Centro Federal de Ensino Tecnológico do Amazonas

Ferramentas para Desenvolvimento em Software Livre

Prof. Tiago Eugenio de Melo, Msc.
tiago@comunidadesol.org

Sumário

- Controle de Versões e Desenvolvimento Colaborativo de Software
- Make
- Gestão de bugs
- Qualidade de Software
- Prototipação de Software
- Perguntas de revisão
- Referências bibliográficas

Controle de Versões e Desenvolvimento Colaborativo de Software

- O desenvolvimento de sistemas é muito dinâmico e as mudanças são inevitáveis.
- O entendimento do cliente sobre as suas necessidades, o ambiente em que o sistema será executado e a legislação relacionada são fatores de tais mudanças.
- Diante deste contexto, surge a necessidade de controlar essas mudanças para que o desenvolvimento não se torne caótico. Surgindo a Gerência de Configuração.

Controle de Versões e Desenvolvimento Colaborativo de Software

- O que é Gerência de Configuração?
  - A Gerência de Configuração de Software (GCS) é um processo aplicado a todas as fases que compõem o ciclo de vida de um software, estabelecendo regras formais para identificar e controlar mudanças por meio de um controle sistemático sobre as modificações realizadas.
  - Pode-se entender também que a Gerência de Configuração de Software é um conjunto de atividades de apoio ao desenvolvimento que permite que as mudanças inerentes ao desenvolvimento sejam absorvidas pelo projeto de maneira controlada, mantendo a estabilidade na evolução do software.

Controle de Versões e Desenvolvimento Colaborativo de Software

- A Gerência de Configuração é útil para responder a algumas questões:
  - Quais mudanças aconteceram no sistema?
  - Por que essas mudanças aconteceram?
  - O sistema continua íntegro mesmo depois das mudanças?

Controle de Versões e Desenvolvimento Colaborativo de Software

- Conceitos
  - Configuração de um sistema é uma coleção de versões específicas de itens de configuração (hardware, firmware ou software) que são combinados de acordo com determinados procedimentos para servir a uma finalidade específica.
  - A Gerência de Configuração (GC) é a disciplina de identificar a configuração de um sistema em diferentes pontos no tempo, com a finalidade de controlar sistematicamente as mudanças realizadas, mantendo a integridade e a rastreabilidade da configuração através do ciclo de vida do sistema.
  - Resumindo, a configuração é um conjunto de versões específicas dos itens que compõem um sistema, enquanto que a GC é o controle da evolução dessas configurações no tempo.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Segundo o CMMi, as atividades relacionadas à GC são:
  - Identificação da configuração dos produtos de trabalho selecionados que compõem as baselines em um determinado ponto no tempo. (baseline é uma configuração formalmente aprovada para servir de referência para o desenvolvimento posterior do sistema.)
  - Controle das mudanças nos itens de configuração.
  - Construção ou fornecimento de especificações para construir produtos de trabalho a partir do sistema de gerenciamento de configuração.
  - Manutenção da integridade das baselines.
  - Fornecimento de dados precisos de status e configuração corrente a desenvolvedores, usuários finais e clientes.

Controle de Versões e Desenvolvimento Colaborativo de Software

- As atividades da GC podem ser resumidas em:

Controle de Versões e Desenvolvimento Colaborativo de Software

- Controle de versão
  - É a espinha dorsal de toda a gerência de configuração, apoiando as atividades de controle de mudança e integração contínua. Fornece os seguintes serviços:
    - Identificação, armazenamento e gerenciamento dos itens de configuração e de suas versões durante todo o ciclo de vida do software.
    - Histórico de todas as alterações efetuadas nos itens de configuração.
    - Criação de rótulos e ramificações do projeto.
    - Recuperação de uma configuração em um determinado momento desejado do tempo.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Controle de mudança
  - Fornece um serviço complementar ao oferecido pelo sistema de controle de versão. O foco deste tipo de ferramenta é nos procedimentos pelos quais as mudanças de um ou mais itens de configuração são propostas, avaliadas, aceitas e aplicadas.
  - Oferece serviços para identificar, rastrear, analisar e controlar as mudanças nos itens de configuração.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Integração contínua
  - Para as necessidades de GC, bastaria um controle de construção de software que cuidasse da identificação, empacotamento e preparação de uma baseline para a entrega a um cliente externo ou interno, tornando-a uma release ou um build, respectivamente.
  - A idéia de utilizar uma integração contínua, entretanto, vai um pouco mais além. O objetivo é garantir que as mudanças no projeto são construídas, testadas e relatadas tão logo quanto possível, depois de serem introduzidas.
  - Em projetos de software, a construção do programa é feita pela recuperação da configuração correta no sistema de controle de versão e a construção dos arquivos executáveis e da instalação do produto. Este processo é executado geralmente após cada mudança publicada no sistema de controle de versão ou em intervalos de tempos definidos.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Existem diversas ferramentas disponíveis para apoiar as atividades de GC.
- A quantidade de funcionalidades, a maturidade, a documentação, suporte disponíveis e a popularidade de cada ferramenta variam bastante.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Exemplos de ferramentas:

Controle de Versão	CVS
	Subversion
	Aegis
	Arch
Controle de Mudanças	Trac
	Mantis
	Bugzilla
	Scarab
Integração Contínua	Scons
	Bitten
	Ant
	Maven
	CruiseControl

Controle de Versões e Desenvolvimento Colaborativo de Software

- O controle de versões é o meio pelo qual a GCS se utiliza para controlar, de forma consistente, as modificações realizadas.
- Identificação das versões
  - Em grandes sistemas, existem centenas de componentes de software, com diferentes versões. O controle das versões devem definir, de maneira não ambígua, uma forma de identificar a versão de cada componente.
  - Numeração das versões
    - Neste esquema, o componente ou sistema é aumentado, controlado pela versão.
    - Por exemplo, a primeira versão é chamada de 1.0, e as versões seguintes são chamadas de 1.1, 1.2 e assim sucessivamente.

Controle de Versões e Desenvolvimento Colaborativo de Software

- O que é CVS – *Current Version System*?
  - Ferramenta *open source* que implementa as principais funções pertinentes ao processo de controle de versões.
  - O CVS armazena em seu repositório as modificações realizadas num arquivo ao longo do tempo; cada modificação é identificada por um número chamado Revisão.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Conceitos:
  - **Recuperação de erros**: frequentemente, durante o desenvolvimento de um software ou durante a correção de um *bug* pode-se gerar outros erros ao software, às vezes mais difíceis de serem corrigidos que os problemas originais. Em casos como este, as ferramentas de controle de versões permitem que seja recuperada uma versão do código anterior ao novo erro.
  - **Controle de mudanças por autor**: nas ferramentas modernas, é possível guardar um histórico das modificações que foram feitas pelos diversos programadores.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Conceitos:
  - **Desenvolvimento concorrente**: os sistemas de controle de versões procuram oferecer aos usuários um ambiente colaborativo, que permita a edição dos diversos arquivos de forma concorrente. Permite-se, dessa forma, que as mudanças efetuadas por diversos autores sejam sincronizadas.

Controle de Versões e Desenvolvimento Colaborativo de Software

- As principais funções do controle de versões são:
  - Recuperar versões anteriores.
  - Auditar as modificações realizadas: quem, quando e o quê.
  - Automatizar o rastreamento de arquivos.
  - Estabelecer meios para obter a situação de um projeto em determinado ponto do tempo.
  - Previnir conflitos entre desenvolvedores.
  - Permitir o desenvolvimento paralelo.

Controle de Versões e Desenvolvimento Colaborativo de Software

- O que não é CVS?
  - O CVS não é um compilador ou interpretador.
  - O CVS não substitui o coordenador ou líder de projeto.
  - O CVS não é um substituto para a comunicação entre as pessoas.
  - O CVS não é uma ferramenta de testes automatizada.
  - O CVS não é uma ferramenta de rastreamento de bugs.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Por que escolher o CVS?
  - O CVS é uma ferramenta *open source*, extremamente madura e amplamente utilizada por diversos projetos em todo o mundo.
  - Há uma enorme quantidade de documentação disponível na Internet sobre o CVS, além de livros, artigos e casos de sucesso.
  - O CVS é uma ferramenta multiplataforma.
- Onde eu posso obter o CVS?
  - Principais endereços:
    - www.cvshome.org
    - http://ximbiot.com/cvs/cvshome

Controle de Versões e Desenvolvimento Colaborativo de Software

- Terminologia no uso de controle de versões:
  - Workspace (área de trabalho)
    - Termo empregado para representar um diretório no computador cliente onde os arquivos de um projeto serão transferidos durante uma retirada.
  - Repository (repositório)
    - Local onde os arquivos submetidos ao controle de versões são armazenados.
  - Revision (revisão)
    - Termo usado para descrever a numeração atribuída pelo CVS a cada modificação de um arquivo.
  - Release (versão)
    - Estado de um conjunto de arquivos em um determinado ponto no tempo.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Terminologia no uso de controle de versões:
  - Commit (submeter)
    - Termo usado para descrever a transferência de um ou mais arquivos da área de trabalho para o repositório.
  - Checkout (recuperar)
    - Termo usado para descrever a transferência de um ou mais arquivos do repositório para a área de trabalho (contrário do commit).
  - Diff (comparação das diferenças)
    - Termo usado para descrever o processo de comparação das modificações entre revisões diferentes de um arquivo.

