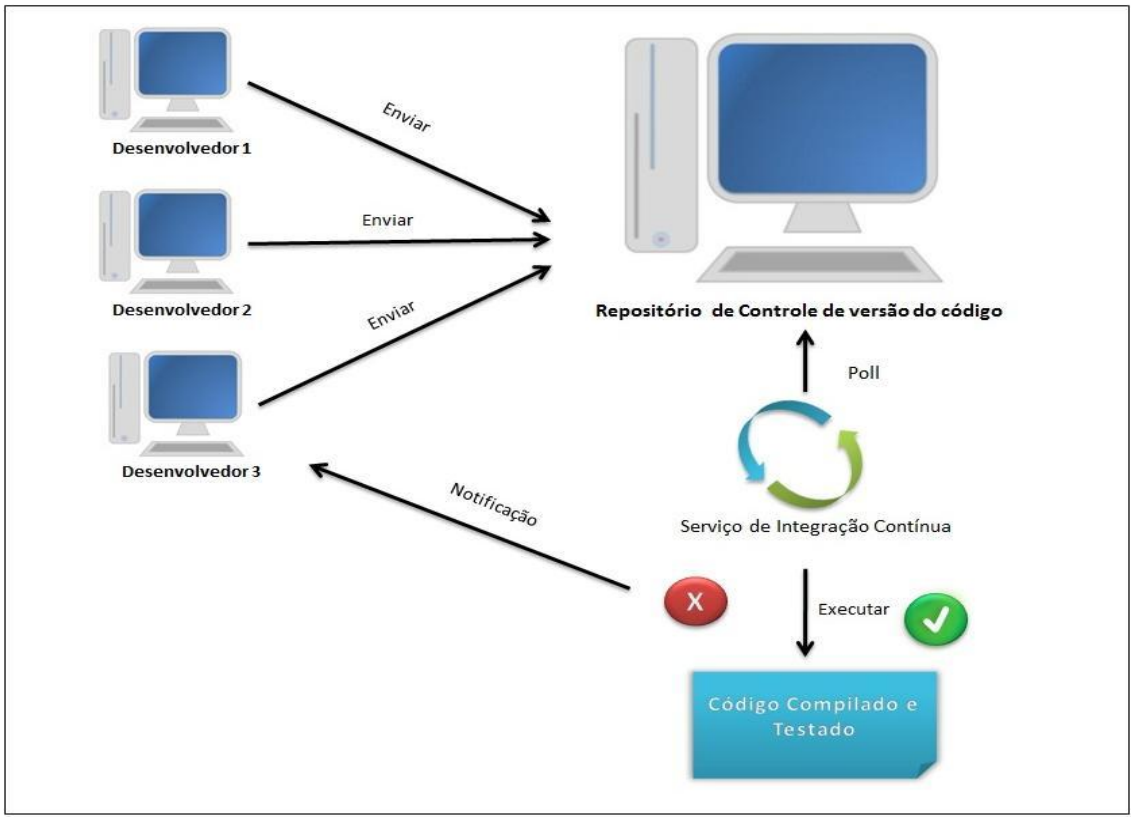
Segundo Andres e Beck (2004), na Integração Contínua é importante se conscientizar que quanto mais tempo a integração demora a ser feita, mais ela será complicada e menos previsível serão as dificuldades.

Uma das principais vantagens da Integração Contínua é a obtenção de informações sobre a condição do código produzido utilizando métricas e verificações oferecidas por algumas ferramentas de *build* automatizado. O *build* é o código que está em processo de desenvolvimento, compilado e transformado em executável. Em algumas boas ferramentas de controle de versão é possível desenvolver artefatos em paralelo e se necessário, o merge desse documento.

Atualmente no mercado existem diversas ferramentas cuja função é fazer o controle de versão de código como: CVS, Microsoft Visual SourceSafe, Subversion, Rational ClearCase, entre outros. A função principal dessas ferramentas é centralizar em um só local (um servidor) o armazenamento dos arquivos do projeto, e também controlar as versões desses arquivos para que o trabalho em equipe não seja prejudicado.

A ilustração da arquitetura de Integração Contínua é apresentada na Figura 1. Essa arquitetura será detalhada nas continuidades do trabalho.

Figura 1 - Ilustração da Arquitetura de Integração Contínua



Fonte: Autoria Própria baseada no Livro *Continuous Integration*

No exemplo, é considerada que uma ferramenta própria para integração contínua instalada em outra estação é utilizada. A partir dela, pode se definir uma hora do dia para executar o *build* da última versão do projeto em desenvolvimento. Além disso, dependendo da ferramenta utilizada, pode se definir que uma notificação por e-mail deve ser enviada para toda a equipe de desenvolvimento informando se a execução foi realizada com sucesso ou se falhou.

1.1 Estilo de Integração Contínua

1.1.1 Síncrono

Modo Síncrono de realização da Integração Contínua é feita apenas por um par que integra seu trabalho de cada vez e outros pares só são liberados para integrar ao serem informados do término de integração corrente. Lembrando que não são todos os projetos que podem usar esse modelo de integração, já que ele exige que os desenvolvedores trabalhem

juntos, normalmente em uma mesma sala. Sendo assim só pode ser garantido que apenas um par integre de cada vez, utilizando computador dedicado à integração, por exemplo.

1.1.2 Assíncrono

No modo Assíncrono, o desenvolvedor integra seu código, executado um subconjunto de passos tais como: assegurar que um projeto compila e todos os testes automatizados executam com sucesso, criar *backup* do projeto na estação de trabalho, fazer *update* do projeto, assegurar que o software continua compilando e os testes executam com sucesso e fazer o *commit* do projeto.

As ferramentas de apoio à integração são executadas após a sequência de passos, para a monitoração do repositório permanentemente. Sempre que detecta mudanças, a ferramenta faz o *checkout* de todo o projeto automaticamente, podendo usar ferramenta como *Maven* para gerenciar o build do projeto e executar todos os testes.

1.2 Vantagens x Desvantagens

Ao utilizar Integração Contínua pode se ter diversos benefícios. Um bom exemplo a ser citado é a utilização de testes unitários, onde é possível detectar erros, e assim tendo tempo de ser corrigido imediatamente. Um desenvolvedor pode experimentar muitos benefícios quando se pratica o teste de unidade em seus espaços reservados que podem ser definidos como espaços privados: *feedback* rápido, localização de erros, a confiança para fazer a mudança, e muito mais. Quando os testes unitários são exercidos na prática da Integração Contínua o seu significado é sutilmente alterado.

Para Fowler (2006), a principal vantagem é a redução de riscos obtida através da detecção rápida de erros, assim a equipe pode então corrigir o problema o mais rápido possível, o que é fundamental para não introduzir erros ao criar novas funcionalidades, refatorar, etc. Integração contínua é mais uma forma de trazer segurança em relação a mudanças: você pode fazer modificações sem medo, pois será avisado caso algo saia esperado.

Quando a Integração Contínua é utilizada, é possível asegurar o não aparecimento de *bug*, porém a rápida detecção de erros facilita a remoção desses. Se introduzir um bug e detectar rapidamente, é muito mais fácil se livrar, lembrando que *bugs* são coisas

desagradáveis, e que bugs encontrados no sistema após a implantação não seria bem visto pelo usuário.

Segundo Soares (2006), a utilização da prática de Integração Contínua facilita o processo de desenvolvimento das equipes, já que nesse processo é utilizado apenas uma máquina reservada para integração onde todos os membro da equipe deve ter livre acesso.

1.2.1 Precauções

Para Dias (2008), para que uma equipe de desenvolvimento tenha resultados satisfatórios em seu produto final é necessário que exista integração de seus componentes, por isso, deve-se tomar cuidado com algumas situações peculiares da integração. Muitas vezes, o trabalho que é feito individualmente funciona perfeitamente.

Muitas vezes o trabalho individual funciona corretamente, os builds de todo o sistema é feito corretamente, os testes são executados com sucesso e os devidos artefatos são produzidos. Porém, assim que as ferramentas de Integração Contínua obtêm as alterações e realiza o *build*, é possível se deparar com alguns erros. Isto pode ocorrer por causa de um *commit* incompleto, por uma falha de configuração no servidor de integração ou por uma característica do ambiente de desenvolvimento que é diferente no servidor e na estação de trabalho. Esse tipo de problema vai acontecer frequentemente e, para torná-lo menos incômodo, é conveniente possuir uma única pessoa responsável por verificar os erros de integração. O papel dessa pessoa é verificar qual tipo de erro está acontecendo: se for um erro de configuração do servidor, ela deve descobrir como corrigi-lo; se for um erro de programação, ela deve repassar a responsabilidade da correção para o programador que fez a alteração (DIAS, 2008).

Segundo Andres e Beck (2004) o conceito de Integração Contínua pode ser quase completamente automatizado, uma vez que a principal parte deve ser feita manualmente por cada desenvolvedor – o *commit*. O envio de alterações para repositório de controle de versões é fundamental para que a integração seja feita. Se os desenvolvedores só enviarem suas atualizações uma vez por semana, por exemplo, pouco importa se a ferramenta de integração está configurada para efetuar *build* a cada hora ou uma vez por dia.

O outro extremo, por outro lado, também pode ser problemático. Levar o conceito de continuidade ao pé da letra pode ser nocivo para um projeto. Configurar o servidor de integração para integrar a cada hora pode gerar erros constantemente. Isso atrapalha o

processo de desenvolvimento de software, pois interfere no trabalho do desenvolvedor que fez as alterações e também dificulta o trabalho dos outros, que podem começar a ter problemas por causa das alterações alheias realizadas de maneira incorreta e com muita frequência. Configurar a integração para que seja feita automaticamente uma ou duas vezes por dia, dependendo da velocidade de produção dos programadores, é mais do que suficiente para a maioria dos projetos. Caso mais de duas integrações sejam necessárias em determinado momento, estas devem ser iniciadas manualmente.

1.3 Build

Para RUP (2007) o *build* é uma versão operacional de um sistema ou parte de um software que demonstra um subconjunto dos recursos a serem fornecidos no produto final. Ele é constituído de um ou mais componentes (geralmente executáveis), que são construídos a partir de outros componentes, normalmente por um processo de compilação e vinculação do código-fonte.

1.4 Commit

Segundo Fowler (2006), *commit* é a operação que permite a validação das atualizações do código-fonte no diretório local de trabalho da máquina do desenvolvedor, através da ferramenta de gerenciamento de configuração, é possível ter o controle de versão dos códigos. O *commit* é feito do diretório local de trabalho para o repositório da ferramenta de gerenciamento de configuração. Quanto mais frequência um desenvolvedor realiza um *commit*, menos lugares o desenvolvedor terá que se preocupar por erros de conflito, e mais rapidamente ele os consertará.

1.5 Ambientes de trabalho

As práticas de código compartilhado, repositório único de código e Integração Contínua exigem uma organização do projeto de forma a facilitar ao máximo a reprodução do ambiente de desenvolvimento e de produção. Desta forma, o repositório único de código deve conter todos os arquivos necessários para construir e rodar os testes automatizados do sistema. Sendo assim, tratam-se vários pontos apontados pelo Objetivo de desenvolvimento já que

basta saber qual o repositório único de código para conseguir montar um ambiente de contribuição ao projeto.

A prática de Código Compartilhado exige uma padronização no estilo de escrita de código e de teste que deve ser compartilhado por todos. Sendo assim, os arquivos descrevendo essas padronizações devem estar sob o controle de versão no repositório único de código de forma a serem obtidos por qualquer membro da equipe.

Por fim, a prática de Integração Contínua exige um sistema automático para obtenção do código, instalação do mesmo num ambiente limpo e execução dos testes automatizados do projeto. Porém, para que seja possível montar esse tipo de ambiente automaticamente, é necessário que a descrição e configuração do ambiente de produção estejam inclusas nessa ferramenta de Integração Contínua servindo tanto de exemplo quanto de documentação.

