CSC 7700: Scientific Computing

Module A: Basic Skills

Lecture 6: Best practices for software development / Revision control systems

Dr Frank Löffler

October 27th 2010

Overview

Communication channels

Version control systems

Issue tracker

Documentation format

Summary

Overview

Overview

Software development, or

- Application Development
- Software Design
- Designing Software
- Software Engineering
- Software Application Development
- Enterprise Application Development
- Platform Development

... development of a software product in a planned and structured process.

Overview

Software development involves some combination of stages:

- Market research
- Gathering requirements for the proposed business solution
- Analyzing the problem
- Devising a plan or design for the software-based solution
- Implementation (coding) of the software
- Testing the software
- Deployment
- Maintenance and bug fixing

Collection of stages: software development lifecycle (SDLC).

- Very different methodologies how to combine stages exist.
- Choice of methodology should be project-dependent.

Project Environment / Community

Before starting development: create "project environment":

- Communication channels
- Version control system
- Bug tracker and tasks list
- Documentation format
- Testing tools
- Package management

Communication channels

Communication channels

Dependent on team distribution, consider possibilities like:

- In-person meetings
- Conference phone calls
- Email, especially dedicated mailing lists
- Instant messaging (e.g. IRC)
- VoIP/Videoconferences (e.g. Skype/EVO)

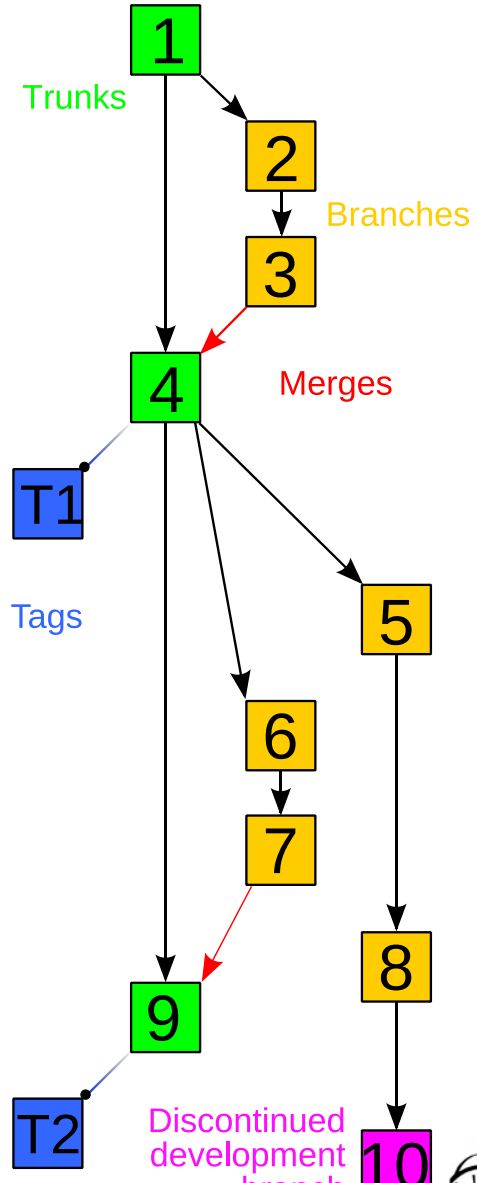
Version control systems

Version control systems

Also known as

- Revision control
- Source control
- Software configuration management (SCM)

Definition: Management of changes to documents, programs, and other information stored as computer files



LSU

Version control systems

Version Control Systems:

- Most commonly stand-alone applications
- Also embedded into various other software, e.g.
  - word processors
  - spreadsheets
  - content management systems
  - wikis

Changes:

- Might be identified by number of letter code
- Termed as “revision number”, “revision level”, or simply “revision”
- Associated with timestamp and changing user
- Can be compared, restored, and with some types of files, merged

Version control systems

Traditionally: centralized model - solve conflicts with one of two methods:

- File locking
  - grant write access to only one checkout at a time
  - benefit: can provide some protection against difficult conflicts
  - drawback: when locked for too long, developers are tempted to bypass RCS
- Version merging
  - Simultaneous edit by multiple developers allowed
  - Merge necessary when transferring change to central server
  - Automatic merging almost only available for simple files: text

Often both options are available, but version merging is most used by far.

