UNIVERSIDADE DO VALE DO RIO DOS SINOS

UNIDADE ACADÊMICA DE EDUCAÇÃO CONTINUADA

ESPECIALIZAÇÃO EM QUALIDADE DE SOFTWARE

DANIEL BALDO

IMPLANTAÇÃO DA INTEGRAÇÃO CONTÍNUA E SEUS BENEFÍCIOS:
um estudo de caso.

São Leopoldo

2012

DANIEL BALDO

IMPLANTAÇÃO DA INTEGRAÇÃO CONTÍNUA E SEUS BENEFÍCIOS:
um estudo de caso.

Trabalho de Conclusão apresentado como requisito parcial para a obtenção do título de Especialista em Qualidade de Software pelo curso de Pós Graduação Lato Sensu em Qualidade de Software da Universidade do Vale do Rio dos Sinos - UNISINOS

Orientador: Esp. Elias Nogueira

São Leopoldo
2012

Dedico este estudo:
à minha noiva Letícia;
à minha família;
aos meus amigos e colegas de trabalho.

RESUMO

A evolução de todo projeto de software é baseado em mudanças. Neste sentido, cabe destacar a importância da utilização do processo de Integração Contínua, afim de que as mudanças no projeto de software não afetem negativamente o seu resultado, mantendo o projeto integro e sempre disponível para ser implantado em produção. Esta pesquisa tem como objetivo apresentar os conceitos sobre Integração Contínua e, através do estudo de caso, investigar e analisar os resultados obtidos com a implantação deste processo.

Palavras-Chave: Integração Contínua. Compilação Automatizada. Gestão de Configuração de Software.

ABSTRACT

The evolution of each software project is based on changes. In this sense it necessary to detach the importance of using the Continuous Integration process to show that changes in software project does not affect negatively its results and the project intact and always available to be deployed in production. The paper has the aim to introduce the concepts of Continuous Integration and with the case study, investigate and analyze the results obtained with the implementation of this process.

Keywords: Continuous Integration. Automated Build. Software Configuration Management.

LISTA DE ABREVIATURAS E SÍMBOLOS

CVS – Current Versioning System (Sistema de Versões Concorrentes)

FTP – File Transfer Protocol (Protocolo de Transferência de Arquivos)

IC – Integração Contínua

SVN – SubVersion

TI – Tecnologia da Informação

TDD – Test Driven Development (Desenvolvimento Direcionado pelos Testes)

XP – Extreme Programming (Programação Extrema)

SUMÁRIO

1 Introdução 8
2 Desenvolvimento 10
2.1 Integração dos Projetos 10
2.2 Extreme Programming (XP) 13
2.2.1 Princípios da XP 14
2.2.2 Boas Práticas da XP 15
2.3 Integração Contínua 17
2.3.1 Valores da IC 17
2.3.2 Práticas da IC 18
2.3.3 O Processo de IC 21
2.3.4 Fases da IC 22
3 Estudo de Caso 24
3.1 Implantação do Processo 26
3.1.1 Processo Antigo 27
3.1.2 Processo Atual 29
3.2 Servidor de Integração Contínua 35
4 Análise de Resultados 40
4.1 Melhorias Obtidas 40
4.1.1 Redução dos Problemas e Retrabalho. 41
4.1.2 Feedback Instantâneo 42
4.1.3 Equipe Mais Ágil 43
4.1.4 Processo Gera Benefícios para Equipe 44
4.1.5 Antecipação dos Problemas 45
4.1.6 Entregas mais frequentes do sistema em produção 46
4.1.7 Identificação e Correção Rápida dos Problemas 47
4.1.8 Redução da Instabilidade do Projeto 48
4.1.9 Segurança para a Equipe 49
4.1.10 Facilidade da Gestão dos Projetos 50
5 Considerações Finais 51

5.1 Melhorias Futuras.....51
5.2 Conclusões.....53
Referências.....54
Glossário.....55
Apêndice A.....56

1 Introdução

Considerando que cada vez mais as empresas buscam a excelência na qualidade do software visando à otimização dos recursos, redução no prazo de entrega e instabilidades causadas pelas mudanças e erros frequentes, o processo de Integração Contínua vem a agregar melhorias no desenvolvimento do software.

No modelo cascata de desenvolvimento a integração do software é o último estágio antes da implantação da release em produção. Porém, nesta fase surgiam alguns problemas durante a integração dos projetos: diversos bugs na integração, atrasos na entrega, incompatibilidades, riscos subestimados.

Para tornar possível a redução destes problemas, minimizar riscos e custos, melhorar a velocidade e qualidade do software, pode-se contar com a Integração Contínua.

Em um projeto na qual a equipe de desenvolvedores realiza diversas modificações, um simples erro de digitação pode comprometer a integridade do software. Para assegurar a integridade e coesão, a Integração Contínua visa a segurança dos dados no código do projeto mesmo após diversas alterações.

Trabalhando no setor de Tecnologia da Informação (TI) de uma empresa de grande porte do ramo varejista tive contato direto com a implantação do processo de Integração Contínua. Nesta empresa o número de projetos e suas dependências estavam aumentando e tornando-se difíceis de serem gerenciadas.

Um fator a destacar seria a dificuldade nesse gerenciamento, uma vez que, cada desenvolvedor alterava o código e, sem intenção, tornava instável a compilação de outro projeto, pois este desenvolvedor não tinha em sua cópia de

trabalho os demais projetos da mesma linha de desenvolvimento. Em alguns casos o desenvolvedor também tornava instável a compilação do projeto por esquecer-se de executar os testes unitários antes de realizar a submissão ao repositório.

A empresa possui fábricas externas (empresas prestadoras de serviços) além dos desenvolvedores, o que torna o processo de desenvolvimento ainda mais vulnerável devido a quantidade de alterações e pessoas envolvidas.

As fábricas não percebem o impacto de suas alterações sobre os demais projetos, devido a não possuírem acesso aos outros projetos por questões de segurança e redução de custos. O trabalho do desenvolvedor se tornaria mais complexo e custoso caso tivesse em seu *workspace* todos os projetos.

A partir dos problemas apresentados na integração do código adotou-se a Integração Contínua visando a qualificação do trabalho. A implantação deste processo pela empresa será utilizada para estudo nesta monografia.

A pesquisa foi realizada a partir de experiências e análise de resultados obtidos através da implantação da Integração Contínua, e teve como objetivo a identificação dos resultados na melhoria da qualidade do software quando aplicado o processo de Integração Contínua.

O trabalho está estruturado em quatro capítulos. No primeiro, consta a introdução, em que será apresentado o tema central da pesquisa. O segundo capítulo relata os conceitos de *eXtreme Programming* e Integração contínua. No capítulo três será apresentado o estudo de caso e o processo de implantação na empresa em qual trabalho. No decorrer do quarto capítulo, destaco a análise dos resultados e os benefícios adquiridos com a implantação deste processo. Ao finalizar este estudo, no último capítulo constam algumas conclusões e reflexões sobre o trabalho realizado.

2 Desenvolvimento

2.1 Integração dos Projetos

Para Humble (2010) sistemas de controle de versão, também conhecido como gerenciamento do código fonte, são mecanismos para manter múltiplas versões dos arquivos. Desta maneira, após o arquivo ser alterado pode-se resgatar a versão anterior. Esta forma de trabalho promove o desenvolvimento de software de forma colaborativa.

Para possibilitar este desenvolvimento, todos os desenvolvedores trabalham em sua cópia de trabalho local, o que possibilita o controle de suas alterações. Esta forma de trabalho facilita o progresso da tarefa do desenvolvedor. Os desenvolvedores efetuam alterações em projetos independentes e devem integrar o código juntamente com outros projetos, e todo o sistema deve ter uma compilação confiável. Este modo de trabalhar ajuda a garantir que o código do sistema compile sem erros em sua cópia de trabalho local.

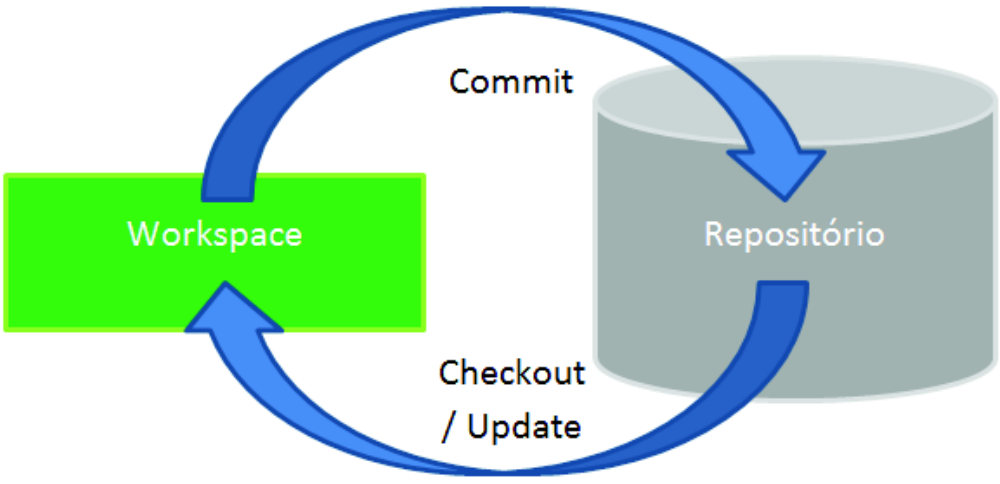


Figura 1 – Sincronização da cópia de trabalho com o repositório.

A figura 1 apresenta o repositório do projeto e o *workspace* (cópia de trabalho) do desenvolvedor. A funcionalidade do repositório é armazenar todas as alterações realizadas no projeto. A ação de submeter estas alterações para o repositório é denominada *commit* e a sincronização com o repositório é definida como *update*.

Na fase de desenvolvimento do projeto as alterações são submetidas ao repositório diversas vezes. Desta forma, não é possível ter cem por cento de certeza que todo o sistema irá ser construído após várias alterações e ser integrado com o *trunk* sem problemas. Neste caso, podemos citar o seguinte exemplo: o desenvolvedor “A” pode fazer uma alteração em paralelo (no mesmo momento) que seja incompatível com a alteração que o desenvolvedor “B” esteja realizando. A comunicação pode ajudar nessas situações, mas o problema ainda permanece. Quando o *commit* é realizado no repositório com as alterações no código, pode-se, apesar de suas melhores intenções, introduzir erros de compilação.

Sendo assim, existe a necessidade de certificar-se que o código esteja sempre integro para ser construído de forma confiável. Para resolver este problema é necessário compilar todos os projetos. Desta maneira, qualquer classe que apresente alguma inconsistência irá falhar durante a compilação. A figura 2 demonstra que a integração pode ser complicada, semelhante ao jogo de quebra-cabeça.