1.6 Controles de Código

Para Andres e Beck (2004), o meio para facilitar a integração entre os desenvolvedores é que todo código produzido seja compartilhado por toda equipe. Assim todos são responsáveis pelo código e qualquer alteração em alguma parte do sistema. Qualquer membro da equipe pode melhorar uma parte do sistema sempre que precisar.

Para que essa prática seja realizada, é necessária a criação de um servidor para que qualquer usuário conectado à rede desse servidor possa obter qualquer versão. Com isso é necessário a utilização de ferramentas que operam sobre uma rede e que oferecem recursos para que os usuários facilmente obtenham o código fonte. Além do mais, incluem um gerenciamento que possibilita recuperar versões prévias dos arquivos mantidos pelo sistema. O custo de ferramentas deste tipo não deveria ser uma questão relevante, pois existe o *Concurrent Versions System (CVS), Microsoft Visual Source Safe, Subversion* e outros (COLLABNET, 2002).

Assim, para viabilizar a tarefa de integração, todo o código fonte deveria ser mantido por uma ferramenta de controle de versão. Isto inclui manter também todos os *scripts*, arquivos de propriedades, de configuração, esquemas de banco de dados, *scripts* de integração e qualquer outro arquivo que seja necessário para integrar o sistema em desenvolvimento. Outro ponto importante é o de tentar manter todo o código fonte sob uma estrutura única do sistema de controle de versão. Algumas vezes, são utilizadas estruturas diferentes para componentes diferentes. A construção de vários componentes deveria ser tratada pelos scripts de integração, e não pela estrutura do sistema utilizado para armazená-los.

1.7 Ferramentas no Mercado

Muitas das tarefas de integração podem ser automatizadas por meio de ferramentas. Algumas dessas ferramentas existentes no mercado são: Cruise Control, Hudson e Continuum. Porém, ferramentas mais simples, tais como Maven[1], Rake[2], Ant[3] e Buildount[4] podem ser usadas pra criar builds de forma simples. A diferença das últimas ferramentas em comparação com as primeiras é que essas são mais completas, podendo até mesmo cuidar de envio de e-mails, agendar *builds* e até comandar diferentes máquinas para executarem *builds*.

1.7.1 CruiseControl

Para garantir que a integração seja feita com a frequência adequada, deve-se empregar uma ferramenta para integração contínua. O CruiseControl é uma ferramenta para *build* automatizado que, valendo-se dos *buildfiles* do *Apache Ant* e do sistema de controle de versão, garante que os projetos estão sendo continuamente integrados. O Cruise Control é escrito em Java e possui os mesmos benefícios de independência de plataforma oferecidos por outras ferramentas como Ant e o JUnit (HIGHTOWER et al., 2004).

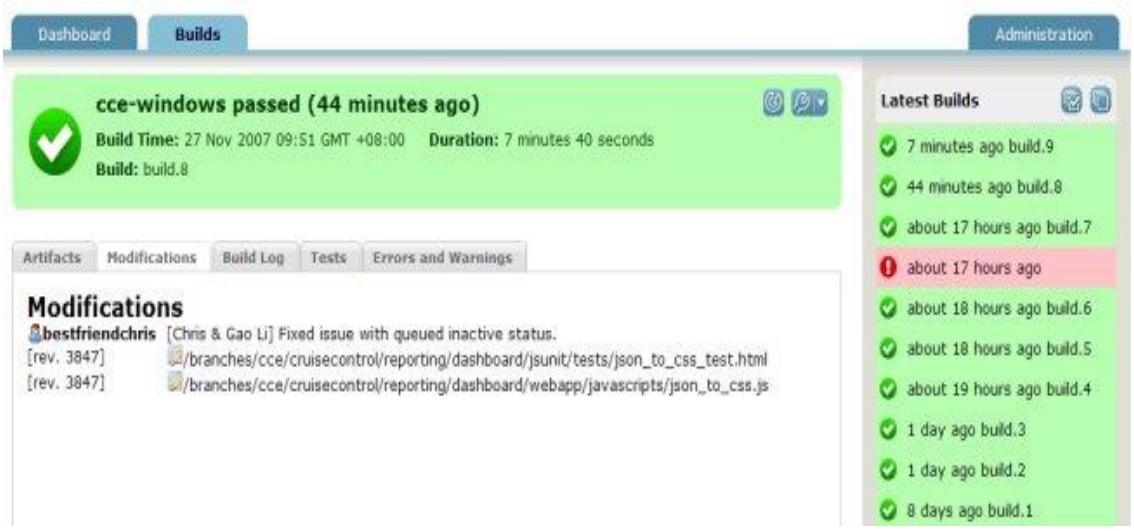
CruiseControl é baseado em um conceito simples. Uma instância desta aplicação é configurada para observar um repositório de controle de versões e detectar mudanças nos arquivos que lá estão. Quando alguma alteração é percebida, a instância atualiza uma cópia local do projeto com as modificações e executa o roteiro de *build* para o projeto. Depois que o *build* é feito (com sucesso ou não), o Cruise Control publica vários artefatos especificados pelo usuário (incluindo um *log* do *build* que foi feito) e informa os membros do projeto sobre sucesso ou a falha na construção.

Na figura 2 apresenta a tela da ferramenta CruiseControl.

---

[1] *Marven* é uma ferramenta para automatizar o processo de *build* de aplicações Java. Fonte: http://maven.apache.org/
[2] *Rake* é uma ferramenta de construção escrita em Ruby, semelhante a fazer, Ant e Phing. Fonte: http://rake.rubyforge.org/
[3] *Apache Ant* é uma biblioteca Java e de linha de comando ferramenta que tem como missão conduzir processos descritos em construir arquivos como alvos e pontos de extensão dependentes uns dos outros. Fonte: http://ant.apache.org/
[4] *Buildout* é baseado em Python para a criação de sistema de construção, montagem e implantação de aplicativos de múltiplas partes, alguns dos quais podem não ser baseado em Python. Fonte: http://www.buildout.org/

Figura 2 - Ferramenta de Integração Contínua CruiseControl



Fonte: Autoria Própria

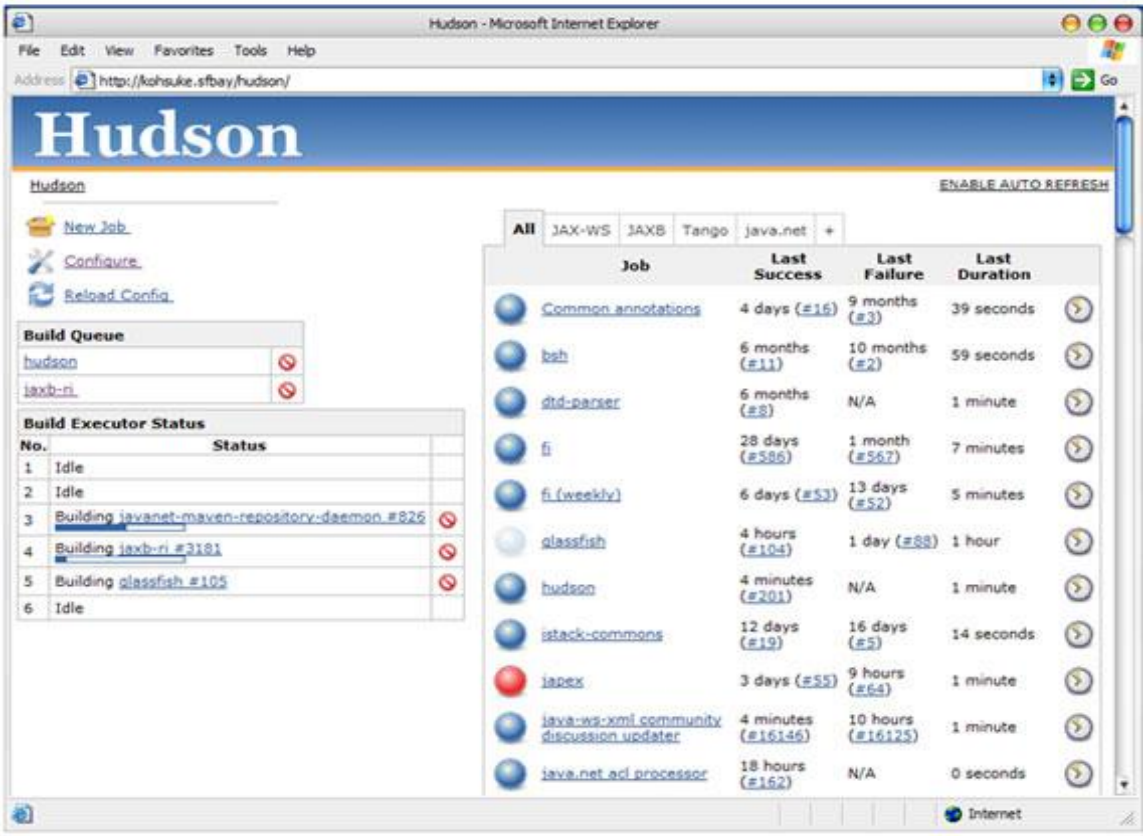
1.7.2 Hudson

O Hudson é uma ferramenta *Open Source* (Licença MIT) para a Automação e Gerenciamento de *build*. Permite tanto fazer construções de projetos quanto monitorar *jobs* executados externamente. Entre as principais características do *Hudson*, destaca se a facilidade de instalação e a possibilidade de distribuir os *build* para múltiplos computadores diferentes, além de integração com diversas ferramentas como Maven, notificação por e-mail, personalização do *job* com *batch scripts* ou *shell* scripts, ANT ou Maven para construção, publicação de resultados, de testes e acompanhamento do *log*, notificação por email.

A função da Integração Contínua é fornecer automatização da construção de aplicações e testes integrados, gerando relatórios que facilitam a homologação da aplicação e para o resgate de uma construção anterior.

Utilizando o Hudson é possível integrar com repositórios como CVS, possibilitando fazer *checkout* do projeto e configurar um *listener* para uma nova construção da aplicação toda vez que existir alteração da versão de qualquer arquivo do projeto. Existe ainda integração com ferramentas para construção de aplicações como ANT, que é onde deve ser configurado para serem executados os testes. Na figura 3 apresenta a tela da ferramenta Hudson listando os builds existentes em uma integração.