Branches - Trunk - Tags

Branch

- Duplication of an object under revision control
- Usually (complete) directory tree

Trunk

- Main, central development branch
- Upstream branch

Tag

- Repository snapshots
- Used especially for releases
- Synonyms: labels, baselines

Some repository systems treat all three identically.

Version control systems - Branches

Branching: duplication of an object under revision control

- Modifications can happen in parallel along multiple branches.
- Branch: also known as trees, streams or codelines
- Originating branch: *parent* branch or *upstream*
- Branch without parent: *trunk* or *mainline*
- In some distributed RCSs: repository $\equiv$ branch
- Branches might be merged, especially into *trunk*
- Branches not intended to be merged usually called *fork*

Distributed revision control

- Peer-to-peer approach
- Theoretically no central repository, only working copies
- Working copies synchronized by exchanging change-sets
- Common operations are fast, because local
- Communication only necessary for exchange with other copies
- In practice, projects tend to have one designated central, "official" copy

Checkouts - Exports - Update - Commits

Checkout

- Act of creating a local working copy from the repository
- User may specify a specific revision or obtain the latest
- Also: synonym for working copy

Export

- Act of obtaining a the files from the repository
- Similar to checkout, but creates clean directory tree without version control metadata.

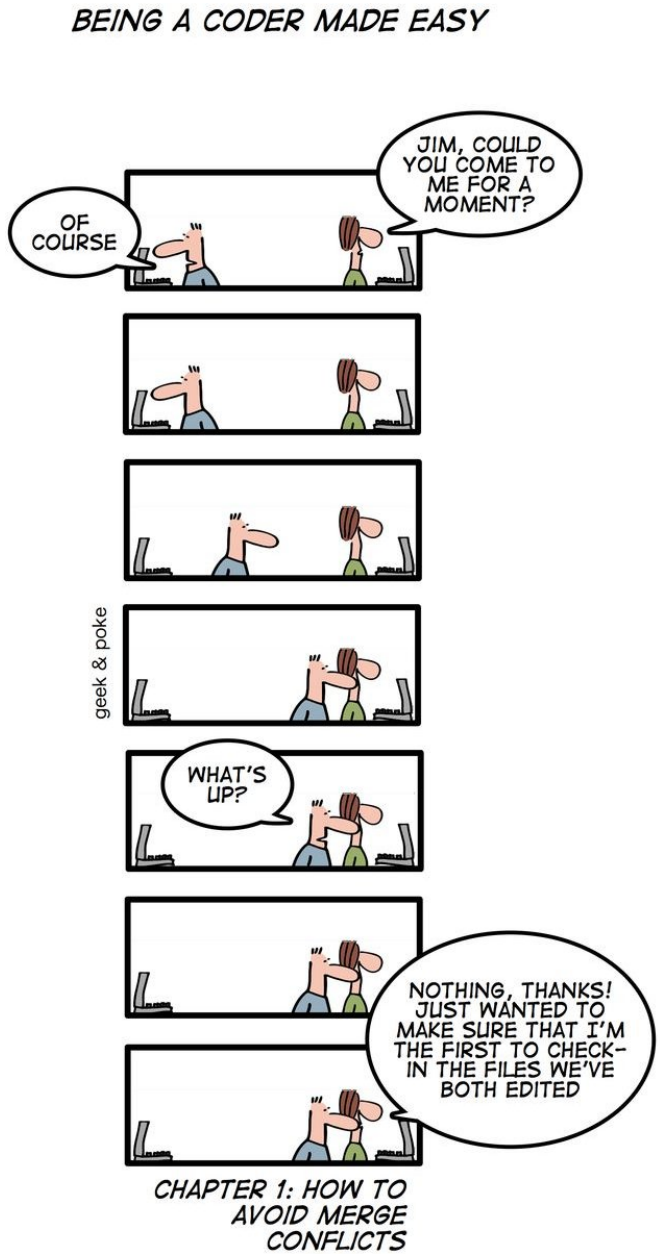
Update

- Merges changes in the repository into the local working copy
- Might create conflicts with local changes that might have to be resolved manually

Commit

- Action of writing or merging the changes made in the working copy back to the repository.
- Might not be possible without an up-to-date checkout
- Also: New revision that is created by committing

Some day on Geek & Poke



http://geekandpoke.typepad.com/

Revision control - Log messages

"If you have nothing to say about what you are committing, you have nothing to commit."