Controle de Versões e Desenvolvimento Colaborativo de Software

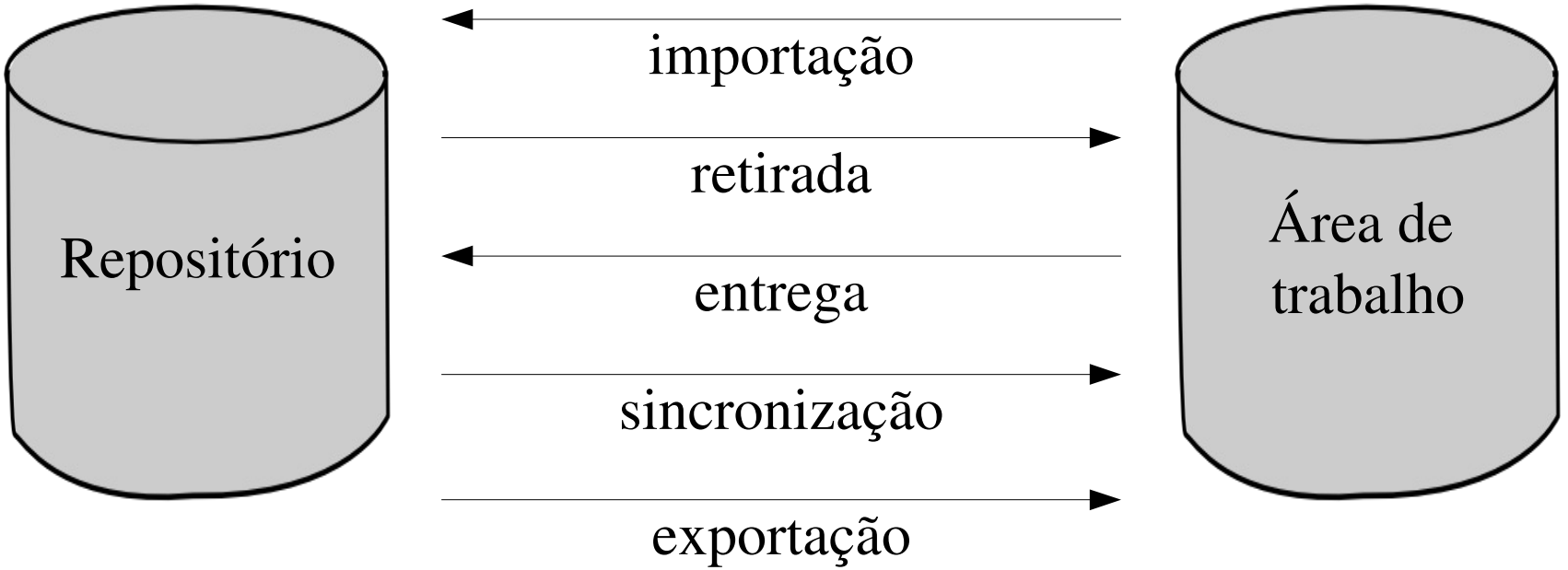
- Terminologia no uso de controle de versões:
  - Import (importação)
    - Termo usado para descrever o processo de transferência dos arquivos de um projeto a um repositório, dando início ao controle de versões.
  - Module (módulo)
    - Termo usado para descrever um nome simbólico associado a um conjunto de diretórios e arquivos que compõem um projeto, a fim de facilitar o acesso e a manipulação.
  - Branch (ramo)
    - Termo usado para descrever o processo de divisão dos arquivos de um projeto em linhas de desenvolvimento independentes.

Controle de Versões e Desenvolvimento Colaborativo de Software

- O que é um repositório?
  - Um repositório é um diretório que tem por objetivo abrigar todos os arquivos de um projeto sob o controle de versões.
- Como ele é dividido?
  - Arquivos administrativos
    - São empregados para configurar e ampliar o comportamento do CVS.
  - Módulos
    - São diretórios que armazenam os arquivos de um projeto sob o controle de versões.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Principais operações realizadas pelo CVS



Controle de Versões e Desenvolvimento Colaborativo de Software

- Processo de instalação
  - Pode ser baixado no endereço: http://www.nongnu.org/cvs
  - Através do comando `apt-get install` cvs.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Configuração do ambiente
  - A variável de ambiente $CVSROOT é um ponteiro que determina a localização do repositório.
  - Todas as operações do CVS serão realizadas no repositório determinado pela variável de ambiente $CVSROOT.
- Criação da variável de ambiente
  - `mkdir` /home/aluno/cvsroot
  - `export CVSROOT`=/home/aluno/cvsroot (para automatizar, basta adicionar esta ao arquivo /etc/profile).
- Inicializando um repositório
  - `cvs init`
  - `cvs -d` (caminho) `init` (explicitamente)

Controle de Versões e Desenvolvimento Colaborativo de Software

- Backup do repositório
  - A princípio, o backup pode ser feito de maneira semelhante ao backup de outros arquivos.
  - Porém, é importante observar que ao realizar o backup, nenhum usuário poderá estar acessando o repositório.
  - Para impedir o acesso ao repositório, basta criar o arquivo #cvs.lock na raiz do repositório.
- Movendo um repositório
  - Assim como o backup, basta mover o repositório como um arquivo qualquer.
  - Observar que variável CVSROOT deverá ser alterada também.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Repositórios remotos
  - O CVS utiliza um mecanismo cliente-servidor para realizar o acesso ao repositório.
  - Esse mecanismo é empregado para suportar múltiplos usuários simultâneos, garantindo a integridade dos arquivos e a atomicidade das operações e, sobretudo, fornecendo uma infra-estrutura mínima para o desenvolvimento colaborativo.
  - Como o CVS não utiliza muitos recursos durante a execução, o servidor não precisa ser muito robusto.
  - O disco rígido deverá ser compatível com o tamanhos dos arquivos armazenados.
  - Existem diversos métodos de acesso ao repositório.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

1. Criação de um novo projeto
    - O primeiro passo para iniciar o controle de versões é chamado de importação.
    - Neste passo, os arquivos de um projeto são transferidos para o repositório.
    - `cvs import -m` "mensagem" nome_modulo vendortag releasetag
2. Obtenção dos arquivos de um projeto
    - Para a realização das modificações necessárias, é preciso retirar os arquivos armazenados no repositório.
    - `cvs checkout` nome_modulo

Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

  3. Adição de arquivos

     - As alterações não são reconhecidas automaticamente pelo CVS, é necessário manipular isso explicitamente.
     - Para adicionar um arquivo:
     - `cvs add` aula1.txt
     - Caso um arquivo tenha sido removido acidentalmente, é possível recuperá-lo através do comando:
     - `cvs add` aula1.txt
     - Para adicionar um arquivo binário, deve-se passar um parâmetro informando ao CVS:
     - `cvs add -kb` aula1.pdf

Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

4. Remoção de arquivos

  - A remoção de um arquivo é feita em duas etapas. Primeiro, deve-se excluir fisicamente o arquivo. Em seguida, deve-se informar ao CVS que o arquivo deverá ser excluído do projeto.
  - `rm` aula1.txt
  - `cvs remove` aula1.txt
  - Se o arquivo não for excluído fisicamente, o CVS apresentará uma mensagem de erro. Uma alternativa é forçar a operação de remoção.
  - `cvs remove -f` aula1.txt
  - É possível também usar caracteres curingas (wildcards).
  - `cvs remove -f` *.txt

Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

5. Renomear arquivos

    - Para renomear um arquivo, deve-se excluir o arquivo e depois inclui-lo no repositório.
    - `mv` aula1.txt aula2.txt
    - `cvs remove -f` aula1.txt
    - `cvs add` aula2.txt
    - Apesar de ser possível renomear um arquivo diretamente no repositório, deve-se evitar este tipo de operação.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

6. Submeter alterações no repositório

    - Após a realização das modificações, estas devem ser submetidas no repositório. Este processo é chamado de entrega.
    - `cvs commit`
    - Somente os arquivos que foram removidos, criados ou alterados é que serão transferidos ao repositório e incorporados ao projeto.
    - Após a confirmação de entrega, o CVS abrirá o editor de texto padrão do sistema operacional para que o usuário introduza um comentário ou descrição sobre as alterações realizadas nos arquivos antes.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

7. Liberação da área de trabalho
    - A liberação da área de trabalho tem duas finalidades:
        - Antes da entrega, para descartar todas as alterações realizadas nos arquivos do projeto.
        - Depois da entrega, para eliminar os arquivos mantidos na área de trabalho e, principalmente, informar ao CVS que você não irá mais trabalhar no projeto.
    - `cvs release -d` nome_modulo
8. Arquivos binários
    - Ao informar, explicitamente, que o arquivo é binário, o CVS definirá um atributo persistente ao arquivo a fim de diferenciá-lo durante as demais operações, evitando que este se corrompa.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

8. Exportando um liberação

    - Para liberar os arquivos para um determinado cliente ou para disponibilizar para uma comunidade de desenvolvedores, é preciso exportar o arquivos no formato nativo, sem levar os arquivos de configuração do CVS.
    - `cvs export` aulas

9. Examinando o status dos arquivos

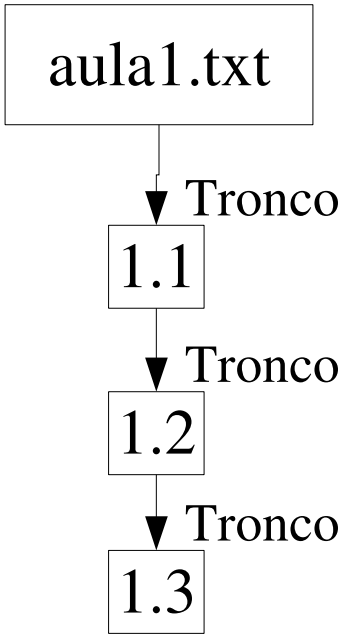
    - Às vezes, é necessário examinar o status dos arquivos mantidos na área de trabalho em relação aos arquivos armazenados no repositório.
    - `cvs status` aula1.txt

Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

10. Ramificação

  - O CVS mantém os arquivos numa estrutura lógica hierárquica, semelhante a uma árvore com um tronco e diversos ramos.
  - O pressuposto é de que o tronco dessa árvore cresça linearmente, conforme a evolução do projeto.

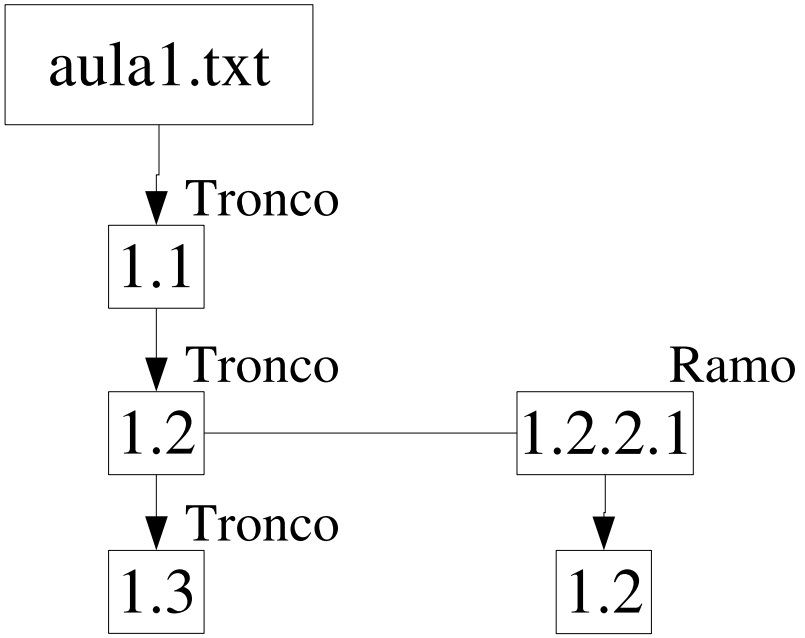


Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

10. Ramificação

- O CVS oferece ainda um mecanismo para a criação de ramos de desenvolvimento paralelo.
- Na maioria dos casos, os ramos são criados para separar as atividades de manutenção das atividades de manutenção evolutiva de um projeto.



Controle de Versões e Desenvolvimento Colaborativo de Software

- Etapas do controle de versões

10. Ramificação

    - A criação de uma ramificação pode ser feita através do comando `cvs tag`.
    - `cvs tag -b` rel2-0_patch

Controle de Versões e Desenvolvimento Colaborativo de Software

- Arquivos administrativos
  - Os arquivos administrativos são empregados para configurar e ampliar o comportamento do CVS.
  - Os arquivos administrativos também fornecem um meio para criar uma interface entre o CVS e as ferramentas externas.
  - Por padrão, os arquivos administrativos são criados durante a inicialização de um repositório por meio do comando `cvs init` e são armazenados no diretório CVSROOT.
  - Os arquivos administrativos são mantidos sob o controle de versões do CVS e não deverão ser modificados diretamente no repositório, com algumas exceções.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Ferramentas gráficas CVS
  - Apesar do CVS ser uma ferramenta de linha de comando, foram desenvolvidos vários programas que possuem uma interface gráfica mais amigável.

