Ministério da Agricultura, Pecuária e Abastecimento

Comunicado Técnico 31

Dezembro, 2002
Campinas, SP

ISSN 1677-8464

Integração Contínua no Projeto SIGI

Marcos Cezar Visoli[1]
Moacir Pedroso Júnior[2]
João Francisco Gonçalves Antunes[3]

Na atividade de desenvolvimento de software existe uma série de boas práticas que deveriam ser realizadas, mas que na maioria das vezes, não são. Uma das mais básicas, e muito importante, é a de utilizar ou construir um processo automatizado de integração e de testes, que possibilite integrar e testar o código produzido várias vezes num dia por uma equipe de desenvolvimento. Não se trata apenas de realizar um *daily build* e *smoke test (*McConnell, 1998*)*, uma prática conhecida e utilizada. Se trata de extremar o processo, na tentativa de identificar o mais cedo possível qualquer problema decorrente da integração de código.

Embora a integração contínua seja uma das práticas associadas a *Extreme Programming* (XP) (Beck, 2000), sua utilização não depende da aplicação de outras práticas de XP. Assim, qualquer projeto pode utilizar integração contínua, mesmo que não siga as outras práticas de XP.

Este documento trata da prática de integração contínua e da experiência do seu uso no projeto SIGI (Pedroso et al., 2001).

Integração contínua

O uso da integração contínua provoca uma mudança no padrão de desenvolvimento de software, o que é difícil de expressar caso nunca se tenha trabalhado em um ambiente com práticas como esta. A maioria dos desenvolvedores trabalham como se estivessem sós, num ambiente onde eles integram apenas o código construído por eles mesmos. Para a maioria, o trabalho em uma equipe de desenvolvimento apresenta problemas de integração, mas isso é encarado naturalmente como algo que faz parte do processo. A integração contínua reduz estes problemas, mas em troca, exige um grau mais elevado de disciplina.

O principal benefício da integração contínua é a possibilidade de se eliminar períodos no processo de desenvolvimento de software em que pessoas gastam muito tempo procurando erros que não foram inseridos por um ou por outro desenvolvedor, mas produzidos pela interação das partes desenvolvidas por cada um deles. Estes erros são difíceis de se encontrar e podem ser inseridos muito tempo antes de serem detectados.

[1] *Bsc. em Ciência da Computação, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo – 13083-970 – Campinas, SP. (e-mail: visoli@cnptia.embrapa.br)*

[2] *Ph.D. em Pesquisa Operacional, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo – 13083-970 – Campinas, SP. (e-mail: pedroso@cnptia.embrapa.br)*

[3] *Bsc. em Matemática Aplicada e Estatística, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo – 13083-970 – Campinas, SP. (e-mail: joaof@cnptia.embrapa.br)*

Embrapa

Com a integração contínua, uma grande quantidade destes erros pode ser detectada no mesmo dia em que foram introduzidos. Além disso, a busca do trecho de código que ocasionou o erro é facilitada pela redução do escopo de pesquisa. E, mesmo que o erro não seja rapidamente encontrado, a liberação do código com problema para a produção pode ser evitado, a não ser que a nova funcionalidade tenha uma grande importância, que se sobrepõe à do erro. De qualquer forma, o erro estará identificado.

A integração contínua não garante que todos os erros sejam encontrados, pois a técnica é baseada em testes, e testes não eliminam todos os erros. O que testes se propõem a fazer é encontrar o maior número possível de erros, aumentando a confiabilidade do produto e, possivelmente, reduzindo seus custos de desenvolvimento e manutenção.

Uma máxima da integração contínua é a de que integrar sempre, com freqüência, é melhor que integrar raramente. Pode parecer óbvio para quem pratica isto, mas para equipes que não o fazem em seus projetos não é. Se a integração é realizada ocasionalmente, a tarefa pode despender tempo e energia razoável, o que necessitará de mais trabalho da equipe de desenvolvimento. Um comentário geralmente elaborado é o de que em grandes projetos não é possível realizar integração diária, porém há vários projetos que o fazem, inclusive projetos de mais de 10 milhões de linha de código, como é o caso de alguns da Microsoft.