Log messages serve at least three important purposes

- To speed up the reviewing process.
- To help us write a good release note.
- To help the future maintainers to find out why a particular change was made

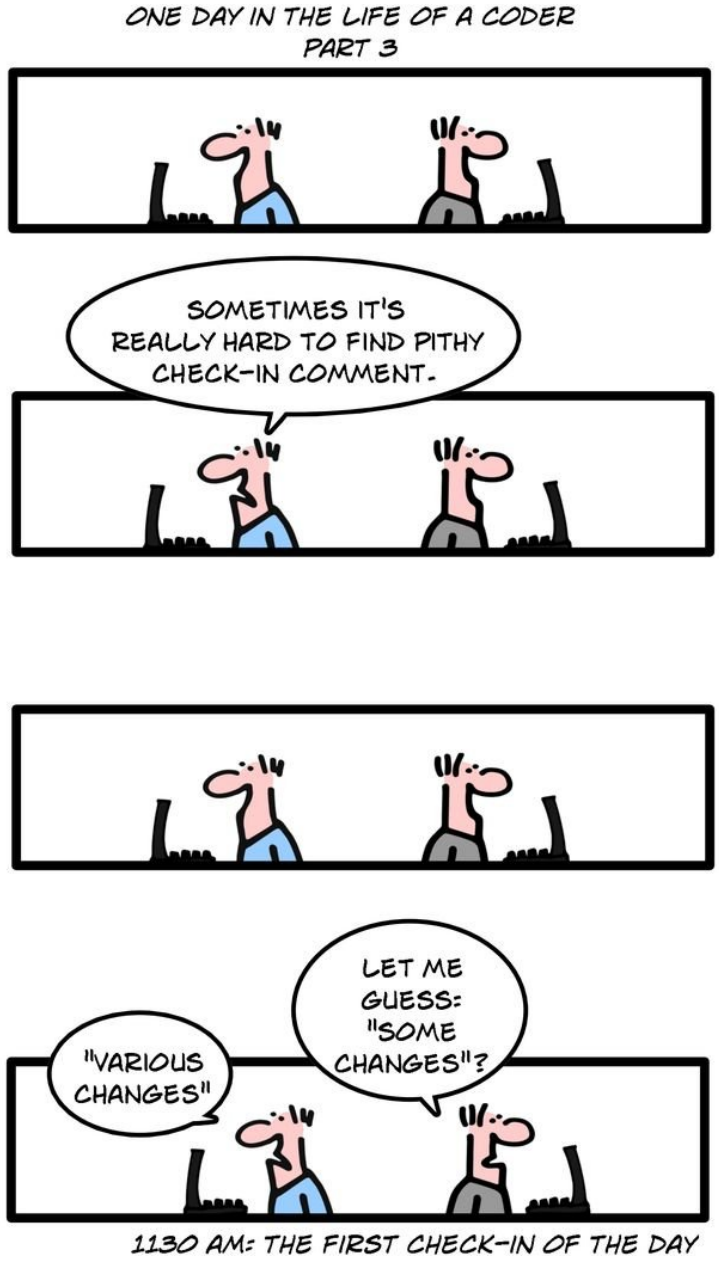
At least try to

- Summarize clearly in one line what this commit is about
- Describe the change, not how it was made

Even better

- Write one-line summary, followed by an empty line and a longer description
- Line-break the commit message

Some day on Geek & Poke



http://geekandpoke.typepad.com/

Revision control - example workflows

Simple change to central repository

1. Checkout
2. Change file locally
3. Test change
4. Update - shows no remote changes
5. Commit change

Revision control - example workflows

Change to central repository with conflicts

1. Checkout
2. Change file locally
3. Test change
4. Update shows changes and merge conflicts
5. Resolve conflict
6. Test change
7. Repeat updating until success
8. Commit change