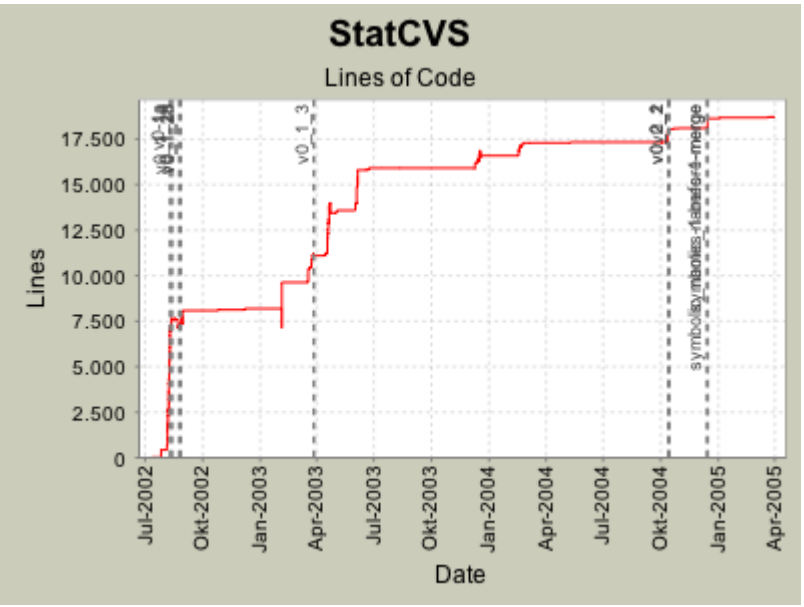
Controle de Versões e Desenvolvimento Colaborativo de Software

- Ferramentas gráficas CVS
  - StatCVS
    - StatCVS gera relatórios HTML a partir dos logs do repositório CVS. O programa oferece uma variedade de gráficos e tabelas descrevendo a evolução do projeto segundo diferentes pontos de vista.
    - Licença: GNU GPL
    - Linguagem de programação: Java
    - Multiplataforma
    - Link: http://sourceforge.net/projects/statcvs

Controle de Versões e Desenvolvimento Colaborativo de Software

- Ferramentas gráficas CVS
  - StatCVS

Author	Lines of Code
cyganiak	26174 (58.2%)
lukasz	8809 (19.6%)
jentzsch	4547 (10.1%)
manschu	4375 (9.7%)
squig	1028 (2.3%)
farkas	55 (0.1%)



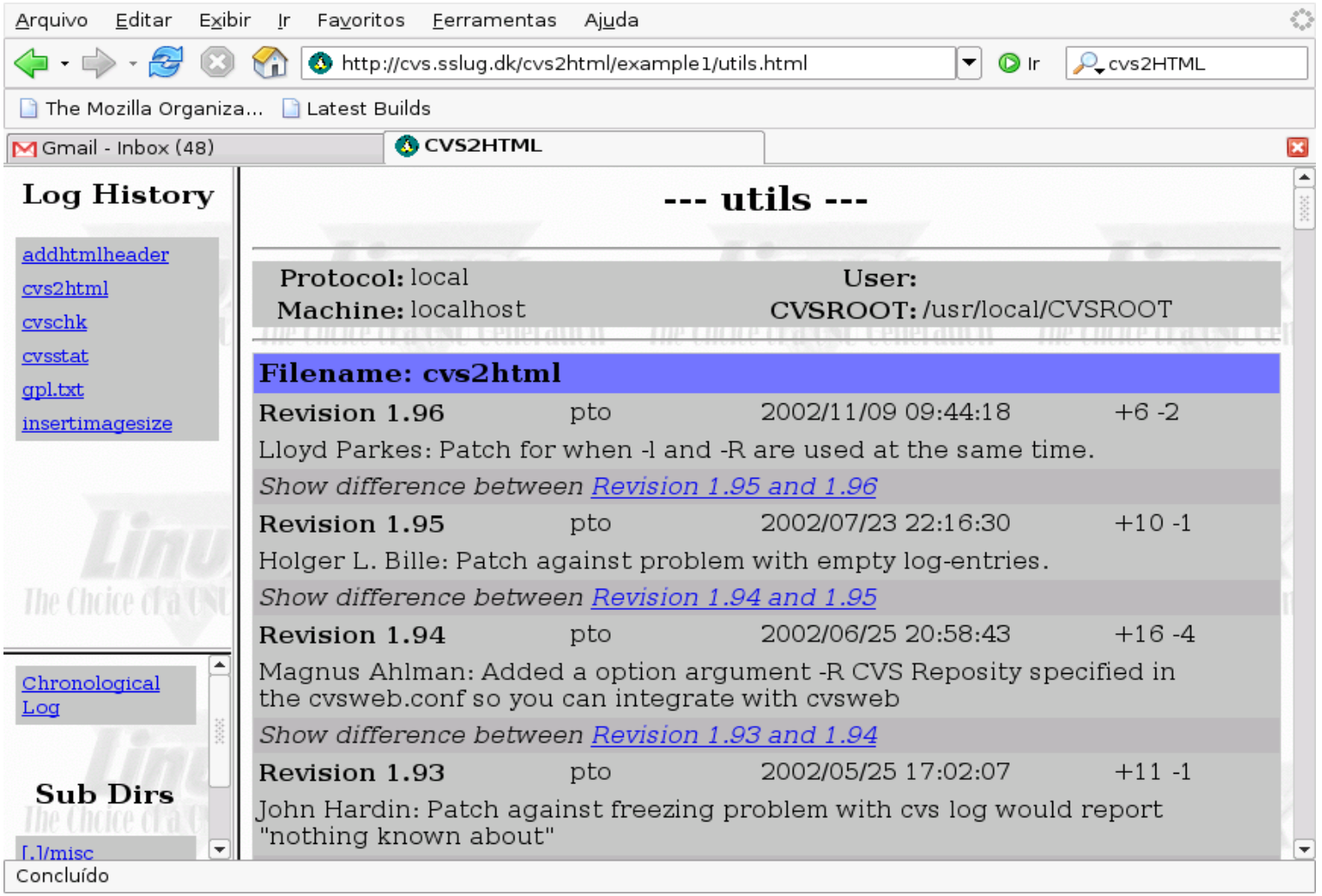
- [root] (5 files, 445 lines)
  - etc (3 files, 542 lines)
  - lib (4 files, 0 lines)
  - src (0 files, 0 lines)
    - net (0 files, 0 lines)
      - sf (0 files, 0 lines)
        - statcvs (7 files, 501 lines)
          - ant (2 files, 225 lines)
          - gui (0 files, 0 lines)
          - input (13 files, 1895 lines)
          - model (10 files, 1635 lines)
          - output (24 files, 3350 lines)
          - renderer (14 files, 1999 lines)
            - xml (0 files, 0 lines)
          - reportmodel (12 files, 1102 lines)
          - reports (13 files, 1026 lines)
          - util (10 files, 1145 lines)
          - web-files (5 files, 152 lines)
    - output (0 files, 0 lines)
  - tests-src (0 files, 0 lines)
    - net (0 files, 0 lines)
      - sf (0 files, 0 lines)
        - statcvs (1 files, 66 lines)
          - input (27 files, 1874 lines)
          - model (7 files, 985 lines)
          - output (2 files, 364 lines)
          - renderer (2 files, 244 lines)
            - xml (0 files, 0 lines)
          - reportmodel (2 files, 222 lines)
          - util (7 files, 946 lines)

Controle de Versões e Desenvolvimento Colaborativo de Software

- Ferramentas gráficas CVS
  - cvs2HTML
    - O cvs2HTML é uma ferramenta que tem por objetivo exportar os dados gerados pelo comando `cvs log` a fim de organizá-los num formato mais consistente e intuitivo.
    - Licença: GNU GPL
    - Linguagem de programação: Perl
    - Link: http://cvs.sslug.dk/cvs2html

Controle de Versões e Desenvolvimento Colaborativo de Software

- Ferramentas gráficas CVS
  - cvs2HTML

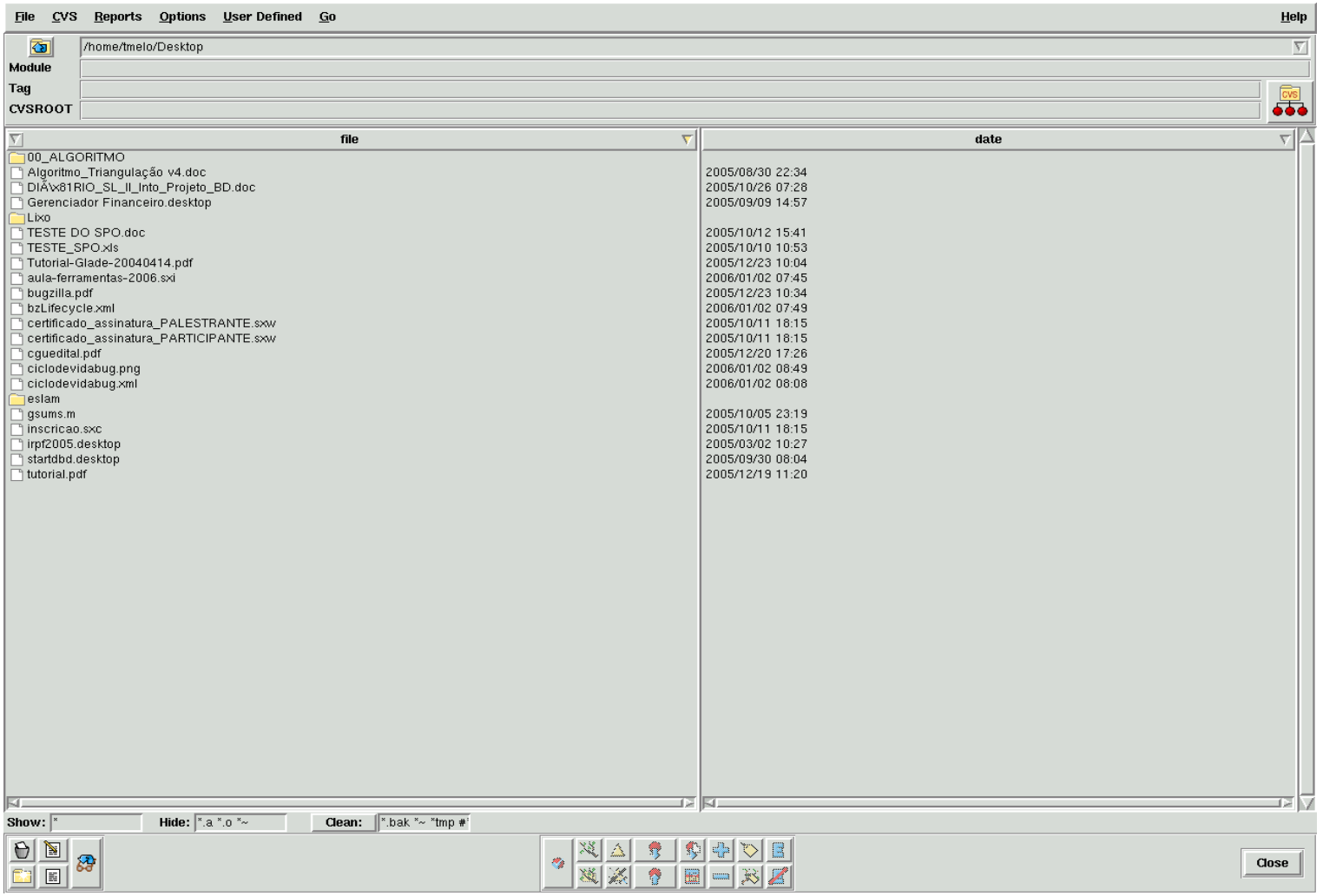


Controle de Versões e Desenvolvimento Colaborativo de Software

- Ferramentas gráficas CVS
  - TkCVS
    - TkCVS é uma ferramenta gráfica para fazer o controle do CVS. Ferramenta bastante completa.
    - Licença: GNU GPL
    - Linguagem de programação: Tcl/TK
    - Link: http://tkcvs.sourceforge.net

