CS 146: Intro to Web Programming and Project Development

Instructor: Patrick Hill

Email: phill@stevens.edu

Introduction to Soft. Engineering (II)

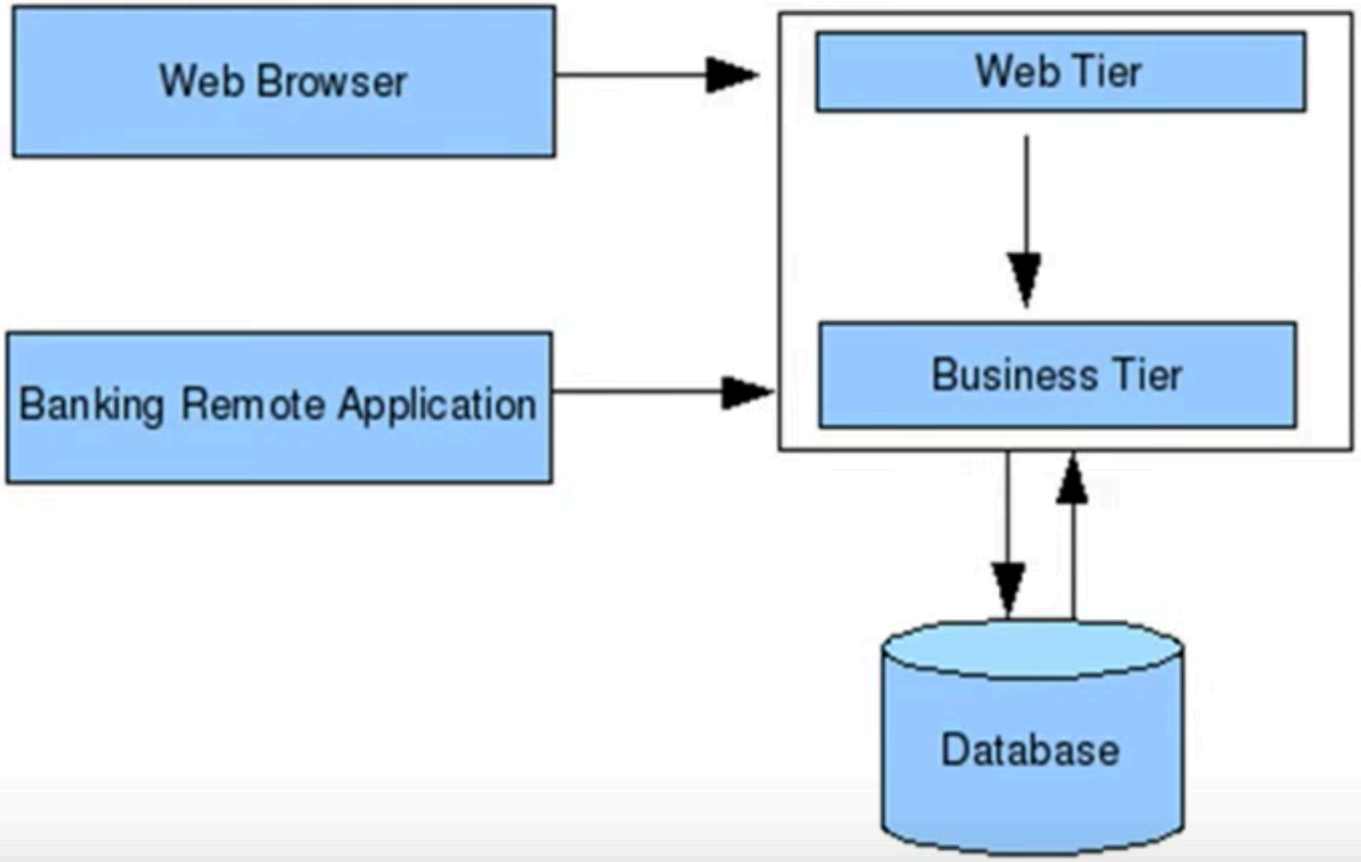
Testing Levels

- **Unit testing**, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