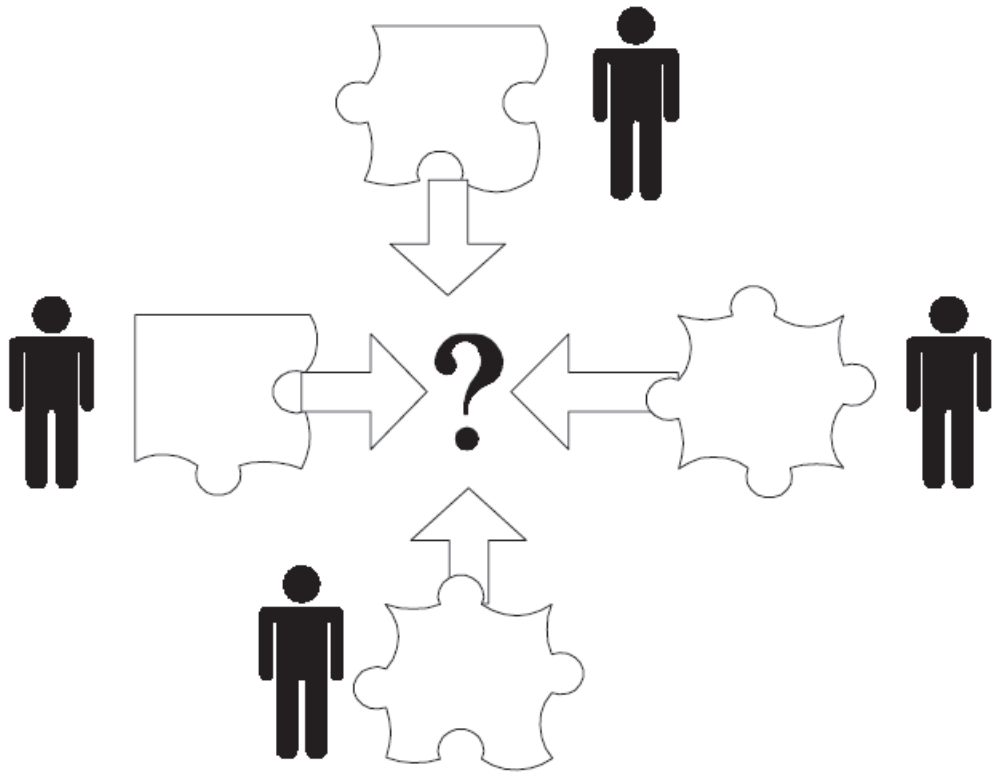


Figura 2 – Integrar pode ser difícil.

A compilação completa e centralizada pode resolver alguns destes problemas, mas a compilação centralizada trabalha com submissão de código, de modo que o dano já está inserido no repositório. Portanto, é necessário ter um processo que seja executado quando alguma modificação é realizada no código do repositório. Neste sentido, a Integração Contínua (IC) é o processo que permite ter garantias e maior confiança que a compilação do projeto está estável e será realizada com sucesso.

A complexidade do projeto aumenta durante sua evolução, existindo assim maior necessidade de integrar e garantir que os componentes do software funcionem juntos. Não sendo mais necessário esperar até o final do projeto para integrar. Este processo causava frequentemente atraso na entrega dos projetos. A Integração Contínua permite apresentar os riscos de forma mais rápida durante os pequenos incrementos do projeto.

2.2 Extreme Programming (XP)

Em um projeto onde vários desenvolvedores trabalham juntos, é necessário estabelecer um processo seguro para a integração das contribuições de cada desenvolvedor. Segundo Beck (2004), no *eXtreme Programming* (XP), utiliza-se a prática de Integração Contínua, na qual os desenvolvedores integram o que produzem com o repositório diversas vezes ao dia. Esta prática é realizada de forma segura, seguindo um roteiro que é automatizado. A integração busca assegurar que o repositório permaneça sempre consistente, possibilitando que qualquer desenvolvedor possa obter todo o código do projeto, a qualquer momento, sendo então capaz de compilá-lo e executar todos os testes com sucesso.

Desde 1998, Booch já realizava estudos sobre a Integração Contínua. O referido autor relatou que o uso de Integração Contínua tinha como objetivo criar oportunidades para reconhecer cedo os riscos e fazer as correções incrementais, sem desestabilizar todo o esforço de desenvolvimento.

De acordo com Duvall (2007), foi com o avanço do XP e outras metodologias ágeis, e com a recomendada prática de IC, que as pessoas começaram a tomar conhecimento do conceito de não somente realizar compilações diárias, mas contínuas.

O que é XP?

Para Beck (2004), eXtreme Programming é uma metodologia leve para times de tamanho pequeno a médio, que desenvolvem software com requisitos vagos e que se modificam rapidamente. A XP é baseada em valores da simplicidade, comunicação, coragem e feedback. Esta metodologia busca manter a equipe unida com práticas simples, permite que a equipe perceba o andamento do projeto e sintonize as práticas para a sua situação desejada.

2.2.1 Princípios da XP

Segundo Beck (2004), a *Extreme Programming* leva princípios e práticas do senso comum a níveis extremos:

- Simplicidade: busca concentrar as forças da equipe para entregar o que é claramente necessário, nada além disto. Isto busca priorizar e maximizar o que é desenvolvido até à data da entrega.

- Comunicação: fator importante para a compreensão de um assunto, quanto mais facilitada é a comunicação, maiores serão as chances de evitar problemas de ambiguidades e entendimento distorcidos.

- Feedback: busca reduzir o tempo entre o desenvolvimento de uma funcionalidade até a percepção do resultado pelo usuário. Quanto menor for este tempo, mais fácil será de resolver o problema e menor será o prejuízo causado.

- Respeito: valor básico, onde todos da equipe se permitam ouvir, compreender e respeitar a opinião do colega.

- Coragem: a única certeza de um projeto de software é que serão realizadas diversas mudanças. Sendo assim, deve-se ter coragem para aceita-las e confiar nos mecanismos de proteção: programação em par, desenvolvimento orientado a testes e Integração Contínua.

2.2.2 Boas Práticas da XP

Beck (2004) afirma que a XP possui boas práticas que auxiliam no desenvolvimento do projeto:

- Cliente sempre disponível: presença constante do cliente para resolução das dúvidas, alterações e prioridades do escopo. Com o objetivo de manter o projeto ativo e dinâmico.

- Uso de metáforas: permitir a comunicação facilitada entre a equipe. Estabelecer metáforas que sejam de conhecimento comum e fácil entendimento de todos.

- Planejando o jogo: estabelecer uma estratégia de como será realizado o desenvolvimento do projeto. Qual será o tempo de cada iteração e o que será entregue.

- Pequenas entregas: conforme as iterações forem concluídas o cliente pode validar o produto e a qualidade entregue. Permitindo a equipe identificar se o projeto está evoluindo corretamente. Estas pequenas entregas reduzem o custo das alterações necessárias para alinhar com os requisitos do usuário.

- Testes de aceitação: definem critérios para identificar se o projeto entregue preenche os requisitos do usuário.

- Primeiro os testes: programar os testes unitários e após codificar o projeto em si. Isto garante a redução de erros de programação e aumenta a padronização do código produzido.

- Integração Contínua: integrar os módulos várias vezes por dia e executar os testes unitários. Qualquer problema de compilação é corrigido no mesmo momento, evitando erros após a conclusão do projeto.

- Projeto simples: escrita do código de forma clara e padronizada, facilitando a compreensão e manutenção pela equipe.

- Refatoração: permitir a constante melhoria do código. Quando possível, procurar sempre reutilizar componentes e tornar o código genérico.

- Programação em par: permite que o conhecimento do projeto seja compartilhado. Enquanto um programador codifica o desenvolvimento seu colega supervisiona, tentando perceber erros no código, erros de lógica, e padronização no que é codificado.

- Rodízio de duplas: trocar as duplas permite que todos os integrantes sigam o mesmo padrão de codificação.

- Propriedade coletiva: com a programação em par e rodízio de duplas permite-se que todos tenham conhecimento dos projetos, facilitando a manutenção do projeto por qualquer integrante da equipe.

- Padronização do código: estabelecer um padrão de codificação para todos os membros da equipe possuam a mesma visão do código.

- Evitar jornadas de horas extras: sempre que possível, a equipe deve evitar a sobrecarga de trabalho. Assim a equipe reduz o desgaste e tem boas condições de trabalho.

2.3 Integração Contínua

Desta forma a Integração Contínua é considerada uma das boas práticas da XP e é a chave para o sucesso do projeto de software. O processo de integração desempenha uma parte importante dos projetos de softwares, porque consiste na construção completa do sistema e execução dos testes automatizados. Segundo Fowler (2006):

> Integração contínua é uma prática onde que os desenvolvedores devem integrar seu trabalho frequentemente, no mínimo uma vez ao dia. Cada integração é verificada por uma compilação automatizada para detectar erros de integração tão rapidamente quando possível. Muitas equipes acreditam que esta abordagem reduz significativamente os problemas de integração e permite que uma equipe desenvolva software coeso mais rapidamente.

2.3.1 Valores da IC

Germano (2008) ressalta que a Integração Contínua quando é aplicada no processo de desenvolvimento, observa-se a presença dos seguintes valores:

- Redução dos riscos;

- Redução nos processos repetitivos que são executados de forma manual;

- Geração do software em qualquer momento e implantado em qualquer lugar;

- Permitir melhor visibilidade do projeto;
- Estabelecer maior confiança na equipe de desenvolvimento ao entregar o software;

2.3.2 Práticas da IC

Fowler (2006) apresenta algumas práticas que tornam a Integração Contínua eficaz, as quais destacam-se:

- Manter um único repositório de fontes: utilizar um repositório de arquivos para armazenar o histórico de mudanças. Este repositório irá permitir o compartilhamento de informações, rastreamento das mudanças ao longo do tempo, possibilitando também que os desenvolvedores sincronizem seu *workspace* com o repositório.
- Automatizar a compilação: permite através da execução de um script a compilação dos fontes, relatórios, dependências e execução dos testes unitários.
- Construir testes para verificar a compilação: para ter certeza que o código desenvolvido está correto pode-se aplicar a técnica de *Test Driven Development* (TDD), desta maneira grande parte do código é executado ao rodar a suíte de testes. Caso esta suíte de testes falhe, isto indicará a equipe que houve problema na compilação do sistema.
- Submeter as alterações na linha principal diariamente: o conflito de alterações ocorre quando o arquivo do *workspace* é alterado, e não está submetido no repositório. Neste momento, o mesmo arquivo foi

alterado por outro desenvolvedor, e ao sincronizar é apresentado o conflito deste arquivo, que deve ser resolvido de forma manual. Sendo assim, a integração permite divulgar aos demais desenvolvedores quais alterações foram realizadas. No momento que um conflito ocorrer este poderá ser facilmente identificado e corrigido. Quanto mais frequentes as submissões forem executadas para o repositório, menores serão os conflitos. Para isto ocorrer, a equipe deve estar habituada a submeter suas alterações no repositório diariamente.

- Executar automaticamente a compilação da linha principal no servidor de integração: a equipe submete suas alterações diariamente e após é disparado automaticamente a compilação e execução dos testes. Desta maneira é possível obter um *feedback* do estado atual da compilação dos projetos.

- Manter a compilação rápida: a XP orienta que o tempo para compilação deve ser no máximo 10 minutos. A compilação não deve ultrapassar muito este tempo, pois perde-se o *feedback* rápido das alterações realizadas e também torna a equipe lenta.

- Testar a aplicação em um servidor que seja clone de produção: os testes do sistema são utilizados para eliminar qualquer problema que o sistema apresente em produção. Caso o ambiente seja diferente, esta diferença será um risco que talvez não ocorra em produção. Desta forma, o ambiente de testes deve ser o mais próximo com o ambiente de produção: versão de banco, bibliotecas, sistema operacional, etc.

- Facilitar que todos consigam buscar a ultima versão executável: permitir que qualquer integrante da equipe consiga obter o executável mais recente para realizar testes exploratórios, demonstrações ou para identificar as mudanças realizadas.

- Permitir que todos visualizem o que esta acontecendo: o principal ponto da Integração Contínua é a comunicação, permitindo que todos integrantes visualizem o estado do sistema e as mudanças realizadas. Muitas equipes ligam um monitor ao servidor de Integração Contínua para dar mais visibilidade aos envolvidos do estado do sistema. No caso da compilação falhar é possível alertar a equipe que houve problema na submissão de algum código. Este alerta pode ser disparado por e-mail, mensagem instantânea ou sons de sirene para quem estiver acessando a ferramenta de Integração Contínua pelo navegador web.