Controle de Versões e Desenvolvimento Colaborativo de Software

- Ferramentas gráficas CVS
  - TKCVS



Controle de Versões e Desenvolvimento Colaborativo de Software

- Comparativo entre o CVS e outros 13 softwares para Controle de Versões pode ser obtido no seguinte endereço:
- http://better-scm.berlios.de/comparison/comparison.html

Controle de Versões e Desenvolvimento Colaborativo de Software

- Subversion (SVN)
  - O que é SVN?
    - É uma ferramenta de Engenharia de Software que é empregada por muitos desenvolvedores para controlar as mudanças na sua árvore de código-fonte.
    - SVN é popular dentro da comunidade de software livre e utilizada por muitos projetos *Open Source*, tais como Apache, KDE, GNOME, Python, entre outros.
    - SVN roda em todas as distribuições GNU/Linux, além de outros sistemas operacionais como Win/32, BeOS, OS/2 e MacOS X.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Subversion (SVN)
  - No início do ano de 2000, a empresa CollabNet tentou escrever um substituto para o CVS devido à sua limitação.
  - Porém, neste período, o CVS já havia se tornado a ferramenta de controle de versão de fato dentro da comunidade de software livre.
  - A alternativa foi criar uma nova versão, tendo como base os seguintes objetivos:
    - Manter uma metodologia de controle de versão.
    - Manter a compatibilidade com os recursos já existentes no CVS.
    - Construir uma ferramenta similar ao CVS, mas com mais recursos, de forma que o usuário tivesse que fazer pouco esforço para mudar de ferramenta.
  - Finalmente, em agosto de 2001, o SVN se tornou livre do CVS.

Controle de Versões e Desenvolvimento Colaborativo de Software

- Características do SVN
  - Versionamento de diretórios. SVN implementa um sistema de arquivos virtual que rastreia as mudanças numa árvore de diretórios sobre o tempo.
  - Controle do histórico de arquivos e diretórios. SVN possibilita adicionar, apagar, copiar e renomear arquivos e diretórios.
  - *Commits* atômicos. Previne problemas que podem ocorrer quando somente uma parte de um conjunto de modificações é enviado com sucesso para o repositório. Números de revisão são por *commit* e não por arquivo. Logs de mensagens são anexadas para a revisão e não armazenada redundantemente as no CVS.
  - Metadados versionados. Metadados, assim como arquivos e diretórios podem ser versionados.

Controle de Versões e Desenvolvimento Colaborativo de Software

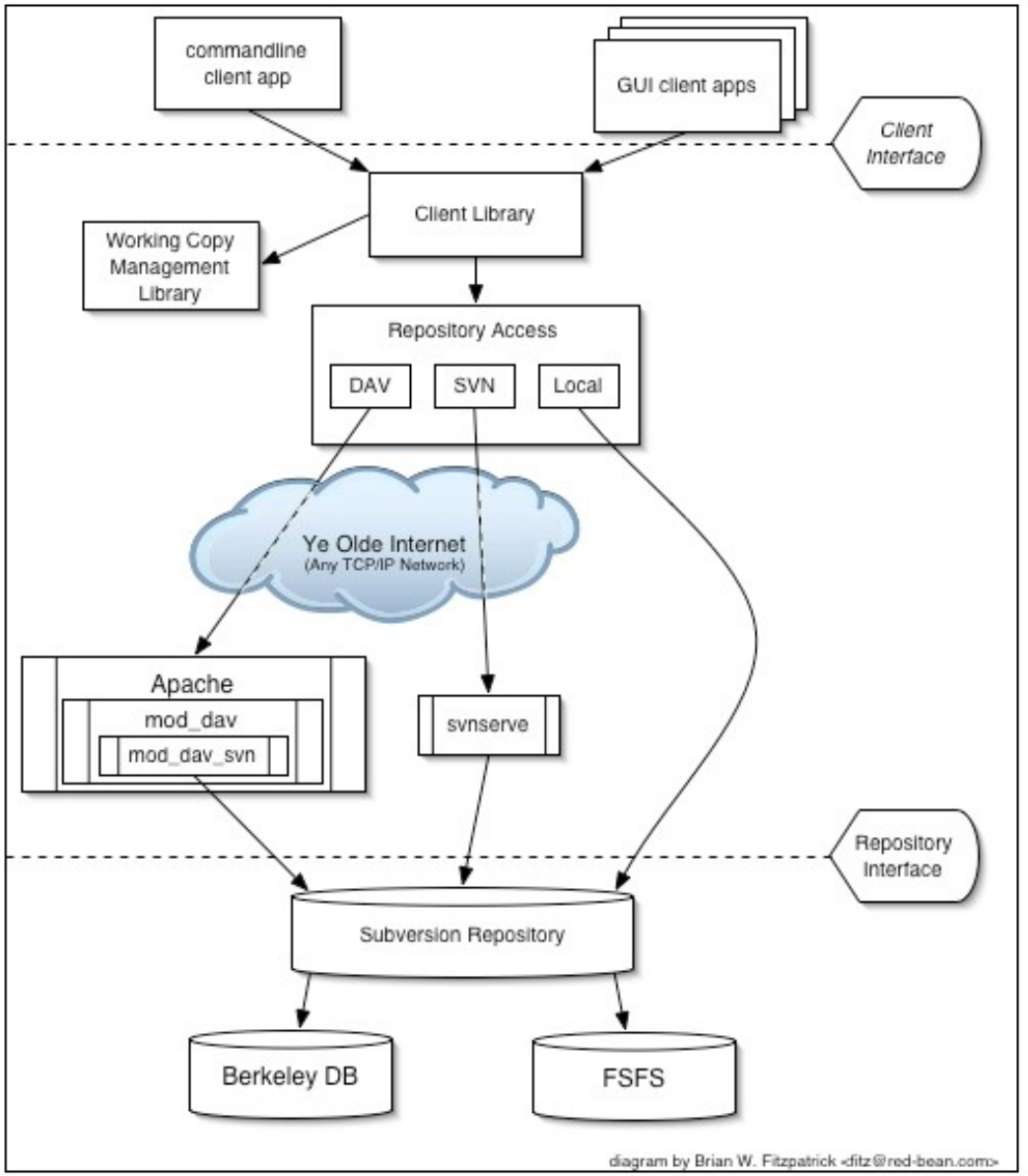
- Características do SVN
  - Escolha do protocolo de redes. É possível escolher entre os diversos protocolos de redes (HTTP, HTTPS, etc).
  - Manipulação consistente de dados. Usando um algoritmo de diferenciação de binários, o SVN consegue trabalha tanto com arquivos textos, como com arquivos binários.
  - Extensibilidade. Implementado como uma coleção de bibliotecas compartilhadas de C. Extremamente manutenível e usável por outras aplicações e linguagens.

Controle de Versões e Desenvolvimento Colaborativo de Software

- COMPARATIVO CVS x SVN
  - CVS
    - PRO: utilizado por um grande número de usuários. Muito bem documentado. Suportado por muitas empresas e profissionais.
    - CONTRAS: permite o *commit* somente de arquivos. É mais lento. É direcionado apenas para arquivos de texto e requer informações especiais para outros tipos de arquivos.
  - SVN
    - PRO: suporte para comandos padrão de diferenciação. Previne o usuário de acidentalmente utilizar o comando *commit* para conflito de arquivos. Forma mais fácil de versionamento e retorno de versões.
    - CONTRAS: necessita de muito mais espaço do que o CVS. Ainda não é amplamente utilizado.

Controle de Versões e Desenvolvimento Colaborativo de Software

- SVN



Controle de Versões e Desenvolvimento Colaborativo de Software

- Problemas atuais do SVN:
  - Operação para renomear: usa *copy* e *delete*, mas ainda utiliza o mesmo nome nas versões antigas.
  - O SVN armazena cópias adicionais de dados na máquina local, que podem ser um problema para projetos ou arquivos muito grandes, ou se desenvolvedores estão trabalhando em muitos *branches* simultaneamente.

Referências

- http://www.cvshome.org
- http://tortoisesvn.tigris.org/
- http://subversion.tigris.org
- http://tortoisesvn.tigris.org
- http://svnbook.org/
- http://software.ericsink.com/scm/source_control.html
- http://www.wush.net/
- http://svnbook.red-bean.com/

Make

- O que o Make faz?
  - Make é uma ferramenta que rastreia as dependências temporais entre módulos (de programas, texto, etc).
  - Permite gerenciar grandes programas ou grupos de programas.
  - Permite a recompilação apenas dos módulos que necessitam ser recompilados (porque foram alterados ou porque necessitam de módulos que foram alterados).

Make

- Como o Make funciona?
  - O Make rastreia as dependências entre arquivos e executa os comandos definidos para manter a consistência.
  - As dependências são descritas em um arquivo texto, geralmente chamado de Makefile, contendo diretivas (ou comandos) Make.

Make

- O que é o arquivo Makefile?
  - É um arquivo capaz de fazer determinadas operações mediante regras determinadas pelo programador.
  - O objetivo de um Makefile é facilitar o trabalho do programador, geralmente quando se usa muitas rotinas em linhas de comando.
  - Compilação/link edição de programas, geração de manfiles e geração de arquivos postscripts/tex/pdf/dvi são exemplos dessas rotinas.

Make

- Quais são as razões para se usar o Makefile?
  - Quando existe a necessidade de compilar somente o código-fonte que foi modificado desde a última compilação, ao invés de ter que compilar todo o código-fonte.
  - Um projeto que consiste de muitos arquivos fontes, pode ter comandos de compilação complexos e longos. Com o Make, isto pode ser reduzido.
  - Durante a programação, algumas vezes é necessários opções de compilação especializadas que raramente são usadas e são difíceis de lembrar, com o Make isto pode ser reduzido.
  - Manter a consistência do ambiente de desenvolvimento.
  - Automatizar o processo de construção porque o Make pode ser chamado de um shell script ou de uma tarefa da cron.

