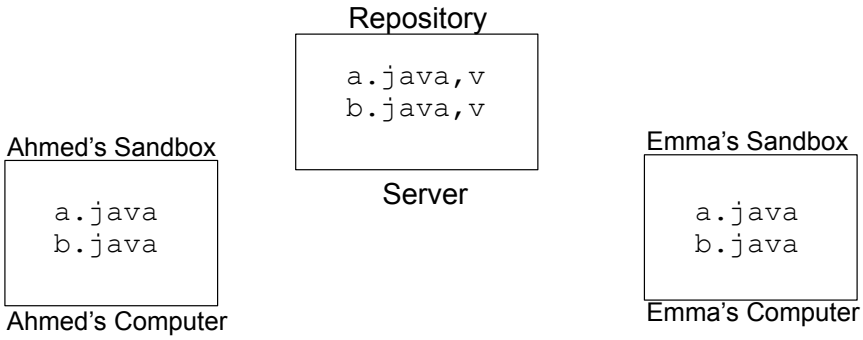
Tuesday, October 18

Configuration Management (Version Control)

How Version Control Works

- Place the official version of source code into a central repository, or database
- Programmers check out a working copy into their personal sandbox or working copy
- When finished and fully tested, programmers check in their code back to the repository



subversion Commands

- checkout
  - Retrieving a file from the repository
- commit
  - Putting changes back into repository
- update
  - Refresh the local or working copy copy with any changes since checkout

- http://www.cs.put.poznan.pl/csobaniec/Papers/svn-refcard.pdf

Conflict Detection and Management

- On check in, official repository copy is compared with new copy
  - Check version number
    - If repository and working copy versions are the same, accept the changes
    - If repository version is newer, reject commit operation.
- Need to update working copy before check in
  - Includes a synchronization step to merge changes from two files (new repository version and modified file from working copy)

Conflict Detection and Management

- Merge algorithm
  - Line by line comparison
  - Changes to different lines are OK
  - Changes to same lines labeled as a conflict
    - Both versions written to the working copy copy
    - Choose which line by editing manually
  - No guarantee of code correctness after merge
- On a successful check in, save new version of files with a set of backwards references to changes
- Can apply detection selectively
  - Binary files only use version numbers or timestamps
  - No conflict detection applied to files that are ignored

Operations are Atomic

- Checkout, update, and commit operations are like database transactions, they are all or nothing
  - In case of network error, machine crash, etc.
  - Avoids partial operations
- Operations work on directories
  - Recursively applied to files
- Every commit transaction is assigned a number, indicating a version of the directory
  - In other words, Version N and Version M of a file may be the same

Updates

- Update command is applied to the directory
- Three possible outcomes for each file
  - U - updated, repository version newer than working copy
  - G - merged, changes don't overlap
  - C - conflict, you need to resolve
- After resolving, tell subversion client that you have done so

What to check in

- Code that compiles cleanly and has been tested
- Don't check in files that are automatically created from others
  - e.g. .class files
- Do check in:
  - Your own little test programs
  - And their expected output
  - Readme files, notes, build logs, etc.
  - Anything else you created by hand

When to check in

- Version control is not a backup system
  - Your computer should have one of those
- Don't check in just because you're taking a break
  - Check in files when they are stable
  - e.g. After adding a new feature
- Or when you have to switch machines
  - e.g. From home to school or vice versa

Comments

- Upon check in, you will have the opportunity to add a comment
  - USE THIS FEATURE!
- You're going to wish you did when you try to revert back to an earlier version

More Uses for Version Control

- Protecting you from yourself
  - Backing out changes
  - Finding where errors were injected
- Working with a team
  - Simultaneous file sharing
  - More complex products
    - Multiple versions, platforms
- Recording an audit trail
  - Hey boss, I've been working…
  - Linus Torvalds vs. SCO

Tagging

- Use tags to label a group of files
  - Makes it easy to check out a release or configuration
- Using a particular version as the baseline for a series of versions
- Reasons for branching
  - Experimental code
  - Bug fix chains
- Happens at the repository, not your working copy

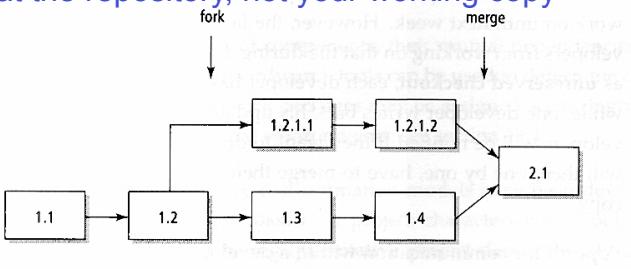


Figure 19.6 Forking and merging of development paths

Thursday, October 20

No Silver Bullet: Essence and Accident in Software Engineering

What is a silver bullet in myth?

What is a silver bullet in software?

“There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability, in simplicity.”

Why might we expect a silver bullet in software?

What are accidental difficulties in software?

What are essential difficulties in software?

"I believe the hard part of building software to be the specification, design and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation."

"We still make syntax errors, to be sure; but they are fuzz compared to the conceptual errors in most systems."

What are the four inherent properties of essential difficulties in software?

Complexity
Conformity
Changeability
Invisibility

Complexity

"…a scaling-up of a software entity is not merely a repetition of the same elements in larger size; it is necessarily an increase in the number of different elements."

"…descriptions of a software entity that abstract away its complexity often abstract away its essence. Mathematics and physical sciences made great strides…by constructing simplified models… It does not work when the complexities are the essence."

Conformity

“The physicist labors on; however, in a firm faith that there are unifying principles to be found…”

“No such faith comforts the software engineer. Much of the complexity he must master is arbitrary complexity, forced without rhyme or reason by the human institutions and systems to which his interfaces must conform.”

Changeability

“In short, the software product is embedded in a cultural matrix of applications, users, laws, and machine vehicles. These all change continually, and their changes inexorably force change upon the software product.”

Invisibility

“The reality of software is not inherently embedded in space. Hence it has not ready a geometric representation in that way that land has maps…”

“In spite of progress in restricting and simplifying the structure of software, they remain inherently unvisualizable.”

What are some of the software technologies that have solved only accidental difficulties?

High-level languages
Unified programming environments
Artificial intelligence
Expert systems
“Automatic” programming
Programming verification

What are the four promising attacks on the conceptual essence?

Buy versus build

Requirements refinement and rapid prototyping

Incremental development-- grow, not build software

Great designers