- Executar a implantação de forma automática: a Integração Contínua será realizada em diversos ambientes. Para facilitar este trabalho é interessante desenvolver scripts para automatização e padronização. Com o desenvolvimento deste script pode-se até mesmo, realizar a implantação das versões em produção. A implantação automática ajuda a acelerar o processo e reduzir número de erros. Desta maneira, as incertezas da implantação do sistema em produção são reduzidas e permite-se gerar versões do sistema com maior frequência.

2.3.3 O Processo de IC

O desenvolvedor é responsável pela submissão no repositório e este deverá acompanhar a compilação no servidor de Integração Contínua. Caso este desenvolvedor quebre a compilação deverá corrigir o problema.

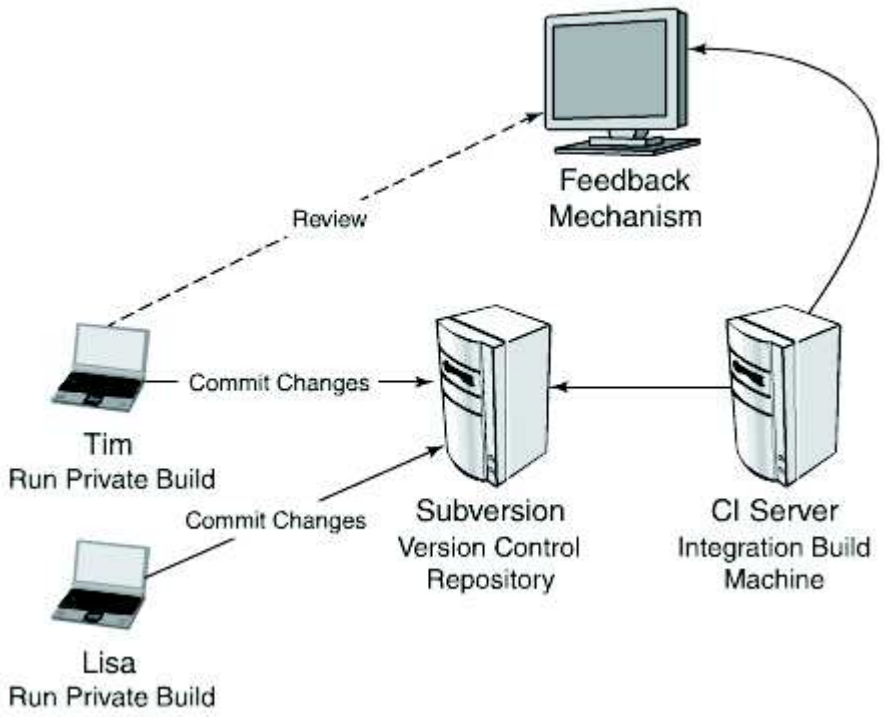


Figura 3 – Processo da Integração Contínua.

No instante em que a alteração é submetida, o servidor de Integração Contínua tem a função de monitorar este repositório dos projetos e disparar a compilação. Se esta tarefa falhar, o desenvolvedor que realizou a codificação é avisado por e-mail ou mensagem instantânea. A chave principal da compilação diária é identificar as falhas e corrigi-las imediatamente. A Integração Contínua permite desta maneira que a equipe trabalhe numa linha (*branch*) evolutiva e estável.

2.3.4 Fases da IC

A implantação da Integração Contínua é um processo realizado em diversas fases. Em cada fase as melhorias são realizadas de forma incremental. A evolução deste processo está relacionada com a configuração da infraestrutura dos servidores, e também com as evoluções ligadas a cultura da equipe de desenvolvimento em si.

Segundo Smart (2011), existem algumas fases necessárias para introduzir a Integração Contínua, sendo estas:

- Fase 1 - Não existe servidor para construção dos pacotes: no inicio a equipe não possui um servidor central para construir a aplicação. Manualmente o desenvolvedor executa a tarefa. O código pode estar armazenado no servidor central, mas os desenvolvedores não necessariamente submetem suas alterações com certa frequência;

- Fase 2 - Compilações noturnas: a equipe possui servidor central e automatiza as compilações através de agendamento com certa frequência. A tarefa simplesmente compila o código e ainda não possui testes unitários confiáveis.

- Fase 3 - Compilações noturnas e alguns testes automatizados: nesta fase a equipe de fato começa a utilizar a Integração Contínua e considerar mais sério a automatização dos testes. O servidor de integração está configurado para executar a compilação da aplicação sempre que o código for alterado no sistema de controle de versão. Como funcionalidade adicional o servidor também alerta a equipe para que de forma proativa consiga corrigir o problema rapidamente.

- Fase 4 - Métricas adicionadas: automatizar a analise da qualidade de código e as métricas de cobertura de código ajudam a avaliar sua relevância e efetividade dos testes. A qualidade do código compilado também pode gerar automaticamente a documentação da aplicação. Isto resulta para a equipe o alto padrão de qualidade de código.

- Fase 5 - Tornando os testes mais importantes: os benefícios da Integração Contínua estão intimamente relacionados a prática de testes. Práticas como *Test-Drive Development* são mais praticadas, gerando maior confiança nos resultados das compilações automáticas.

- Fase 6 - Automatização dos testes de aceitação e implantação automatizada: este nível de maturidade ocorre quando a equipe trabalha com testes de aceitação automatizados. Possibilita de forma clara a identificação de quais as funcionalidades que foram implementadas e o que falta ser codificado do projeto. Também nesta fase ocorre a instalação automática do sistema no ambiente de testes para validação das funcionalidades.

- Fase 7 - Implantação contínua: automatização dos testes unitários, integração e aceitação resultam maior certeza do código alterado. Isto possibilita para que a equipe realize a implantação automática ocorra direto em produção.

3 Estudo de Caso

Trabalho no setor de TI de uma empresa, de grande porte do ramo varejista, onde é realizado o desenvolvimento dos projetos pelo setor de TI da companhia e também pelas fábricas externas. Atualmente os projetos são repassados para três fábricas de software localizadas em diferentes cidades do estado. Todas as fábricas realizam diversas alterações nos projetos, tornando difícil mantê-los íntegros e coesos.

Desta forma, a Integração Contínua foi implantada na empresa para atender a esta dificuldade, pois o número de projetos e suas dependências estavam aumentando e tornando-se complexas de serem gerenciadas.

Estas dificuldades acabavam gerando para a equipe alguns problemas, as quais destacam-se abaixo:

- Envio de e-mail solicitando implantação de release: quando a equipe de testes necessitava validar uma release do sistema era sempre necessário o envio de e-mail solicitando a geração e implantação da release. Sendo assim, esta solicitação entrava na fila das tarefas do responsável para execução desta atividade;

- Processo manual da geração da release: o processo de geração das releases era executado manualmente, o que permitia a ocorrência de erros durante esta execução;

- Disponibilidade de uma pessoa para geração do sistema: existia apenas um responsável pela geração e empacotamento do sistema, quando esta pessoa se ausentava do trabalho, isto resultava no atraso da liberação do sistema para a equipe de testes;

- Poucas releases eram geradas: quando a geração era executada de forma manual isto resultava em poucas releases liberadas para testes, e algumas vezes a equipe de testes ficava parada, aguardando a implantação da release no ambiente de testes;

- Falha na compilação: quando o responsável realizava *checkout* dos projetos e identificava falhas de compilação, existia a necessidade de conversar com o líder do projeto para que estas falhas fossem corrigidas, e após este ajuste seria possível continuar a geração e empacotamento do sistema;

- Erros de empacotamento: estes erros eram ocasionados pelo responsável da geração das *releases*. Por ser um processo manual e passível de erros, alguns problemas acabavam ocorrendo. Por engano, o responsável realizava a geração e empacotamento do *branch* incorreto, que não continha as últimas correções do projeto;

- Falha na instalação da release: após o *deploy* do sistema gerado no ambiente de testes, algumas vezes uma nova funcionalidade necessitava a adição de alguma biblioteca no servidor de aplicação, para que o sistema funcionasse corretamente. Porém, como este passo era executado de forma manual, muitas vezes por esquecimento, não era executado. Outro erro comum que ocorria nas

instalações é que o responsável pela execução não realizava o acompanhamento dos logs do servidor de aplicação após sua inicialização. Desta maneira, a release era liberada para testes com erros críticos, onde nem mesmo o *login* do sistema era executado com sucesso;

- Inexistência da análise da qualidade de código: não existia a análise da qualidade dos projetos assim estes apresentavam baixa qualidade e resultavam na ocorrência de erros básicos (como por exemplo: erro *NullPointerException*) no ambiente de produção;

- Falta da verificação das pendências do código: alguns projetos possuem relação e dependência, assim esta compilação deve seguir uma ordem de execução.

3.1 Implantação do Processo

Há cerca de três anos iniciou-se o processo de IC, pois havia a necessidade de gerar versões do sistema de forma rápida e eficiente. Abaixo descrevo o processo realizado anteriormente a implantação da IC e após relato as mudanças executadas até o presente momento.

3.1.1 Processo Antigo

O processo de geração do sistema era realizado antigamente de forma manual, o que resultava em diversos problemas mesmo antes dos testes iniciarem de fato. Na figura 4, observam-se quais as etapas eram realizadas pelo desenvolvedor.

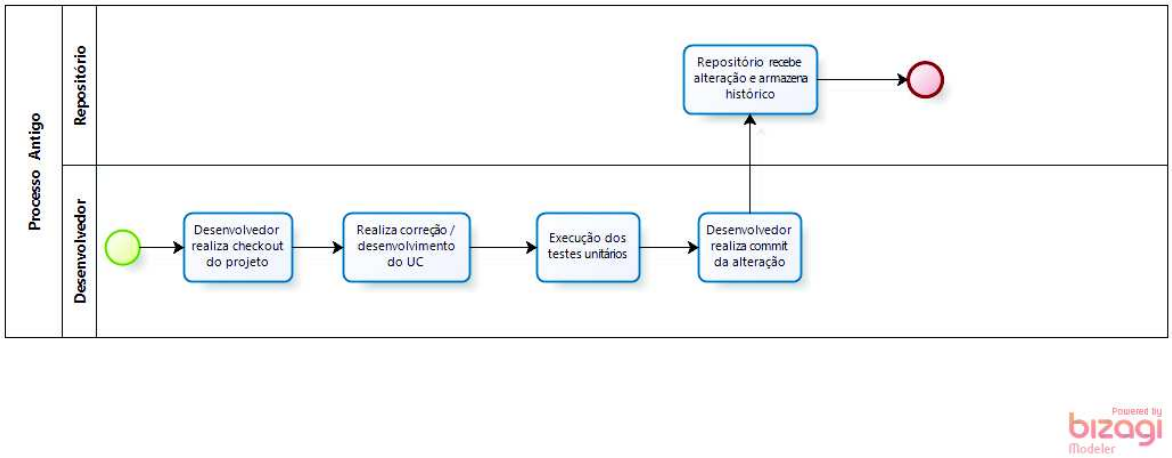


Figura 4 – Processo antigo.

Para resolver um problema ou codificar uma melhoria no código, o desenvolvedor realizava as seguintes etapas:

a) Através da ferramenta Eclipse, efetuava *checkout* do projeto desejado;
b) Alterava e testava o código em seu *workspace*;
c) Executava os testes unitários;
d) Caso os testes unitários tinham sua execução sem erros, realizava-se então o *commit* para enviar as alterações ao repositório.