A chave para um processo de integração contínua bem sucedido é automatizar. Grande parte da integração pode, e deveria, ser feita de forma automatizada. Obter o código, compilar, realizar testes significativos e outras tarefas podem ser realizadas automaticamente. Ao final do processo, deveria ser emitido uma indicação de como a integração ocorreu, com sucesso ou não. Caso a integração tenha ocorrido com sucesso nenhuma ação é necessária. Caso tenha ocorrido uma falha, rapidamente deve ser identificado onde a integração não funcionou, para que as ações de correção sejam devidamente planejadas e executadas.

Se a integração contínua estiver devidamente automatizada, é possível integrar tantas vezes quanto possível, pois o esforço para isto é muito pequeno. A única limitação é a quantidade de tempo que uma iteração da integração utiliza.

Localização única do código fonte

Para que a integração seja facilitada, é necessário que qualquer desenvolvedor seja capaz de obter o código fonte facilmente. Não é mais admissível que seja necessário solicitar a cada um dos desenvolvedores as últimas versões do código, verificar onde copiá-los e realizar as configurações necessárias antes de iniciar uma integração.

O padrão que pode ser estabelecido é o de que qualquer desenvolvedor possa, de uma máquina qualquer, conectada à rede, com um simples comando obter cada um dos arquivos necessários para construir o sistema sob desenvolvimento.

A solução mais óbvia, espera-se, pelo menos, é a de usar um sistema de gerência de configuração do código fonte de todo o sistema. Estas ferramentas geralmente são projetadas para operar sobre uma rede e 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 temos o *Concurrent Versions System* (CVS) (Collabnet, 2002), uma excelente ferramenta com código aberto (*open source*).

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. O problema que surge é o de que, para integrar, deve-se lembrar quais as versões de código fonte para cada componente devem ser utilizadas. A construção de vários componentes deveria ser tratada pelos *scripts* de integração, e não pela estrutura do sistema utilizado para armazená-los.

Scripts de construção automatizados

Para um projeto pequeno, com uma dúzia ou mais de arquivos, a construção da aplicação pode ser feita através de um pequeno comando, como por exemplo javac *.java (caso o código da aplicação esteja em java). Em projetos maiores, o código fonte geralmente está organizado em muitos diretórios e o processo de compilação muitas vezes deve garantir não só a ordem da compilação, mas também a localização exata dos objetos produzidos.

Outro ponto é que o tempo para compilar e ligar os objetos pode ser grande, e muitas vezes entre uma integração e outra, uma pequena parte do código ou alguns arquivos tenham sido alterados. Uma boa ferramenta de integração deveria verificar quais arquivos foram alterados e somente compilar estes. Isto pode ser realizado através da verificação das datas de alteração de arquivos e das datas em que os objetos foram gerados. Então dependências podem ser usadas: se para um objeto, outro do qual ele dependa foi alterado, o primeiro deve ser compilado novamente.

Alguns compiladores gerenciam dependências, outros não.

Também, dependendo do que é necessário construir, podem ser usados *scripts* diferentes ou *scripts* parametrizados. Por exemplo, pode ser necessário construir a aplicação com ou sem testes, ou ainda com diferentes conjuntos de testes. Alguns componentes podem ser construídos independentes dos outros. Enfim, os *scripts* deveriam fornecer alternativas do que produzir para os diferentes cenários existentes num ambiente de produção de software.

De simples linhas de comando, os *scripts* tornam-se mais sofisticados, e podem passar a ser *shell scripts* ou construídos em linguagens mais sofisticadas, como *perl* (Perl Foundation, 2002) ou *python* (Python Software Foundation, 2003). De qualquer forma, torna-se evidente a necessidade de se usar um ambiente projetado para este tipo de demanda.

O uso de Integrated Development Environment (IDEs) no processo de desenvolvimento também deve ser levado em consideração. Muitos deles utilizam arquivos proprietários para configuração do sistema, o que aumenta a complexidade na criação de *scripts* de integração. Além do mais, cada IDE pode ser configurado de forma diferente por cada um dos desenvolvedores (propositadamente ou acidentalmente) e isto deve ser uniformizado na fase de integração.

Suporte a testes unitários

Os compiladores podem identificar uma série de problemas existentes na construção de programas, mas ainda assim existe uma série de outros erros que não são identificados. Para identificar um número maior de erros com antecedência, uma alternativa é o uso da disciplina de testes automatizados, defendido nas práticas de XP (Beck, 2000).