Revision control - Conflicts

Updating from repository will

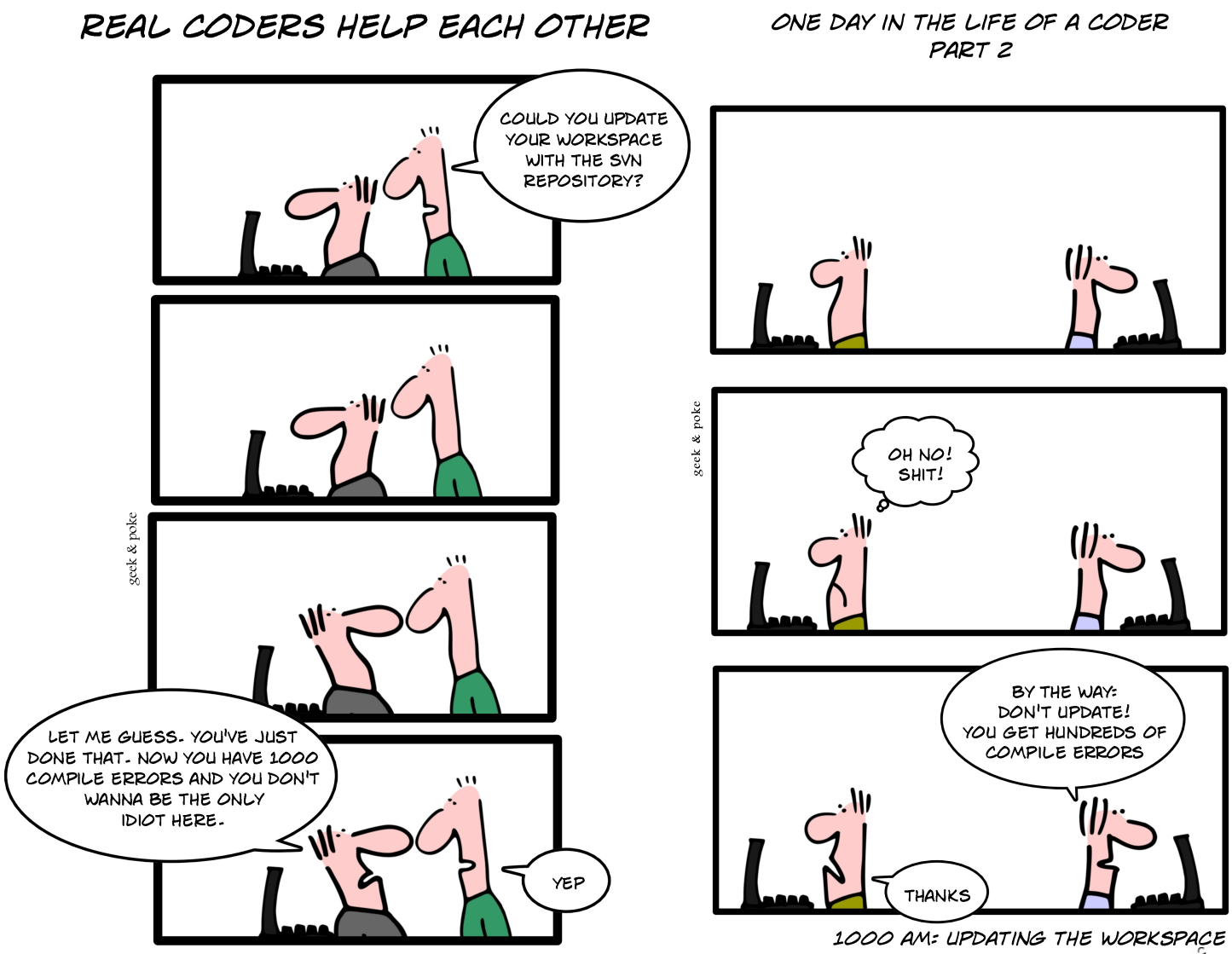
- Try to merge remote changes with local changes
- Create a conflict if this fails
  - Binary files: only option between 'theirs full' and 'mine full'
  - Text files: fine-grained control of multiple changes in single file

Example:

```
$svn update
  U index.html
  G changed-b.html
  C rubbish-b.html
  Updated to revision 46.
```