Testing Scenario

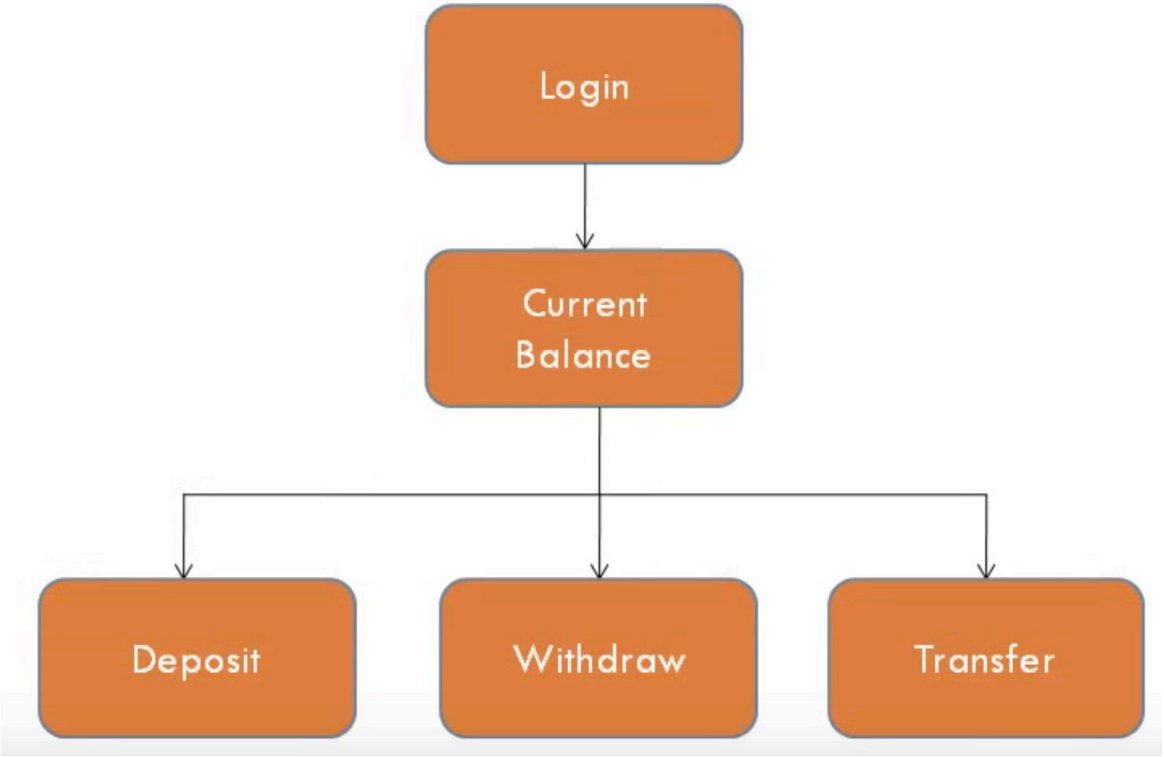
- Your company is hired by a bank to develop an online banking application
- Requirement Analysis
  - Log on
  - View Current Balance
  - Deposit
  - Withdraw
  - Transfer