Figura 3 - Ferramenta Hudson listando os Builds existentes em uma Integração



Fonte: Autoria Própria

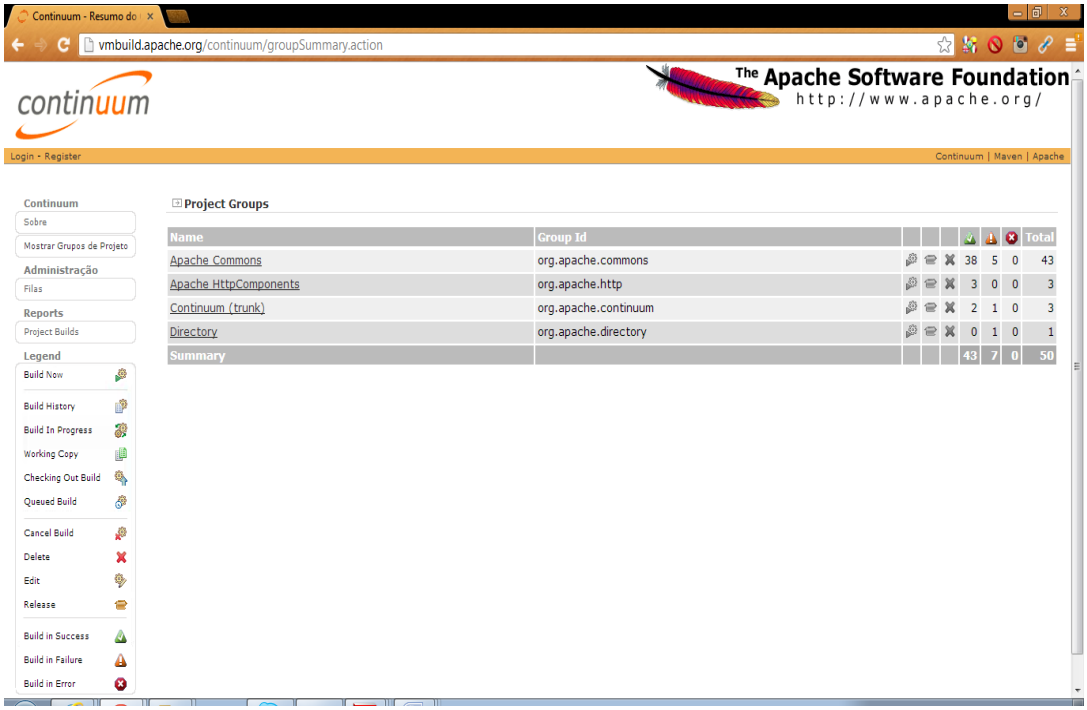
1.7.3 Continuum

A ferramenta Continuum que é executada em um *Servlet Container* qualquer onde pode se baixar o código alterado do controle de versão, executar o build, distribuir no servidor de aplicação e executar os testes unitários. Se o build quebrar, manda e-mail pra quem quebrou o build e pra quem mais estiver interessado no processo. A interface dele não é das melhores, mas a configuração inicial é simples: basta apenas apontar seu “pom.xml” e a aplicação executa o resto do processo.

Segundo MASSOL et al.(2006), o Continuum é um servidor *build* e integração contínua para o Maven. Este funciona da mesma forma que o Cruise Control, mas se adequa trivialmente a projetos que são gerenciados pelo Maven.

Na figura 4 apresenta a tela da ferramenta Continuum.

Figura 4 - Interface Gráfica da Ferramenta Continuum



Fonte: Autoria Própria

1.8 Considerações Finais

A Integração Contínua é considerada um dos princípios ágeis, sendo muito difundida nas equipes de desenvolvimento de software. Esta prática possibilita que inúmeras integrações sejam realizadas diariamente por diversas pessoas e tendo a garantia de que ao gerar um *build* final, este não acusará erros clássicos de integração de código fonte, caso contrário no momento em que for gerada alguma inconsistência o time poderá ser avisado imediatamente através de ferramentas de apoio que realizam procedimentos de auditoria e inspeção do código submetido para geração de uma nova versão (DUVALL et. al., 2007).

Desta forma evita-se o retrabalho garantindo a consistência de integração de todo um sistema a cada *commit* realizado por membros do time. Indicadores que garantem a qualidade deste tipo de prática podem ser expressos no número de *build* gerados com sucesso ou então problemas na geração dos *builds* por: Dia, Semana, *Sprint*/Iteração, entre outras métricas conforme necessidade (FOWLER, 2006).

Muitas ferramentas possuem características comuns à integração contínua, em sua grande maioria ferramentas open source, apesar de as grandes ferramentas dotadas de inúmeras funcionalidades e altamente difundidas serem pagas. Ferramentas com este intuito

podem ser configuradas para realizarem determinadas verificações a cada *commit* realizado, consistindo a integridade do software em execução frequentemente ao longo do dia, por exemplo, podendo ser configurado de tal forma a validar uma suíte de testes com cobertura para toda a aplicação, sempre que um novo *commit* for realizado assegurando o bom funcionamento do produto desenvolvido.

CAPÍTULO 2 – O PAPEL DO INTEGRADOR NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Para Silva (2006), o papel de Integrador não se limita apenas em integrar processos, mas sim podendo participar de todo o processo de desenvolvimento de software. O integrador na verdade também pode ser um desenvolvedor, em projetos pequenos o indivíduo de uma equipe pode vir a exercer vários papéis. Para que se tenha o melhor processamento de informações é necessário que os desenvolvedores realizem estudos de processos de desenvolvimento de um sistema, passando entre hardware, software e o usuário final. As soluções que os desenvolvedores determinam serão padronizadas e transcritas de forma que o computador possa executar.

Em última instância, é possível perceber que a qualidade é um conceito com múltiplas facetas (perspectivas de usuário, desenvolvedor e cliente) e que envolve diferentes características (por exemplo: usabilidade, confiabilidade, eficiência, manutenção, portabilidade, segurança, produtividade) que devem ser alcançadas em níveis diferentes, dependendo do propósito do software.

Para Pressman (2006), a qualidade de software é definida como "conformidade com requisitos funcionais explicitamente documentadas e características implícitas, que são esperadas em todo software desenvolvido profissionalmente".

Com base nessa afirmação Pressman (2006), é possível relacionar qualidade com conformidade em relação aos requisitos funcionais e não funcionais presentes nos documentos de especificação do sistema. Porém, ele não menciona a possibilidade de as especificações estarem incorretas – o que não atenderia ao cliente. Portanto, para um software ser considerado de qualidade satisfatória, é importante que ele atenda a requisitos funcionais e não funcionais, além de possuir uma análise de requisitos coerentes com o desejo do cliente.

O que há de comum nas várias perspectivas discutidas acima é que todas elas estão focadas no produto de software. Ou seja, estamos falando de qualidade do produto.

Para garantir que o produto final de um software apresente características citadas anteriormente é necessário avaliar esse produto e se ele apresentar uma abordagem indesejável para a equipe de desenvolvimento, tendo em vista que a constatação a posterior de que o software não apresenta a qualidade desejada pode implicar na necessidade de refazer grande parte do trabalho. Sendo assim é necessário que a qualidade seja incorporada ao produto ao longo de seu processo de desenvolvimento. De fato, a qualidade dos produtos de

software depende fortemente da qualidade dos processos usados para desenvolvê-los e mantê-los. (ESMIRELLO, 2005)

Seguindo uma tendência de outros setores, a qualidade do processo de software tem sido apontada como fundamental para a obtenção da qualidade do produto. Abordagens de qualidade de processo, tal como a série de padrões ISO 9000, sugerem que melhorando a qualidade do processo de software, é possível melhorar a qualidade dos produtos resultantes. (ISSO 9000,2000)

Para Esmirello (2005), seguir a premissa por detrás dessa afirmativa é a de que processos bem estabelecidos, que incorporam mecanismos sistemáticos para acompanhar o desenvolvimento e avaliar a qualidade, no geral, conduzem a produtos de qualidade. Por exemplo, quando se diz que um fabricante de eletrodomésticos é uma empresa certificada ISO 9001 (uma das normas da série ISO 9000), não se está garantindo que todos os eletrodomésticos por ele produzidos são produtos de qualidade, mas sim que ele tem um bom processo produtivo, o que deve levar a produtos de qualidade.

2.1 Desenvolvimento de Sistemas Integrados

No desenvolvimento de sistemas integrados é importante a agregação de artefatos para que o sistema seja capaz de integrar todas as funcionalidades. O papel do integrador é liberar os componentes já testados e um ambiente de integração, e assim combinarem todos os artefatos e criar um *build*.

O que ocorre nos subsistemas e no sistema é responsabilidade também do integrador que planeja toda a integração, sendo que cada um tem um ambiente de trabalho. Todos os componentes testados são liberados no ambiente de trabalho de desenvolvimento particular de um desenvolvedor para o ambiente de trabalho de integração do subsistema, enquanto os subsistemas de implementação integrados são liberados no ambiente de trabalho de integração do subsistema para no ambiente de trabalho de integração do sistema. (RUP, 2007).

2.1.1 Teste de Integração

Segundo Lima (2004), a atividade de teste de software é dividida em três fases: teste de unidade, teste de integração e teste de sistema.

Para Barbosa (2004), no paradigma de Programação Orientação a Objeto, um componente pode ser entendido como uma unidade de teste básica, podendo corresponder a uma classe num sistema ou um método específico de uma classe. Embora não exista uma padronização para a definição de um componente, é necessário que um componente pode ser entendido como um sistema de componentes.

É necessário definir os testes de unidade como os testes de componentes individuais, os testes do sistema resultam dos testes da união dos componentes e os testes de integração são os testes das interações entre componentes, ou seja, é o caminho no qual o teste é conduzido para integrar componentes no sistema. (BARBOSA, 2004).

Teste de integração é o processo de verificar se os componentes do sistema, juntos, trabalham conforme descrito nas especificações do sistema e do projeto do programa (LIMA, 2004).

Uma estratégia de integração deve responder a três questões: quais componentes é foco dos testes de integração, em que sequência as interfaces de componentes deverão ser exercitadas e qual a técnica de teste devemos empregada para exercitar a interface. Ou seja, teste de integração é uma busca por defeitos que causam as falhas entre componentes (BRINDER, 2000).

2.1.2 Gerenciamento de Integração

Segundo RUP (2007), os componentes são desenvolvidos de acordo com o caso de desenvolvimento definido para cada projeto, e passam esses componentes dos espaços de trabalho de desenvolvimento para o espaço de trabalho de integração do projeto.

Os *builds* são produzidos conforme os integradores combinam os componentes liberados no espaço de trabalho integrado. Quando é criada uma nova *baseline*, o integrador precisa bloquear o espaço de trabalho de integração para garantir que haja um conjunto estático de arquivos e que novos arquivos não sejam liberados pelos desenvolvedores engajados no projeto (RUP, 2007).

Existem dois tipos de visão que podem ser associados ao espaço de trabalho de integração, sendo eles:

- Visão dinâmica: Modo que oferece acesso imediato e transparente aos arquivos e diretórios contidos no repositório do projeto.

- Visão estática: As visões de imagem copiam os arquivos e diretórios do repositório do projeto para o computador do desenvolvedor

De acordo com RUP (2007), a visão de integração deve ser uma visão dinâmica, pois ela garante que o integrador tenha a última versão dos arquivos e diretórios liberados pelo desenvolvedor.

O papel do integrador é realizar as atividade de integração durante o processo de desenvolvimento de um projeto e seguir o plano de build de integração, adicionado os subsistemas de implementação, liberandos ao espaço de trabalho de integração do sistema e criando os *builds*. A integração de cada build é testada por um testador e após o último incremento, o build deve ser gerado e testado por um testador (RUP, 2007).

2.2 Papéis no Processo de Desenvolvimento

Segundo RUP (2007), um papel (perfil) define o comportamento e as responsabilidades de um indivíduo ou um grupo deles numa ação em equipes. O processo de desenvolvimento inclui conceitos de gerência de orçamento, equipes, conhecimento e outros. Um indivíduo pode ter vários papéis já que os papéis não são indivíduos e nem cargos ou função.

2.2.1 Analista

Quem assume o papel de Analista tem a função de coordenar o levantamento dos requisitos e a modelagem dos casos de uso, identificando funções do sistema e estabelecendo o escopo do sistema.

Para RUP (2007), o indivíduo que assume o papel de Analista de Sistema tem que habilidades de comunicação acima da média, pois seu papel na comunicação com os clientes (e outros interessados) abrange o levantamento de requisitos e outras atividades relacionadas. É fundamental que os profissionais que desempenham este papel tenham conhecimento dos domínios do negócio e da tecnologia.