Em XP, os testes são divididos em duas categorias: testes unitários e testes de aceitação (ou funcionais). Os testes unitários são escritos pelos desenvolvedores e tipicamente testam uma classe ou um pequeno grupo de classes. Testes de aceitação geralmente são escritos pelo cliente ou por uma equipe de testes (assessorado pelos desenvolvedores) e testam o sistema como um todo. É possível automatizar, tanto quanto for possível, os dois tipo de testes.

No processo de integração, todos os testes unitários podem ser realizados. E pode-se colocar uma condição de que um processo de integração está realizado quando, além de compilado, nenhum erro for encontrado executando-se todos os testes unitários. A etapa de execução de testes de aceitação, que segue à integração, deverá obrigatoriamente ser realizado por uma equipe de testes sobre uma versão do programa que foi construído com todos os testes unitários realizados, e todos sem nenhum erro encontrado.

O princípio básico de teste unitário é o de que quando os desenvolvedores escrevem algum código, eles também escrevem testes para aquele código. Quando uma tarefa está completa, não somente o código correspondente está pronto, mas todo o código necessário para testar (e garantir que está correto) está pronto também. Isto segue a prática de XP, que ainda diz que o teste deve ser escrito primeiro, ou seja, nenhum código deveria ser escrito sem que existisse um teste que acusasse uma falha para ele. Quando uma funcionalidade deve ser adicionada, primeiro deve ser escrito um teste que não acusasse falha se a funcionalidade estivesse lá. Somente depois disto é que a funcionalidade apenas deve ser escrita.

Existem ferramentas para construir testes, como JUnit (Object Mentor, 2002) para a linguagem java, um *framework* muito simples para escrever testes, organizá-los em conjuntos, e executá-los interativamente ou em lote. JUnit é uma ferramenta da família xUnit (Jeffries, 2002), que possui este tipo de ambiente para várias outras linguagens.

Os testes podem ser automatizados, e assim incluídos num processo de integração do código fonte. Executá-los várias vezes ao dia, rapidamente acusam problemas nos testes que foram escritos (e que deveriam estar corretos) ou no código que é testado. E quando estes erros acontecem, já que estão localizados numa determinada classe ou conjunto de classes, são rapidamente encontrados. Para facilitar o processo de localização de um erro nesta fase, a versão atual do código fonte, onde está o problema pode ser comparada com a versão anterior, mantida pela ferramenta de controle de versão, para que as diferenças sejam apontadas e o erro devidamente localizado, e então corrigido.

Cabe ressaltar que os testes, mesmo que quando executados não acusem falha, não garantem a ausência de erros. Testes imperfeitos (que não buscam todos os erros) que são executados freqüentemente são muito melhores que testes perfeitos que nunca são escritos.

Uso da integração contínua no projeto SIGI

O projeto SIGI trata do desenvolvimento de um sistema informatizado para o gerenciamento de projetos de pesquisa, de desenvolvimento institucional e de negociação tecnológica. Ainda, suporta módulos que agrupam informações destes projetos em planos táticos e operacionais, gerenciamento de tabelas do sistema, controle de acesso, etc.

O Sigi é composto por uma aplicação chamada SigiCliente, que é executada na plataforma Windows (95, NT, 2000), construído em Borland Delphi (Borland Software Corporation, 2003), para editar os projetos gerenciados pelo sistema, e uma outra parte, chamada

SigiServidor construída na plataforma J2EE, que contém os módulos que agregam as informações editadas pelo SigiCliente e incorporam outras informações gerenciais. Tanto a comunicação do SigiCliente com o SigiServidor, como a operação do SigiServidor se dá via Web.

A Fig. 1 apresenta a arquitetura do ambiente de construção e integração do sistema, totalmente automatizados, sendo que as cores distinguem os equipamentos onde os processos são realizados. As seções posteriores descrevem cada uma das etapas.