Testing Scenario - Functional Design



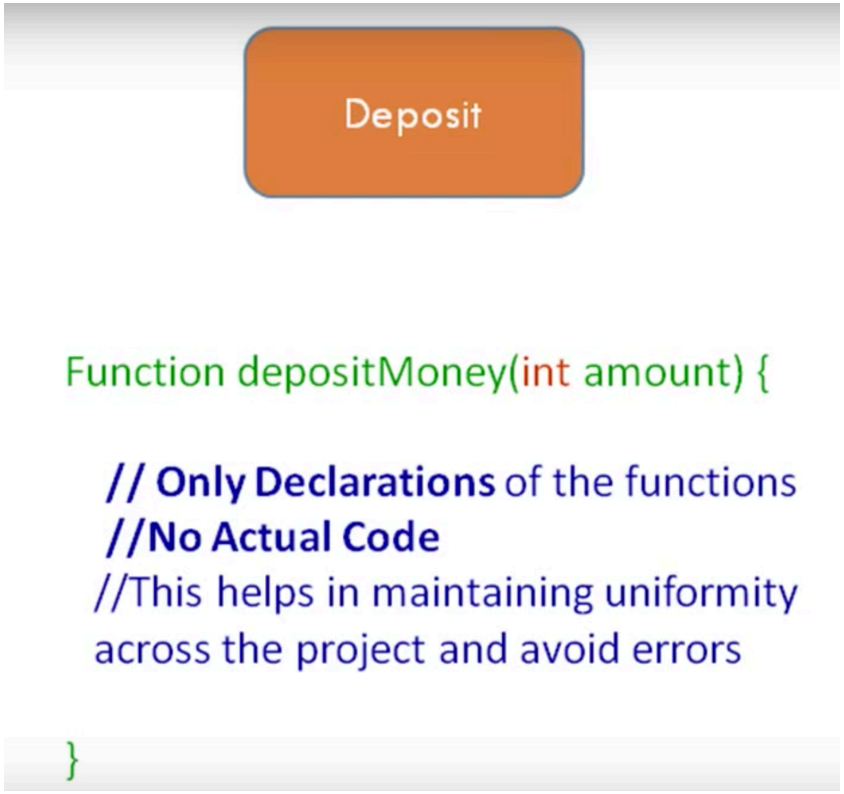
Testing Scenario – High Level Design

- Break down into modules



Testing Scenario – Detail Design

- Code Architecture/Skeleton and Documentation



Testing Scenario – Unit Testing

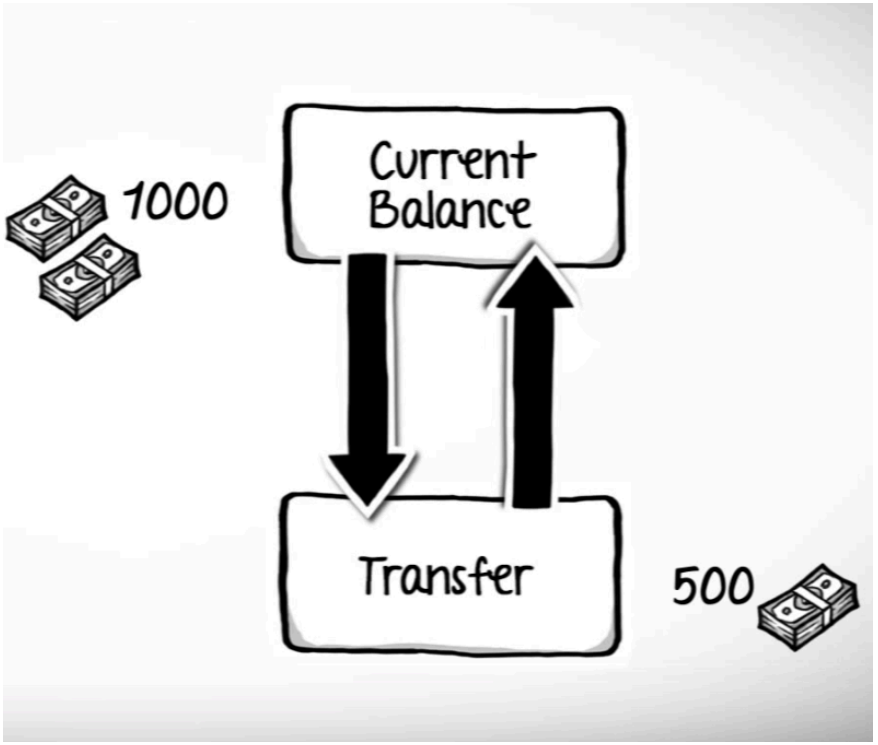
Testing Levels

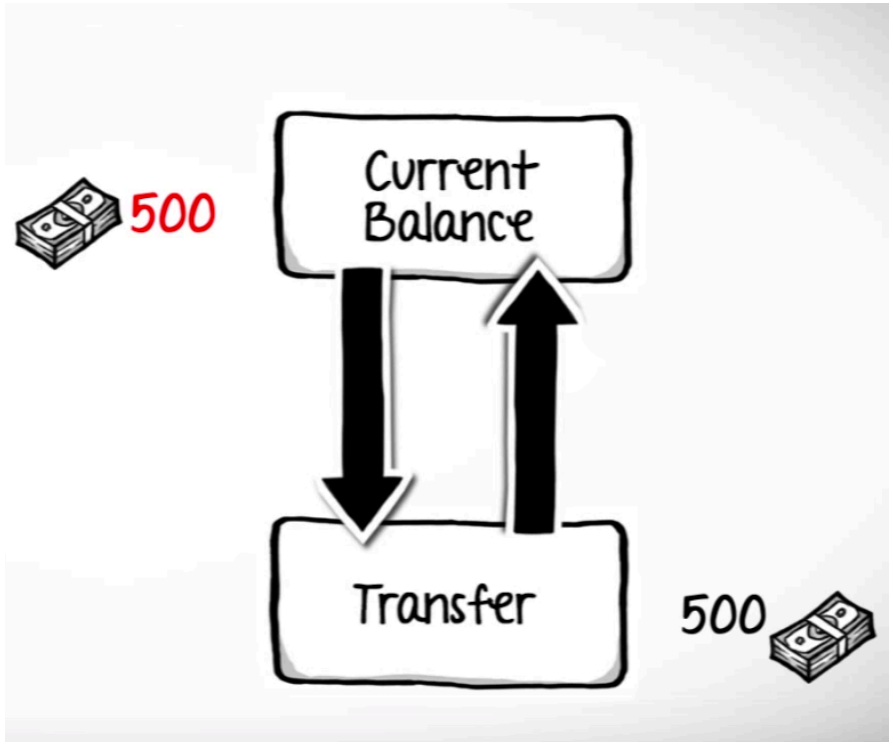
- **Integration testing** is any type of software testing that seeks to verify the interfaces between components against a software design.
