Principles of Software Construction: Objects, Design, and Concurrency

Version control with git

Michael Hilton  Bogdan Vasilescu
Administrivia

• Midterm 2 Thursday March 28\textsuperscript{th}
  – Midterm Review March 27 6 pm in NSH 3305

• Form teams for HW 5
Part 1: Design at a Class Level
Design for Change: Information Hiding, Contracts, Unit Testing, Design Patterns
Design for Reuse: Inheritance, Delegation, Immutability, LSP, Design Patterns

Part 2: Designing (Sub)systems
Understanding the Problem
Responsibility Assignment, Design Patterns, GUI vs Core, Design Case Studies
Testing Subsystems
Design for Reuse at Scale: Frameworks and APIs

Part 3: Designing Concurrent Systems
Concurrency Primitives, Synchronization
Designing Abstractions for Concurrency
Last week Thursday recap
Characteristics of a Good API

Review

- Easy to learn
- Easy to use, even if you take away the documentation
- Hard to misuse
- Easy to read and maintain code that uses it
- Sufficiently powerful to satisfy requirements
- Easy to evolve
- Appropriate to audience
Try API on at least 3 use cases before release

- If you write one, it probably won’t support another
- If you write two, it will support more with difficulty
- If you write three, it will probably work fine
- Ideally, get different people to write the use cases
  - This will test documentation & give you different perspectives
- This is even more important for plug-in APIs
- Will Tracz calls this “The Rule of Threes”
  *(Confessions of a Used Program Salesman, Addison-Wesley, 1995)*
Names Matter – API is a little language

*Naming is perhaps the single most important factor in API usability*

- Primary goals
  - **Client code should read like prose** (“easy to read”)
  - **Client code should mean what it says** (“hard to misread”)
  - **Client code should flow naturally** (“easy to write”)

- To that end, names should:
  - be largely self-explanatory
  - leverage existing knowledge
  - interact harmoniously with language and each other
Aside: Software engineering research on names
[Vasilescu et al, ESEC/FSE 2017]
Today:

VERSION CONTROL WITH GIT
Versioning entire projects
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
Three ways to move work around between branches
1) `git merge bugFix (into master)`
git checkout bugfix; git merge master (into bugFix)
Activity:
Move work from bugFix directly onto master

2) git rebase master
But master hasn't been updated, so:

git checkout master; git rebase bugFix
Activity:
Copy a series of commits below current location

3) `git cherry-pick C2 C4`
Activity:
Ways to undo work (1)

git reset HEAD~1

HEAD is the symbolic name for the currently checked out commit
Ways to undo work (2)

**git revert HEAD**

```
C0
  |
  v
C1
  |
  v
C2
```

- `master*` points to `C2`

**git reset does not work for remote branches**

```
C0
  |
  v
C1
  |
  v
  |
  v
C2`

```

- `master*` points to `C2`
Activity:
Highly recommended

• (second) most useful life skill you will have learned in 214

Summary

• Version control has many advantages
  – History, traceability, versioning
  – Collaborative and parallel development

• Collaboration with branches
  – Different workflows

• From local to central to distributed version control