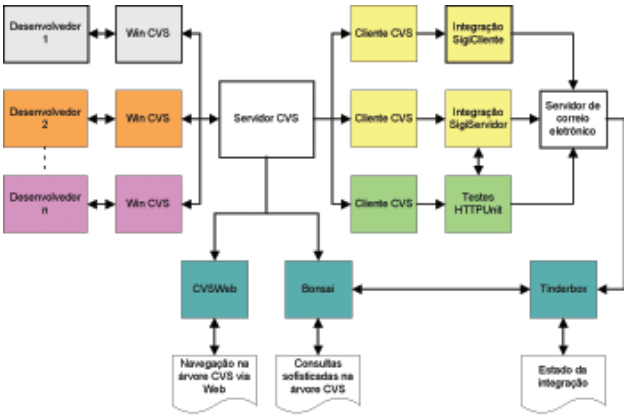

Fig. 1. Integração contínua no projeto Sigi

Código fonte sob controle de versão

Todo o código fonte do Sigi está centralizado num repositório, sob controle de versão, com o uso de um servidor CVS. Além do código fonte do projeto Sigi, o servidor mantém todos os *scripts* necessários para a construção e integração, esquema do banco de dados, documentação, fontes do site do projeto, etc.

A partir do repositório, os desenvolvedores atualizam os ambientes de desenvolvimento de seus equipamentos, implementam 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. Para interagir com o servidor CVS, os desenvolvedores utilizam o WinCVS (CVSGUI, 2002), uma ferramenta gráfica para o sistema operacional Windows que permite acesso ao CVS.

Processo de integração contínua

O processo de integração é realizado de forma independente das atualizações, efetuadas pelos desenvolvedores, no repositório mantido pelo CVS. 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 deve-se 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 os seguintes passos:

- envio de mensagem de início para o Tinderbox (Mozilla Organization, 2002b). Tinderbox é uma ferramenta de inspeção que, através do recebimento de informações de processos de compilação/construção de código, e da integração com outras ferramentas, permite rapidamente a verificação do sucesso ou não de um processo, a identificação das pessoas que atualizaram os arquivos e o 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;
- descarga (*checkout*), do repositório mantido pelo CVS, 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;
- envio de mensagem de *log* de todo o processo de integração e término de execução para o Tinderbox.

No Sigi existem três *scripts* de integração: do SigiCliente, do SigiServidor e de testes de interface Web, construídos com HttpUnit (2002). Em princípio, seguem os itens listados acima, porém para cada um existem tarefas específicas. HttpUnit é uma ferramenta que permite a emulação de partes do comportamento de um navegador (*browser*), incluindo submissão de

formulários, *scripts* em Java, autenticação http, *cookies* e redirecionamento automático de páginas, e permite, através de código Java, examinar as páginas retornadas como texto, XML DOM (*Document Object Model*), conjuntos de formulários, tabelas e *links*.

A comunicação dos *scripts* com o Tinderbox é realizada via correio eletrônico, sendo necessário a existência de um servidor.

Os *scripts* são executados de hora em hora, ou seja, a cada hora tem-se um resultado de uma integração. A escolha do intervalo de tempo respeita o tempo necessário para uma integração, inferior a uma hora, e depois opta pelo primeiro número inteiro de horas. Por que não foi escolhido outros valores como duas, três ou mais horas? Aplica-se uma regra muito simples: se a integração é uma prática importante, por que não extremá-la? Assim, optar por um intervalo de uma hora é, por enquanto, a escolha mais adequada.

Além dos *scripts* de integração executam-se outros para a produção automática da documentação do código fonte do sistema, tanto da parte cliente como da parte do servidor. Estas atividades estão agendadas para serem realizadas uma vez por dia.

Os *scripts* de integração estão escritos em *Practical Extracting and Reporting Language* (Perl) (Perl Foundation, 2002), que invoca comandos CVS, comandos de *shell*, comandos que processam arquivos *makefile* (Free Software Foundation, 2002), compiladores, etc. O Tinderbox é responsável por apresentar, via Web, os resultados dos *scripts* de integração, de acordo com a mensagem e seu conteúdo, recebidos do *scripts* em execução. São quatro os possíveis resultados de uma integração e são identificados por cores diferentes:

- verde: indica que o script foi realizado com sucesso;
- amarelo: indica que o script está em execução;
- vermelho: indica que o script falhou, e que a falha é de compilação;
- laranja: indica que o *script* falhou, e que a falha é de execução de testes unitários.