- U - Updated
- G - Merged
- C - Conflict

Some day on Geek & Poke



http://geekandpoke.typepad.com/

LSU

Revision control - Conflicts

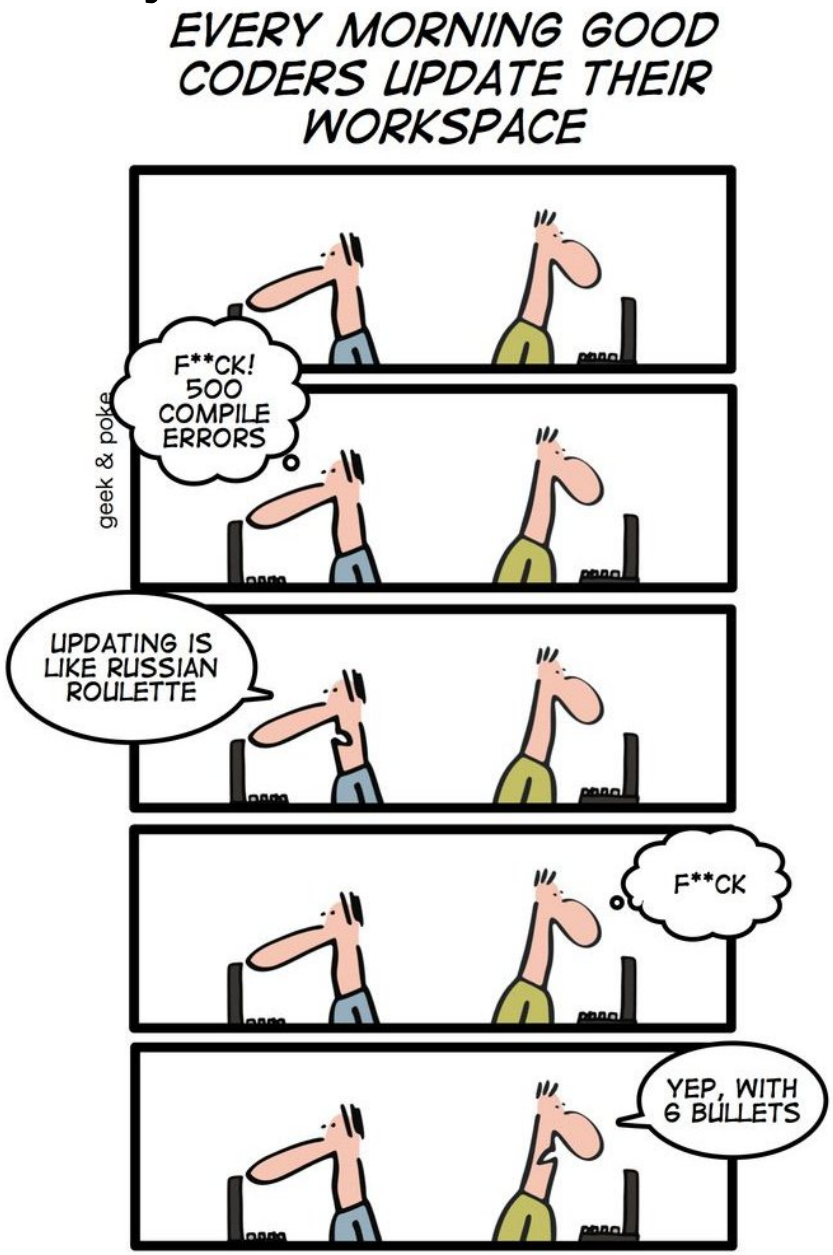
Conflict markers in text files:

