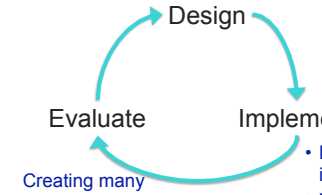
Objectives

- Collaboration with Version Control Tools
- SLogo Design

VERSION CONTROL

Problems in Collaborating on Code



Creating many prototypes
- What if don't like recent prototype? Need to go back to older version

- Different parts (e.g., user interface and backend)
- > 1 developer implementing concurrently
  - What if introduce a bug?

Version Control Systems

- Backup and Restore
  - Files are saved as they are edited
  - Revert to a specific version/checkpoint
- Synchronization
  - Lets people share files
  - Stay up-to-date with the latest version
- Track changes to code
  - Save comments explaining why change happened
  - Stored in the VCS, not the file
  - Track how, why a file evolves over time
- Track Ownership
  - Tags every change with the name of the person who made it

Version Control

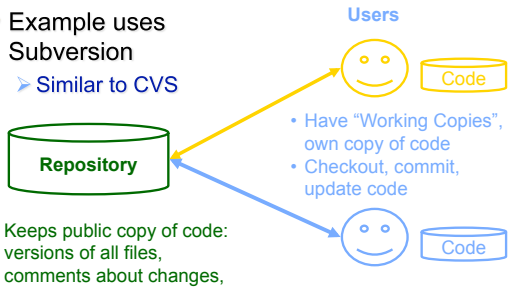
- Short-term undo
  - Messed up a file? Go back to the last **good** version
- Long-term undo
  - Created a bug a year ago? Jump back to see change you made.
- Sandboxing
  - Making a big change? Make temporary changes in isolated area, test, work out kinks before "checking in" your changes
- Branching and merging
  - Branch a copy of your code into a separate area, modify it in isolation (tracking changes separately)
  - Later, merge work into common area

CVS and Subversion

- Popular Version Control Systems
- Subversion is newer, more flexible
- Terms used are common for all version control systems

Using Version Control

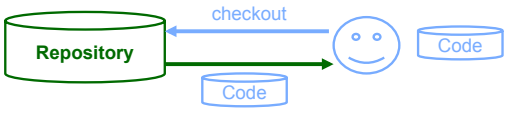
- Example uses Subversion
  - Similar to CVS



- Have "Working Copies", own copy of code
- Checkout, commit, update code
- Keeps public copy of code: versions of all files, comments about changes, who made changes

Using Version Control: `checkout`

- To start, need to `checkout` your working copy of the code



Using Version Control: `commit`

- After you make changes that you want others to see, **commit** your version

- Checks for conflicts -- code conflicts with recent changes in the public copy
- Update code, fix conflicts
- Try commit again

Using Version Control: `commit`

- After you make changes that you *want others to see*, **commit** your version
  - Include comments about what changes you made and why

- Checks for conflicts
- Updates each modified file
- Records comments with updated files

Using Version Control: `commit`

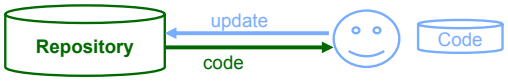
- After you make changes that you *want others to see*, **commit** your version
  - Include comments about what changes you made and why

- Checks for conflicts
- Updates each modified file
- Records comments with updated files

Other people's code doesn't change

Using Version Control: `update`

- To see the *current* version of the code, **update** your repository
  - Resolve conflicts



Using Version Control: `add`, `delete`

- You need to **add** and **delete** files and directories to the repository, then **commit**

- Create new records for added files
- Close records for deleted files
  - Files not deleted from repository

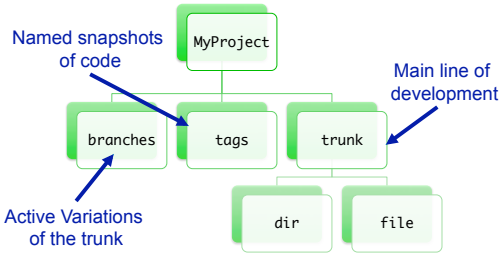
- Add, delete files and directories
- Commit

Version Control Advice

- Does not eliminate need for communication
  - Process becomes much more difficult if developers do not communicate
- Before picking up again, **update** your working copy
- **Commit** only after you've tested code and you're fairly sure it works
  - Write descriptive comments so your team members know what you did and why

Code Organization

- Organize code into appropriate structure



Subclipse

- Plugin for Eclipse
- Installation:
  - Help --> Software Update --> Find and Install, Search for New Features to Install.
  - Create two remote site:
    - Name: Subclipse
    - `http://subclipse.tigris.org/update_1.4.x`
      - Need Required, SVNKit

Checking Out Code

- Create a new SVN Repository:
  - File → New → Other → SVN
  - Repository:
    `file:///home/courses/cs209/shared/svn/SLogo/trunk`
- Copy SLogo project to SLogo.orig
- Checkout from repository
  - As a new project (Wizard)
  - Java project, named SLogo

Practice with Subclipse

- Named: your name
- Put some text into it
- **Add** the file to the Repository:
  - Right-click on the file you created → Team → Add
- **Commit** your file *(Save for group to see)*
  - Right-click on top-level directory/project → Team → Commit
  - Add an appropriate comment
- **Update** your repository *(Get latest working version)*
  - Right-click on top-level directory/project → Team →Update
  - Do you have any one else's files?

PLANNING THE PROJECT

Feedback on Prep Document

- Impressive!
- Well thought out plans
  - Good analysis
  - Anticipating what will need to be done
  - Shows foresight
- Continue discussion as a group now…

What Steps Need To Be Completed?

What Steps Need to be Completed?

- Turtle
  - Drawing the trail
  - Helper movement/state methods
    - E.g., Pen up or down
- GUI
  - Command interface
  - More options/buttons (optional)
  - Listeners
  - Improve GUI
- **TESTING!**
- Parsing SLogo language
  - Handle 30 SLogo commands
  - Handle errors appropriately
- Executing SLogo language
  - Turtle does what's appropriate
- SLogo files
  - Reading, writing, saving
  - (Mostly GUI)

Dependencies

Dependencies

- SLogo Execution requires Turtle methods
- GUI (testing) requires Turtle methods, Parsing/Execution

Effect of Extensions

- Extensions could affect your code design
  - Where could change --> abstraction
- Decide on Wednesday
  - May change your minds after start working on the code
  - Top vote getters so far (mostly in GUI)
    - Different turtle images
    - Undo/redo options (what does this entail?)
    - Changing Turtle's pen's characteristics

Plan

- Tasks/Steps
  - Testing
  - Think about iterative development
    - Friday deadline: basic functionality of all parts
- Division of tasks
  - # of people per part
- Deadlines

Goals

- Implement a version of application
  - Iterative development
- Don't go too far in depth, more breadth
  - See design issues sooner
    - "We need method/functionality X in class Y"

Secondary Goals

- You're going to figure out that your design isn't perfect--maybe not even good!
  - Could be partially fault of given code
  - Fix smaller things
    - Refactoring!
  - Note larger things (analysis/post-mortem due at end of finals week)

Good judgment comes from experience.
How do you get experience?
Bad judgment works every time.

SLogo Timeline

- Friday, Dec 5: demo application (group)
- ??: final implementation due (group)
- Fri, Dec 12: "Post-mortem" (individual)

Course Evaluations - Wednesday

- Favorite topics
- Least favorite topics
- Anything you definitely wanted covered?
- Wanted covered longer?
- Didn't want covered?