Para cada iteração do *script* de integração, o Tinderbox mantém um link para o arquivo de log. Assim, é possível verificar com detalhes, todos os comandos executados e seus resultados. Ainda, o Tinderbox interage com o Bonsai (Mozilla Organization, 2002a), informando quais usuários atualizaram o repositório CVS no período de cada iteração, e apresentando *links* para acessar os arquivos que estes alteraram. Bonsai é uma ferramenta que auxilia na execução de consultas sobre uma estrutura controlada pelo CVS, obtendo uma lista de atualizações efetuadas ou uma lista de atualizações efetuadas por cada pessoa, podendo ser restringida por espaço de tempo pré-determinado. Também pode recuperar os *logs* e comentários incluídos, realizar comparações entre diferentes versões de arquivos, e recuperar qual a pessoa que alterou determinada linha de código de um arquivo, fazendo do Bonsai uma boa ferramenta para auxiliar no controle da estrutura mantida pelo CVS.

A Fig. 2 apresenta uma tela de informação sobre a execução dos *scripts* de integração do projeto Sigi. Pode-se verificar a existência de três colunas, uma para cada processo de integração, onde estão disponibilizados as informações de data e hora em que foi realizado, a situação de cada um deles, um *link* para um arquivo de *log*, com o registro de todos os comandos executados na iteração do processo.

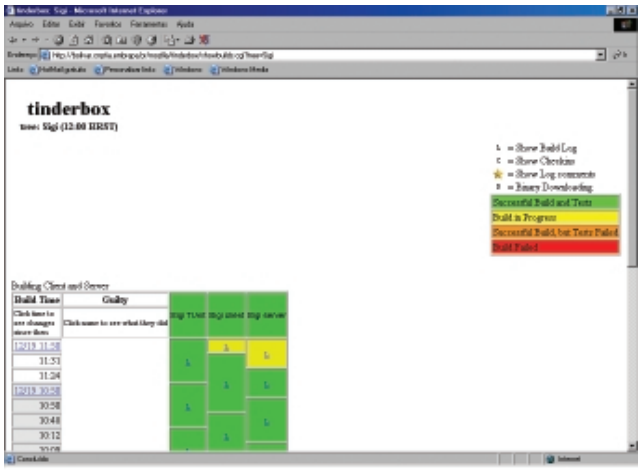

Fig. 2. Resultado do processo de integração do Sigi, emitido pelo Tinderbox.

Outra preocupação na elaboração de *scripts* é a de que deve ser possível também executá-los numa máquina de desenvolvimento. O *script* é configurado para identificar se está em execução na máquina de integração ou na máquina de desenvolvimento. Na máquina de desenvolvimento é necessário que alguns passos não sejam executados, como, por exemplo, o de remoção do código fonte e descarga do repositório sob controle do CVS.

Integração do SigiCliente

O SigiCliente é desenvolvido em Borland Delphi 5.0 e utiliza uma série de outras ferramentas adicionais. A construção do ambiente de integração contínua deve levar em consideração que para o desenvolvimento do SigiCliente utiliza-se o IDE, e para a máquina de integração tudo deve ser feito através de linhas de comando, para viabilizar a automatização. É necessário analisar os arquivos de configuração do IDE e adaptá-los para o ambiente de integração contínua, com

atenção para cada uma das opções de compilação, diretórios das ferramentas utilizadas, opções de montagem, etc. Além disso, é necessário adicionar *scripts* para produção automática de arquivos de *help* e do instalador do software.

Algumas ferramentas podem ser adicionadas durante o processo de desenvolvimento do projeto e isto requer um procedimento adequado para não comprometer o ambiente já em funcionamento. A estratégia é a de realizar a instalação primeiro em uma máquina de desenvolvimento, adaptando os *scripts* necessários, testar e quando tudo estiver correto, repetir os procedimentos na máquina de integração.

Integração do SigiServidor

O SigiServidor é construído na plataforma J2EE (Sun Microsystems, 2002) e seu ambiente alvo para execução é o *servlet container* Tomcat (Apache Software Foundation, 2002b). Assim, é necessário um trabalho adequado na organização dos diretórios necessários para produzir a aplicação de acordo com a estrutura requerida pelo Tomcat.

Em cada iteração, é realizada a criação da base de dados do sistema, com a carga de dados pré-definidos, através da execução de *scripts* mantidos também pelo CVS. A base de dados é necessária para alguns testes unitários, inclusive testes de produção de relatórios.