- Integration testing works to expose defects in the interfaces and interaction between integrated components (modules).

Testing Levels

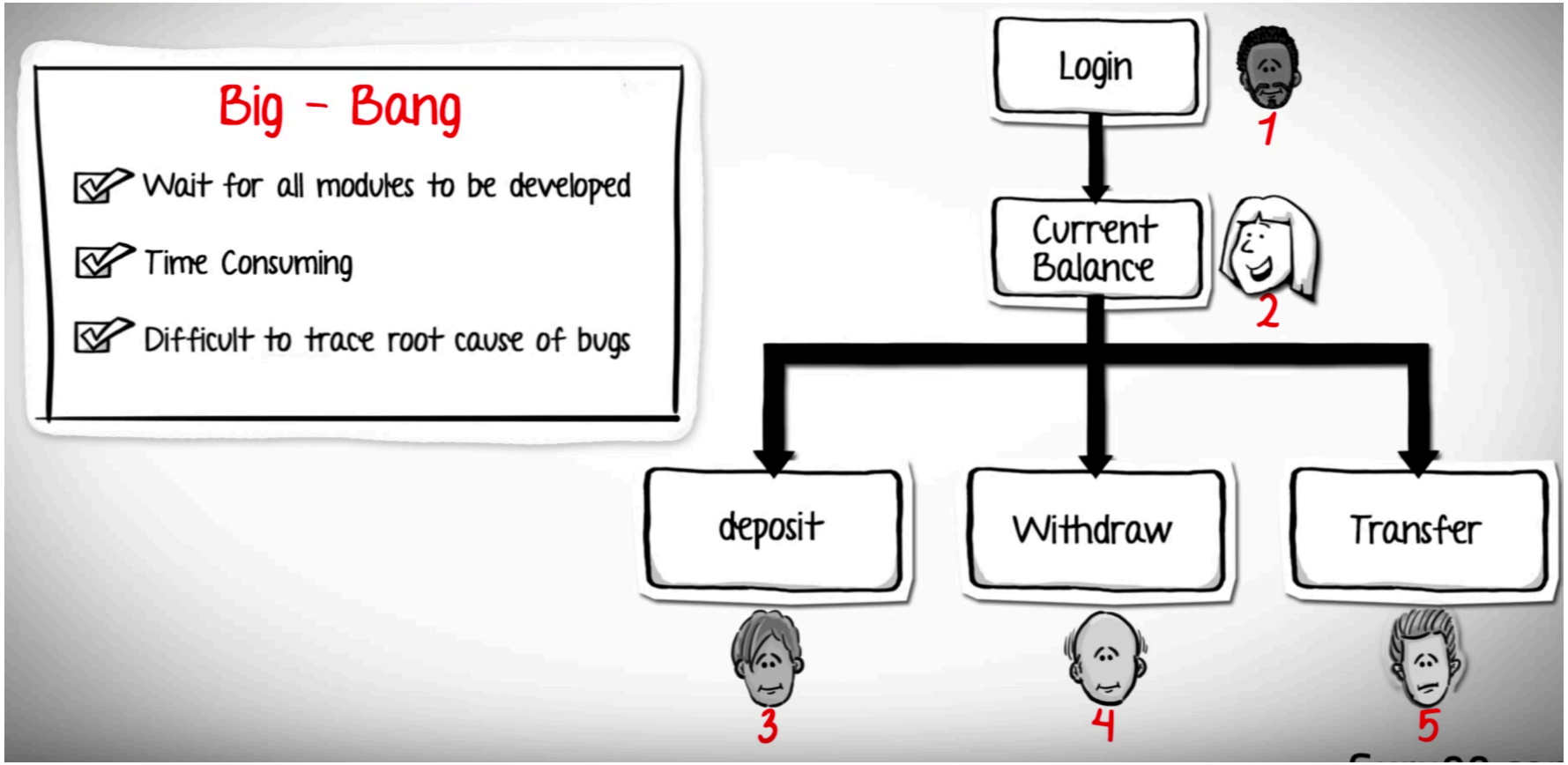
- The practice of **component interface testing** can be used to check the handling of data passed between various units, or subsystem components, beyond full integration testing between those units.
- The data being passed can be considered as "message packets" and the range or data types can be checked, for data generated from one unit, and tested for validity before being passed into another unit.