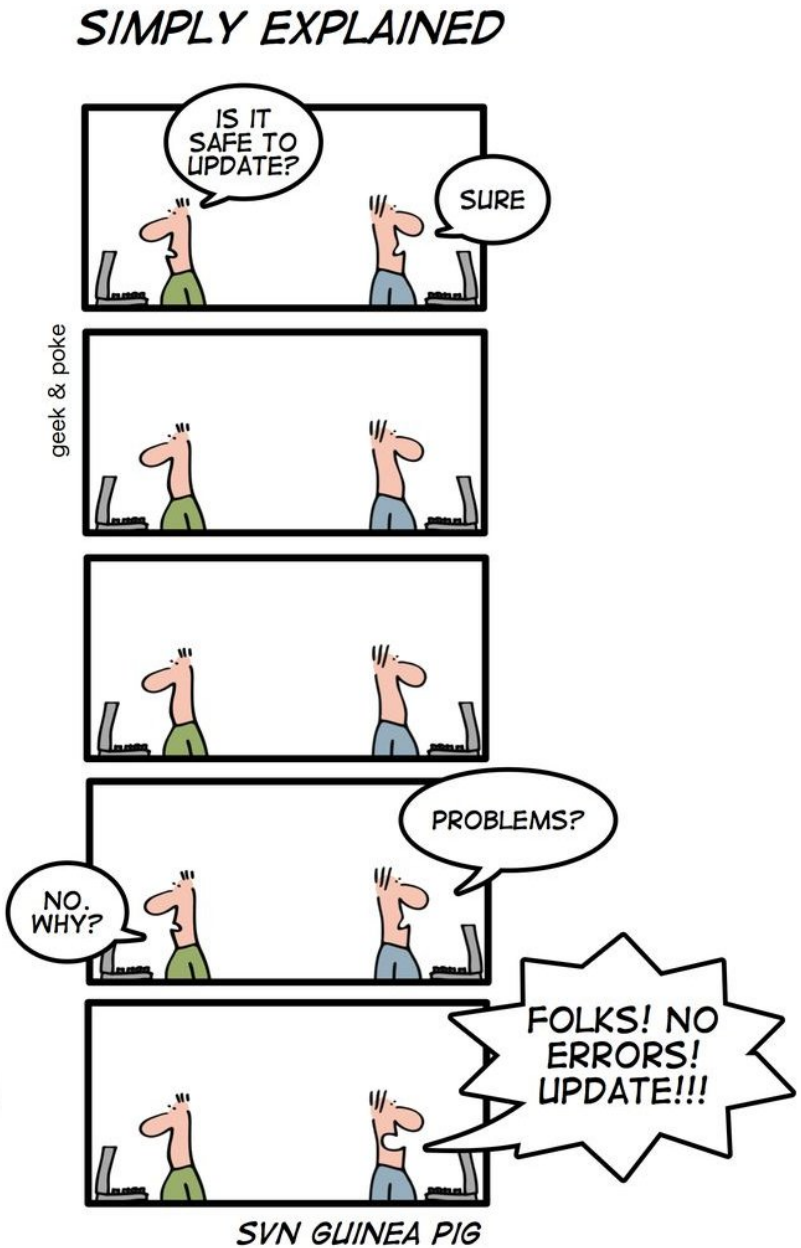
```
<<<<<<< filename
    your changes
=======
    code merged from repository
>>>>>>> revision
```

Resolve conflict by

- Reviewing both changes and manually merging them
- Test merged version
- Tell RCS that conflict is resolved
- Commit

Some day on Geek & Poke





http://geekandpoke.typepad.com/

Revision control - example workflows

Simple branch example

1. Create branch from trunk
2. Checkout branch
3. Change files locally, test and commit like on trunk
4. Possibly merge changes from trunk, resolve conflicts as usual
5. Eventually, merge changes from branch back into trunk
6. Remove branch

Revision control - example workflows

Applying change using distributed RCSs

1. Checkout / Clone
2. Change file locally
3. Test change
4. Commit change
5. Update from other working copy (e.g. “central” repository)
6. Resolve possible conflicts, commit needed changes
7. Test again
8. Repeat until success
9. Push commit(s) to other working copy (e.g. “central“ repository)

Revision control - history examination

History in RCSs can be used to find out

- Why something was implemented (log messages)
- When something was implemented
- Who did a certain change

Typical example:

