CSC 440/540: Software Engineering
Continuous Integration
Topics

1. Continuous integration
2. Configuration management
3. Types of version control
   1. None
   2. Lock-Modify-Unlock
   3. Copy-Modify-Merge
Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.
CI Processes

Continuous Integration

1. Source Code
2. Version Control System
3. Source Code Build

Development Team

4. Static Code Analysis
5. Run Automated Unit Tests

Publish Reports

6. Code Coverage Analysis

Run Automated Functional Tests

7. Built Artefact

Provision and Deploy to Test Environment

Set-up Test ‘Fixtures’

Slide #4
If it hurts, do it more often
CI Practices

- Maintain a single source repository
- Keep everything in repository (configuration, docs, etc.)
- Automate the build
- Make your build self-testing
- Everyone commits to mainline each day
- Every commit should be built (integrated)
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get latest deliverables
- Everyone can see results of latest build
- Automate deployment
CI Benefits

- Errors are found faster
- Fewer errors are encountered
- Releases become easier
- Releases are available for QA and comparisons
- Avoid last minute chaos near release date
Open Source CI Tools

Build software based on
- Version control system commits
- Scheduled times
- User request from IDE

Can build on
- Multiple hardware configurations
- Multiple OS and middleware configs

Notifies developers about
- Build failures
- Test failures

Configuration management refers to the process by which all artifacts relevant to your project, and the relationships between them, are stored, retrieved, uniquely identified, and modified.
CM enables developers to

- Exactly reproduce the application environment, including OS version, patch level, network configuration, middleware, and all configuration files.
- Easily make incremental changes to any environmental aspects and deploy those changes to any or all development, testing, or production machines.
- View the history of configuration changes, including what the change was, who made the change, when the change was made, and why the change was made.
Application Configuration

- How do you represent your configuration information?
- How do deployment scripts access it (DB, file, LDAP, registry, web service)?
- How does configuration vary between environments, applications, and versions of applications?
- Is configuration information such as passwords or keys properly secured?
Testing Configuration

1. Verify all references to external services are usable.
   1. Are external servers up and reachable?
   2. Can we communicate with them with needed protocols?

2. Test functionality of services.
   1. Can app retrieve or write information (e.g., login to database server and read data from a known table)?
What is Version Control?

Recording changes made to a file or set of files.

- Backups only record a snapshot of a file tree.
- Version control typically also records metadata:
  - What changes were made
  - Who made the changes
  - Why the changes were made

A version control system is like a time machine, allowing you to retrieve your file from points in time in the past.
Why Version Control?

1. To enable collaborative editing and sharing of files.
2. To maintain a history of a project, tracking what changes were made and who made them.
3. To manage multiple releases of a software project.
Using Version Control

- Keep absolutely everything in version control
  - Requirements, design docs, build instructions, tests
  - External library versions and locations
  - Configurations: OS, network, application
- Check in regularly to mainline (master, trunk, etc.)
  - Regularly means *at least* every work day.
- Use meaningful commit messages
  - Let’s team members know what to expect.
  - Let’s you track down bugs when they appear later.
Terms

Repository
Place where systems stores tree of your files, with their revision history.

Client
Programs that connect to repository to read or write these files.

Check-in
Writing changes to the repository

Check-out
Reading a version of your file set from the repository

Working Copy
User’s local copy of the repository, including any changes they have not yet checked in
No Version Control

Problem: how can two users share files, but not overwrite each other’s changes

- User A and User B each read file Spam
- Both edit the file in their editor
- User A saves his changes
- User B saves his changes

Result:
- User B’s version of the file is saved
- User A’s changes are lost
Manual Version Control

- Create copies of each file or directory with version numbers.
- Ad-hoc, still doesn’t avoid devs stomping on each others’ changes.
Lock-Modify-Unlock

Solution: Only let one person modify a file at a time

- User A locks file Spam
- User A reads file Spam
- User B attempts to lock file Spam, but is denied access
- User A saves his changes
- User A unlocks file Spam.