O *script* de integração do SigiServidor trabalha em conjunto com o *script* de integração e de testes unitários da interface Web do sistema, os testes construídos com HttpUnit são descritos na próxima seção. Quando uma integração é bem sucedida, o arquivo de instalação do SigiServidor é disponibilizado em uma área comum, para que os testes unitários em HttpUnit sejam executados. Este, ao finalizar seus testes, indica que uma nova integração do SigiServidor pode ser iniciada.

Execução dos testes de interface Web

Os testes de interface *Web* do SigiServidor são realizados em um equipamento diferente daquele onde são realizados os *scripts* anteriores, principalmente por não ser adequado instalar as ferramentas de desenvolvimento, necessárias para a construção do SigiServidor, no mesmo ambiente de execução do SigiServidor, o Tomcat.

Além dos itens citados no escopo da seção principal, o *script* de integração aguarda a disponibilidade de uma versão do SigiServidor, automaticamente o instala no computador, inicia o *servlet container* Tomcat, realiza todos os testes, desativa o Tomcat e finalmente emite a informação para o script de integração do SigiServidor, que já pode produzir uma nova versão.

Avanços e melhorias

O ambiente de integração contínua está operacional, devidamente testado e aprovado após quase dois anos em operação. Porém, como em qualquer processo, várias melhorias podem ser realizadas, decorrentes da experiência acumulada e a disponibilidade de novas ferramentas.

Um melhoria significativa, é a organização de *scripts* para módulos menores, principalmente do processo de integração do SigiServidor. Pode-se criar vários outros *scripts*, um para cada componente do sistema, o que oferece uma informação mais detalhada da situação da integração.

Outra melhoria está relacionada ao uso da ferramenta make (Free Software Foundation, 2002), que, entre outras coisas, processa arquivos de dependências de compilação dos componentes. A ferramenta Ant (Apache Software Foundation, 2002a) é muito utilizada para a organização destas dependências no ambiente de desenvolvimento de aplicações com a tecnologia Java, mas pode também ser aproveitada para outros ambientes.

Outra melhoria é verificar com mais detalhes novas ferramentas para o monitoramento do processo de integração contínua. CruiseControl (2002) e Krysalis Centipede (Krysalis Community Project, 2002) são, por exemplo, ferramentas ou pacotes que oferecem uma série de recursos para facilitar a construção de ambientes de integração contínua.

Conclusões

Desenvolver um processo de integração disciplinado e automatizado é essencial para o controle de um projeto de desenvolvimento de software. O sucesso do uso da integração contínua está direta e fortemente associado à busca pela automatização de todo processo que seja necessário ser realizado na tarefas de compilação, montagem, testes unitários, etc. Como conseqüência, exige uma disciplina maior no processo diário de desenvolvimento de software. Verificar se o código está testado, integrar o novo módulo ou a sua atualização na máquina de desenvolvimento, atualizar o repositório sob o CVS freqüentemente são algumas das tarefas do desenvolvedor. Por outro lado, o acompanhamento dos resultados das integrações também exigem freqüência e disciplina. Uma integração diária, exige um acompanhamento com a mesma freqüência, pois não há sentido em gastar um esforço em automatizar o processo de integração se o acompanhamento não segue a mesma freqüência.

No projeto Sigi, o uso da integração contínua quase completa dois anos. O início da implantação desta prática exigiu certo esforço, para estabelecer padrões de configuração, permitir extensibilidade à estrutura de armazenamento do código fonte, dominar as

ferramentas utilizadas no processo de integração e das configurações das ferramentas utilizadas no desenvolvimento do sistema, e também com a integração das ferramentas como CVS, Bonsai, Tinderbox, etc. Porém, passado este período inicial, as tarefas se resumiram ao acompanhamento e pequenos ajustes quando da inserção de uma nova ferramenta.

Pode-se dizer que o resultado do uso da prática de integração contínua é excelente. Os resultados práticos são vários, e destacam-se os seguintes:

- identificação rápida de erros de integração, e posterior identificação da localização do erro e correção imediata;
- maior controle sobre a configuração das ferramentas de desenvolvimento;
- maior segurança e confiabilidade sobre as versões do produto geradas.

