INSTITUTO DE INFORMÁTICA
UFG

ENGENHARIA DE SOFTWARE

Atividade: Dia de Trabalho

Tarefa: Integração Contínua

Versão 0.1

Histórico da Revisão

Data	Versão	Descrição	Autor
26/12/2012	0.1	Elaboração Inicial	Emerson José Porfírio
26/12/2012	0.1	Diagrama da tarefa	Emerson José Porfírio

Sumário

Tarefa: Integração Contínua

1. Objetivo

Apresentar e documentar a tarefa Integração Contínua da atividade Dia de Trabalho (Processo Testar&Fixar) que faz parte do LMP – Logiciel Mobile Process a ser utilizado pelo Grupo de Estudo Logiciel como trabalho prático para as disciplinas de Integração I e de Desenvolvimento de Software para Dispositivos Móveis do curso de Bacharelado em Engenharia de Software do INF - UFG.

1.1 Escopo

Tarefa 3 da atividade Dia de Trabalho do LMP – Logiciel Mobile Process (Processo Testar&Fixar).

2. Introdução

O processo de desenvolvimento LMP – Logiciel Mobile Process abrange atividades acadêmicas referentes aos processos de engenharia de software do INF-UFG. Este oferecerá o apoio ao processo de desenvolvimento para dispositivos móveis, permitindo que o mesmo seja realizado de acordo com o planejamento de tempo e de recursos e com os requisitos funcionais e de qualidade definidos para os projetos propostos.

O modelo foi baseado no Processo MobileD e no RUP, além de seguir as orientações do Guia do MPS.Br 2011 (nível F).

3. Tarefa: Integração Contínua

A finalidade da integração contínua é continuamente integrar o novo código com o código existente em um repositório próprio. Ao integrar continuamente, as integrações maciças podem ser evitadas o que, caso contrário, podem tomar uma grande quantidade de tempo e esforço da equipe. A Integração Contínua é uma prática que permite aos desenvolvedores obter um *feedback* rápido sobre andamento do projeto de desenvolvimento como um todo. Ela ajuda a controlar as constantes mudanças do software.

4. Metas

Os objetivos de **Integração Contínua** são:

1. Assegurar que os membros da equipe trabalhem sempre com as últimas versões;
2. Certificar-se de que há sempre uma versão de trabalho do sistema; e
3. Permitir o desenvolvimento simultâneo.

5. *Inputs*

5.1 Pré-condições

1. **Ambiente adequado para Integração Contínua**. O ambiente de trabalho tem de ser adequado à integração contínua. Conforme estudos da área, a maioria da integração pode, e deve, ser feita automaticamente. Uma ferramenta de controle de versão pode ser utilizada para automatizar e apoiar a integração contínua. Um projeto pode sobreviver sem uma ferramenta de controle de versão utilizando práticas manuais na integração. No entanto, uma ferramenta de controle de versão é recomendada, porque torna mais fácil e flexível a integração contínua.

2. **Ferramenta de controle de versão instalada**. Se no projeto decide-se usar uma ferramenta de controle de versão significa que esta deva ser instalada corretamente com um servidor para gerenciar o repositório de código fonte e as máquinas clientes para o desenvolvimento.

3. **Projeto sob controle de versão**. Antes de qualquer código ser integrado, o projeto precisa ser criado e compartilhado, utilizando a ferramenta de controle de versão.

5.2 Entrada

1. **Código fonte para ser integrado**. Desenvolvedores tenham terminado o seu trabalho e estão prontos para a integração de código.

