Legal Notices

There are various product or company names used herein that are the trademarks, service marks, or trade names of their respective owners, and March Hare Software Limited makes no claim of ownership to, nor intends to imply an endorsement of, such products or companies by their usage.

This document and all information contained herein are the property of March Hare Software Limited, and may not be reproduced, disclosed, revealed, or used in any way without prior written consent of March Hare Software Limited.

This document and the information contained herein are subject to confidentiality agreement, violation of which will subject the violator to all remedies and penalties provided by the law.

LIMITED WARRANTY.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, March Hare Software Limited AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THIS DOCUMENT, AND ANY ADVICE OR RECOMMENDATION CONTAINED IN THIS DOCUMENT.

NO OTHER WARRANTIES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL March Hare Software Limited OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE FOLLOWING DOCUMENTATION INCLUDING ANY RECOMMENDATION OR ADVICE THEREIN, EVEN IF March Hare Software Limited HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, March Hare Software Limited’s ENTIRE LIABILITY UNDER ANY PROVISION OF THIS DOCUMENT INCLUDING ANY RECOMMENDATION OR ADVICE THERIN SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE DOCUMENT OR £5.00; PROVIDED.

© Copyright 2004 - 2010 March Hare Software Ltd

march-hare.com
sales@march-hare.com
Promotion model

A promotional model for managing changing documents or software is very common in many organizations. Using a promotion model it is easy to ensure that the correct people only “see” the documents and objects that are at the appropriate level of the promotion process.

Example 1

A government department is required to draft a new piece of legislation. The document evolves using a clearly defined model:
- Draft
- Legal Review
- Ministerial Approval
- Parliament

The document may go through several revisions during this entire process, however each time it is promoted to the next level it cannot be changed at that level except by authorised people. For example the public servant who authors the document cannot change the Legal review copy.

Frequently the document is only ever changed at the lowest level of the hierarchy, however meta data may be added at different levels. For example the legal review may wish to tag certain paragraphs as needing changes, or they may make the changes themselves.

It is important that the department that prints the legislation to table before parliament cannot accidentally print a copy that has not been through the entire promotion process.

Example 2

A software development company releases software four times a year. The software for each release evolves using a clearly defined model:
- Development
- Review
- Test
- Integration Test
- Production

The software goes through several revisions during this entire process, however each time it is promoted to the next level it cannot be changed at that level except by authorised people. For example the programmer who authors the bug fix cannot change the Test copy.

The software is only ever changed at the lowest level of the hierarchy, however meta data may be added at different levels. For example the code review may wish to tag certain functions as needing changes to comply with company coding standards, or they may make the changes themselves.

It is important that the users only run the Production version of the software and do not accidentally run another version. If a CD is produced and shipped out, the distribution department must have a fail safe way to ensure that they cannot accidentally deploy an untested release.
Mixed model

Mixing up the development models
Most organizations implementing CVS will want to use a mixture of the branching and promotional models. For example:
- Develop Version 1 on the Trunk
- When Version 1 is feature complete Branch Version 1 Maintenance
- Begin work on Version 2 on the Trunk
- Finalise Version 1 on the Branch
- Promote Version 1 Branch to Test
- Fix Version 1 bugs on the Version 1 Branch and promote to test again

Using a mixture of the techniques can lead to a balance with the strengths of both systems.

Regardless of the choices you make CVS is always capable of reapplying the changes between any two revisions to another revision – whether it is on the same branch or a different one.

Patch management – getting fixes to customers
The CM model that you choose may be heavily influenced by business concerns such as needing to deliver fixes to current software while also allowing development to continue on newer versions. This is known as patch management.

Service Packs
Often organizations deliver stable combinations of patches to customers as a service pack – or a point release. For example, version 1 is released and several bugs are found and fixed, and three months after the first release version 1.1 (or version 1 service pack 1) is released containing all bug fixes.

This service pack example can also be described in a time line similar to:
- Version 1
- Fix bug 1
- Fix bug 2
- Fix bug 3
- Release Version 1.1

Patches
What happens if one of those bugs seriously effects the customer’s current day-to-day operations? In this case it may be necessary to release one of the bug fixes immediately. Since most customers are not affected the release version 1.1 is not created earlier – but a patch is produced.

This patching example can also be described in a time line similar to:
- Version 1
- Fix bug 1
- Fix bug 2
- Release Patch 1
- Fix bug 3
- Release Version 1.1
However the customer does not want any changed functionality except their bug fix. In the list above it can be seen that bug fix 1 and bug fix 2 have already been completed and a combined Release Patch 1 contains both fixes. A typical development environment may have made 50 changes, and the customer does not want the responsibility of testing all those fixes to get their environment working again.