- Como o Makefile trabalha?
  - Um Makefile trabalha usando regras. Dentre essas regras inclui-se:
  - Teste de dependências.
  - Comparação entre datas (para verificar se houve modificações).

Make

- Como um Makefile pode ser executado?
  - O Makefile é executado através do comando `make`.
  - Existem diversas opções para serem usadas com o comando `make`.
  - Exemplos:
    - `make`
    - `make all`
    - `make install`
    - `make clean`
    - `make cleanall`
    - `make cleandist`
    - `make tar`

Make

- Como um Makefile pode ser executado?
  - O arquivo normalmente é chamado de makefile ou Makefile. A última opção é a mais comum, pois é mais fácil de se ver com um comando `ls`.
  - É possível empregar outro nome, desde de que você use a opção -f. Exemplo: `make -f` outronome.

Make

- Como escrever um Makefile?
  - A escrita de um Makefile é bem simples e possui a seguinte estrutura:

    alvo:dependências

        comandos
  - Dependências são coisas ou códigos-fontes necessários para realizar o alvo.
  - O alvo normalmente é um arquivo executável.
  - Os comandos são as instruções necessárias para realizar o objetivo.
  - Ele só executa o COMANDO se existir as DEPENDÊNCIAS para criar o ALVO.

Gestão de bugs

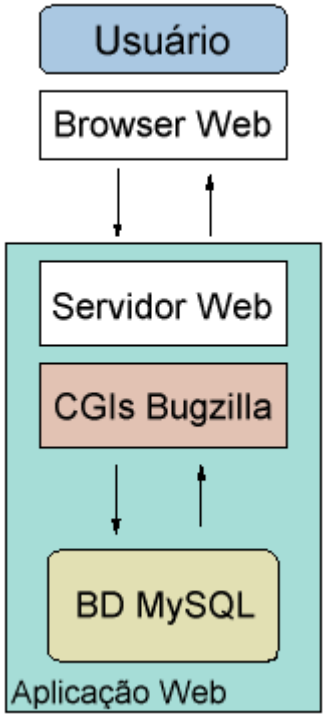
- Bugzilla
  - É uma ferramenta WEB para gestão de erros.
  - É um projeto de software livre, mantido por voluntários, e que faz parte do projeto Mozilla.
  - Esta aplicação permite que indivíduos ou grupos de programadores acompanhem os relatórios de erros ou pedidos de melhoramentos.
  - É o sistema para a informação de novos bugs, para o monitoramento dos bugs já informados e notificações sobre o status de desenvolvimento de correções para cada falha.
  - Suporta uma grande comunidade de usuários e desenvolvedores.
  - Permite avaliar, em tempo real, o estado dos diversos 'bugs' e o progresso geral, de um produto.
  - Proporciona um espaço para todas as etapas do acompanhamento de defeitos, da criação à validação.

Gestão de bugs

- Uso do Bugzilla
  - Um sistema de gestão de erros permite aos desenvolvedores manter contato com os seus clientes e revendedores e comunicar problemas eficientemente durante o desenvolvimento.
  - O acompanhamento de defeitos ajuda a reduzir os custos ao fornecer suporte responsável, bases de conhecimento de suporte telefônico e um sistema para acompanhar problemas de hardware e software comuns.
  - Aumenta a produtividade e a responsabilidade dos empregados ao fornecer um ritmo de trabalho documentado e um feedback positivo para boas performances.

Gestão de bugs

- Características técnicas do Bugzilla
  - Aplicação WEB convencional.
  - Escrita em HTML e Perl/CGI.
  - Utiliza base de dados relacional (MySQL).
  - Roda em diversas plataformas.



Gestão de bugs

- Processo Bugzilla
  - O que caracteriza um bug?
    - Cada bug tem um conjunto de informações associadas importantes.
    - As consultas às bases são feitas com estas informações.
    - Componentes:
      - Identificador do bug;
      - Produto/Componente;
      - Plataforma;
      - Comentários;
      - Anexos;
      - etc.

Gestão de bugs

- Processo Bugzilla
  - Apesar de existir alguma flexibilidade, o processo de funcionamento é predominantemente seqüencial.
  - O processo do Bugzilla é composto das seguintes etapas ou estados:
    - Criação
    - Confirmação
    - Atribuição
    - Resolução
    - Verificação
    - Fechamento
    - Reabertura

Gestão de bugs

- Processo Bugzilla – Criação de um bug
  - Um usuário descobre um defeito – algo inesperado – ou tem uma idéia ou opinião sobre algo a ser desenvolvido.
  - Os bugs podem ser classificados em defeitos, melhorias ou metabugs.
  - O bug será adicionado à lista de bugs e deverá ser processado.
  - O bug será atribuído a algum desenvolvedor ou passará, diretamente, para o estado de resolvido.
  - A qualidade de um projeto pode ser avaliada através dos bugs e das suas soluções.

Gestão de bugs

- Processo Bugzilla – Confirmação de um bug
  - Alguns bugs são claros, outros sutis e pouco freqüentes
  - Após a descoberta do bug, é preciso verificar se ele é relevante e se já foi informado.
  - Surge aí a necessidade de um líder de projeto ou um trabalho em equipe, através de votações.

Gestão de bugs

- Processo Bugzilla – Atribuição do bug
  - Nesta etapa, define-se quem será o responsável por solucionar o bug.
  - Nesta fase, ocorre a discussão em cima da forma como será reparado ou atendida a solicitação.
  - Patches, testes e imagens são anexados a um bug.
  - Nesta etapa, o bug pode ser resolvido ou atribuído a outra pessoa, tornando-se um novo bug.

Gestão de bugs

- Processo Bugzilla – Resolução do bug
  - Resulta da implementação e integração de uma resolução aceitável na base de testes.
  - Pode ser necessária uma revisão formal e aprovação para se considerar um bug como resolvido.
  - Se o Controle de Qualidade (CQ) verificar a solução como adequada, o bug vai para o estado de Verificado. Caso a solução não seja adequada, o bug será reaberto.

Gestão de bugs

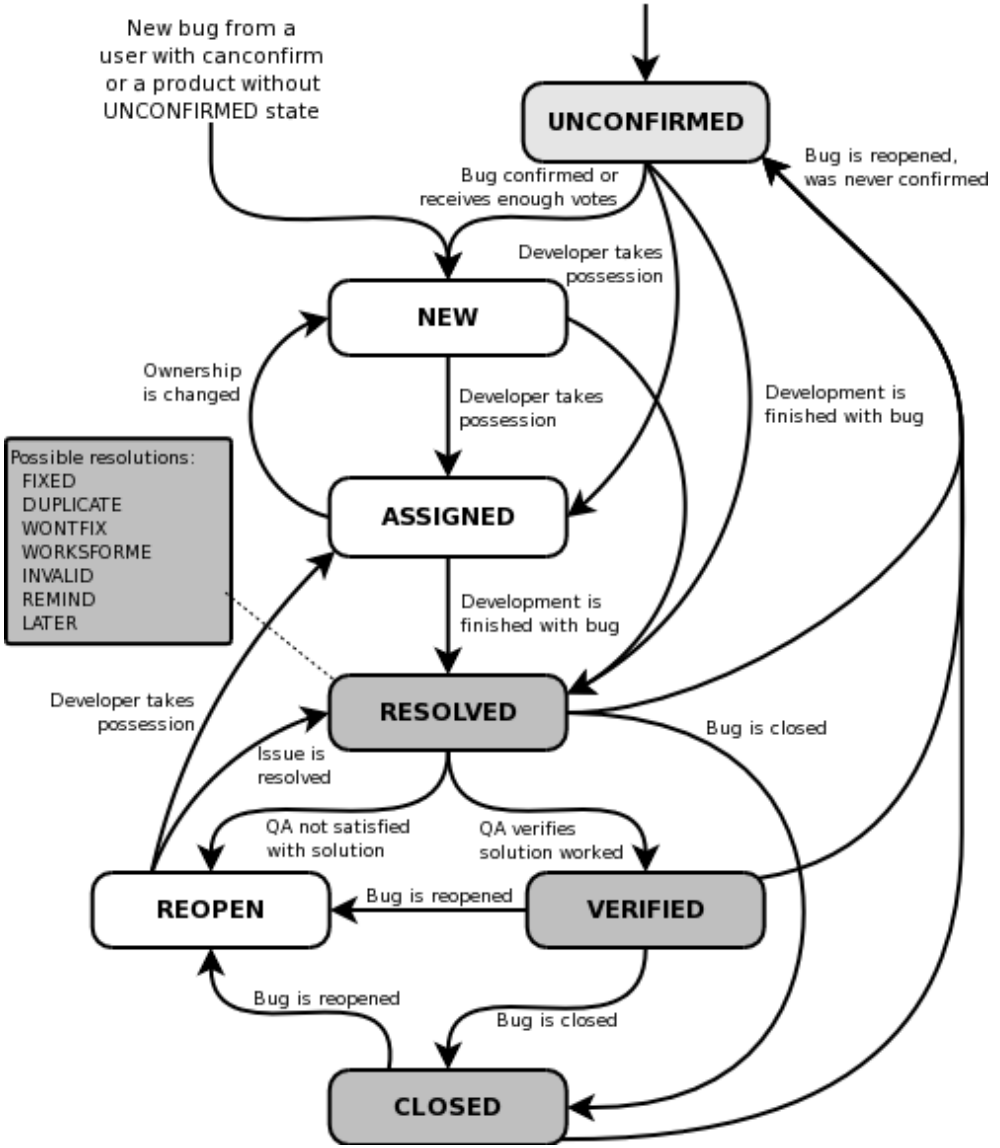
- Processo Bugzilla – Verificação
  - O líder ou a equipe responsável verifica se o bug está realmente resolvido na árvore principal e o marca como verificado.
  - Daqui em diante, há menos interação dos usuários desenvolvedores e mais gerência do bug.
  - Muitos bugs ficam como resolvidos até que o responsável possa revê-los.

Gestão de bugs

- Processo Bugzilla – Fechamento
  - Ocorre quando a próxima versão “release” de um produto é lançada com a resolução do bug.
  - Aqui o bug é fechado e a resolução é considerada como correta.
- Processo Bugzilla – Reabertura
  - Ocorre quando um defeito resolvido reaparece ou não foi devidamente resolvido.

Gestão de bugs

- Máquina de Estados do Bugzilla



Gestão de bugs

- Resoluções dos bugs
  - Corrigido (*fixed*)
    - Uma resolução é confimada para a árvore e testada.
  - Inválido (*invalid*)
    - O problema descrito não é um bug.
  - Sem correção (*wontfix*)
    - O problema descrito é um bug e não será corrigido.
  - Futuro (*later*)
    - O problema descrito é um bug, mas não será resolvido nesta versão do produto.
  - Possível (*remind*)
    - O problema descrito é um bug e que provavelmente não será corrigido nesta versão do produto, apesar de ser possível.