2.2.2 Desenvolvedor

Com o surgimento da prática de Integração Contínua se tornou comum a sua utilização em projetos de desenvolvimento de software. Nessa prática é comum que uma equipe de desenvolvimento seja composta por um grupo de pessoas onde elas, por sua vez, ficam responsáveis por funções direcionadas a cada membro de uma equipe que compõem o projeto. No dia a dia de um projeto de desenvolvimento de software, existem momentos em que, por alguma necessidade, é necessário que se altere parte do código que foi escrito anteriormente para que o sistema passe a funcionar de uma nova maneira, e assim surge a questão de como alterar esse projeto sem que se altere sua integridade.

2.2.3 Testador

Segundo RUP (2006), o testador é o indivíduo responsável pelas atividades centrais do esforço de teste e que possa definir a abordagem de teste que assegurar sua correta implementação. Ele identifica as técnicas, ferramentas e diretrizes apropriadas para implementar os testes necessários e dar orientação sobre os correspondentes requisitos de recursos para o esforço de teste e incluindo:

- Identificar a abordagem de implementação mais apropriada para um dado teste.
- Gerar testes individuais.
- Configurar e executar os testes.
- Registrar os resultados e verificar a execução dos testes.
- Analisar erros de execução e recuperar-se deles.

Os papéis organizam a responsabilidade de executar atividades e desenvolver artefatos em grupos lógicos. Cada papel pode ser designado a uma ou mais pessoas, e cada pessoa pode desempenhar um ou mais papéis. Ao definir o perfil do papel Testador, as suas habilidades devem ser levadas em consideração, já que são fundamentais para a realização de suas funções no decorrer do processo.

O conhecimento e as habilidades podem variar de acordo com os tipos de testes a serem executados e as fases do ciclo de vida do projeto; no entanto, geralmente o profissional a desempenhar o papel Testador deve ter as seguintes habilidades: capacidade das abordagens e técnicas de teste; capacidade de diagnosticar e resolver problemas; conhecimento do sistema

ou do aplicativo em teste se desejável e conhecimento da arquitetura de rede e do sistema se desejável.

Em alguns projetos existe a necessidade de realizar testes automatizados, com isso o testador deve possuir habilidades a mais do que as já citadas anteriormente, tais como: experiência no uso de ferramentas de automatização de testes; habilidades de programação; habilidades de depuração e diagnóstico. (RUP, 2007)

2.2.4 Gerente

O Gerente de Projeto aloca recursos, define prioridades, coordena interações com clientes e usuários, e geralmente mantém a equipe de projeto focada no objetivo correto. O gerente de projeto também estabelece um conjunto de práticas que garantem a integridade e a qualidade dos artefatos do projeto (RUP, 2007).

As habilidades necessárias para cumprir o papel de gerente de projeto irão depender do tamanho e da complexidade técnica e gerencial do projeto, mas o gerente de projeto, como define o RUP, precisa:

- Possuir experiência e domínio da aplicação e em desenvolvimento de software.
- Possuir habilidades de análise e gerência de riscos, estimativa, planejamento e decisão.
- Possuir apresentação e habilidades de comunicação e negociação
- Mostrar capacidades de liderança e construção de equipes;
- Possuir boa experiência com gerência e um histórico de tomar decisões rapidamente momentos de grande tensão.
- Possuir boas habilidades interpessoais e mostrar julgamento na seleção de pessoal.
- Possuir foco na entrega de um produto de software que satisfaça as necessidades dos clientes.

Com todos os requisitos citados anteriormente o papel de Gerente tem se tornando necessário nos projetos de desenvolvimento para garantir a integridade e a qualidade de seus produtos. As principais atividades de um gerente são de ajustar as prioridades, alocar recursos, coordenar interações com clientes e usuários e geralmente manter o foco da equipe em suas metas nos períodos determinado. (RUP, 2007)

Conforme a complexidade do projeto o gerente pode assumir outras responsabilidades além do gerenciamento. Por exemplo, o gerente do projeto poderá auxiliar com a coleta das necessidades do negócio, ou poderá auxiliar no projeto de um sistema de gerenciamento de banco de dados, ou redigir a documentação do projeto. O gerenciamento do projeto é uma função especial que uma pessoa deve realizar, mesmo se a pessoa estiver trabalhando também em outras funções.

2.2.5 Desenvolvedor

Os desenvolvedores são responsáveis pelo desenvolvimento de parte do sistema e testes unitários. Os papéis do Desenvolvedor organizam os papéis envolvidos principalmente no designer e desenvolvimento de software.

Segundo RUP (2007) o desenvolvedor pode ter os seguintes papeis:

- **Implementador** - A responsabilidade do implementador é por resolver e testar componentes de acordo com os padrões adotados para o projeto, para fins de integração com subsistemas maiores. O implementador precisa ter habilidades e conhecimentos apropriados que são: conhecimento do sistema ou aplicativo que está em teste; familiaridade com ferramentas usadas para testes e automatização de testes; habilidades de programação.
- **Revisor de Código** - O revisor de código garante a qualidade do código fonte, além de planejar e conduzir revisões do código-fonte. As principais habilidades de um revisor são parecidas com as do implementador e o indivíduo que desempenha esse papel geralmente é considerado especialista na linguagem de programação usada para o código que esta sendo revisado.
- **Designer de Banco de Dados** - O designer de banco de dados define os componentes que fazem parte do banco de dados do software a ser criado tais como: tabelas, índices, visões, restrições, *triggers*, procedimentos e funções. Para exercer esse papel o indivíduo deve ter sólidos conhecimentos práticos de: administração de banco de dados, conhecimento do ambiente de linguagem de implementação e técnica de análise.
- **Integrador** - Após os implementadores liberarem os componentes testados em um espaço de trabalho de integração os integradores combinam esses componentes para criar um *build*. A responsabilidade de planejar a integração

dos subsistemas e no sistema fica com integrador, sendo que cada um tem um espaço de trabalho de integração. Muitas vezes, o papel de integrador pode ser direcionado a uma pessoa na qual realizará também o papel de testador.

- **Arquiteto de Software** - O arquiteto de software atua como liderança e coordenação das atividades e os artefatos técnicos no decorrer do projeto. Eles estabelecem a estruturação geral de cada visão de arquitetura: a decomposição da visão, o agrupamento dos elementos e as interfaces entre esses principais agrupamentos. Em resumo, o arquiteto de software deve ter grande conhecimento geral, possuir maturidade, visão e profunda experiência que permita identificar problemas rapidamente e dar opiniões sensatas e criteriosas na falta de informações completas. Mais especificamente, o arquiteto de software ou os membros da equipe de arquitetura devem combinar as seguintes habilidades: experiências, liderança, comunicação e Orientação por metas e por atividade.

2.3 Considerações Finais

A definição de papéis visa simplificar a distribuição de tarefas durante a execução do processo, de maneira que várias funções pudessem ser acumuladas por um mesmo papel. Além disso, um mesmo membro pode exercer mais de um papel em momentos diferentes do processo, de forma que fossem mais bem aproveitados a aptidão das pessoas e o paralelismo de suas ações.

Nos últimos anos, os processos de desenvolvimento de software vêm recebendo grande atenção principalmente no que diz respeito à sua definição, automação, medição e melhoramento.

De acordo com Pressman (2006), no contexto da engenharia de software, um processo é um framework para as tarefas necessárias à construção de softwares com alta qualidade, definindo a abordagem que será adotada enquanto o software estiver em desenvolvimento.

CAPÍTULO 3 – FRAMEWORK DE DESENVOLVIMENTO DE SOFTWARE

A Engenharia de Software é a área de conhecimento da Ciência da Computação responsável pelo estudo dos elementos envolvidos no processo de desenvolvimento de software (ou simplesmente, processo de software). Esse processo é o principal objeto de estudo da Engenharia de Software e pode ser definido como o mecanismo pelo qual um determinado problema do usuário, descrito por meio de um conjunto de requisitos, é convertido em uma solução automatizada, baseada em software. Para tal, incorpora princípios, métodos, metodologias e ferramentas, que permitem a gerência do processo de software; e fornece mecanismos para a construção de software de forma produtiva.

À medida que a importância do software cresceu a comunidade dos desenvolvedores de software tem continuamente tentando desenvolver tecnologias que tornem mais fácil, mais rápido e menos dispendioso construir e manter programas de computador de alta qualidade (PRESSMAN, 2006). Algumas dessas tecnologias são voltadas para um domínio de aplicação específica como projeto de sites da web; ou em se focam em bases mais amplas tais como exemplo sistemas orientados a objeto ou programação orientada a objeto.

Uma dessas alternativas é fazer uso de framework. A partir do conceito de reutilização de código, os frameworks estão sendo desenvolvidos e pesquisados (ASSIS et. al., 2003).

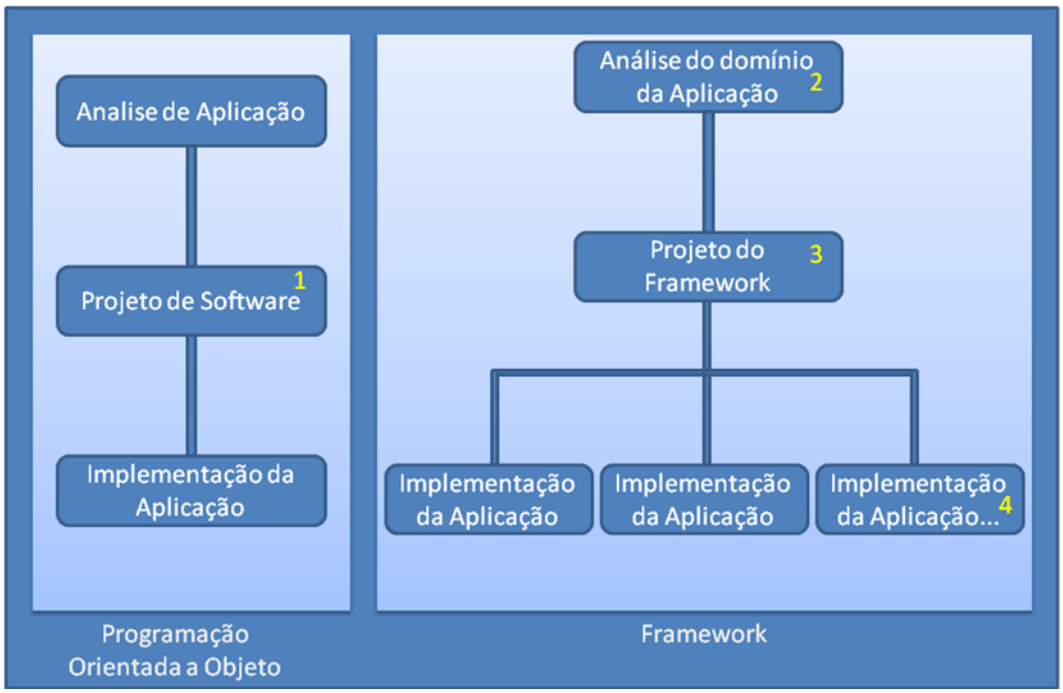
3.1 Framework

Um dos principais objetivos da Engenharia de Software é o reuso. Através dessa atividade é possível obter o aumento da qualidade do produto final reduzindo tempo, esforço e custo no processo de desenvolvimento de software. (GIMENES, et. al. 2005).

Conforme citado anteriormente a alternativa a ser utilizada são os Frameworks, na qual tem como propósito ajudar no processo de desenvolvimento de aplicações. Frameworks funcionam como molde ou estrutura para construções de aplicações e subsistemas dentro de um mesmo domínio (BERGAMO, 2000).

Na Figura 5 é possível observar a comparação entre dois processos de desenvolvimento de *software*, no qual um processo utiliza a programação Orientação Objeto e a outra a pura utilização de *framework*.compara o processo de desenvolvimento de *software* utilizando programação Orientada a Objeto pura e utilizando *frameworks*.