A responsabilidade de manter a compilação do projeto e execução dos testes unitários sempre funcionando ficava dependente das tarefas executadas pelo desenvolvedor. O principal problema era ocasionado quando o desenvolvedor

alterava o código e, por esquecimento, não alterava os projetos relacionados ou os testes unitários. Este erro era identificado após algum tempo por outro desenvolvedor que estivesse trabalhando no mesmo projeto. Este processo tornava instável o código fonte armazenado no repositório, o que gerava diversos problemas para a equipe.

As falhas também eram encontradas durante a execução da geração e empacotamento do sistema. Como antes da implantação do processo não existia métricas dos problemas encontrados, o gráfico 1 foi gerado com base na experiência dos colegas de trabalho.

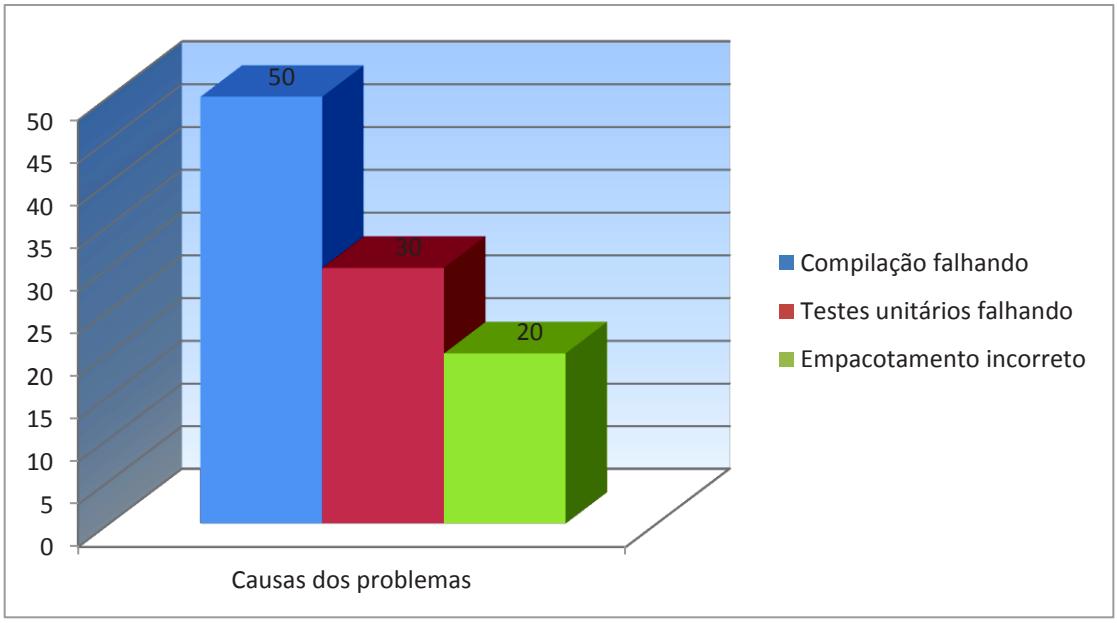


Gráfico 1 – Causa dos problemas

No gráfico 1 é possível visualizar que a maioria dos problemas eram causados por falhas de compilação. Seja falha na compilação do projeto em si, como falhas na execução dos testes. O restante dos problemas eram causados pela pessoa responsável pela geração do sistema. Onde era realizado o *checkout* do *branch* incorreto para geração e empacotamento do sistema.

Devido a estes inúmeros problemas, a equipe de testes adicionou uma fase chamada de "*quicktest*", que era executada antes dos testes. Esta atividade procurava validar se a versão implantada no servidor era a mesma que foi solicitada e também verificava se o sistema estava integro. Este teste era executado de forma exploratório procurando encontrar erros de gravidade alta que impediam que os testes iniciassem. Este "*quicktest*" tinha o objetivo de encontrar problemas como: erros no login da aplicação e erros ao abrir as telas do sistema.

3.1.2 Processo Atual

Para solucionar os problemas identificados foi instalada a ferramenta Jenkins com o objetivo de reduzir o trabalho manual, automatizar a compilação e empacotamento dos projetos. Juntamente com a instalação desta ferramenta houve a necessidade de padronização dos projetos, permitindo desta maneira que a execução de um script conseguisse compilar e empacotar todos os projetos. Nesta padronização todos os projetos foram alterados para que possuíssem a mesma estrutura de pastas e assim utilizassem a mesma forma de compilação das classes, relatórios e empacotamento.

No inicio a ferramenta foi configurada apenas para compilar os projetos da linha do *trunk*. Esta primeira tentativa de utilizar a ferramenta falhou, pois não era acessada pelos desenvolvedores e, por esse motivo, apresentava várias falhas na compilação e acabou ficando sem utilização por um período.

Após alguns meses, configurou-se a ferramenta para realizar a Integração Contínua do *branch* de suporte à produção. Esta configuração da ferramenta

realizava a compilação do código, execução dos testes unitários e análise da qualidade do código. Porém, a execução destes três passos acabava sendo um processo que demorava muito para ser concluído. Para a equipe a maior necessidade era ter o *feedback* na qual identificava se o projeto estava compilando e executando os testes unitários corretamente. Conforme boas práticas do XP este *feedback* deve ser rápido, entre dez a quinze minutos. Desta forma, a análise da qualidade de código foi configurada para ser executada automaticamente pela ferramenta durante a madrugada.

Com esta configuração estabelecida, tornou-se necessário a orientação à equipe para que todos realizassem o acompanhamento do estado das compilações. Este *branch* de suporte a produção era utilizado para correções dos erros encontrados em produção, assim, iniciou-se o trabalho de integração e acompanhamento das compilações dos projetos. Logo depois, houve também a necessidade de configurar a ferramenta para enviar alertas por e-mail aos desenvolvedores, caso o processo de compilação falhasse.

Esta implantação da ferramenta Jenkins no ambiente de desenvolvimento possui para o processo um papel muito importante. Foram necessários alguns dias de trabalho para a instalação e configuração dos projetos, porém, o retorno obtido foi satisfatório.

No processo atual, as tarefas que são executadas pelo desenvolvedor são disparadas novamente pela ferramenta, com o objetivo de encontrar falhas na compilação do projeto e execução dos testes unitários. Na figura 5, as tarefas realizadas pela ferramenta desde a submissão do desenvolvedor até a publicação da nova versão.

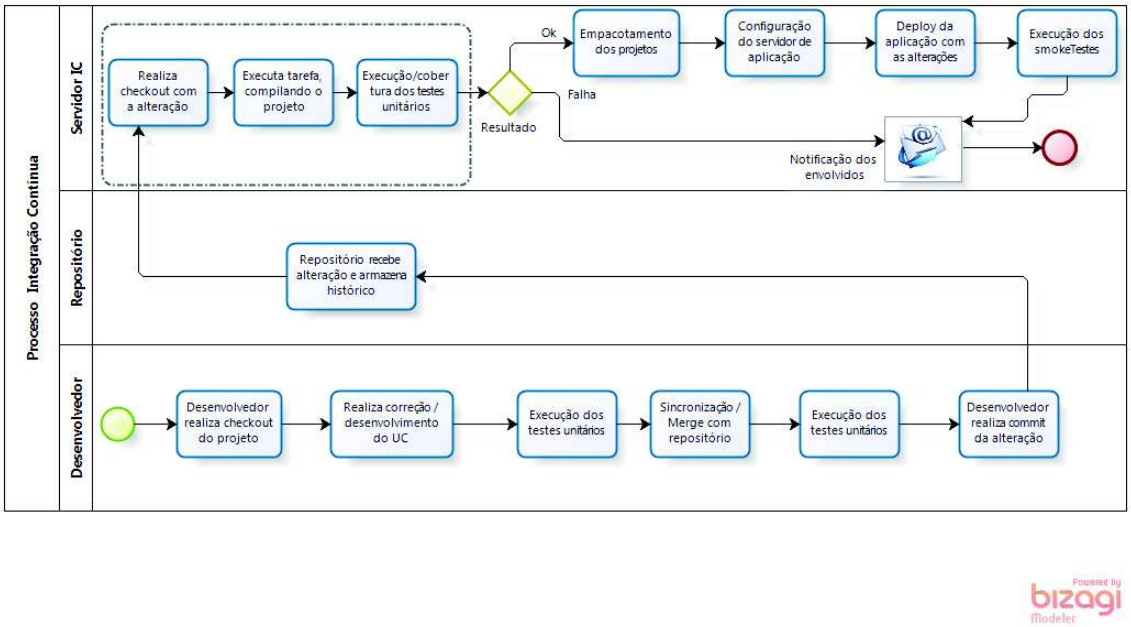


Figura 5 – Processo atual: integração dos projetos.

Pode-se observar neste fluxo que o servidor de Integração Contínua identifica as alterações realizadas no repositório e executa algumas tarefas para certificar-se que a compilação está correta. Estas tarefas são executadas na seguinte ordem:

a) Sincronização com o repositório, para isto realiza-se o *checkout* do projeto;
b) Execução do script ant para compilação do código fonte;
c) Execução do script ant para compilação, execução e análise de cobertura dos testes unitários;
d) Caso a compilação ou a execução dos testes unitários resultem em falha, as etapas seguintes são abortadas e esta tarefa dispara um e-mail para alertar os envolvidos;
e) O empacotamento do projeto é a tarefa responsável por criar um arquivo de extensão “.jar”, que possui todos os arquivos compilados do projeto;
f) Com o projeto compilado e empacotado realiza-se então a configuração do servidor de aplicação;

g) Publicação do sistema gerado;

h) Execução dos smokeTestes;

No servidor de Integração Contínua, além da compilação dos projetos e execução dos testes unitários, realiza-se também a análise estática da qualidade de código. A figura 6 representa as etapas que são executadas para verificar a qualidade em cada projeto.

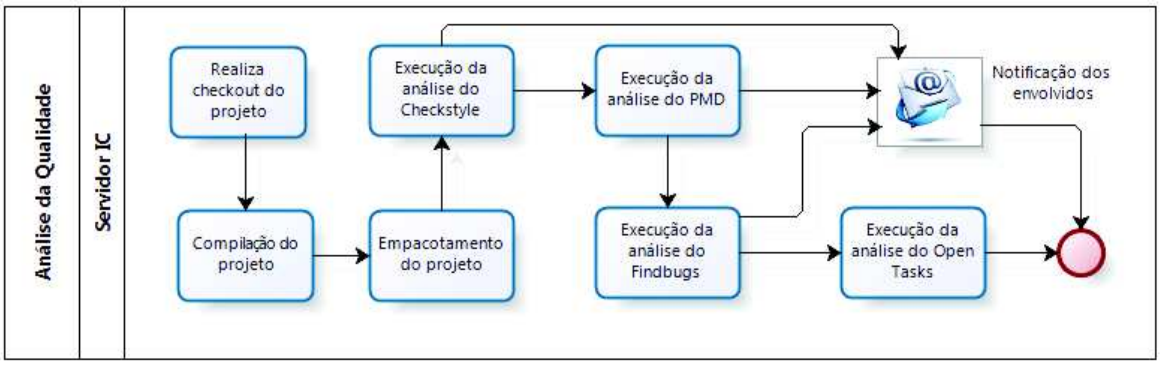


Figura 6 – Processo atual: análise da qualidade do código.