Gestão de bugs

- Resoluções dos bugs
  - Duplicado (*duplicate*)
    - O problema é um bug já descrito.
    - Para marcar um bug como duplicado é necessário o número do bug já descrito.
  - Não-repetitivo (*worksform*)
    - Apesar de todas as tentativas, não foi possível reproduzir o comportamento do bug.

Gestão de bugs

- Administração do Bugzilla
  - Gerência dos usuários
    - Criação do usuário padrão
      - Após a instalação do Bugzilla, você será solicitado a criar o administrador do sistema (nome e senha).
      - Se você quiser criar outros usuários administradores, basta criar um usuário e configurar as suas permissões.
      - Você pode criar novos usuários através do link users, na página principal.

Gestão de bugs

- Administração do Bugzilla
  - Produtos
    - Produtos pertencem à categoria mais alta e tendem a representar os produtos do mundo real.
    - Para criar um novo produto
      - Selecione “products” do rodapé.
      - Clique em “Add” na tabela à direita.
      - Preencha o formulário com o nome do produto e a sua descrição.
  - Componentes
    - Componentes são subpartes de um produto.
    - Para criar um novo componente
      - A partir de Produtos, clique no link “Edit components”.
      - Para adicionar um novo componente, clique em “Add” na tabela à direita.
      - Preencha o formulário com o nome do novo componente, a descrição e o responsável pela correção.

Gestão de bugs

- Administração do Bugzilla
  - Versões
    - Versões são revisões do produto, assim como Produto 1.0, Produto 1.1 e assim por diante.
    - Para criar uma nova versão
      - A partir da tela “Edit Products”, seleciona “Edit Versions”.
      - Digite a nova versão.
  - Milestones
    - Milestones são metas de datas em que você planeja ter um bug corrigido.
    - Para criar um novo miliestone
      - Selecione “Edit Miliestones” da página “Edit Product”.
      - Clique em “Add”.

Gestão de bugs

- Plataforma
  - Este campo indica a plataforma de hardware em que ocorreu o bug.
  - Macintosh, PC, Sun, HP e DEC são exemplos de plataformas.
- Sistema Operacional
  - Este campo indica o sistema operacional em que ocorreu o bug.
  - Linux, Sun, Mac e Windows são exemplos de sistemas operacionais listados.
- Prioridade
  - Este campo descreve a importância e a ordem em que os bugs devem ser corrigidos.
  - O campo é utilizado pelos programadores para priorizar o seu trabalho.
  - As prioridades vão de P1 (mais importante) a P5 (menos importante).

Gestão de bugs

- Severidade
  - Este campo indica o impacto do bug.
  - A severidade pode ser:
    - Blocker – Bloqueia o desenvolvimento e/ou o trabalho de testes.
    - Critical - Crashes, perdas de dados ou uso excessivo de memória.
    - Major – Perda de funções principais.
    - Normal – Problema comum.
    - Minor – Perda de funcionalidades menos importantes ou outros problemas em que existe facilidade em ser contornado.
    - Trivial – Problemas simples de serem resolvidos, como uma palavra com grafia errada.
    - Enhancement – Pedido de melhorias.

Gestão de bugs

- Como reportar um bug no Debian?
  - Pontos importantes a serem considerados:
    - Não reportar diversos bugs não relacionados – especialmente em pacotes não relacionados.
    - Verificar se o relatório de bug já não foi reportado anteriormente. Se já existir, é possível comentar o bug já cadastrado.
    - Caso você não descubra qual pacote contém o problema, envie uma mensagem para a lista de discussão debian-user@lists.debian.org.
  - Usando uma ferramenta automática para relatar bugs
    - O programa **reportbugs** auxilia no processo de relatar bugs.

Gestão de bugs

- Como reportar um bug no Debian?
  - Enviando o relatório via e-mail
    - As mensagens devem ser enviadas para submit@bugs.debian.org
    - Como em qualquer mensagem, você deve incluir uma linha do assunto de forma clara e descritiva no cabeçalho principal.
    - O assunto que você usar será usado como o título inicial do bug no sistema de gerenciamento.
    - Você precisa colocar um pseudo-cabeçalho no início do corpo da mensagem. Exemplo: Package: <nome do pacote>
    - A segunda linha deve conter a versão do pacote. Exemplo: Version: <versão do pacote>

Gestão de bugs

- Como reportar um bug no Debian?
  - Devem ser incluídos no relatório:
    - O texto exato e completo de quaisquer mensagens impressas ou logadas. Isto é muito importante !
    - Exatamente o que você digitou ou fez para demonstrar o problema.
    - Uma descrição do comportamento incorreto : exatamente qual comportamento você estava esperando e o que você observou. Uma transcrição de uma sessão de exemplo é uma boa maneira de mostrar isso.
    - Uma correção sugerida, ou até mesmo um patch, caso você tenha preparado um.
    - Detalhes da configuração do programa com o problema. Inclua o texto completo dos arquivos de configuração do programa em questão.

Gestão de bugs

- Como reportar um bug no Debian?
  - Devem ser incluídos no relatório:
    - As versões de quaisquer pacotes dos quais o pacote com problemas dependa.
    - A versão de kernel você está usando (digite `uname -a`).
    - A sua biblioteca C compartilhada (digite `ls -l` /lib/libc.so.6 ou `dpkg -s` libc6 | `grep` Version).
    - Quaisquer outros detalhes sobre seu sistema Debian, caso pareçam aproriados. Por exemplo, caso você tenha um problema com um script Perl você pode informar a versão do binário 'perl' (digite `perl -v` ou `dpkg -s` perl | `grep` ^Version).
    - Detalhes apropriados do hardware em seu sistema. Caso você esteja reportando um problema com um controlador de dispositivo por favor liste todo hardware em seu sistema, uma vez que problemas são normalmente causados por conflitos de endereços de I/O e IRQ.

Gestão de bugs

- Como reportar um bug no Debian?
  - Exemplo

```
To: submit@bugs.debian.org
From: diligent@testing.linux.org
Subject: Hello diz `goodbye'

Package: hello
Version: 1.3-16

Quando eu invoco `hello' sem argumentos a partir de um shell comum
      o mesmo imprime `goodbye' ao invés do esperado `hello, world'.
      Aqui está uma transcrição :
$ hello
goodbye
$ /usr/bin/hello
goodbye
$

Sugiro que a string de saída, em hello.c, seja corrigida.

Estou usando GNU/Linux 2.2, kernel 2.2.17-pre-patch-13
e libc6 2.1.3-10.
```

Gestão de bugs

- Como reportar um bug no Debian?
  - Níveis de severidade
    - Caso um relatório seja de um bug particularmente sério ou seja meramente uma requisição de um novo recurso você pode definir o nível de severidade do bug no momento em que o reporta.
    - Porém, isto não é requerido e os desenvolvedores atribuirão um nível de severidade apropriado para seu relatório caso você não o faça.
    - Para atribuir um nível de severidade, coloque uma linha como essa no pseudo-cabeçalho : pseudo-cabeçalho.
    - Exemplo: Severity: <severidade>

Gestão de bugs

- Como reportar um bug no Debian?
  - Atribuição de TAGs
  - Você pode definir tags em um bug quando relata o mesmo.
  - Por exemplo, caso você esteja incluindo um patch junto a seu relatório de bug você pode desejar definir a tag patch.
  - Porém, isso não é requerido e os desenvolvedores irão definir tags em seu relatório quando apropriado.
  - Para definir tags, inclua um alinha como essa no pseudo-cabaçalho
  - Exemplo: Tags: <tags>

Gestão de bugs

- Como reportar um bug no Debian?
  - Caso o sistema de gerenciamento de bugs não saiba quem é o mantenedor do pacote relevante, o mesmo encaminhará o relatório para debian-bugs-dist.
  - Ao enviar para maintonly@bugs ou nnn-maintonly@bugs, você deve certificar-se de atribuir o relatório de bug ao pacote correto, colocando um Package correto no topo de um envio original de um relatório ou usando o serviço control@bugs para (re)atribuir o relatório apropriadamente primeiro caso o mesmo já não esteja correto.

Qualidade de Software[1]

- A sociedade está cada vez mais dependente de sistemas computacionais.
- Os sistemas dependem cada vez mais de software.
- O mau funcionamento do software pode ter altos custos e prejuízos para os seus usuários.

1. Adaptado de notas de aula do prof. Ricardo Anido
http://www.ic.unicamp.br/~ranido/mc626

Qualidade de Software

- O que é qualidade de software?
  - Conjunto de características que devem ser alcançadas em um determinado grau para que o produto atenda às necessidades de seus usuários.
  - Totalidade de características de uma entidade que lhe confere a capacidade de satisfazer às necessidades explícitas e implícitas [NBR ISO 1994].
  - Conformidade a:
    - requisitos funcionais e de desempenho;
    - padrões e convenções de desenvolvimento pré-estabelecidos;
    - atributos implícitos que todo software desenvolvido profissionalmente deve possuir.

Qualidade de Software

- Características de qualidade
  - Operação
    - Características operacionais
      - Correção
      - Confiabilidade
      - Integridade
      - Eficiência
  - Revisão
    - Habilidade para ser alterado
      - Manutenibilidade
      - Flexibilidade
      - Testabilidade
  - Transição
    - Adaptabilidade para novos ambientes
      - Portabilidade
      - Reusabilidade
      - Interoperabilidade

Qualidade de Software

- Com relação ao uso do produto (características operacionais):
  - Correção: o quanto um programa satisfaz a sua especificação e cumpre os objetivos visados pelo cliente.
  - Confiabilidade: o quanto um programa executa a função pretendida com a precisão exigida.
  - Eficiência: a quantidade de recursos computacionais e de código exigida para que um programa execute sua função.
  - Integridade: o quanto o acesso ao software ou aos dados por pessoas não autorizadas pode ser controlado .
  - Usabilidade: o quanto de esforço é necessário para aprender, preparar a entrada e interpretar a saída de um programa.

- Com relação às alterações do produto (habilidade para ser alterado):
  - Manutenibilidade: o quanto de esforço é necessário para localizar e eliminar erros em um programa.
  - Flexibilidade: o quanto de esforço é necessário para modificar um programa.
  - Testabilidade: o quanto de esforço é necessário para testar um programa a fim de garantir que ele execute a função pretendida.

Qualidade de Software

- Com relação às alterações do produto (habilidade para ser alterado):
  - Portabilidade: o quanto de esforço é necessário para transferir um programa de uma plataforma de hardware e/ou software para outra.
  - Reusabilidade: o quanto um programa (ou partes dele) pode ser reutilizado em outros programas.
  - Interoperabilidade: o quanto de esforço é necessário para se acoplar um programa a um outro.

- Características de Qualidade

Funcionalidade	O software satisfaz às necessidades explícitas e implícitas do usuário?
Confiabilidade	O software, durante um período de tempo, funciona de acordo com as condições pré-estabelecidas?
Usabilidade	O software é fácil de usar?
Eficiência	O software não desperdiça recursos?
Manutenibilidade	O software é fácil de alterar?
Portabilidade	O software é facilmente adaptável a diferentes plataformas

Qualidade de Software

- Funcionalidade: o software satisfaz às necessidades explícitas e implícitas do usuário?
  - Adequação: propõe-se a fazer o que é apropriado?
  - Acurácia: gera resultados corretos ou conforme acordado?
  - Interoperabilidade: é capaz de interagir com os sistemas especificados?
  - Conformidade: está de acordo com normas e convenções previstas em leis, normas e descrições similares?
  - Segurança de acesso: evita acesso não autorizado, acidental ou deliberado acesso a programa e dados?

Qualidade de Software

- Confiabilidade: o software, durante um período de tempo, funciona de acordo com as condições pré-estabelecidas ?
  - Maturidade: com que freqüência apresenta falhas?
  - Tolerância a falhas: ocorrendo falhas, como ele reage?
  - Recuperabilidade: é capaz de recuperar os dados após uma falha?

Qualidade de Software

- Usabilidade: o software é fácil de usar?
  - Inteligibilidade: é fácil entender os conceitos utilizados?
  - Apreensibilidade: é fácil de aprender a usar?
  - Operacionalidade: é fácil de operar e controlar a operação?

Qualidade de Software

- Eficiência: o software não desperdiça recursos?
  - Comportamento em relação tempo: qual é o tempo de resposta e de processamento?
  - Comportamento em relação aos recursos: quanto recurso usa?
  - Durante quanto tempo?

Qualidade de Software

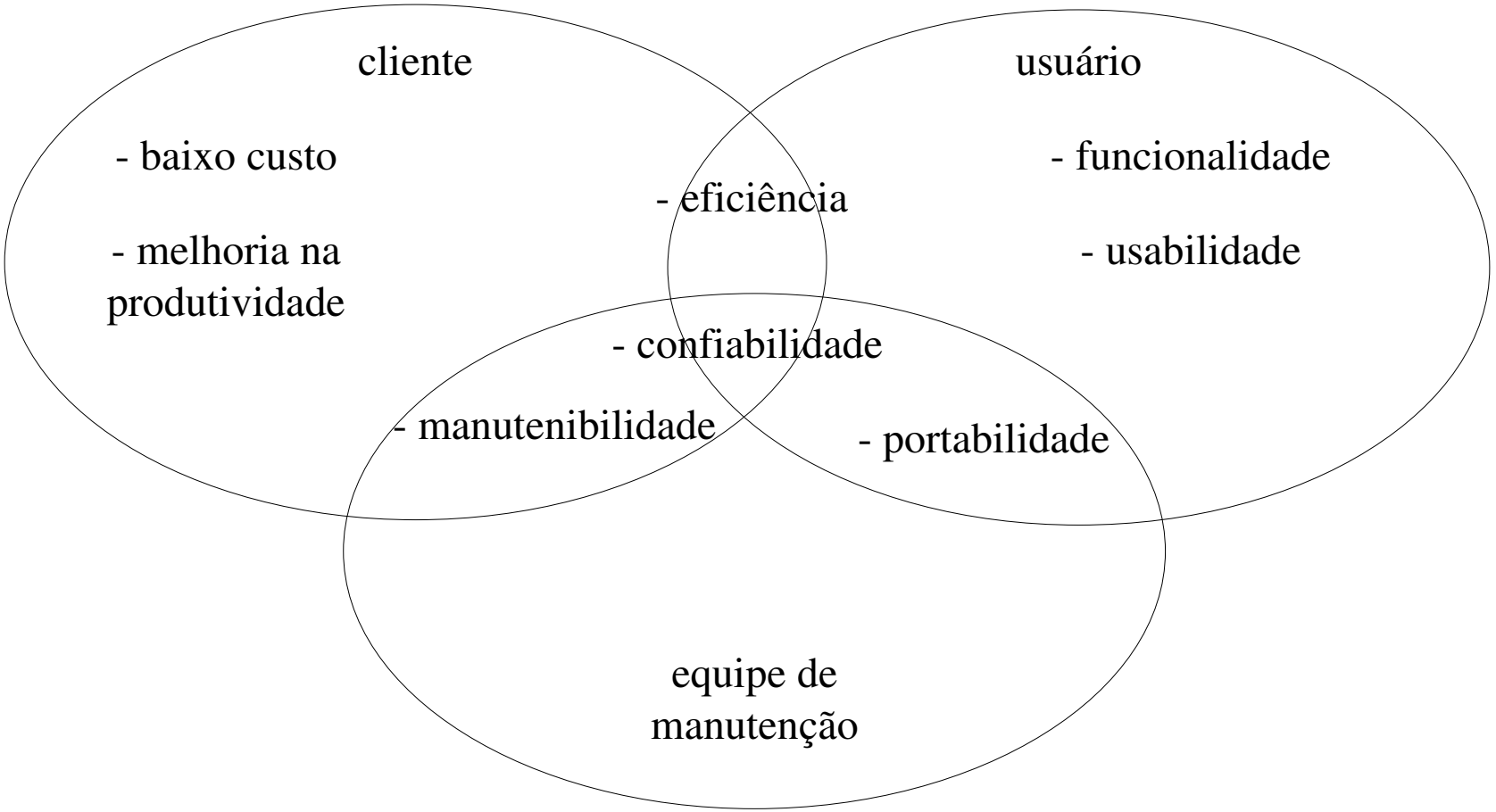
- Manutenibilidade: o software é fácil de alterar?
  - Analisabilidade: é fácil encontrar um erro quando ocorre?
  - Modificabilidade: é fácil modificar e remover erros?
  - Estabilidade: há grandes riscos de erros quando se faz alterações?
  - Testabilidade: é fácil testar quando se faz alterações?

- Portabilidade: o software é facilmente adaptável a diferentes plataformas?
  - Adaptabilidade: é fácil adaptar a outras plataformas sem aplicar outras ações ou meios além dos fornecidos para esta finalidade no software considerado?
  - Capacidade para instalar: é fácil instalar em outras plataformas?
  - Capacidade para substituir: é fácil substituir por outro software?
  - Conformidade: está de acordo com padrões e convenções de portabilidade?

- Qualidade versus tipo de software

Qualidade de Software

- Qualidade versus tipo de software



- Avaliação da qualidade
  - Objetivos:
    - aprimorar o processo de desenvolvimento e, em conseqüência, melhorar a qualidade do produto resultante.
    - avaliar a qualidade do produto visando emitir documento oficial sobre a qualidade de um software e sua conformidde em relação a uma norma ou padrão.
    - adquirir um software, com o intuito de escolher o produto mais adequado dentre um conjunto de produtos selecionados.

Qualidade de Software

- Aprimoramento do produto de software
  - Iniciativas que visam melhorias do processo de software:
    - SEI/CMM (*Capability Maturity Model*), modelo desenvolvido pelo Instituto de Engenharia de Software (SEI) da Universidade Carnegie- Mellon, EUA, visando dar às organizações diretrizes sobre como aprimorar o processo.
    - ISO/SPICE (*Software Process Improvement & Capability dEtermination*), cujo objetivo é gerar normas.
    - ISO/IEC para a avaliação de processos de software.
    - Norma ISO/IEC 12207, define um processo de ciclo de vida do software.
    - Norma ISO/IEC 9000-3, apresenta diretrizes para a aplicação da ISO 9001 (voltada para indústria), por empresas que desenvolvem software, para o processo de desenvolvimento e manutenção de software.

- Avaliação da qualidade do produto
  - Algumas normas:
    - ISO/IEC 9126 (NBR 13596), define as características de qualidade de software que devem estar presentes em todos os produtos.
    - ISO/IEC 12119, estabelece os requisitos de qualidade para pacotes de software e instruções para teste, considerando esses requisitos.
    - ISO/IEC 14598-5, define um processo de avaliação da qualidade de produto de software.

- Avaliação da qualidade do produto
  - Como fazer:
    - Organismos de certificação:
      - No Brasil, para fornecer o certificado ISO 9000, existem empresas credenciadas pelo INMETRO.
    - Avaliar in-house:
      - Utilizar equipe multidisciplinar com especialistas da área de tecnologia e especialistas da área que se utilizará do software (i.e., que vão olhar para o software a partir do ponto de vista do cliente - grupo de Garantia da Qualidade do Software).
    - Contratar empresas para avaliação
      - Existem empresas que fazem avaliação do software mas, por não serem credenciadas pelo INMETRO, não emitem certificado oficial. São no entanto mais acessíveis e mais ágeis que os organismos credenciados.

- Entraves à qualidade [IEEE610.12-1990

erro (engano)	ação humana que produz um resultado incorreto	**mistake**
falha	incorreção em um passo, processo ou definição de dados; manifestação no software de um engano cometido pelo desenvolvedor	**fault (bug)**
erro	diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do software	**error**
defeito	incapacidade de fornecer o serviço conforme especificado	**failure**

- Por que surgem as falhas?
  - Alterações:
    - As alterações degradam a estrutura do software, tornando-o cada vez mais difícil de alterar.
  - Tempo:
    - Com o tempo, os custos da implementação de alterações aumenta, e a capacidade do sistema em prestar os serviços esperados diminui.
  - Complexidade:
    - Difícil de desenvolver: um único desenvolvedor não é capaz de entender o sistema como um todo.
  - Difícil de usar
    - Difícil de entender: código incompreensível, falta de documentação.

Qualidade de Software

- Garantia da qualidade do software
  - Definição de um arcabouço para se atingir a qualidade do produto de software.
  - Padrão sistemático e planejado de ações que são exigidas para garantir a qualidade do software.
  - Visa responder às seguintes questões:
    - O software atende às características de qualidade desejadas ?
    - O desenvolvimento do software foi conduzido conforme os padrões pré-estabelecidos ?
    - As disciplinas técnicas cumpriram adequadamente seus papéis como parte da atividade de Garantia da Qualidade ?
  - Norma ISO/IEC 14598 pode ser usada para definir o processo de avaliação.

Prototipação

- Prototipação é o desenvolvimento rápido de um sistema.
- Protótipos são uma valiosa ferramenta no projeto de aplicações.
- O uso principal é ajudar clientes e desenvolvedores a entender os requisitos do sistema.
  - Os usuários podem experimentar com um protótipo para ver como o sistema apóia o seu trabalho.
  - O protótipo pode revelar erros e omissões nos requisitos.
- A prototipação pode ser considerada como uma atividade de redução de risco, pois reduz os riscos com os requisitos.

Prototipação

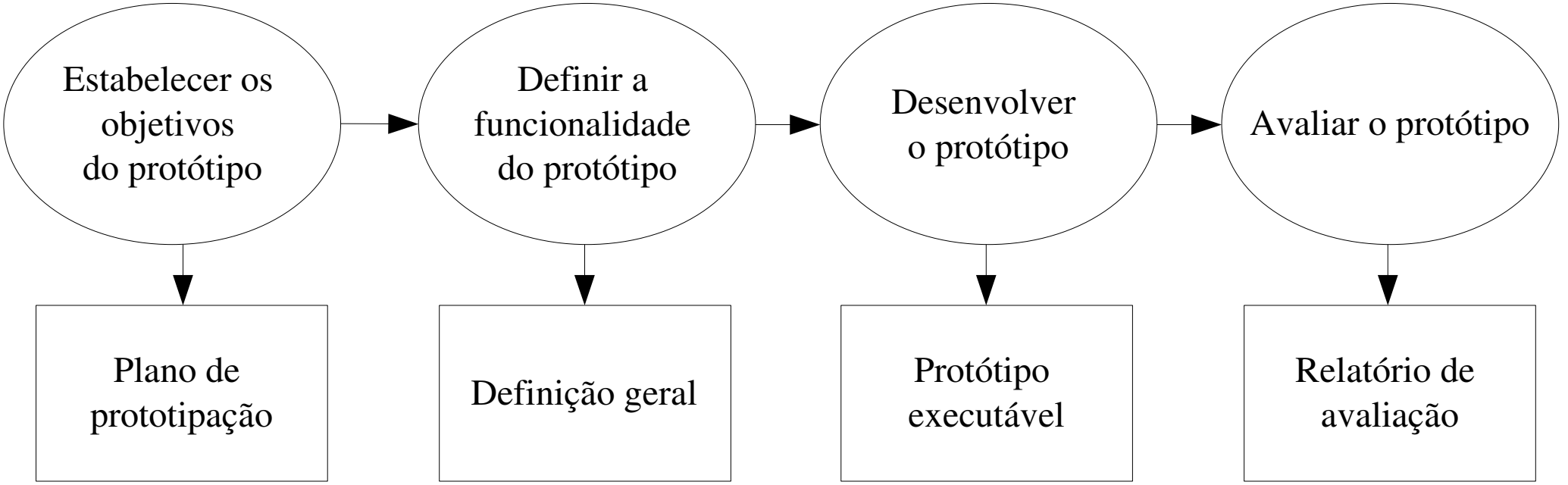
- Benefícios da prototipação:
  - Resolve possíveis desentendimentos entre clientes e desenvolvedores.
  - Serviços omitidos podem ser detectados e serviços confusos podem ser esclarecidos.
  - Um sistema estará disponível no início do processo.
  - O protótipo pode servir como uma base para derivar uma especificação do sistema.
  - O protótipo pode apoiar o treinamento dos usuários e os testes do sistema.
  - Melhoria na usabilidade do sistema.
  - Manutenibilidade melhorada.

Prototipação

- Benefícios da prototipação:
  - Qualquer um pode participar do designer de protótipos em papel – não há necessidade de conhecimento de nenhuma ferramenta especial.
  - Não existem barreiras impostas aos usuários quanto a fazerem críticas ao protótipo porque ele é de natureza temporária.
  - Redução de custos de desenvolvimento.
  - Redução do esforço de desenvolvimento.
  - Melhorias são feitas mais rapidamente, antes de investir esforços em codificação de programas.

Prototipação

- Processo de prototipação



Prototipação

- Tipos de prototipação
  - Prototipação evolucionária 
    - Uma abordagem do desenvolvimento de sistemas em que um protótipo é inicialmente produzido e refinado por meio de uma série de estágios até o sistema final.
    - O objetivo desta abordagem é entregar um sistema funcional para o cliente final.
    - O desenvolvimento começa com os requisitos que são melhor entendidos.

Prototipação

- Tipos de prototipação
  - Prototipação descartável 
    - Um protótipo é produzido para ajudar a descobrir problemas nos requisitos e depois será descartado.
    - O sistema então é desenvolvido, usando algum outro processo de desenvolvimento.
    - O objetivo desta abordagem é validar ou derivar os requisitos do sistema.
    - O processo inicia com aqueles requisitos que estão menos compreendidos.

Prototipação

- Prototipação evolucionária 
  - Deve ser usada para sistemas em que a especificação não pode ser desenvolvida antecipadamente.
  - Baseia-se em técnicas que permitem rápidas iterações de sistema.
  - A verificação é impossível, uma vez que não existe especificação. Validar significa demonstrar a adequação do sistema.
  - Entrega acelerada do sistema.
  - Compromisso do usuário com o sistema.

Prototipação

- Prototipação evolucionária 
  - Especificação, projeto e implementação são intercaladas.
  - O sistema é desenvolvido como uma série de incrementos que serão entregues ao cliente.
  - São usadas técnicas para desenvolvimento rápido de sistemas, tais como ferramentas CASE e de 4a Geração.
  - As interfaces com o usuário são normalmente desenvolvidas usando um toolkit de desenvolvimento de GUI (*Graphical User Interface*).

Prototipação

- Problemas da prototipação evolucionária 
  - Problemas de gerenciamento.
  - Problemas de manutenção.
  - Problemas contratuais.

Prototipação

- Prototipação descartável 
  - Usada para reduzir os riscos na análise dos requisitos.
  - O protótipo é desenvolvido a partir de uma especificação inicial, entregue para testes, sendo posteriormente descartado.
  - O protótipo descartável NÃO deve ser considerado como um sistema final. Razões:
    - Algumas características importantes podem ter sido deixadas de lado.
    - Não há especificações para manutenção a longo prazo.
    - O sistema pode ser fracamente estruturado e de difícil manutenção.

Prototipação

- Problemas na prototipação descartável 
  - Os desenvolvedores podem ser pressionados a entregar um protótipo descartável como um sistema final.
  - Isso não é recomendável:
    - Pode ser impossível ajustar o protótipo para satisfazer os requisitos não funcionais.
    - O protótipo não possui documentação.
    - A estrutura do sistema sofrerá degradação com as mudanças feitas durante o desenvolvimento.
    - Os padrões de qualidade normais da organização podem não ter sido aplicados.

Prototipação

- Escolha da linguagem de programação
  - Qual é domínio da aplicação do problema?
  - Que tipo de interação é exigida com o usuário?
  - Que ambiente de suporte acompanha a linguagem?
  - Partes diferentes do sistema podem ser programadas em linguagens diferentes?

Prototipação

- Montagem de componentes e aplicações
  - Os protótipos podem ser criados rapidamente a partir de um conjunto de componentes reusáveis, acrescidos de um mecanismo para colar esses componentes.
  - O mecanismo de composição deve incluir recursos de controle e um mecanismo para a comunicação entre os componentes.
  - A especificação do sistema deve levar em conta a disponibilidade e a funcionalidade dos componentes existentes.

Prototipação

- Prototipação com reuso
  - Desenvolvimento em nível de aplicação
    - Aplicações inteiras são integradas com o protótipo, de tal maneira que a sua funcionalidade possa ser compartilhada.
  - Desenvolvimento em nível de componente
    - Componentes individuais são integrados dentro de um framework padrão para implementar o sistema.

Prototipação

- Programação visual
  - Ferramentas como Glade e Qtdesigner suportam programação visual, onde o protótipo é desenvolvido pela criação de uma interface com o usuário a partir de itens padrão e associação de componentes com esses itens.
  - Existe uma vasta quantidade de bibliotecas livres para apoiar este tipo de desenvolvimento.
  - Eles podem ser ajustados para suprir os requisitos específicos da aplicação.

Prototipação

- Problemas com a programação visual
  - Difícil de coordenar o desenvolvimento baseado em equipes.
  - Não há uma arquitetura explícita do sistema.
  - Dependências complexas entre partes do sistema podem gerar problemas de manutenibilidade.

Prototipação

- Prototipação de interface com o usuário
  - É impossível pré-especificar a apresentação de uma interface com o usuário de uma forma efetiva.
  - O desenvolvimento de interfaces do usuário (IU) consome uma parte crescente dos custos globais de desenvolvimento de sistemas.
  - Os itens gerados de IU podem ser usados para desenhar a interface e simular a sua funcionalidade com componentes associados a entidades da interface.

Prototipação

- Algumas diretrizes
  - A prototipação deve ser utilizada quando os usuários puderem participar ativamente do projeto.
  - Os desenvolvedores devem ter experiência em prototipação ou um treinamento adequado.
  - Os usuários envolvidos no projeto devem ter experiência em prototipação ou serem educados no seu uso e finalidade.
  - Os protótipos devem se tornar parte integrante do sistema final apenas se os desenvolvedores tiverem acesso às ferramentas de suporte à prototipação.
  - Se os processos de experimentação e aprendizado forem necessários, antes que exista um compromisso completo com um projeto, a prototipação pode ser utilizada com sucesso.
  - A prototipação poderá não ser necessária se o desenvolvedor já está familiarizado com a linguagem a ser utilizada no projeto do sistema.

Prototipação

- Pontos principais
  - Um protótipo pode ser usado para dar aos usuários finais uma impressão concreta dos recursos do sistema.
  - A prototipação está se tornando mais usada para desenvolvimento de sistemas onde o desenvolvimento rápido é essencial.
  - A prototipação descartável é usada para compreender os requisitos do sistema.
  - Na prototipação evolucionária, o sistema é desenvolvido pela evolução de uma versão inicial até a versão final.
  - O desenvolvimento rápido dos protótipos é essencial. Isso pode exigir deixar funcionalidades de fora ou relaxar requisitos não funcionais.

Prototipação

- Pontos principais
  - Técnicas de prototipação incluem o uso de linguagens de altíssimo nível, programação de banco de dados e a construção de protótipos a partir de componentes reusáveis.
  - A prototipação é essencial para partes do sistema tais como a IU, que não podem ser efetivamente pré-especificadas. Os usuários devem estar envolvidos na avaliação do protótipo.

Perguntas de revisão

- Explique a importância de uma ferramenta de controle de versões no processo de desenvolvimento de software?
- É possível renomear um arquivo diretamente do repositório? Se for possível, esta operação é recomendável? Justifique a sua resposta.
- Por que o CVS tem dificuldade em controlar os arquivos binários?
- Qual é a vantagem no uso de um sistema de gestão de bugs?
- Qual é a finalidade do Make?
- Qual é o conteúdo de um arquivo Makefile?

Perguntas de revisão

- Qual é a importância da análise da qualidade de software?
- Qual é a importância das normas para a melhoria dos processos e produtos de software?
- Comente os principais critérios para avaliação da qualidade de software?
- Quais são as principais vantagens no uso da prototipação?
- Quais são as principais dificuldades no uso da prototipação?
- Qual é a diferença entre prototipação evolucionária e prototipação descartável?

Referências Bibliográficas

(1) Anido, Ricardo. Notas de aula. Disponível em: http://www.ic.unicamp.br/~ranido/mc626.

(2) Caetano, Cristiano. *CVS – Controle de Versões e Desenvolvimento Colaborativo de Software*. Editora Novatec, 2004.

(3) Fogel, Karl & Bar, Moshe. *Open Source Development with CVS*. 3rd. Disponível em: http://cvsbook.red-bean.com.

(4) Pronus Engenharia de Software. Disponível em: http://www.pronus.eng.br. Acesso em dezembro de 2005.

(5) Reis, Christian. Bugzilla: Acompanhamento de defeitos no projeto Mozilla. Disponível em:http://www.async.com.br/~kiko/bugzilla. Acesso em dezembro de 2005.

(6) Sommreville, Ian. *Software Engineering*. 6th Edition. Addison Wesley, 2001.

(7) SourceForge. Disponível em: http://sourceforge.net. Acesso em janeiro de 2006.