Figura 5 - Desenvolvimento com Programação Orientada a Objeto e Framework



Fonte: Apostila da disciplina de Programação Orientada a Objetos da Universidade Federal de Pelotas

A principal ideia de um *framework* não é desenvolver a solução para uma determinada aplicação, mas sim definir um comportamento geral de um domínio de aplicação e criar uma estrutura para representá-lo. A realização de um software para uma implementação específica consiste em instanciar o referido framework, através da especialização de seus componentes (CAVALHEIRO, 2007).

Em um Framework, a abstração ocorre na parte em que se definem quais funções possíveis podem ser implementadas numa aplicação específica. Uma de suas características importantes é a confiabilidade, pois boa parte de seu código já está pronto e testado, o que diminui significativamente a margem de erros juntamente com os esforços de manutenção (ASSIS et. al., 2003).

Na construção de uma aplicação é possível a utilização de vários frameworks para facilitar a escolha e o uso dos frameworks durante o desenvolvimento de softwares, e garantir a integração, evolução e manutenção dos mesmos, Macias (2008) propõe uma estrutura chamada Framework Integrador.

Para Macias (2008), um *framework* define parâmetros de projeto; a forma como é realizada a divisão de artefatos e suas responsabilidades; a maneira como colaboram entre si, e o fluxo de controle, ditando, assim, a arquitetura de software das aplicações que os utilizam

e fornecendo a padronização do projeto, estruturas essenciais para o desenvolvimento de projetos de softwares.

Como os *frameworks* absorvem as decisões de projeto que são comuns ao seu domínio de aplicação, eles priorizam a reutilização de projetos ao invés da reutilização de código, apesar de incluírem classes concretas que podem ser imediatamente utilizadas. O uso de *frameworks* e a reutilização em nível de projeto implicam em uma inversão de controle entre um código específico de aplicação e o *software* na qual ela se baseia. (GAMA et. al., 1995)

3.2 Framework de Processo

Um *framework* de processo é estabelecido pela definição de um pequeno número de atividades na qual são aplicados em todos os projetos de desenvolvimento de software, independentemente de seu tamanho ou complexidade. (PRESSMAN, 2006)

Para Pfleeger (2004), quando existe um conjunto de tarefas ordenadas pode ser considerado um processo: uma série de etapas que envolvem atividades, restrições e recursos para alcançar a saída desejada.

Um *framework* de atividades engloba dois tipos diferentes de atividades: as atividades de processo, que envolvem necessariamente as atividades de desenvolvimento de software, e as atividades de suporte, que envolvem as atividades de apoio ao processo de desenvolvimento, englobando as atividades de gerência (também chamadas de "atividades guarda-chuva") (PRESSMAN, 2006).

Nos últimos anos, vários processos de desenvolvimento de software foram propostos e desenvolvidos. Alguns mais rigorosos com as disciplinas a serem seguidas e os artefatos formais a serem produzidos no ciclo de vida do produto de software (PFLEEGER et al. 2005).

Com o crescimento de demandas que visam softwares de qualidade e eficiência em seus processos, se tornou imprescindível que o desenvolvimento e aprimoramento de processos de software evoluam constantemente de forma que possa atender as necessidades atuais, bem como sirvam como um arcabouço de conhecimentos para a elaboração de novos processos e metodologias de desenvolvimento de software, mais eficientes e adaptáveis.

De acordo com Pressman (2006), a elaboração de software é um processo de aprendizado na qual resulta na incorporação de conhecimentos coletados, destilados e organizados à medida que o processo é conduzido. Processo é o alicerce da engenharia de software. É ele que permite o desenvolvimento racional e oportuno de softwares de

computador. Ele pode ser definido para atividades como desenvolvimento, manutenção, aquisição e contratação de software (FILHO, 2009).

Processos de software formam a base para o controle gerencial de projetos de software e estabelece o conteúdo no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, formulários, etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas (PRESSMAN, 2006).

Em desenvolvimento de software é possível definir um processo de software como sendo um conjunto de atividades que leva à produção de um produto de software. (SOMMERVILLE, 2007).

Pressman (2006) define processos de software como sendo *frameworks* para tarefas que são necessárias para construir software de alta qualidade.

Em Filho (2009), faz uma analogia interessante, um processo de software é como uma receita a ser seguida.

Os Processos de softwares são complexos e como todos os processos intelectuais e criativos dependem de julgamento humano. A existência de um processo de software não garante que o software será entregue no prazo, de que ele irá satisfazer as necessidades do cliente, ou exibirá os atributos arquiteturais que levarão às características de qualidade em longo prazo. Um processo deve ser acoplado a uma sólida prática de engenharia de software e deve ser avaliado para garantir que satisfaça a um conjunto de critérios básicos de processo que demonstram ser essenciais para uma engenharia de software bem sucedida (PRESSMAN, 2006).

Não existe um processo ideal. As organizações devem criar fazer verificações, validar e aperfeiçoar seus próprios métodos (CMMI 2006). Várias destas desenvolvem abordagens inteiramente diferentes, adequadas à sua realidade, para o desenvolvimento de software. Os processos evoluem para explorar as capacidades das pessoas em uma organização e as características específicas dos sistemas que estão sendo desenvolvidos.

No caso de alguns sistemas, como os sistemas críticos, é necessário um processo de desenvolvimento muito bem estruturado. Nos sistemas de negócios, com requisitos que mudam rapidamente, um processo flexível e ágil é provavelmente mais eficaz (SOMMERVILE, 2007).

Existem vários processos de Desenvolvimento Software, porém algumas atividades fundamentais são comuns a todos eles (SOMMERVILE, 2007), como:

- **Especificação:** Define a funcionalidade do software e as restrições sobre sua operação.
- **Projeto e implementação:** O software que atenda a especificação deve ser produzido.
- **Validação de software**: O software deve ser validado para garantir que ela faça o que o cliente deseja.
- **Evolução:** O software deve evoluir para atender aos novos requisitos que naturalmente surgirão.

Processos de software têm como base modelos de processo genéricos. Esses modelos genéricos não são descrições definitivas de processos de software. Ao contrário, são abstrações do processo que podem ser usadas para explicar diferentes abordagens para o desenvolvimento de software.

Esses processos podem ser considerados como *framework* de processo que podem ser ampliados e adaptados para criar processos mais específicos de engenharia de software. Os modelos genéricos de processos de software amplamente utilizados atualmente são o modelo em cascata, o modelo de desenvolvimento evolucionário e o modelo de desenvolvimento baseado em componentes. Estes, não são mutuamente exclusivos e comumente são utilizados em conjunto, especialmente para desenvolvimento de sistemas de grande porte (SOMMERVILLE, 2007).

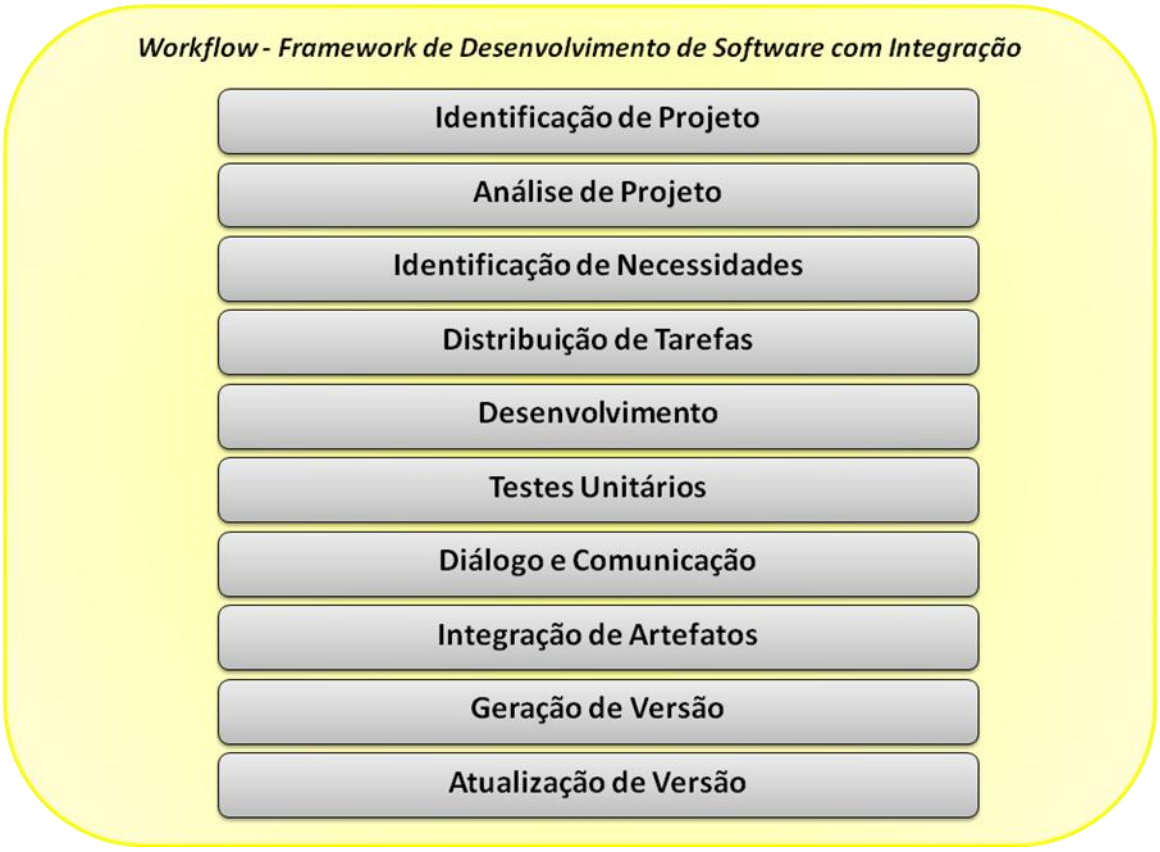
3.3 Framework para Desenvolvimento de Software

O Framework elaborado nesse trabalho visa trazer benefícios no desenvolvimento de software, aumentando a produtividade no desenvolvimento de aplicações além de promover um grande aumento de desempenho em tempo de *runtime*, e facilitar o trabalho com testes unitários.

No processo de desenvolvimento de um software pode se tomar vários caminhos, porém a decisão de seguir um framework para esse processo faz com que as atividades do desenvolvimento se padronizem e assim tendo os benefícios que já foram citados anteriormente nesse projeto.

A figura 6 contém as fases do framework a ser seguido em um projeto de desenvolvimento de software.

Figura 6 - Workflow do Framework de Desenvolvimento de Software com Integração



Fonte: Autoria Própria

Para que se que seja possível entender os passos a serem seguidos em um framework de desenvolvimento para uma equipe, é preciso seguir o workflow descrito na Figura 6.

- **Identificação do Projeto:** A primeira fase é quando o gerente da equipe se reúne com o cliente para que possam avaliar o status da organização na qual o sistema resultante deve ser implantado (a organização-alvo), conforme definido em Identificação do Projeto junto ao cliente.
- **Análise de Projeto:** Com base nos resultados das avaliações, a próxima fase a ser seguida é a de Análise do Projeto, onde após receber as definições funcionais os analistas definem as necessidades do sistema, o que são funcionais técnicas e o que não são funcionais, essa fase também serve para projeto de desenvolvimentos que já existem e que vão apenas sofrer alterações.
- **Identificação de Necessidades:** Nessa fase os analistas já têm o conhecimento do projeto, e então são definidas quais as necessidades do projeto e assim tendo qual prazo para o término do projeto, quais e quantos recursos irão ser usados, em que cenário o software será utilizado e qual tecnologia utilizar. Nessa fase é muito importante saber quais os recursos utilizar nesse projeto, saber se a

equipe a trabalhar no projeto tem bons conhecimentos do domínio do negócio e também saber como os sistemas atuais são utilizados para automatizar os negócios.