6. *Outputs*

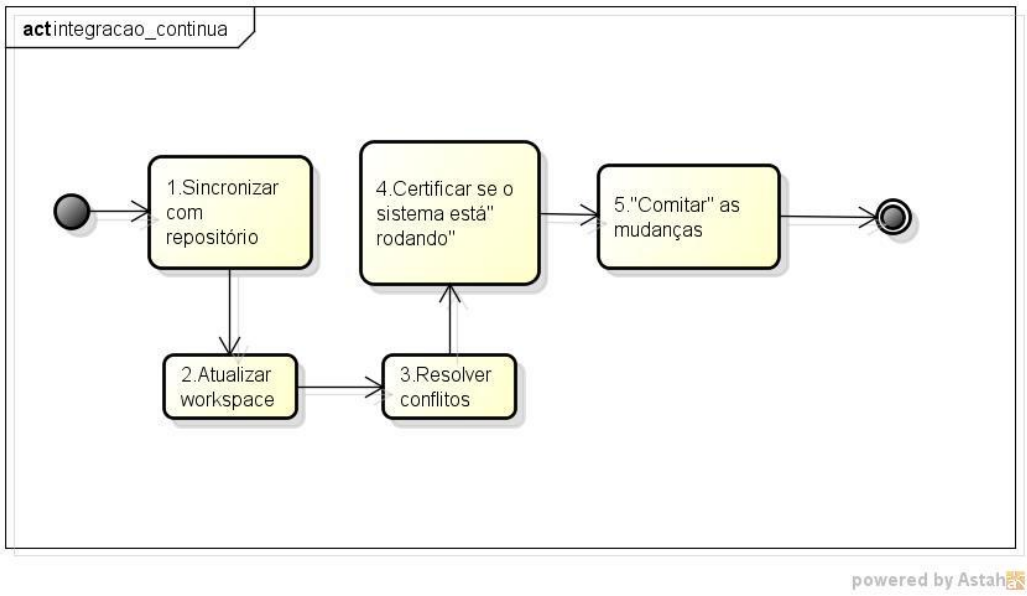
6.1 Pós-condição

1. O sistema não foi desintegrado. Alterações foram integradas e o sistema ainda está funcionando corretamente.

6.2 Saída

1. **Código fonte Integrado**. Código fonte testado e 100% funcionando com sucesso e integrado ao repositório de código. Outros desenvolvedores podem agora utilizar as versões mais recentes dos arquivos de desenvolvimento.

7. Diagrama da tarefa: Integração Contínua



8. Passos

As etapas individuais de **Integração Contínua** são:

1. **Sincronizar com o repositório**. Sincronizar o espaço de trabalho com o repositório. Isso torna possível visualizar o que foi alterado desde a última integração.

2. **Atualizar o espaço de trabalho**. Atualizar os arquivos que você não modificou, mas que alguém tenha modificado ou criado.

3. **Resolver conflitos**. Se tanto o espaço de trabalho como o repositório possuem o mesmo arquivo que tiver sido modificado, ocorrerá um conflito. Os conflitos precisam ser resolvidos por ***merge*** (fusão) destes dois arquivos.

4. **Verificar se o sistema está funcionando**. Devido à atualização, é preciso garantir que o sistema ainda esteja funcionando (por exemplo, executando testes de unidade novamente).

5. **Confirmar as alterações**. Finalmente, depois que todos os conflitos foram resolvidos e o sistema está funcionando, os arquivos podem ser "comitados" no repositório de código.

9. Papéis

As seguintes funções podem ser identificadas na execução de **Integração Contínua**:

1. **A equipe de projeto ou de desenvolvimento**. A pessoa que acrescenta um novo código, remove o antigo ou modifica o existente e assegura que estas mudanças são integradas regularmente.

10. Padrões Relacionados

Outros padrões que fazem parte deste ou são associados com a atividade são identificados abaixo:

- **Dia de Trabalho**. A Integração Contínua faz parte do padrão Dia de Trabalho.
- **Programação em Par**. A Integração Contínua está associada a Programação em pares, uma vez que é recomendável fazer integração contínua durante a programação.
- **TDD**: A Integração Contínua está associada ao Test-Driven Development.

11. Riscos

Os possíveis riscos que podem resultar de Integração Contínua, bem como as soluções incluindo ações preventivas para evitá-los e medidas a tomar para minimizar seus efeitos são discutidos aqui:

- **O Código não está integrado regularmente**. Talvez o maior risco da integração contínua é não segui-la completamente e de forma contínua. Isto pode causar integrações maciças que geram muito tempo e esforço. Solução: Para evitar as integrações em massa e respectivas resoluções de conflitos, devem-se integrar as alterações frequentemente. Integrações frequentes minimizam a chance de que o código seja irregularmente integrado no repositório.
- **Perder alterações**. A principal causa da perda de alterações é fundir incorretamente as alterações no código. Solução: Ao utilizar uma ferramenta de controle de versão, uma nova versão do arquivo será sempre criada ao "comitar" a alteração no código em repositório. Então, basicamente nenhuma das alterações é realmente uma mudança perdida. Com uma ferramenta de controle de versão pode-se facilmente reverter para versões anteriores do arquivo.
- **Integração "quebra" do sistema**. Às vezes, pode acontecer que o sistema não funcione mais após a integração. Solução: O teste de unidade abrangente é uma boa prática preventiva para evitar este risco. Uma vez que todos os testes de unidade devem funcionar 100 por cento antes de qualquer código ser integrado, garante que outros desenvolvedores não consigam quebrar seu código por acidente.

12. Referências

- http://agile.vtt.fi/mobiled.html