CSE 403

Tools and runtimes for your project
Fall 2012

September 25, 2012

Tools and Runtimes

- Product documents, "paper designs", models
- Implement code as a team (dev tools and envs)
- Write tests
- Communication tools (wiki, email, chat, forums(?))
- Deployment stack (web servers, app servers, databases)
- Deployment tools

We're biased towards web applications

~~Strongly biased for using github~~

You are required to use github

- Platform of choice in 2012
- Git repository (version control)
- Wiki (you are required to have one of these)
- Bug tracking system
- Good for your career
- Free (if you need a private repo, come see me)
- Find a good tutorial

Your local development environment

- You decide as a team as to what to use
- Use your own personal machines?
- Department machines?
- How do you bring up a full working system?
- Get git on your system

Version control systems: why?

- Allow developers to work concurrently
  - Personal changes do not affect each other unless published
- Preserves history of the changes
  - If something goes bad, developers can roll back to a "stable" state (maybe)
- Disaster management
- Using a VCS is a natural requirement
  - If not, believe us that you will regret it

Version control system (git)

- A central repository (github)
- Distributed private (personal) repositories
- Personal workspaces
- Make changes->commit local->push to share

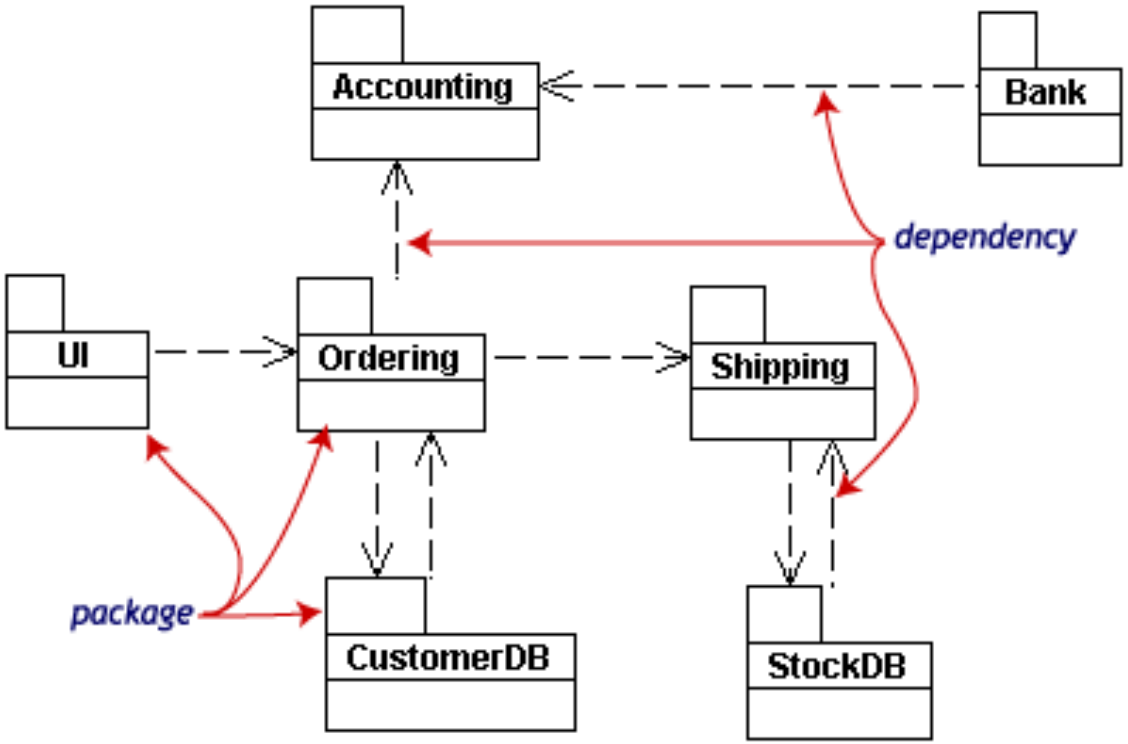
Requirements and UI Tools

- Word (Google docs) for writing requirements
- UI prototypes (paper/pencil, sticky notes, balsamiq, Powerpoint, others)
- UML tools for data model (maybe)

Runtime Resources

- WebFaction
- Heroku
- Amazon EC2
- Department resources

UML tools


Accounting
Bank
dependency
UI
Ordering
Shipping
package
CustomerDB
StockDB

Implementation tools

- Programming language
  - Python, Ruby (on Rails), php....
  - Compilers, interpreters, debuggers, profilers(?)
- IDE: Textmate, Eclipse, vi/emacs!
- Twitter Bootstrap (for generating front ends)
- Xcode?

Testing

- Find bugs in your implementation
- Check for correctness
- Unit tests, integration tests, system tests
- Frameworks -- language specific?
- “Monkey” testing
- Functional testing: Does it match up with the requirements?

Hints/Advice

- Quarter is short, don't get stuck with details
- Try to keep up with internal dead lines
- Hack around, play, experiment early
- Ask for help (but we might not know the answer)
- Don't take too much risk
- Understand the conflict between building product and building tools/infrastructure/process

More advice

- Building a django/python/postgres environment? Read: http://www.jeffknupp.com/blog/2012/02/09/starting-a-django-project-the-right-way/
- How can a programmer learn to design websites that don't suck http://vascop.github.com/blog/2012/07/25/how-can-a-programmer-learn-to-design-websites-that-dont-suck.html
- git and github tutorials
  - http://www.vogella.com/articles/Git/article.html
  - http://www.github.com

Still more advice

- Get started early
- Experiment with the tools
- Hack aggressively in the first four weeks -- pull stuff into the "environment" at week 4
- Ignorance, confusion, frustration, discovery, mastery are all part of the process
- Have fun, build something cool!
- Ask questions, suffer and celebrate together!