Testing Scenario – Integration Testing





Testing Scenario – Integration Testing



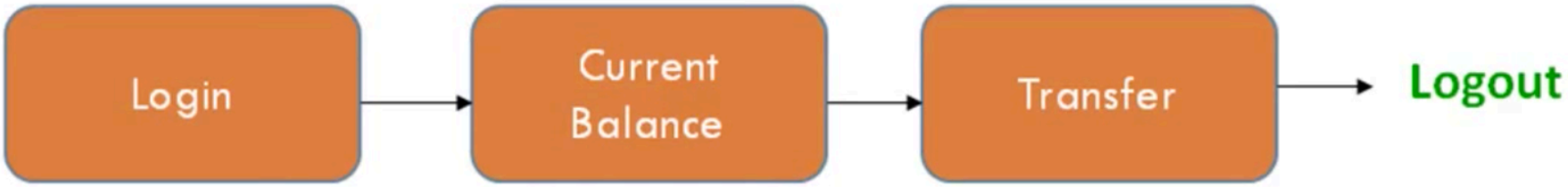
- We can also have incremental Integration Testing!

Testing Levels

- **System testing**, or end-to-end testing, tests a completely integrated system to verify that it meets its requirements
- At last the system is delivered to the user for **acceptance testing**
- Helps to avoid cases where the sponsor says, “This is not what I asked for!” after deployment

Testing Scenario – System Testing

- Whole System, instead of interaction of modules tested in integration testing
- Also, non-functional aspects such as the performance are tested here



- Acceptance Testing:
  - Alpha/beta testing

Testing Methods

- Within testing there are 2 main approaches:
  - Black-box testing: Testing without the knowledge on how the methods work. This means just testing input/outputs without caring about what goes on inside the method.
  - White-box testing: Testing every single line of code in your program. Check internal structures or workings of an application, as opposed to its functionality (i.e. black-**box testing**).