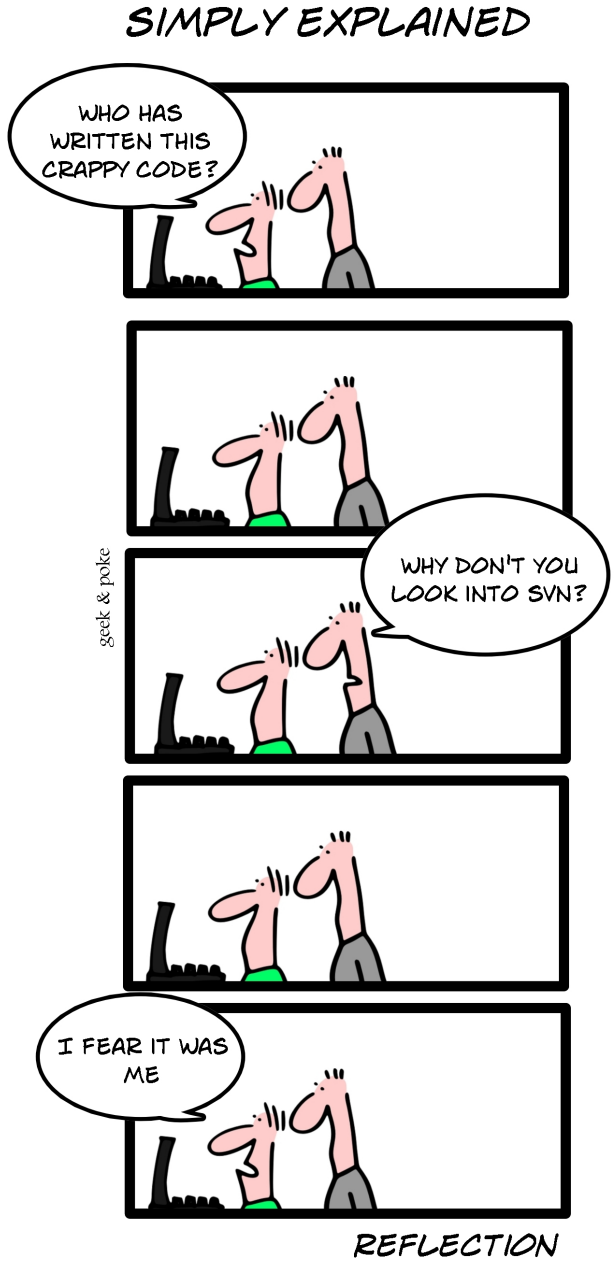
- Test of checkout: ok
- Test of checkout after local changes: ok
- Test after update from repository: failure

Narrow down location of bug by

- List log messages since last known working version
- Test without some of remote changes (go back in history)

Tools can help with this process.

Some day on Geek & Poke



http://geekandpoke.typepad.com/

Issue tracker

Issue tracker

- Synonyms: trouble ticket system, support ticket or incident ticket system
- More restricted: bug tracking system, bug tracker
- Database of “tickets”, describing issues/incidents/bugs

Workflow

1. User notices bug/issue/problem
2. (User tries to create small test case, presenting the problem)
3. User creates/opens ticket in issue tracker
4. Developer reproduces problem
5. Developer fixes problem
6. Developer closes ticket, notifying User

Issue tracker

Tickets/Issues can have attached

- Type (e.g. defect/enhancement)
- Priority (e.g. minor, major, critical, blocker)
- Project component
- Target project milestone
- Version of project component
- List of people CC'ed on changes of ticket
- Owner
- Files (e.g. patches)

Benefits of issue trackers over, e.g. direct developer contact

- Issues are recorded in database, cannot be forgotten
- Users can lookup if specific problem was already reported
- Users can automatically get change notifications

Issue tracker

- A large number of stand-alone issue tracker implementations exists
  - Trac
  - Bugzilla
  - GNATS
- Open source hosting sites usually automatically provide issue tracking systems, e.g.
  - sourceforge
  - savannah
  - seul
  - github
  - google code

Documentation format

Documentation format

Depending on need, various formats possible

- Plain text
- Man pages / Help system documents
- Application-internal
- Print-oriented, e.g. LaTeX, word processor
- Wiki
- Website as in plain HTML and typically in RCS

Summary

Summary

User- and development-friendly project environment provides:

- Information about project: e.g. website
- Communication channels for developers
- Infrastructure for shared code development
  - Project standards
  - Revision control system
- Communication channels for users, especially
  - Channel for problems/issues, directed at developers
  - Users-for-users channel

Summary - Best RCS usage

Basic

- Use it, learn (much) about it!
- Put as much as possible under Version Control
- Only put original source in Version Control, not built objects
- Test changes before committing
- Commit often and in logical chunks
- Update as often as possible (close open files beforehand!)
- Write meaningful commit messages

Advanced

- Branch only when necessary
- Don't copy when you mean to branch
- Branch late
- Propagate / Merge early and often

Reading: Subversion, Git, Mercurial