- **Distribuição de Tarefas**: O gerente de projeto inicia o processo de distribuição de tarefas dentro da equipe, onde ele se certificará que as tarefas serão distribuídas corretamente, de forma que as funções determinadas o membro capacitado.
- **Desenvolvimento:** Em seguida é o início do desenvolvimento do software, onde cabe a cada membro de uma equipe exercer seu papel.
- **Teste:** Os testes a serem realizados nessa fase são os testes unitários, onde cabe a cada desenvolvedor se certificar que o que está sendo desenvolvido está correto sem erro no seu código.
- **Comunicação da Equipe:** Essa é a fase do framework em que pode acontecer em qualquer período do processo, isso dependerá da necessidade da equipe, lembrando que esse é uma interação de extrema importância já que a comunicação em uma equipe é fundamental para o bom funcionamento do processo.
- **Integração de Artefatos:** Essa é a fase em que o Integrador normalmente executa o processo de integrar os processos, ou seja, agrupar seus subsistemas ao todo do sistema. A integração normalmente é executada por uma única pessoa (em um pequeno projeto no qual o processo de build é simples) ou uma pequena equipe (em um grande projeto no qual o processo de build é complexo).
- **Geração de Versão**: Após os testes integrados é gerado um pacote com todos os artefatos daquele sistema. Pode se dizer que essa é a fase da geração do *build*. Os integradores precisam ter experiência em gerenciamento de *builds* de software, em gerenciamento de configuração e na linguagem de programação em que os componentes que serão integrados são escritos. Como a integração frequentemente envolve um alto grau de automatização, também é essencial que haja habilidade no *shell* do sistema operacional ou nas linguagens de scripts e ferramentas como 'make' (no Unix).
- **Atualização no repositório de versão:** Atualizar repositório com versão atual sempre que necessário e em seguida é necessário a verificação de que tudo este ok com o software.

3.4 Consequências ao adotar Frameworks

A utilização de *frameworks* no desenvolvimento software traz benefícios, originados de suas características principais: são modulares, reusáveis extensíveis e eventualmente assumem o controle da execução invocando métodos da aplicação quando necessário (inversão de controle). Por outro lado, há certos desafios na criação e uso de frameworks que não podem ser ignorados. Esta seção lista as principais vantagens e desafios decorrentes da adoção de *frameworks.*

3.4.1 Benefícios Decorrentes da Utilização de Frameworks

Segundo Fayad et al. (2000), os principais benefícios decorrentes da utilização de frameworks, advêm da modularidade, reusabilidade, extensibilidade e inversão de controle que os frameworks proporcionam.

Frameworks oferecem pontos de extensão explícitos que possibilitam aos desenvolvedores estenderem suas funcionalidades para gerar uma aplicação. Os pontos de extensão desacoplam as interfaces estáveis do framework e o comportamento de um determinado domínio de aplicação das variações requeridas por uma determinada

3.4.2 Desafios Decorrentes da Utilização de Frameworks

Quando usado em conjunto com padrões de projetos, bibliotecas de classes, componentes, entre outros, frameworks de aplicação têm o potencial para aumentar a qualidade de software e reduzir o esforço de desenvolvimento.

Contudo, instituições que tentam construir ou usar frameworks frequentemente falham a menos que resolvam os seguintes desafios: esforço de desenvolvimento, curva de aprendizagem, integração, eficiência, manutenção, validação e remoção de defeitos e falta de padrões (FAYAD et al., 1999b).

Destes desafios, os esforços de desenvolvimento afetam principalmente os projetistas de *frameworks*. A curva de aprendizagem, a integração e a eficiência afetam principalmente os desenvolvedores de aplicação. A manutenção e validação e remoção de

defeitos afetam tanto desenvolvedores de aplicação quanto mantenedores de frameworks e a falta de padrões afetam a todos envolvidos no desenvolvimento e uso dos frameworks.

Quanto à curva de aprendizagem, aprender a usar um framework de aplicação leva tempo e incorre em custos. A menos que o esforço de aprendizado possa ser amortizado através do uso do framework em muitos projetos, ou em projetos duradouros, o investimento pode não compensar. Deve ser analisado se os benefícios trazidos pelo framework compensam os custos com o treinamento da equipe de desenvolvimento. (FAYAD et al., 1999b).

O desafio da integração ocorre quando diversos frameworks são usados com outras tecnologias e com sistemas legados não previstos pela arquitetura do framework. Antes de usar um framework o desenvolvedor de aplicações deve analisar possíveis problemas que possam surgir ao incorporar novos frameworks à aplicação. (FAYADA et al., 1999b).

3.5 Considerações Finais

Conforme citado anteriormente o principal da engenharia de software é a reutilização, onde pode obter o aumento da qualidade e redução do esforço de desenvolvimento. Os frameworks possibilitam às equipes a utilização de famílias de aplicações, onde geradas a partir de uma estrutura que captura os conceitos gerais das aplicações.

Fowler (2006) define que um rápido desenvolvimento de sistemas novos devem ser construídos a partir de classes que já existam e que possam ser adaptadas às circunstâncias.

O desenvolvimento com utilizando frameworks traz benefícios. Frameworks diminuem o esforço para entender e manter a aplicação. Frameworks incentivam o reuso, pois capturam o conhecimento de desenvolvedores em determinado domínio ou aspecto de infra-estrutura ou de integração de *middleware*. Além disso, frameworks são expansíveis por construção e, por fim, o uso de inversão de controle simplifica o desenvolvimento de aplicações.

CAPÍTULO 4 – ESTUDO DE CASO

Neste capítulo, todos os conceitos anteriormente apresentados e discutidos serão aplicados no framework de desenvolvimento de software com prática de Integração Contínua.

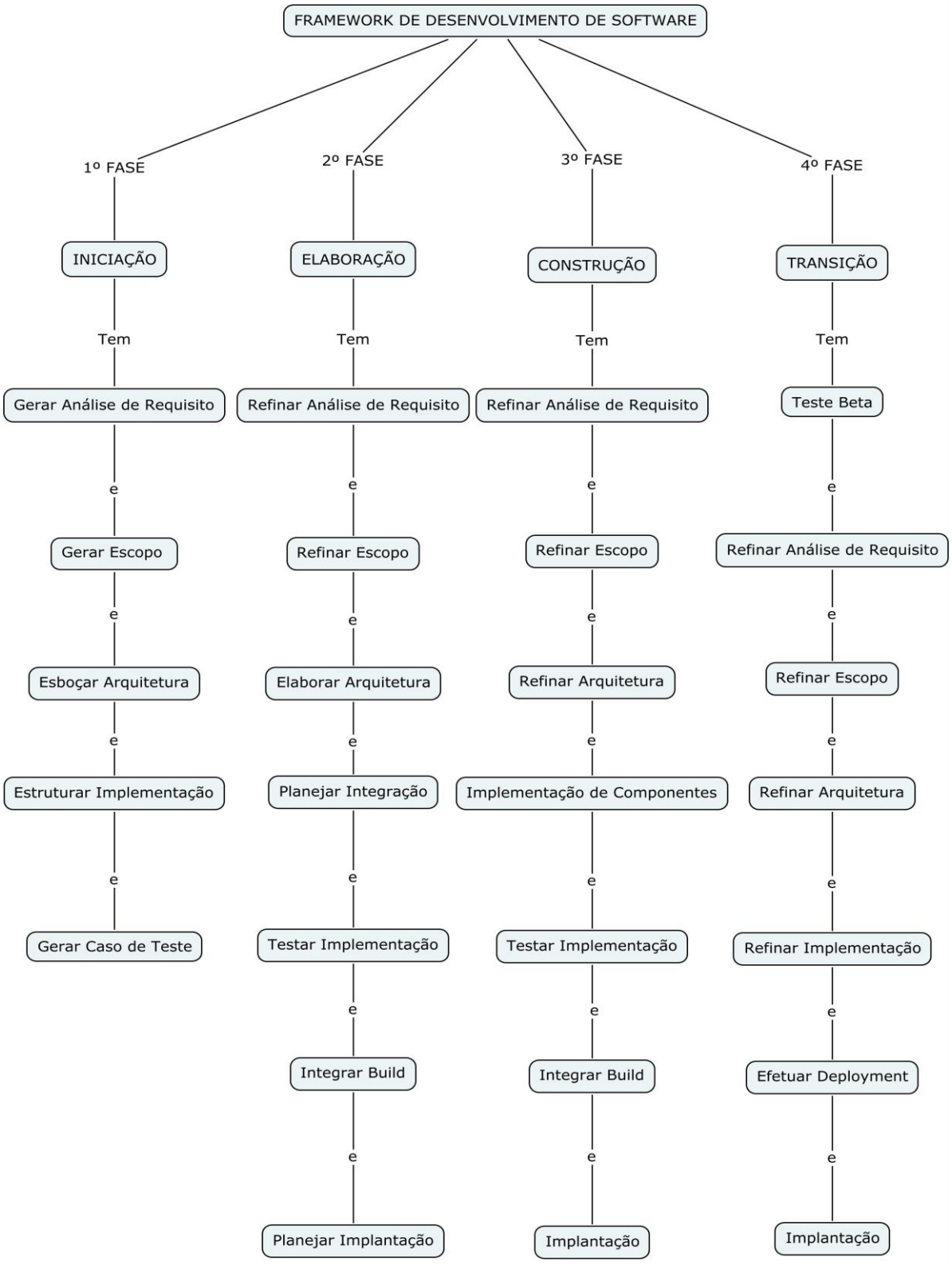
A proposta do deste trabalho é elaborar um *framework* de processo de desenvolvimento de software completo e ágil. Os processos derivados desse framework podem ser considerados leves e voltados para a necessidade de qualquer projeto de desenvolvimento. Sua aplicação visa a facilitar o dia a dia de equipes de desenvolvimento de software com a distribuição eficaz das tarefas em um processo de desenvolvimento de projetos.

4.1 Fases do Framework

Acompanhando a metodologia de desenvolvimento de software, o framework foi dividido em fases, sendo elas: Iniciação, Elaboração, Construção e Transição. Para que o framework elaborado nesse trabalho tenha uma perspectiva de gerenciamento as fases foram divididas em iterações que visam se integrarem no final.

Encontra-se na Figura 7 o mapa conceitual do framework baseado na metodologia de desenvolvimento de software proposto neste trabalho para que se possa ter um melhor entendimento do seu funcionamento.

Figura 7 - Mapa Conceitual Esquema de Fases do Framework de desenvolvimento de Software



Fonte: Autoria Própria

4.1.1 Iniciação

Na fase de Iniciação, o foco principal dos levantamentos das necessidades juntamente com o cliente e assim estabelecendo a comunicação e tendo seguimento nas demais fases. As entrevistas com o cliente são realizadas nesta fase para que todo levantamento possa servir como base no plano de projeto da equipe, na qual seja possível avaliar os possíveis riscos, estimativas de custos e prazos, estabelecendo as prioridades, levantando requisitos do sistema e preliminarmente analisá-los. Assim, ambas as partes podem aprovar as definições do escopo do projeto, onde são examinados os objetivos para se decidir sobre a continuidade do desenvolvimento.

Esta é uma fase muito importante para novos produtos em desenvolvimento, nas quais existem muitos riscos de negócio e de requisito que precisam ser tratados com cuidado para que não prejudiquem a continuidade do projeto. As próximas fases existem apenas o refinamento dos requisitos e os projetistas devem se assegurar que nesta fase seja compensatório dar continuidade ou não.