White Box Vs Black Box Testing

BLACK-BOX TESTING

- based on a description of the software (specification)
- cover as much specified behavior as possible
- cannot reveal errors due to implementation details

WHITE-BOX TESTING

- based on the code
- cover as much coded behavior as possible
- cannot reveal errors due to missing paths

Black Box Testing

BLACK-BOX TESTING EXAMPLE

Specification: inputs an integer and prints it

```
1. void printNumBytes( param )
2.     if (param < 1024) printf("%d", param);
3.     else printf("%d KB", param/1024);
4. }
```

White Box Testing Example

WHITE-BOX TESTING EXAMPLE

Specification: inputs an integer param and returns half of its value if even, its value otherwise

```
1. int fun(int param){
2.     int result;
3.     result = param/2;
4.     return result;
5. }
```

Version Control: Bitbucket/GIT

Version Control Systems

- Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all "real" projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a "central" repository; users "check out" files, work on them, and "check them in"
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

Why Version Control?

- For working by yourself:
  - Gives you a "time machine" for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- **For getting an internship or job:**
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work
  - **If you haven't already, create a github with samples of your code.**

Why Git?

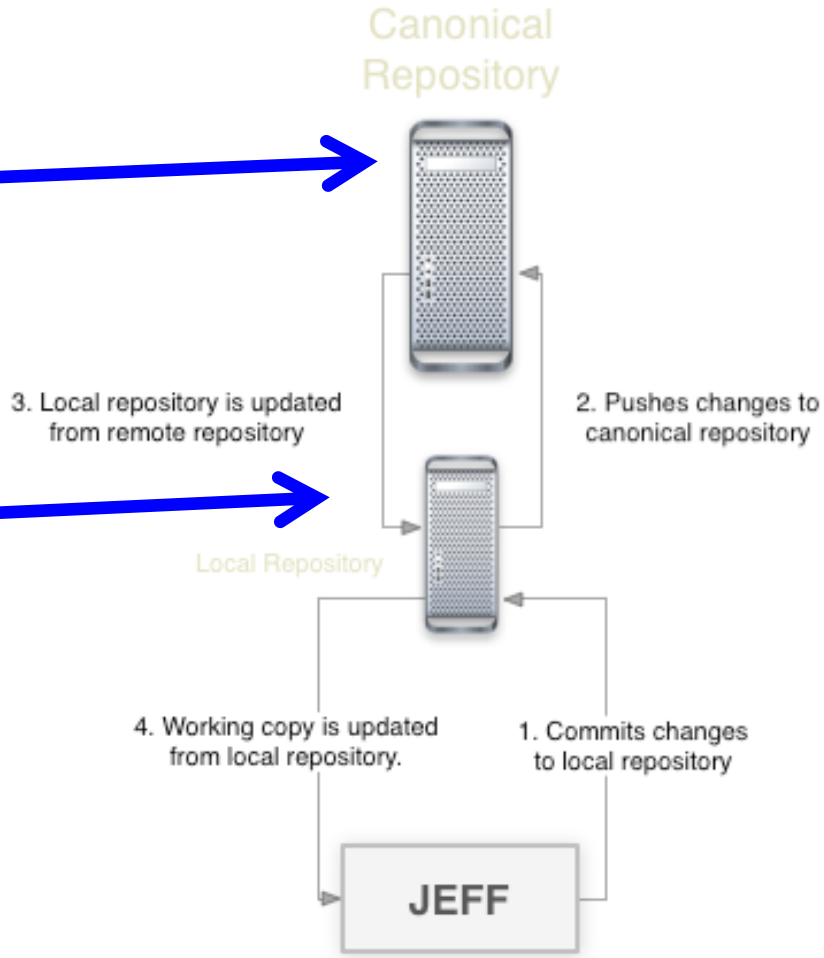
- Git has many advantages over earlier systems such as CVS and Subversion
  - More efficient, better workflow, etc.
  - See the literature for an extensive list of reasons
  - Of course, there are always those who disagree (we don't talk to them)
  - Most companies use Git
- Best competitor: Mercurial
  - Same concepts, slightly simpler to use
  - Much less popular than Git

Atlassian Bitbucket

GitHub

Your repo in bitbucket/github

Your local repo



Typical Workflow

- `git pull` *`remote_repository`*
  - Get changes from a remote repository and merge them into your own repository
- `git status`
  - See what Git thinks is going on
  - Use this frequently!
- Work on your files (remember to `add` any new ones)
- `git add *.js`
- `git commit -m` *“What I did”*
- `git push`

Don’t forget!