As tarefas representadas na figura 6 são executadas para identificar a qualidade dos projetos e realizar a coleta de métricas. Em cada projeto estas tarefas são executadas na seguinte ordem:

a) Sincronização com o repositório, para isto realiza-se o *checkout* do projeto;

b) Execução do script *ant* para compilação do código fonte;

c) O empacotamento do projeto;

d) Execução do script *ant* para execução do *plugin Checkstyle*;

e) Execução do script *ant* para execução do *plugin PMD*;

f) Execução do script *ant* para execução do *plugin Findbugs*;

g) Execução do script *ant* para execução do *plugin Open Tasks*;

h) Em caso de falha, notificação dos envolvidos;

Segundo SMART (2011), é importante executar esta tarefa em separado, porque muitas ferramentas de análise estática podem ser muito lentas quando executadas. Assim configurou-se para que esta analise seja executada após o expediente de trabalho, durante a madrugada.

A análise do código permite a equipe realizar o acompanhamento da qualidade de código que está sendo construído. A verificação da qualidade de código é realizada pela ferramenta através dos plugins: *Checkstyle* (padrão de formatação do código), *Findbugs* (possíveis erros na codificação), *PMD* (analise estática do código) e *Open Tasks* (tarefas em aberto, correções a serem realizadas).

Os problemas que podem ser encontrados pelos plugins de análise estática durante a codificação nos projetos:

a) Plugin *Checkstyle*: código duplicado, Javadoc incorreto, espaços em branco, complexidade ciclomática (ou complexidade condicional), etc;
b) Plugin *Findbugs*: comparação incorreta de variáveis, identificação de variáveis que devem ser estáticas, possíveis *NullPointerException*, etc;
c) Plugin *PMD*: variáveis inicializadas incorretamente, métodos com muitos parâmetros, classe com muitos atributos públicos declarados, etc;
d) Plugin *Open Tasks*: procura no código pela expressão “FIXME”, “TODO” e “@Deprecated”, onde estão respectivamente configurados com prioridade alta, média e baixa;

Na figura 7 é possível visualizar um exemplo contendo os gráficos gerados pelo plugins da ferramenta após a análise da qualidade de código.

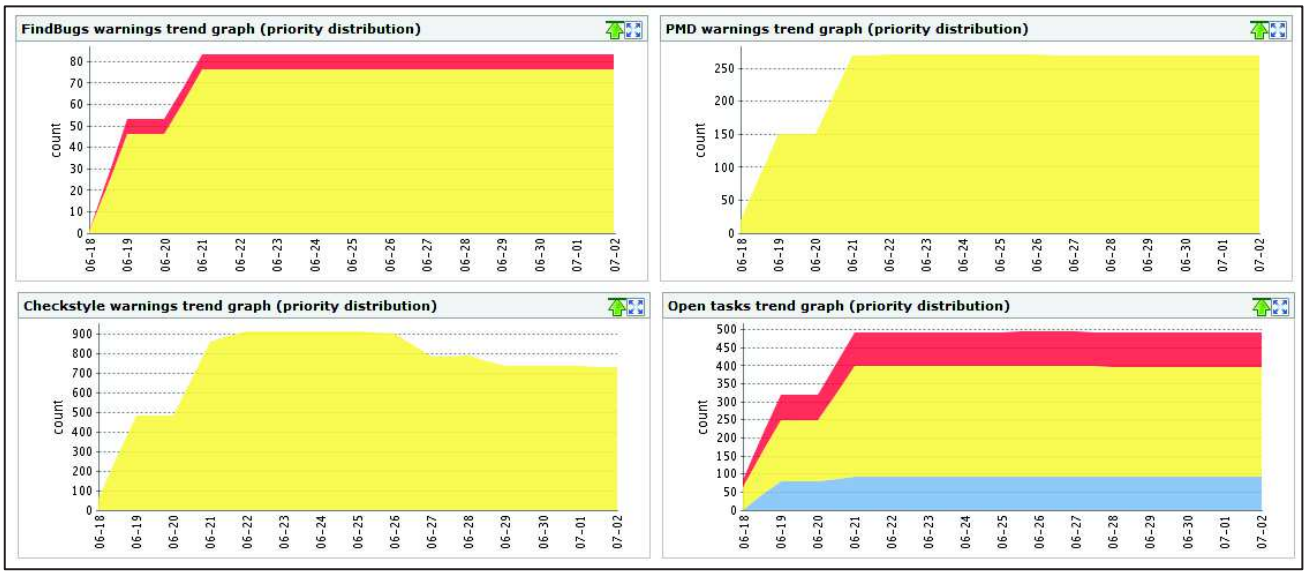


Figura 7 – Análise da qualidade de código dos projetos.

A criticidade dos erros dos gráficos da figura 7 é diferenciada pelas cores. A cor vermelha indica erro grave, amarelo é prioridade normal e a cor azul indica erro de nível baixo. Na linha horizontal os gráficos representam os builds executados diariamente e na vertical estão representados a quantidade de problemas encontrados pelos plugins.

Com a implantação da Integração Contínua, também houve a mudança no software de controle de versão do código. Houve a migração do CVS (Current Versioning System) para o SVN (SubVersion). Esta mudança foi ocasionada, pois, o SVN está ativo no mercado e possui funcionalidades importantes para controle do projeto. Como as principais funcionalidades obtidas com esta mudança foram: o *commit* atômico (o *commit* ocorre totalmente ou falha, não existe o problema de submeter um arquivo e outro não) e número da revisão (número sequencial atribuído ao *commit* realizado no repositório).

3.2 Servidor de Integração Contínua

Os motivos que influenciaram a equipe em escolher a ferramenta foram a facilidade de instalação e a grande quantidade de *plugins* disponíveis. Conforme o autor Smart, em 2010, a ferramenta Hudson havia se tornado líder na solução de Integração Contínua sendo utilizada por mais de 70% das empresas. E após a compra da Sun pela Oracle, em janeiro de 2011, a comunidade de desenvolvedores da ferramenta decidiu em alterar o nome da ferramenta para Jenkins.

A ferramenta (Jenkins) é focada em atingir dois objetivos:

- Construir e testar projetos de software continuamente. A ferramenta permite fácil implementação da Integração Contínua dos sistemas, tornando mais fácil aos desenvolvedores a integração de suas alterações do projeto e permite aos usuários obterem a construção atualizada do sistema.

- Monitorar as execuções das tarefas externas, como agendador de tarefas e e-mails, este monitoramento pode ser utilizado também para as tarefas que são executadas de forma remota. Jenkins mantem o resultado desta execução remota e torna fácil para a equipe ser notificada quando algo errado ocorrer.

Esta ferramenta possui interface web de fácil utilização. Através do navegador da web é possível acessar a ferramenta e criar as tarefas para configurar e customizar os passos necessários para que a aplicação seja construída. Em cada tarefa pode-se disparar chamadas a comando batch, como *shell script* e *ant scripts*.

Estas tarefas podem ser agendadas, ou disparadas de forma automática após qualquer submissão no repositório do projeto.

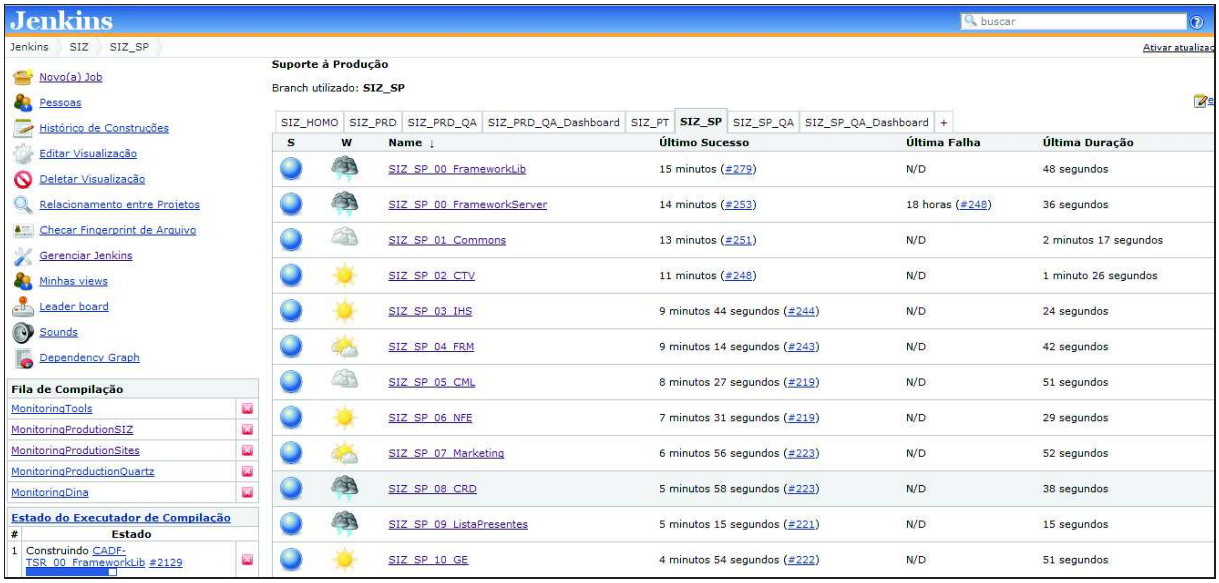


Figura 8 – Interface web da Ferramenta Jenkins.

Atualmente, as tarefas estão agendadas em determinados horários que foram estabelecidos pela equipe. Em cada execução são disparadas as seguintes atividades:

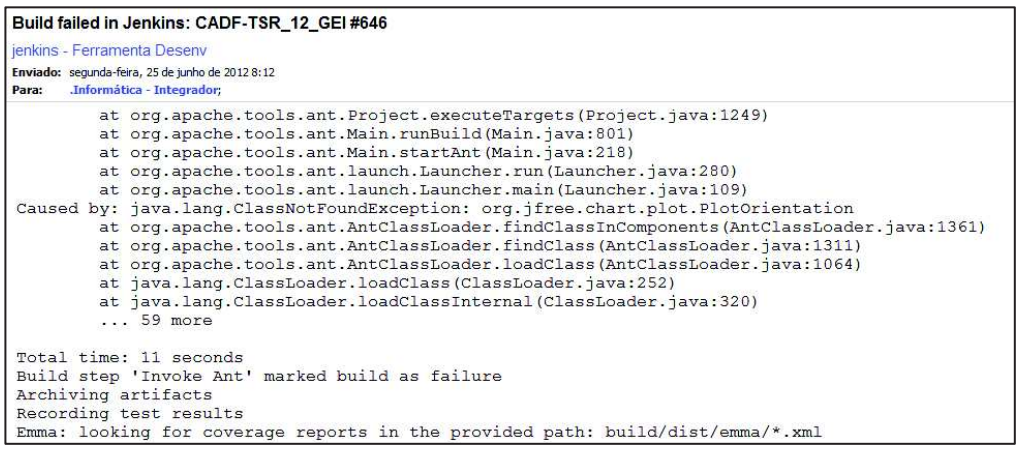
a) Sincronização com o repositório: após a tarefa ser disparada é realizada a sincronização do *workspace* do projeto com o código armazenado no repositório. Assim, é efetuado o *checkout* das correções e outras alterações submetidas no repositório.

b) Compilação do projeto: através de scripts *ant*, o código fonte (.java) é compilado e gera o arquivo *bytecode* (.class), permitindo encontrar falhas de uma alteração parcial do desenvolvedor, falhas em bibliotecas e erros de dependência.

c) Execução dos testes unitários: cada projeto possui uma suíte com os testes unitários que devem ser executados no projeto. Para todo método codificado deve existir um método de teste, que visa validar se o método retorna o resultado esperado.

d) Analise de cobertura dos testes unitários: quanto maior for o número de linhas de código que são percorridas pelos testes unitários, menores serão os problemas da aplicação. Esta atividade faz que o programador codifique a funcionalidade e escreva o teste buscando revisar o que foi desenvolvido. Esta etapa também permite que uma alteração mal realizada acabe ocasionando falhas em outros pontos do projeto, reduzindo as chances deste erro ocorrer em produção.

e) Envio de alertas por e-mail ou mensagem caso o build falhar: em caso de falha, é possível alertar por e-mail ou mensagem (*jabber*) os envolvidos no projeto. Isto permite dar visibilidade aos problemas e reduzir o tempo que o repositório permanece instável. Na figura 9, pode-se observar que o desenvolvedor se esqueceu de adicionar uma biblioteca necessária para a correta compilação do projeto.