Os objetivos principais da fase de iniciação incluem:

- Entendimento do que deve ser construído para que possa ser determinada a visão geral, incluindo o escopo do sistema e seus limites.
- Identificação das funcionalidades chaves para decidir quais requisitos são mais críticos e devem ser priorizados.
- Avaliar se a visão é tecnicamente factível. Podendo ser necessário a criação de protótipo técnico e/ou definir uma arquitetura de alto nível, para que possa determinar uma possível solução.
- Compreensão das estimativas de alto nível para avaliação de custos, prazos e riscos associado ao projeto.
- Adquirir um entendimento mais detalhado dos requisitos priorizados para criação de planos mais detalhados de cada iteração.
- Construir e validar uma arquitetura e um incremento de valor para o cliente.

4.1.2 Elaboração

A segunda fase abrange a modelagem do modelo genérico do processo. Esta fase envolve uma análise detalhada sobre as necessidades e problemas gerais do projeto e a

definição de como os sistemas serão desenvolvidos em termos tecnológicos, considerando os requisitos, limitações e restrições identificadas durante a fase de Iniciação, arquitetura do projeto começa a ter sua forma básica. Dúvidas como se o plano do projeto pode ser confiável ou se os custos são admissíveis podem ser esclarecidas nessa etapa.

Criar a *baseline* para a arquitetura do sistema a fim de fornecer uma base estável para o esforço da fase de construção. Para que seja possível determinar com segurança os custos e o tempo de programação de um projeto precisamos assegurar que a arquitetura, os requisitos e os planos sejam estáveis o suficiente e não comprometa o projeto.

Os objetivos principais da fase de elaboração incluem:

- Tratar de riscos significativos do ponto de vista da arquitetura do projeto.
- Estabelecer uma arquitetura derivada do tratamento dos cenários.
- Determinar com segurança que os requisitos, arquitetura e planos sejam estáveis o suficiente e diminuído a fim de determinar com segurança os custos.
- Elaborar um protótipo para que possa diminuir os riscos específicos como troca de requisitos, reutilização de componentes e demonstrativos para o cliente.
- Ter um suporte de ambiente.

4.1.3 Construção

Na fase de construção ocorre o processo de desenvolvimento do software em que um produto é desenvolvido de maneira frequente até que esteja pronto para ser avaliado por um Gestor de projeto.

Essa fase é de certa forma um processo de manufatura, em que a ênfase está no gerenciamento de recursos e controle de operação para que possa aperfeiçoar os custos, desenvolvimento de código onde o desenvolvedor possa realizar teste e integrar seus artefatos e garantir a qualidade de seu produto. Esta é uma fase em que exige a integração entre desenvolvedores, analistas e administradores de dados para uma analise, construção e teste do software.

Os objetivos principais da fase de construção incluem:

- Desenvolver soluções para que o projeto possa chegar à fase de transição e visando ter ótima qualidade e sem algum problema.

- Minimizar os custos focando a o aperfeiçoamento de recursos, evitando retrabalho e atividades desnecessárias. Utilizando práticas de reutilização de artefato.
- Atingir certo grau de paralelismo entre equipes de desenvolvimento planejando o uso dos recursos em sinergia com foco na minimização de custos e com a priorização dos itens a serem desenvolvidos a cada iteração.
- Deve atingir uma versão de entrega com qualidade adequada, rapidez e eficiência agregando o valor para o cliente.

4.1.4 Transição

Nesta fase o software é testado no geral, com todos os módulos integrados e então avaliados para a verificação se os requisitos foram atendidos. A evidência desta fase é assegurar que o software esteja pronto e disponível para o seu usuário final. Ao chegar nos objetivos principais da fase de transação significa que o projeto está pronto e pode ser encerrado.

A Fase de Transição pode atravessar várias iterações e incluir testar produtos em fases de certificação do produto e realizar pequenos ajustes com base no *feedback* do usuário. Os problemas estruturais mais graves devem ter sido encontrados em fases anteriores e assim não permitindo que ocorra esse fato nesta fase.

Em alguns projetos esta fase pode coincidir com a liberação total dos artefatos a terceiros que poderão ser responsáveis pela operação, manutenção e melhorias no software liberado, a mesma pode ser muito fácil ou muito complexa isso irá depender do tipo do projeto.

Como citado anteriormente o projeto é concluído quando todos os requisitos foram atendidos e para que isso aconteça o nível de qualidade aceitável e documentação do usuário tenha sido concluída e que o usuário possa obter resultados positivos de todas as partes.

Os objetivos principais da fase de transição incluem:

- O teste beta é para validar o novo software em confronto com as expectativas do usuário. Realização de operações paralelas relativas a um sistema legado que está sendo substituído, caso haja um sistema anterior ao do novo produto.
- Migrações e/ou carga de dados em banco de dados para conversões de banco de dados operacionais

- Treinamento de usuário e equipe de manutenção para que adquiram conhecimento para as equipes de operação e manutenção.
- Ajustar e corrigir os erros focando a melhoria do desempenho, qualidade e usabilidade do produto.
- Terno de aceite obtendo o consentimento dos envolvidos em relação à realise entregue. A geração de uma *baseline* final de todos os artefatos do projeto.
- Documentar as lições aprendidas no projeto.

4.2 Integração Contínua no Framework

Em uma das fases já citadas anteriormente foi possível identificar o foco da utilização da Integração sendo elas: Elaboração e Construção. Essas fases focam em atividades de planejamento de integração e a integração de artefatos. Para realizar o processo de integração neste o projeto, será utilizado o conceito de Integração Contínua.

4.2.1 Planejamento de Integração

O planejamento da integração deve ser realizado com antecedência quando se utiliza arquitetura com versões de artefatos em repositórios. Não se torne obsoleto pelas mudanças na arquitetura. É fundamental o planejamento da implementação dos subestimas a serem trabalhados na qual deve ser integrados.

A Integração Contínua conforme citado anteriormente é uma prática bastante utilizada pelas equipes de desenvolvimento de software e consegue dividir o trabalho de toda equipe que evita o desperdício de tempo e o retrabalho desnecessário.

O processo de integração na maioria das vezes é executado por uma pessoa apenas ou uma pequena equipe, isso dependerá do tipo de projeto que está sendo executado.

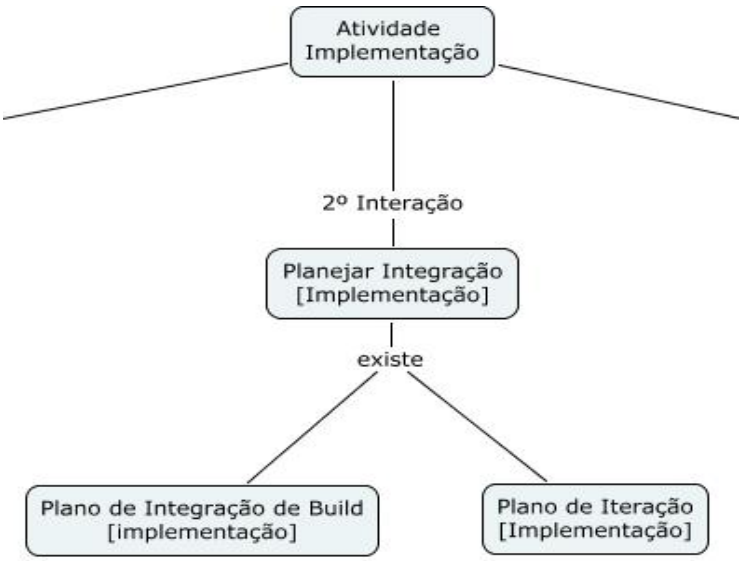
Em ambos os casos a Integração Contínua é executada quando o código integrado e testado após algumas horas ou no máximo no final do dia do desenvolvimento. A maneira simples de fazer isso é ter uma máquina dedicada apenas à integração (ou um servidor). Quando a máquina estiver livre, um par com código para ser integrado a ocupa baixa a versão atual e as modificações e executa testes até que eles passem totalmente sem erro.

É importante que o integrador tenha experiência com a linguagem a ser integrada, com o gerenciamento de *builds*, para que se possa garantir a boa qualidade do produto após a

integração. Devido a grande frequência de integração o grau de automatização em um processo se torna elevado e com isso o integrador também deve possuir habilidades no Shell do sistema operacional ou nas linguagens de script.

Na figura 7 é apresentado esquema de planejamento de integração a ser usado na fase de elaboração na atividade implementação.

Figura 8 - Mapa conceitual ilustrando a atividade de Implementação utilizando Planejamento Integração



Fonte: Autoria Própria

4.2.2 Implementar Componentes

Todo código fonte de um software está centralizado em um repositório sobre o controle de versões em um servidor. Além do código fonte do projeto o servidor mantém todos os *scripts* necessários para a construção e a integração, esquema do banco de dados e documentação.

A partir do repositório, os desenvolvedores atualizam os ambientes de desenvolvimento de seus equipamentos, programam novas funcionalidades, compilam, integram, testam e então atualizam o repositório com o novo código. Esta tarefa é realizada em intervalos que podem variar de horas até dias, dependendo da complexidade da nova funcionalidade.

Antes de buscar os componentes no repositório o integrador precisa se certificar que o código fonte escrito já passou por compilação, testes unitários e implementado as classes no modelo designer para iniciar o processo de integração desses componentes a um todo, porém se houver erro o código deve ser enviado novamente para o retrabalho para os desenvolvedores que realizaram as mudanças necessárias e passando novamente por uma revisão de código tais como testes unitários e avaliações da qualidade do código para que o integrador não tenha problemas na hora de integrar todos os componentes.

O processo de integração é realizado de forma independente das atualizações, efetuadas pelos desenvolvedores no repositório. Sua execução é realizada em equipamentos reservados exclusivamente para isto, com todos os pacotes e ferramentas necessários mantidos pela gerência de configuração, e com acesso controlado. O controle de acesso em uma máquina de integração é necessário e fundamental, pois se deve manter controle sobre todas as configurações do ambiente de compilação e montagem dos sistemas. Isto evita que, acidentalmente, sejam substituídas opções de configuração e o produto seja produzido incorretamente. O gerenciamento das configurações destas ferramentas também é importante, pois independente das configurações adotadas por um desenvolvedor, a máquina de integração garante a montagem e integração da aplicação com as opções adequadas.

Basicamente, o algoritmo de integração, executado na máquina de integração obedece aos seguintes passos:

- As ferramentas de controle de versão têm uma identificação de usuário, já que toda alteração realizada na ferramenta é registrada com o nome do usuário. Assim no primeiro momento é realizada a identificação das pessoas que atualizaram os arquivos e os estados dos arquivos envolvidos no processo;
- Remoção de todo e qualquer código fonte da aplicação. Isto elimina definitivamente qualquer possibilidade de se usar um código fonte desatualizado;
- Remoção de qualquer objeto já compilado, arquivos temporários, scripts, e qualquer outro código que seja passível de atualização pelos desenvolvedores;
- A descarga *(checkout)*, repositório mantido pelo controlador de versão de todo o código fonte e outros scripts necessários para a integração da aplicação;
- Execução dos scripts de configuração da estrutura de diretórios temporários e de saída de resultados da integração;
- Execução dos scripts de compilação do código fonte e testes unitários;

- Execução dos scripts de testes unitários;

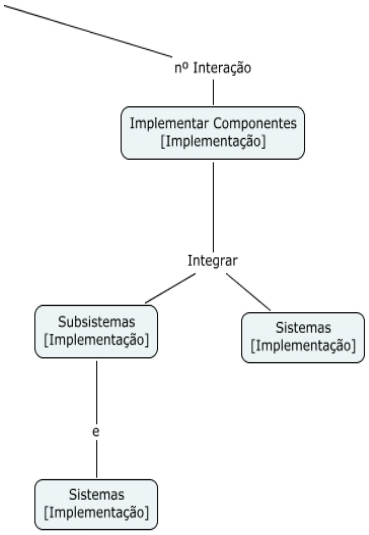
4.2.2.1 Integração de Subsistemas

A integração do subsistema é realizada de acordo com o planejamento de integração, para a qual a ordem de integração de componentes e subsistema foi projetada. Quando o projeto possui subsistemas grandes pode ser utilizado um plano de integração secundária criado especificamente para ele.

É recomendável integrar os componentes de forma incremental e hierárquica para que ao ser compilado não exista erro. A cada incremento, adicione um ou alguns componentes ao sistema.

Quando dois ou mais desenvolvedores trabalha paralelamente em um mesmo subsistema o trabalhando deve ser integrado em um espaço de trabalho reservado especialmente para integração de subsistemas na qual liberarão componentes a partir dos respectivos espaços de trabalho de desenvolvimento privado e onde o integrado possa construir o build. Para que o trabalho em equipe possa funcionar perfeitamente é importante que os desenvolvedores estejam sempre compartilhando os resultados com frequência quando se tem mais de um desenvolvedor trabalhando em um subsistema.

Figura 9 - Mapa conceitual ilustrando a atividade de Implementação utilizando Implementação de Componentes



Fonte: Autoria Própria

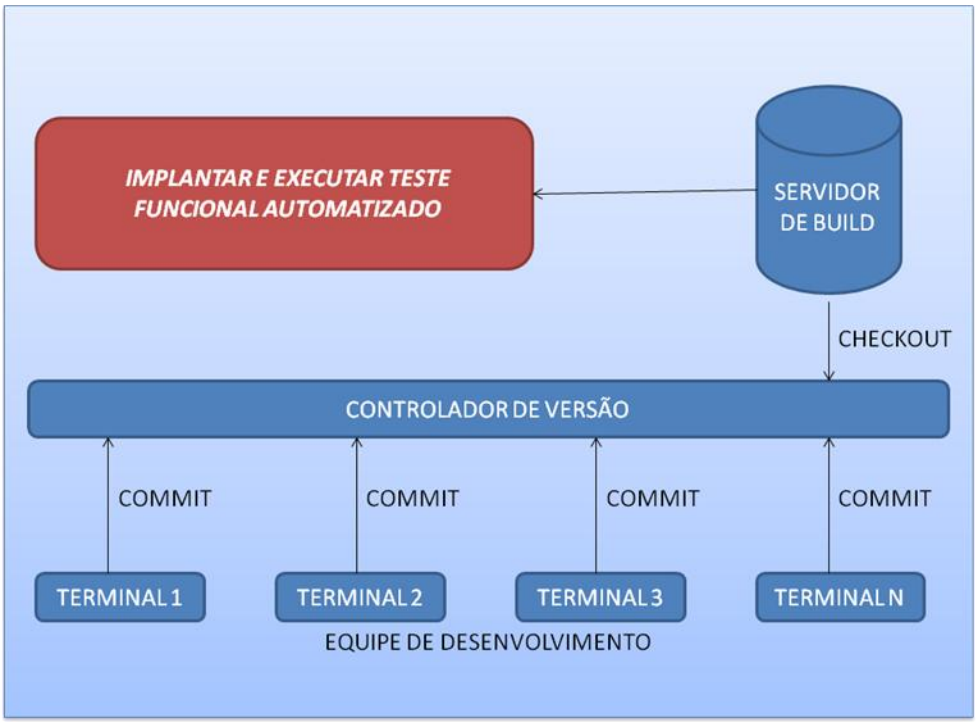
A liberação do subsistema implementado deve ser após o último incremento, quando subsistema de implementado estiver pronto e o *build* associado tiver sido testado, quando houver á integração, o subsistema será liberado para o espaço de trabalho de integração do sistema.

4.3 Processo de Integração de Build

Para o integrador é importante seguir o plano de integração de build para que não haja erro no processo de Integração do *build.* Uma importante na atividade de integração do build é a identificação de quais subsistemas de implementação participam dos casos de uso e cenários da iteração atual.

O *build* é gerado a partir de outros componentes no modelo de implementação e liberando subconjuntos testáveis dos recursos e funções em tempo de execução do sistema. A construção do *build* deve ser instituída conforme o andamento da implementação dos componentes e seus aprimoramentos. É possível construir *build* em todos os níveis de um sistema, abrangendo um único sistema ou vários.

Figura 10 - Integração Contínua no Framework nas fases de Elaboração e Transição



Fonte: Autoria Própria

O *framework* elaborado nesse trabalho prevê que os *builds* sejam definidos ao término das integrações de componentes. Os *builds* informais podem ser construídos por um desenvolvedor por vários motivos (teste unitário, por exemplo), usando componentes provenientes do espaço de trabalho de desenvolvimento particular do desenvolvedor e dos espaços de trabalho de integração do sistema e subsistema, conforme apropriado. Contudo, como o termo é usado aqui, os builds são construídos por um integrador a partir de versões identificadas de componentes liberados pelos desenvolvedores aos espaços de trabalho de integração do sistema ou subsistema.

CAPÍTULO 5 – CONCLUSÃO

Este projeto propôs a elaboração de um framework para desenvolvimento de software com práticas de Integração Contínua. Com embasamento adquirido foi possível concluir que o processo de Integração de software disciplinado e automatizado é essencial para controle de um projeto de desenvolvimento de software. Esse Framework destina-se ao desenvolvimento de qualquer tipo de software independente de seu tamanho já que foi seguido a metodologia já existente de desenvolvimento de software.

O uso efetivo da prática de Integração Contínua está associado à busca pela automatização de todo processo que seja necessário realizar tarefas tais como: compilação, montagem, testes unitários e outros. A função de um desenvolvedor é verificar se o código passou por teste, se integrou um novo módulo ou a sua atualização na máquina de desenvolvimento, atualizar o repositório sobre uma ferramenta de controle de versão com frequência.

Como perspectiva para trabalhos futuros a sugestão é implantar este framework em um ambiente real com uma equipe de desenvolvimento de software para que possa ser avaliado e todos resultados obtidos sejam estudados minuciosamente para se ter certeza da eficácia do Framework e se houver a necessidade realizar melhorias no mesmo.

REFERÊNCIA

ANDRES, C; BECK, K.; **Extreme Programming Explained:** Embrace Change. 2ª edição. Addison Wesley Professional, 2004

ARAUJO, José Marcelo Pereira; VIANA, Alberto Guimarães, Artigo: "**Integração Contínua no Processo de Desenvolvimento de Software**", 2011. Disponível em < http://revista.faculdadeprojecao.edu.br/revista/index.php/projecao2/article/viewFile/85/73> Acessado 12 mar.2012

ASSIS, S. R.; SUZANO, R. **Framework: Conceitos e Aplicações**. CienteFico, Salvador, v.2, Jun./Dez. 2003.

BARBOSA, Ana Emília Victor, Trabalho de Conclusão de Curso "**Teste de Integração a Partir de Modelos UML**", 2004. Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia.

BERGAMO, L. E. C. Dissertação (Mestrado em Ciência da Computação**) Um Framework de Acompanhamento para Aplicações Multimídia Distribuídas de Ensino a Distância Utilizando a Plataforma JAMP**. 2000. - Departamento de Computação, Universidade Federal de São Carlos, São Carlos.

BRINDER, R. V., Testing object-**oriented systems: models, patterns and tools**, AddisonWesley, 2000

CAVALHEIRO, G. G. H. Material de Apoio 14 - Framework. [2007], Apostila da disciplina de Programação Orientada a Objetos. Universidade Federal de Pelotas.

CMMI (2006*), **"CMMI for Development, Version 1.2***"**,** CMU/SEI-2006-TR-008. Software Engineering Institute.

COLLABNET. **Concurrent Versions System**. Disponível em: <http://www.cvshome.org>. Acesso em: 05 abril. 2012

DIAS, Klessis Lopes. Dissertação (Mestrado) – "**Um framework orientado a aspectos para monitoramento e análise de processos de negócio** ". 2008 - PUC do Rio de Janeiro, Departamento de Informática.**DUVALL, Paul M.; GLOVER, Andrew; MATYAS, Steve;** Conti-

nuous Integration: improving software quality and reducing risk**. Boston: Addison-Wesley, 2007**

ESMIRELLO, Rodrigo Alcântra, Trabalho de Conclusão de Curso do Curso de Ciência da Computação. "**Técnicas e Procedimentos de Software com Qualidade",** 2005. Faculdade de Jaguariúna

FAYAD, M. E. & Johnson, R. E. (2000): **Domain-specific application frameworks: frameworks experience by industry**. New York: J. Wiley, c2000. 681 p. ISBN

FAYAD, M. Schmidt, D. **Object-Oriented Application Frameworks. Communications of the ACM**, New York, Set 1999.

FILHO, Paula; PÁDUA, Wilson, **Engenharia de Software: Fundamentos, Métodos e Padrões.** 3ª edição, LTC, Rio de Janeiro, 2009.

FOWLER, Martin, Artigo: "**Continuous Integration**", 2006-a. Disponível em: <http://martinfowler.com/articles/continuousIntegration.html > Acessado em 07 mar. 2012.

FOWLER, Martin**.** ***Padrões de Arquitetura de Aplicações Corporativas***. Porto Alegre. Bookman, 2006-b.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph "et al." **Design patterns: elements of reusable object-oriented software**. 1995.

GIMENES, I. M. de S.; HUZITA, E. H. M.. **Desenvolvimento baseado em componentes: conceitos e técnicas**. Rio de Janeiro: Ciência Moderna, 2005.

HIGHTOWER, R. et al. **Professional Java Tools for Extreme Programming**. Wrox Press, 2004

NBR ISO 9000 – **Sistemas de Gestão da Qualidade**, ABNT, 2000.

LIMA, G. M. P. S.; G. H. Travassos. **Testes de Integração Aplicados a Software Orientados a Objetos: Heurísticas para Ordenação de Classes**. Relatório Técnico ES-632/04, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 2004

MACIAS, A. M**. Frameworks de Desenvolvimento** – *Visão Geral*. Tematec, Ano X, nº XX, p.1, 2008. Disponível em <www.serpro.gov.br/clientes/serpro/serpro/imprensa/publicacoes/tematec/2008/ttec92_a>. Acesso em 24/07/2012.

MASSOL, V.; VAN ZYL, J. **Better Builds With Maven: How-to Guide for Maven 2.0.** Mergere Library Press, 2006.

NARDON, Fabiane Bizinella, **Integração contínua com Hudson-Configuração, Extensão e Diversão!,** Disponível em < http://www.sucesusp.org.br/mailing2009/congresso/justjava/apresentacoes/AuditorioPrincipal/Hudson.pdf**>,** Acessado em 06 abr. 2012

PFLEEGER, Shari Lawrance. **Engenharia de Software: teoria e prática**, Segunda Edição, Prentice Hall, 2004.

PINTO, S. C. C. S. (2000): Composição em WebFrameworks, tese de doutorado, Departamento de Informática PUC-Rio.

PRESSMAN, R. S. (2006), **Engenharia de Software,** McGraw-Hill, sexta edição.

RYSER, Johannes; GLINZ, Martin. A practical approach to validating and testing software systems using scenarios. Disponível em: <http://www.ifi.unizh.ch/groups/req/ftp/papers/QWE99_ScenarioBasedTesting.pdf>. Acessado em 23 set. 2012

RUP, **Rational Unified Process**, Disponível em: < http://www.wthreex.com/ > Acessado em 08 mar. 2012.

SILVA, Nelson Peres da. **Análise e estruturas de sistemas de informação**. Editora Érica, 2006 - São.Paulo

SOARES, Michel dos Santos, **Artigo Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**, 2006. Disponível em: < www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf >, Acessado em 15 mar. 2012

SOMMERVILLE, I. (2007): Engenharia de Software. 8 ed. São Paulo: Addison Wesley, 2007.