The software company needs to balance the customer’s requirements with their own. Specifically they need to ensure that the change is only made once but it is then applied to release 1.1 and also later to release 2.0.

Careful consideration of the business requirements is necessary to design an effective CM process. CVS is technically capable of supporting all these decisions.

In this particular example the choices the software organization would make would depend largely on the frequency and the billing methods. If these individual patch releases are rare or are charged to the customer then they will be designed as an exception. If they are common then the SCCM solution will be designed with some level of automation for reliability and reproducibility.

Bug fixing workflow with Promote

A typical request from QA managers is how to track what files have changed for each job or bug. The following section describes an example of this:

- Make a change to source code using Eclipse and commit to repository.
- View changes via Bugzilla
- View changes via SQL Query on Audit database or Bugzilla database
- View changes via CVS Suite command line client
Make changes to source code using Eclipse and commit to repository
In a typical commercial software development environment a programmer is given a task number, job number or bug number before beginning work. This bug number is then used to track the changes from requirements gathering through to release:

The developer can use the synchronize view to visually compare the changes in the eclipse workspace with the CVS Suite Server repository:
When the developer is finished with the changes they can commit them back to the repository and link them to the job / bug using the bug number:

View changes in Bugzilla
The QA manager can see at a glance in Bugzilla all the files modified by any bug:

**Bug 5820**

<table>
<thead>
<tr>
<th>Summary:</th>
<th>Fix this sample problem reported by customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product:</td>
<td>Eclipse Sample</td>
</tr>
<tr>
<td>Component:</td>
<td>User Sample</td>
</tr>
<tr>
<td>Reporter:</td>
<td>Arthur Barnett <a href="mailto:arthur.barnett@march-hare.com">arthur.barnett@march-hare.com</a></td>
</tr>
<tr>
<td>Assignee:</td>
<td>sample default assignee <a href="mailto:sample@march-hare.com">sample@march-hare.com</a></td>
</tr>
<tr>
<td>Status:</td>
<td>ASSIGNED</td>
</tr>
<tr>
<td>Severity:</td>
<td>enhancement</td>
</tr>
<tr>
<td>Priority:</td>
<td>P3</td>
</tr>
<tr>
<td>Version:</td>
<td>1.0</td>
</tr>
<tr>
<td>Hardware:</td>
<td>All</td>
</tr>
<tr>
<td>OS:</td>
<td>All</td>
</tr>
<tr>
<td>Time tracking:</td>
<td>Orig. Est. 0.0 Actual Hours 0.0 Hours Worked 0.0 Hours Left 0.0 %Complete 0.0 Gain 0.0</td>
</tr>
<tr>
<td>Deadline:</td>
<td>j2sample/src/customer.java revision 1.2</td>
</tr>
<tr>
<td>Attachments:</td>
<td>j2sample/src/customer.java revision 1.2</td>
</tr>
</tbody>
</table>

Fix this sample problem reported by customer

**Comment 1**

CVS Commit by user Administrator@March-Hare.com
j2sample/src/customer.java 1.2 (+2,-2)
this resolves the problem bug 5820 by fixing the message displayed to the user.
The diffs can even be code reviewed directly in Bugzilla:

View changes via SQL Query on Audit database or Bugzilla database

An SQL client (e.g., Microsoft Excel) can query the audit database or the Bugzilla database to find all the files changed by a bug or all of the bugs affected by a file.

Promote to test or production by bug number

The CVS Suite tools all understand the bug number, and you can promote to any defined promotion level using a bug number, e.g., promoting to test:

Firstly the QA manager creates a disk area to perform the promote:

Drag the promotion level to a secure location:
Secondly use the right click promote menu

Right click the new promotion directory and choose the bug number to promote – there is no need to know what files are affected by the bug – all files affected are promoted:

Lastly use the right click commit menu

Right click the new promotion directory and choose commit – you can use the same bug number, or a new bug number (eg: a new bug that combines several other bug fixes into a release or service pack):
You can use the revision graph feature to visually confirm the promotion:
Set and Manage ACL’s

CVS Suite Studio is the recommended tool graphical front end for CVSNT and provides a range of powerful commands for performing the most common CVSNT tasks in a clear and simple user interface.

To use ACL’s you first must configure the ACLMode, see the Administration section for more information. To deny access by default set `aclmode = normal` (if you do not set this CVSNT will allow access by default). Once the repository wide default is set, you can then use CVS Suite Studio to add or remove permissions to folder on the server: