Principles of Software Construction: Objects, Design, and Concurrency

Managing change

Charlie Garrod  Bogdan Vasilescu
Administrivia

- Homework 6 checkpoint deadline (Monday, April 30th)
- Homework 6 due Wednesday, May 2nd
- Final exam Monday May 7th 5:30-8:30 PH 100
- Review session Saturday May 5th
Key concepts from Carnival
Scenario

A customer wants a bug fix to software version 8.2.1, which was released 2 years ago.

How to make sure we can fix, build, and release?
Configuration Management (CM)

Pressman:
“is a set of tracking and control activities that are initiated when a software engineering projects begins and terminates when software is taken out of operation”

Configuration management originates from the 50s, when spacecraft failures resulted from undocumented changes.
The Modern World

Complex Source Languages, Directories, Dependencies Source Files Data

Versioning Branching

Many Tools Compilers, Linkers, Code gens, Translators

Traceability Scalability Configuring

Complex Systems Executables Libraries Dependencies Config Files Data

Consistency Flexibility

Cloud Deployment Distributed Data

Virtualization Load Balancing Security

Diverse User Base Many Platforms Product Lines Shared Libraries

Security Localization
The Modern World

- Which Version?
- How to recreate?
- How to fix?
- Where to apply the fix?
- How/when to Redistribute?
Components of Modern CM

Version Control + Workflows

Build Managers

Package Managers

Deployment Managers + VMs/Containers

App Markets + Update Managers
Configuration management as safety net

• Doing software development without CM is “working without a safety net”

• Configuration management refers to both a process and a technology
  – The process encourages developers to work in such a way that changes to code are tracked
    • changes become “first class objects” that can be named, tracked, discussed and manipulated
  – The technology is any system that provides features to enable this process
Activity

In pairs, discuss other reasons why we may want configuration management

Some reasons
• “Works for me”; difficulty onboarding new devs, installing dependencies
• Audits: Discovery request on changes made to system (e.g. no tracking in breathalyzer lawsuit)
• Product lines (Home, Business, Professional); different customer types.
• Markets: Asia, Europe, America (Language + feature variance)
• Platforms: Windows, Mac OS, Android, iOS
CM is a key part of DevOps (more later)
Components of Modern CM

Version Control: Branches/Forks/Workflows
Task and Build managers
Build machines, virtual environments (dev stacks)
Package managers
Containers, VMs, in the Cloud
Deployment – Infrastructure as Code.
Data migration

Other issues: orchestration, inventory, compliance
Config. management vs version control

- “version control” is “versioning” applied to a single file while “configuration management” is “versioning” applied to collections of files
VERSION CONTROL WITH GIT
A. GOAL: COLLABORATION ON FILES
Collaborating on Files

• How to exchange files
  – Send changes by email
  – Manual synchronization at project meeting
  – All files on shared network directory

• Permission models
  – Each file has an owner; only person allowed to change it
  – Everybody may change all files (collective ownership)
Concurrent Modifications

• Allowing concurrent modifications is challenging
• Conflicts (accidental overwriting) may occur

Common strategies
  – Locking to change
  – Detecting conflicts (optimistic model)
Change Conflicts

Two users read the same file

They both begin to edit their copies

Harry publishes his version first

Sally accidentally overwrites Harry’s version

source „Version Control with Subversion“
Locking Files

Practical problems of locking model?
Merging (1/2)

Two users copy the same file

They both begin to edit their copies

Sally publishes her version first

Harry gets an “out-of-date” error
Merging (2/2)

Harry compares the latest version to his own

Repository

Read

A

A″

Harry

Sally

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″

A″
Example

```java
import java.util.LinkedList;
public class Stack<T> implements Cloneable {
    private LinkedList<T> items = new LinkedList<T>();
    public void push(T item) {
        items.addFirst(item);
    }
    public T pop() {
        if(items.size() > 0) return items.removeFirst();
        else return null;
    }
}
```
Example

```java
import java.util.LinkedList;
public class Stack<T> implements Cloneable {
    private LinkedList<T> items = new LinkedList<T>();
    public void push(T item) {
        items.addFirst(item);
    }
    public int size() {
        return items.size();
    }
    public T pop() {
        if(items.size() > 0) return items.removeFirst();
        else return null;
    }
}
```
Einführung in die Softwaretechnik

Example

```java
import java.util.LinkedList;
public class Stack<T> implements Cloneable {
    private LinkedList<T> items = new LinkedList<T>();
    public void push(T item) {
        items.addFirst(item);
    }
    public int size() {
        return items.size();
    }
    public T pop() {
        if(items.size() > 0) return items.removeFirst();
        else return null;
    }
    public T top() {
        return items.getFirst();
    }
    public int size() {
        return items.size();
    }
}
```

System cannot decide order
3-way merge

• File changed in two ways
  – Overlapping changes -> conflicts
  – Merge combines non-conflicting changes from both

• Merging not always automatic
  – diff tool to show changes
  – Manual resolution of conflicts during merge (potentially requires additional communication)

• Automatic merge potentially dangerous
  -> syntactic notion of conflicts

• Merging of binary files difficult

• In practice: most merges are conflict free
B. GOAL: RELEASE MANAGEMENT
Challenge:

• Refer to concrete consistent versions of the project (code and all dependencies and infrastructure)

• Why?
  – Parallel development of independent features
  – Bug fixes for old releases; patches
  – Variants for different customers
  – Traceability and accountability of changes (provenance)
Branching

• Parallel copies of the source tree
• Can be changed independently, versioned separately, and merged later (or left separate)
• Often used for exploratory changes or to isolate development activities
• Many usage patterns, common:
  – Main branch for maintenance OR main development
  – New branches for experimental features; merge when successful
  – New branches for nontrivial maintenance work
  – Branches for maintenance of old versions
Release management with branches
Release cycle of Facebook’s apps
Variants and Revisions

• **Revision** replaces prior revision (temporal)
• **Variant** coexists with other variants
• **Version** describes both
• **Release**: Published and named version

<table>
<thead>
<tr>
<th></th>
<th>V1.0</th>
<th>V1.1</th>
<th>V2.0</th>
<th>V3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base system (Windows)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Linux variant</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Server variant</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Extension for customer A</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Extension for customer B</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Semantic Versioning for Releases

• Given a version number MAJOR.MINOR.PATCH, increment the:
  – MAJOR version when you make incompatible API changes,
  – MINOR version when you add functionality in a backwards-compatible manner, and
  – PATCH version when you make backwards-compatible bug fixes.

• Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

http://semver.org/
Managing variants

• Branching for variants does not scale well
• Requires special planning or tooling

• Many solutions
  – Configuration files
  – OO polymorphism
  – Preprocessors
  – Build systems
  – DSLs
  – Software product lines
  – ...

```plaintext
/* common parts */
...
/* dependent on operating system */
#if (OS == Unix)
...
#elif (OS == VMS)
...
#else
...
#endif
...```
C. TYPES OF VERSION CONTROL
Centralized version control

• Single server that contains all the versioned files
• Clients check out/in files from that central place
• E.g., CVS, SVN (Subversion), and Perforce

Distributed version control

• Clients fully mirror the repository
  – Every clone is a full backup of all the data
• E.g., Git, Mercurial, Bazaar

SVN (left) vs. Git (right)

- SVN stores changes to a base version of each file
- Version numbers (1, 2, 3, ...) are increased by one after each commit

- Git stores each version as a snapshot
- If files have not changed, only a link to the previous file is stored
- Each version is referred by the SHA-1 hash of the contents

Which files to manage (both types)

• All code and noncode files
  – Java code
  – Build scripts
  – Documentation
• Exclude generated files (.class, ...)
• Most version control systems have a mechanism to exclude files
  (e.g., .gitignore)
Activity

• In pairs, discuss advantages and disadvantages of centralized (e.g., SVN) vs decentralized (e.g., git) version control
D. GIT BASICS

Graphics by https://learngitbranching.js.org
git commit
git branch newImage
git commit
git checkout newImage; git commit
Activity: Make a new branch named bugFix and switch to that branch
1) git merge bugFix
git checkout bugfix; git merge master
Activity:
2) git rebase master

Move work from bugFix directly onto master
git rebase bugFix
To be continued ...
Summary

• Version control has many advantages
  – History, traceability, versioning
  – Collaborative and parallel development
• Locking vs. merging and merge conflicts
• Collaboration with branches
• From local to central to distributed version control