Result:
- User A’s changes are saved.
- User B is prevented from accessing the file.

Example: RCS
Lock-Modify-Unlock Problems

What if user A forgets to unlock the file?
- User B waits til user A returns to office to ask for lock
- or User B asks Admin to unlock file.

What if user A’s and user B’s changes don’t overlap?
- User A wants to edit top of file, User B the bottom
- User B has to wait for User A to finish

What if dependencies exist between multiple files?
- User A modifies file Spam, User B modifies file Eggs.
- The class in Spam uses the interface of the class in Eggs.
- What if the changes are incompatible?
Solution: copy, modify, then merge changes
- User A checks out a working copy of Spam
- User B checks out a working copy of Spam
- Users A and B edit their own copies
- User A checks in Spam
- User B attempts to check in Spam, but receives an out of date error
  - Solution 1: User B tells client to merge changes.
  - Solution 2: Changes conflict (same lines were modified). User B manually merges changes, then tells client that conflict is resolved.
Copy-Modify-Merge Advantages

Users can work in parallel, without waiting

- Most concurrent changes can be automatically merged
- Time to resolve conflicts is less than time required to wait for locks
Centralized vs. Distributed VC

Computer A
- Checkout
  - file

Central VCS Server
- Version Database
  - version 3
  - version 2
  - version 1

Server Computer
- Version Database
  - version 3
  - version 2
  - version 1

Computer B
- Checkout
  - file

Computer A
- file

Computer B
- file

Version Database
- version 3
- version 2
- version 1
## Version Control Software

<table>
<thead>
<tr>
<th>Open Source Software</th>
<th>Proprietary Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch</td>
<td>Bitkeeper</td>
</tr>
<tr>
<td>CVS</td>
<td>IBM Clearcase</td>
</tr>
<tr>
<td>Git</td>
<td>Perforce</td>
</tr>
<tr>
<td>RCS</td>
<td>UNIX SCCS</td>
</tr>
<tr>
<td>Mercurial</td>
<td>MS TFVC</td>
</tr>
<tr>
<td>Subversion</td>
<td></td>
</tr>
</tbody>
</table>

Additional client software exists for many systems
- GUI interfaces (WinCVS, tkCVS, tortoiseCVS)
- Editor integration (emacs, vim)
- IDE integration (Eclipse, VisualStudio)
Adding Files to a Git Repo

1. **Stage**: Tell git which files constitute a single change.
2. **Commit**: Save change to repository with a descriptive comment.
Interacting with Remote Repos

Local
- working directory
- staging area
- local repo

Remote
- remote repo

- git add
- git commit
- git push
- git fetch
- git checkout
- git merge
Git History is Nonlinear

- `origin/ks/xser32.dll`: Updated `msys-1.0.dll` to `MSYS-g670d78f` Signed-off-by: Johannes Schindelin
  - Hack: `sh.exe` -- link to `xser32.dll` instead of `user32.dll`
  - Hand edited on Linux in hte[1]. This is how.

- `xser32.dll`: Fake `user32.dll` like stub (for `sh.exe` not to load `user32.dll`)
  - Signed-off-by: Johannes Schindelin

- `/src/xser32/release`: cd to correct directory signed-off-by: Johannes Schindelin

- `origin/ks/msys`: Updated `msys-1.0.dll` to `MSYS-g670d78f`
  - Signed-off-by: Johannes Schindelin

- `xser32.dll`: Don't pull `user32.dll` & friends just to detect whether right alt should be used as main
  - Issue #535: support UNC paths for the "Git Bash Here" explorer menu item.
  - As pointed out in the bug report, Associate .sh files to be run with bash in the installer
  - This adds a checkbox to the installer to support this.
  - Update Inno Setup to version 5.4.1 (Unicode)

- Merge remote-tracking branch 'work/gettext' into devel

- `Git-1.7.4-preview20110204`: Updated to Git v1.7.4
  - Signed-off-by: Pat Thoyts <patthoyts@use
References