Referências bibliográficas

APACHE SOFTWARE FOUNDATION. **The Apache Ant Project**. Disponível em: <http://jakarta.apache.org/ant>. Acesso em: 17 dez. 2002a.

APACHE SOFTWARE FOUNDATION. **The Apache Jakarta Project**: The Jakarta site: Apache Tomcat. Disponível em: <http://jakarta.apache.org/tomcat/>. Acesso em: 17 dez. 2002b.

BECK, K. **Extreme programming explained**: embrance change. Boston: Addison-Wesley, 2000. 224 p.

BORLAND SOFTWARE CORPORATION. **Borland Delphi Studio Enterprise**. Disponível em: < http://www.borland.com/delphi/delphi_enterprise/index.html>. Acesso em: 10 jan. 2003.

COLLABNET. **Concurrent Versions System**. Disponível em: <http://www.cvshome.org>. Acesso em: 16 dez. 2002.

CRUISECONTROL. **[CruiseControl home page]**. Disponível em: <http://cruisecontrol.sourceforge.net>. Acesso em: 17 dez. 2002.

CVSGUI [home page]: a set of GUI front-end for CVS. Disponível em: <http://www.wincvs.org>. Acesso em: 17 dez. 2002.

FREE SOFTWARE FOUNDATION. **GNU Make - GNU Project**. Disponível em: <http://www.gnu.org/software/make/make.html>. Acesso em: 17 dez. 2002.

HTTPUNIT [home page]. Disponível em: <http://www.httpunit.org>. Acesso em: 17 dez. 2002.

JEFFRIES, R. E. **XProgramming.com**: an extreme programming resource. Disponível em: <http://www.xprogramming.com/software.htm>. Acesso em 17 dez. 2002.

KRYSALIS COMMUNITY PROJECT. **Krysalis centipede**. Disponível em: <http://www.krysalis.org/centipede>. Acesso em: 17 dez. 2002.

MCCONNELL, S. **Software project survival guide**. Redmond: Microsoft Press, 1998. 304 p.

MOZILLA ORGANIZATION. **Bonsai - CVS query form**: a query interface to the CVS source repository: Bonsai version 1.3. Disponível em: <http://bonsai.mozilla.org/cvsqueryform.cgi?cvsroot=/cvsroot>. Acesso em: 17 dez. 2002a.

MOZILLA ORGANIZATION. **Tinderbox**. Disponível em: <http://www.mozilla.org/tinderbox.html>. Acesso em: 17 dez. 2002b.

OBJECT MENTOR. **JUnit, testing resources for extreme programming**. Disponível em: <http://www.junit.org/index.html>. Acesso em: 17 dez. 2002.

PEDROSO JÚNIOR, M.; ANTUNES, J. F. G.; VISOLI, M. C.; CASTRO, A. M. G.; LIMA, S. M. V. **Sistema de Informação Gerencial de Projetos de Pesquisa Agropecuária para o Instituto Nacional de Investigaciones Agrícolas da Venezuela**. Campinas: Embrapa Informática Agropecuária, 2001. 14 p. (Embrapa. Programa 12 – Automação Agropecuária. Projeto 12.2001.950). Projeto em andamento.

PERL FOUNDATION. **Perl Mongers**. Disponível em: <http://www.perl.org>. Acesso em: 17 dez. 2002.

PYTHON SOFTWARE FOUNDATION. **Python language website**. Disponível em: <http://www.python.org>. Acesso em: 10 jan. 2003.

SUN MICROSYSTEMS. **Java 2 platform, enterprise edition J2EE**. Disponível em: <http://java.sun.com/j2ee> . Acesso em: 05 dez. 2002.

Comunicado Técnico, 31

Embrapa Informática Agropecuária
Área de Comunicação e Negócios (ACN)
Av. André Tosello, 209
Cidade Universitária - "Zeferino Vaz"
Barão Geraldo - Caixa Postal 6041
13083-970 - Campinas, SP
Telefone (19) 3789-5743 - Fax (19) 3289-9594
e-mail: sac@cnptia.embrapa.br

1ª edição
2002 - on-line

Expediente

Supervisor editorial: *Ivanilde Dispato*
Normalização bibliográfica: *Marcia Izabel* Fugisawa Souza
Capa: *Intermídia Publicações Científicas*
Editoração Eletrônica: *Intermídia Publicações Científicas*