Build failed in Jenkins: CADF-TSR_12_GEI #646

jenkins - Ferramenta Desenv

Enviado: segunda-feira, 25 de junho de 2012 8:12

Para: .Informática - Integrador;

```
        at org.apache.tools.ant.Project.executeTargets(Project.java:1249)
        at org.apache.tools.ant.Main.runBuild(Main.java:801)
        at org.apache.tools.ant.Main.startAnt(Main.java:218)
        at org.apache.tools.ant.launch.Launcher.run(Launcher.java:280)
        at org.apache.tools.ant.launch.Launcher.main(Launcher.java:109)
Caused by: java.lang.ClassNotFoundException: org.jfree.chart.plot.PlotOrientation
        at org.apache.tools.ant.AntClassLoader.findClassInComponents(AntClassLoader.java:1361)
        at org.apache.tools.ant.AntClassLoader.findClass(AntClassLoader.java:1311)
        at org.apache.tools.ant.AntClassLoader.loadClass(AntClassLoader.java:1064)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:252)
        at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:320)
        ... 59 more

Total time: 11 seconds
Build step 'Invoke Ant' marked build as failure
Archiving artifacts
Recording test results
Emma: looking for coverage reports in the provided path: build/dist/emma/*.xml
```

Figura 9 – E-mail alertando da falha na construção do projeto.

Assim que este desenvolvedor receber o e-mail indicando a falha no *build*, este deve acessar a ferramenta, entender e corrigir o problema. A correção é realizada através de um *commit* no repositório. Assim este desenvolvedor pode aguardar a próxima execução agendada ou disparar manualmente a compilação para se certificar que o *build* volte a ficar novamente estável.

Caso este desenvolvedor demore a encontrar a solução ou nenhum outro integrante da equipe esteja disponível, o próprio integrador (responsável pela integração dos projetos) realiza a correção. Isto acontece para evitar que outras fábricas de software sejam impedidas de trabalhar, pois alguns projetos possuem dependência.

f) Empacotamento dos projetos: após a compilação de todos os projetos, a aplicação é empacotada e incrementada a versão.

g) Configuração do servidor de aplicação: nesta etapa foi realizado o desenvolvimento de um projeto que contem todas as configurações do servidor de aplicação. Este passo contribuiu para a redução do tempo de configuração de cada release. Este projeto tem a função de armazenar as seguintes configurações, como: bibliotecas, conexões do banco de dados, agendamento de tarefas e parâmetros de inicialização do servidor de aplicação. Desta forma, o Jenkins executa a tarefa de configuração do servidor de aplicação utilizando este projeto, onde são aplicadas as configurações no servidor informado, e após, realiza-se a instalação da aplicação gerada.

Esta atividade também reduziu de forma significativa os problemas de bibliotecas e falhas de configuração no servidor. Atualmente o próprio

desenvolvedor adiciona neste projeto as bibliotecas e outras configurações necessárias para utilizar em seu ambiente de desenvolvimento e depois também será aplicada no ambiente de testes.

h) *Deploy* da aplicação: realiza-se a publicação da nova versão do sistema no servidor de aplicação.

Foi definido por padrão que estas etapas são executadas quatro vezes ao dia, sendo duas compilações na parte da manhã e duas durante a tarde. Para releases com maior volume de alterações no código executam-se instalações com maior frequência, ocorrendo 1 compilação a cada hora.

i) Execução dos *Smoke-Test* (teste de fumaça): esta etapa é executado os testes onde procura-se encontrar problemas após implantação do sistema no servidor de aplicação. Este teste executa algumas validações para se certificar de que o sistema está corretamente instalado e funcionando. O teste de fumaça também verifica se os serviços que a aplicação requer estão funcionando. Estes serviços são, por exemplo: servidor de FTP, comunicação com banco de dados, serviços *webservice* de empresas terceiras, existência de procedures de banco.

4 Análise de Resultados

A empresa obteve diversas melhorias com a implantação da Integração Contínua. Para avaliar os resultados das melhorias da implantação deste processo foram enviadas 10 questões para os colaboradores da empresa e também para os colaboradores localizados nas fábricas externas. Estas questões foram respondidas pelas pessoas que participam do processo de desenvolvimento e desempenham diversas funções. Os colaboradores que participaram da pesquisa exercem na empresa função de: desenvolvedor, testador, analista de testes, projetista e analista de dados. A pesquisa obteve um total de 24 participantes que responderam ao questionário na qual avaliaram o processo de Integração Contínua. O questionário enviado aos colaboradores era composto de perguntas fechadas na qual as respostas deveriam ser sinalizadas com: concordo totalmente, concordo, não sei opinar, discordo e discordo totalmente.

4.1 Melhorias Obtidas

Serão apresentados os gráficos com os resultados do questionário (Anexo I) que foram enviados para as equipes de desenvolvimento e testes.

4.1.1 Redução dos Problemas e Retrabalho.

Quando questionados sobre “A implantação do processo de Integração Contínua contribuiu para a redução de problemas (falhas de compilação) e retrabalho”, obtive as seguintes respostas:

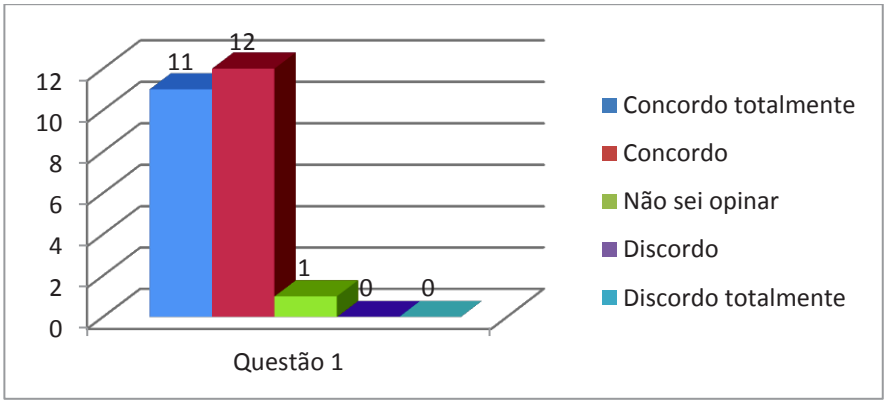


Gráfico 2 – Reduz os problemas e retrabalho

O desenvolvimento de sistemas é formado por um conjunto de atividades extremamente complexas. Nestas atividades, durante a evolução do projeto, podem ocorrer diversas mudanças, sejam estas, alterações dos requisitos, de ambientes, dos testes. Cada mudança possui determinado impacto sobre o projeto já codificado e o gráfico 2 comprova que a equipe concorda com a ideia que a Integração Contínua proporciona a redução dos problemas no projeto, tornando a codificação uma evolução previsível e diminuindo assim o retrabalho. Esta redução dos problemas evita o desperdício de tempo e possibilita que esta equipe se concentre em melhorias e inovações para o projeto.

4.1.2 Feedback Instantâneo

Abaixo segue gráfico 3 na qual podemos visualizar as respostas dos entrevistados referente ao seguinte questionamento: “A ferramenta de Integração Contínua fornece um feedback instantâneo com e-mail e mensagens jabber (Pandion)”.

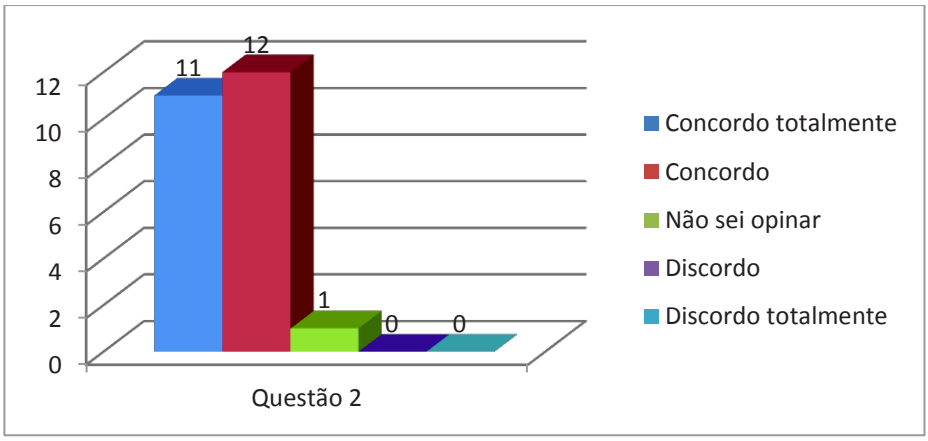


Gráfico 3 – Fornece *feedback* instantâneo.

Durante a codificação do projeto, diversas alterações são submetidas ao repositório pelo desenvolvedor, e o servidor de Integração Contínua dispara a compilação do projeto e torna os problemas visíveis para a equipe. A equipe é alertada caso ocorrer algum erro nesta execução, possibilitando que os envolvidos permaneçam no controle das alterações dos projetos, mesmo durante as mudanças grandes e complexas.

4.1.3 Equipe Mais Ágil

Referente a questão “O processo de geração e implantação automática torna o trabalho da equipe mais ágil” obtive as seguintes respostas:

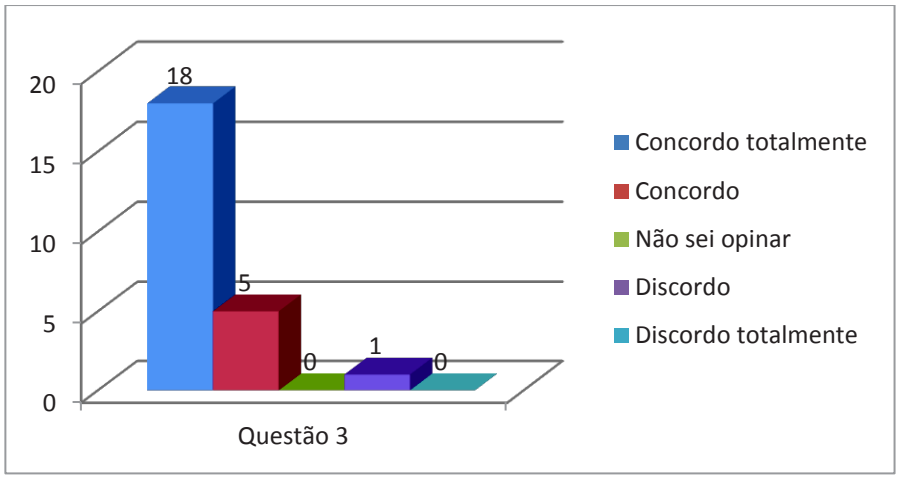


Gráfico 4 – Reduz o tempo entre os ciclos de correções.

O processo de geração automatizada torna a equipe mais dinâmica e reduz o tempo entre os ciclos de correção e testes do projeto. Isto possibilita que o projeto permaneça ativo e estável durante as diversas compilações.

Como resultado da implantação deste processo obteve-se a agilidade na geração do sistema. Atualmente em 10-15 minutos consegue-se gerar uma release e implantá-la com sucesso. Onde antes eram necessários alguns dias para conseguir implantar e liberar uma release para testes sem problemas. Com a execução desta atividade, alguns dos problemas existentes anteriormente foram solucionados: falta de bibliotecas para que a aplicação seja executada sem problemas, falta de acompanhamento na inicialização do Jboss, erro de configuração na conexão com o banco de dados.

4.1.4 Processo Gera Benefícios para Equipe

No gráfico abaixo visualizamos as respostas referente a questão “A implantação deste processo gera maior confiança, motivação e comprometimento da equipe”

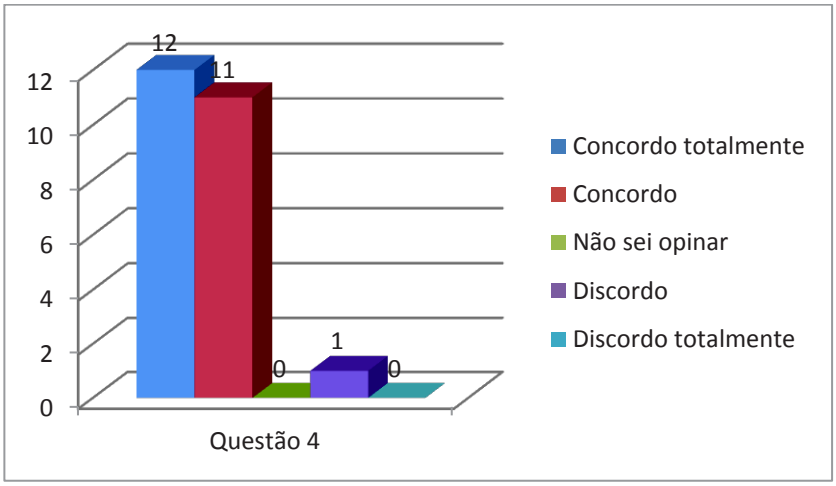


Gráfico 5 – Gera confiança, motivação e comprometimento da equipe.

A equipe obtém maior confiança e certeza do produto que está sendo desenvolvido a cada compilação gerada. Esta atividade faz com que todos os envolvidos permaneçam comprometidos com o objetivo de manter a compilação estável e permita que o projeto seja entregue com a qualidade esperada.

No instante em que a equipe mantém a cultura de sempre analisar o status da compilação, surge então um compromisso de manter a compilação estável para todos os envolvidos no desenvolvimento. Isto resulta em maior confiança e certeza do produto que está sendo desenvolvido a cada compilação executada.

4.1.5 Antecipação dos Problemas

Quando questionados sobre “A análise de código realizada pelos plugins (PMD, FindBugs, Checkstyle, Open Tasks, ...) da ferramenta antecipa os problemas que antes eram identificados somente em produção” os entrevistados responderam que:

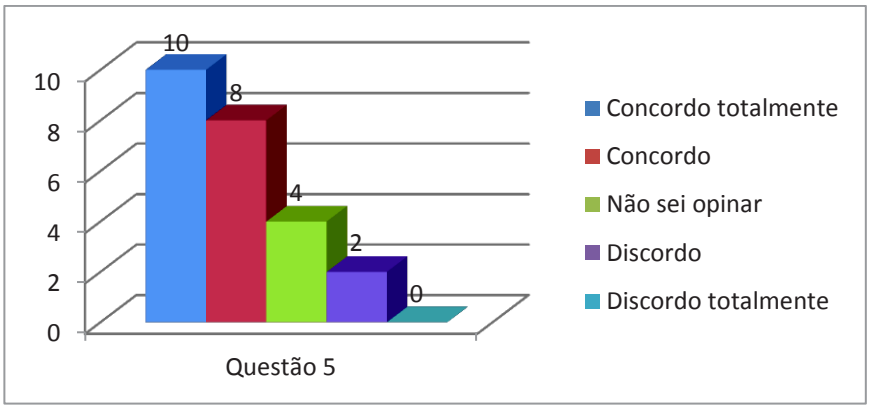


Gráfico 6 – Visualização dos problemas através da analise de código

A análise da qualidade de código possibilita que erros básicos sejam visualizados e não passem despercebidos pela equipe. Sem a execução desta atividade, os problemas do projeto são provavelmente encontrados quando este estiver sendo utilizado pelo cliente.

A execução da análise da qualidade de código torna-se muito importante, pois mantem um padrão de qualidade em todos os projetos. Para aqueles projetos complexos, que são desenvolvidos por equipes distribuídas, esta execução da análise da qualidade ganha ainda mais importância, pois, além de estabelecer um padrão de qualidade, auxilia também na antecipação dos problemas que antes eram encontrados somente no ambiente de produção.

4.1.6 Entregas mais frequentes do sistema em produção

Em relação a questão “A implantação da Integração Contínua possibilita entregas mais frequentes do sistema em produção” obtive as seguintes respostas:

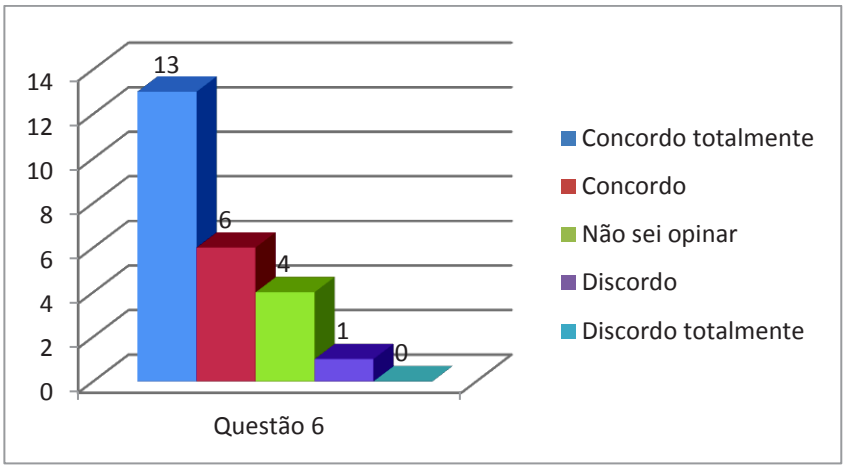


Gráfico 7 – Possibilita entregas frequentes

Os processos manuais e repetitivos, quando automatizados, geram maior qualidade do processo de desenvolvimento. Torna estável a compilação do projeto, permitindo que este projeto permaneça compilando corretamente e esteja disponível para ser implantado a qualquer momento em produção.

4.1.7 Identificação e Correção Rápida dos Problemas

O gráfico abaixo representa as respostas sobre o questionamento “No modelo cascata a integração dos projetos era realizada quando o projeto estava concluído. Agora com a implantação da Integração Contínua percebe-se que os problemas foram antecipados permitindo assim que estes sejam corrigidos rapidamente”.

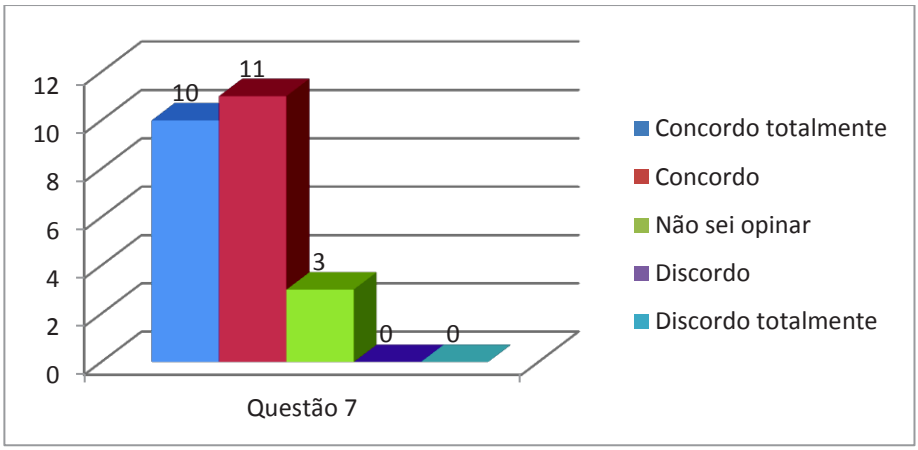


Gráfico 8 – Antecipação dos problemas

A codificação do projeto é validada a cada compilação executada, assim evita-se surpresas na entrega do projeto. Caso algum problema surgir, este é rapidamente detectado e corrigido. Antes, no modelo cascata, a integração dos projetos acontecia somente após sua conclusão e isto apresentava alguns problemas na integração e resultava no atraso da entrega.

4.1.8 Redução da Instabilidade do Projeto

Sobre a questão “A implantação da Integração Contínua reduz o tempo que o projeto permaneça instável no repositório do SVN, possibilitando que este fique sempre disponível para ser compilado e implantado” os entrevistados informaram as respostas:

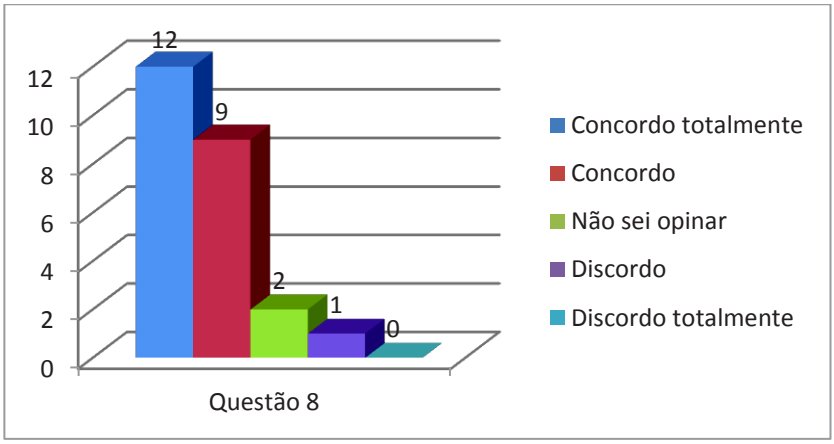


Gráfico 9 – Reduz a instabilidade do projeto

A compilação do projeto quando realizada de forma contínua permite que este permaneça estável e fique disponível para qualquer membro da equipe realize o *checkout*, compilação e execução do projeto sem problemas.

4.1.9 Segurança para a Equipe

Para a questão “Um projeto com alta cobertura de testes automatizados resulta em maior segurança para a equipe quando o código precisa ser alterado. E quando a Integração Contínua executa este teste automatizado é possível perceber efeitos colaterais não percebidos antes da implantação deste processo.” obtive as seguintes respostas:

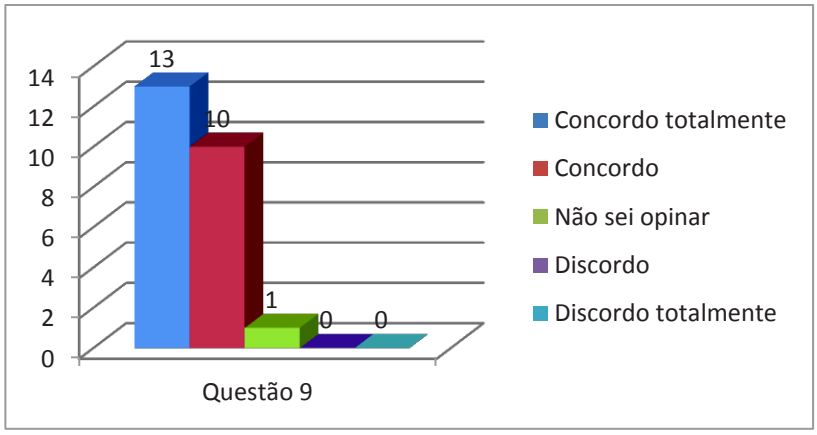


Gráfico 10 – Testes automatizados resulta em maior segurança

A execução de testes automatizados permite que os defeitos sejam capturados e eliminados de forma rápida. Quanto maior for cobertura de linhas dos testes automatizados, maior será a certeza que uma alteração crítica não tenha causado efeito colateral no projeto. Isto proporcionará maior segurança para a equipe, pois irá auxiliar na visualização dos erros causados por uma mudança, incentivando assim o *refactoring* dos projetos.

4.1.10 Facilidade da Gestão dos Projetos

O gráfico abaixo representa as respostas ao questionamento “A implantação da Integração Contínua facilita a gestão dos projetos independente do tamanho de cada projeto”.

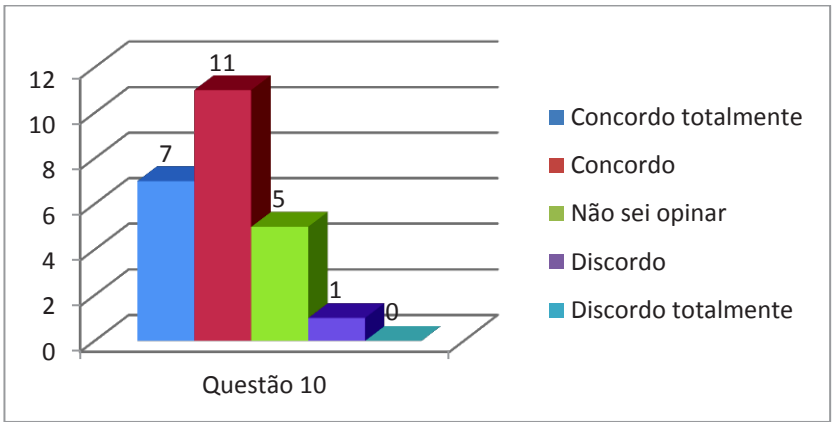


Gráfico 11 – Gestão dos projetos

A Integração Contínua resulta em maior certeza do projeto que está sendo entregue, existe uma redução nos problemas inesperados e assim é gerado um produto de melhor qualidade e estável. Estas características resultam em maior certeza no prazo de finalização do projeto.

5 Considerações Finais

O trabalho foi desenvolvido com o objetivo de analisar os problemas existentes antes da implantação da Integração Contínua e identificar melhorias obtidas após o processo implantado.

Foram enfatizados alguns conceitos e identificados problemas no processo antigo da empresa em que foi realizado o estudo de caso. Além disso, realizou-se a análise do processo e a pesquisa dos benefícios obtidos com a implantação do processo da Integração Contínua. Para tanto, foi aplicado um questionário com os colaboradores que participam dos projetos desenvolvidos, a fim de avaliar os benefícios da implantação do processo de Integração Contínua.

5.1 Melhorias Futuras

Como melhoria futura neste processo seria a alteração na configuração da ferramenta para executar a compilação dos projetos a cada *commit* efetuado. O servidor de Integração Contínua ainda não foi configurado desta maneira por falta de recursos de *hardware*. Quando esta configuração estiver habilitada irá fornecer um *feedback* mais rápido a equipe.

O processo de Integração Contínua agrega qualidade no desenvolvimento das tarefas, porém esse não é o único processo necessário para eficácia do trabalho. É importante ressaltar a necessidade de existir um processo que facilite a publicação das versões em produção, neste caso destaco o conceito *Continuous*

Delivery. Segundo Humble (2010), o conceito *Continuous Delivery* é um conjunto de práticas que tem o objetivo de entregar software mais frequentemente aos usuários.

Para automatizar a execução dos scripts de banco, a equipe realizou estudos, visando melhorias no processo, pois houveram problemas em algumas versões do sistema, na qual ocorreram esquecimentos na execução de alguns scripts. Desta forma, esta tarefa quando automatizada e padronizada irá gerar maior confiança e certeza da eficácia no trabalho da equipe.

Atualmente estamos tralhando numa tarefa para melhorar nossas estatísticas da qualidade de código (*PMD, CheckStyle, FindBugs e Open Tasks*). A partir do momento que os problemas de qualidade do código forem corrigidos será necessário criar e manter a cultura de alerta a todos os integrantes. Assim todos estariam cientes que a qualidade do código está estável e quando algum problema surgir, a equipe deverá corrigir imediatamente.

Também será necessário aumentar a quantidade dos *Smoke-Test* para gerar maior confiança quando uma versão é instalada no servidor. Esta funcionalidade irá permitir a equipe encontrar de forma mais rápida os problemas como: falha de banco, erro de permissões no servidor, erro de FTP.

A equipe tem como objetivo adquirir uma televisão para visualizar os *builds* com falha, além dos *feedbacks* recebidos por e-mails e mensagens.

A configuração de clusters da ferramenta Jenkins irá proporcionar uma resposta mais rápida para a equipe. Existe a ideia de manter dois servidores de Integração Contínua, onde um servidor seria configurado como *Master* para realizar a compilação dos projetos a cada *commit* e no *Slave* (em outra máquina virtual) seria executado a análise da qualidade do código dos projetos. Hoje a equipe possui somente uma máquina virtual para executar estas duas tarefas. Assim com a adição

deste servidor *Slave* seria possível analisar a qualidade do projeto no mesmo momento em que a compilação do projeto fosse disparada. Desta forma, a analise da qualidade do código não precisaria mais aguardar até a madrugada para que seja executado, permitindo um *feedback* mais rápido para a equipe.

5.2 Conclusões

O processo de Integração Contínua demonstra-se muito eficaz durante a evolução dos projetos. Este processo é uma forma de manter o repositório estável e também oferece maior segurança para a equipe quando existe a necessidade de realizar alterações em diversos projetos.

Através das respostas obtidas do questionário comprovou-se que a Integração Contínua oferece diversos benefícios para a equipe. Entre estas melhorias estão o *feedback* rápido e a garantia de que o projeto estará sempre disponível pra ser implantado em produção. Quando o projeto é formado por equipes distribuídas, este *feedback* torna-se ainda mais importante, pois auxilia que este projeto consiga evoluir com estabilidade, mesmo após diversas alterações.

A partir deste estudo, comprovou-se que a Integração Contínua agrega valores importantes para a evolução do projeto: garante a estabilidade do código, maior segurança para a equipe durante a realização das modificações e também proporciona um projeto de melhor qualidade.

Referências

BECK, Kent. **Extreme programing explained**. Ed. Bookman. 2004

BERCZUK , Stephen P. ; Brad Appleton. **Software Configuration management patterns**: Effective teamwork, practical integration: Ed.: Addison Wesley. 2008.

BOOCH, Grady. **Best of Booch**: Designing Strategies For Object Technology. Ed Cambridge University Press. 1998.

DUVALL, Paul M. **Continuous Integration**: Improving Software Quality and Reducing Risk. Ed. Addison Wesley. 2007.

GERMANO, Victor Hugo. **Revista Visão Ágil**; Edição 04 - 2008 – Disponível em: <http://www.visaoagil.com/static/downloads/edicoes/VA_04.pdf>
Acesso em: 16 de fev. 2012.

HUMBLE, Jez; David Farley. **Continuous Delivery**: Reliable Software Releases through Build, Test, and Deployment Automation. Ed. Addison Wesley. 2010.

SMART, John Ferguson. **Jenkins - The Definitive Guide**: Continuous Integration for the Masses. Ed. O'Reilly Media. 2011.

FOWLER, Martin. **Continuous Integration**. Disponível em: <http://martinfowler.com/articles/continuousIntegration.html> - Acesso em: 10 de fev. 2012.

Glossário

Ant - Apache Ant é uma biblioteca do Java e ferramenta executada por linha de comando;

Bytecode – é o arquivo resultante da compilação do código Java;

Branch – Ramificação do projeto, utilizado para isolar as mudanças quando necessárias;

Deploy – Publicação da aplicação no servidor de aplicação;

Jabber - Padrão aberto para a comunicação por Mensagem Instantânea;

Workspace – Pasta que contém o ambiente de desenvolvimento dos projetos;

NullPointerException – Erro que ocorre na linguagem Java, quando tenta-se utilizar um atributo ou método de um objeto que não foi inicializado;

Repositório – local de armazenamento utilizado para controlar as alterações do projeto;

Refactoring – alteração do código sem mudar seu comportamento;

Trunk - Linha principal (tronco) de desenvolvimento do repositório.

Apêndice A

Objetivo

O objetivo desta pesquisa é identificar, junto a profissionais da área de Desenvolvimento e Testes de Software, quais são as melhorias obtidas com a implantação do processo de Integração Contínua.

Avaliação Processo Integração Contínua

Estou cursando a pós-graduação na Unisinos e gostaria de contar com sua colaboração para avaliação do processo de Integração Contínua (IC).

As respostas obtidas serão utilizadas para avaliar o processo de Integração Contínua e a ferramenta Jenkins nos projetos desenvolvidos pela equipe.

1. Nome do respondente:
2. Tempo (meses) que trabalha/presta serviços para a empresa:
3. Profissão:
    a. Desenvolvedor
    b. Testador
    c. Projetista
    d. Analista de Dados
4. A implantação do processo de Integração Contínua contribuiu para a redução de problemas (falhas de compilação) e retrabalho.
    a. Concordo totalmente
    b. Concordo
    c. Não sei opinar

d. Discordo

e. Discordo totalmente

5. A ferramenta de Integração Contínua fornece um *feedback* instantâneo com e-mail e mensagens jabber (Pandion).

    a. Concordo totalmente

    b. Concordo

    c. Não sei opinar

    d. Discordo

    e. Discordo totalmente

6. O processo de geração e implantação automática torna o trabalho da equipe mais ágil.

    a. Concordo totalmente

    b. Concordo

    c. Não sei opinar

    d. Discordo

    e. Discordo totalmente

7. A implantação deste processo gera maior confiança, motivação e comprometimento da equipe.

    a. Concordo totalmente

    b. Concordo

    c. Não sei opinar

    d. Discordo

    e. Discordo totalmente

8. A análise de código realizada pelos *plugins* (*PMD, FindBugs, Checkstyle, Open Tasks, ...*) da ferramenta antecipa os problemas que antes eram identificados somente em produção.
    a. Concordo totalmente
    b. Concordo
    c. Não sei opinar
    d. Discordo
    e. Discordo totalmente
9. A implantação da Integração Contínua possibilita entregas mais frequentes do sistema em produção.
    a. Concordo totalmente
    b. Concordo
    c. Não sei opinar
    d. Discordo
    e. Discordo totalmente
10. No modelo cascata a integração dos projetos era realizada quando o projeto estava concluído. Agora com a implantação da Integração Contínua percebe-se que os problemas foram antecipados permitindo assim que estes sejam corrigidos rapidamente.
    a. Concordo totalmente
    b. Concordo
    c. Não sei opinar
    d. Discordo
    e. Discordo totalmente

11. A implantação da Integração Contínua reduz o tempo que o projeto permaneça instável no repositório do SVN, possibilitando que este fique sempre disponível para ser compilado e implantado.
    a. Concordo totalmente
    b. Concordo
    c. Não sei opinar
    d. Discordo
    e. Discordo totalmente
12. Um projeto com alta cobertura de testes automatizados resulta em maior segurança para a equipe quando o código precisa ser alterado. E quando a Integração Contínua executa este teste automatizado é possível perceber efeitos colaterais não percebidos antes da implantação deste processo.
    a. Concordo totalmente
    b. Concordo
    c. Não sei opinar
    d. Discordo
    e. Discordo totalmente
13. A implantação da Integração Contínua facilita a gestão dos projetos independente do tamanho de cada projeto.
    a. Concordo totalmente
    b. Concordo
    c. Não sei opinar
    d. Discordo
    e. Discordo totalmente