CS48 Lecture Schedule

- Today (Week 2)
  - Intro to SWE: Process & requirements analysis
- Next week: Requirements analysis and specification
  - Basics: project description, vision statement, initial features as use cases or user stories
  - Requirements Analysis Doc due 2/2/18

- Weeks 4-6
  - In depth analysis: modelling system behavior (UML)
  - Design principles
  - Implementing and testing
  - Draft project due 2/23/18: build up/evolve your Doc

- Rest of class: Midterm, demo prep, demos
Software Engineering

- Set of "Activities" – not "steps"
  - Often happening simultaneously
  - Not necessarily discrete
  1. Planning: mostly study the requirements
  2. Domain analysis: study the problem area
  3. System design: devise computer solution
  4. Implementation: write the code
  5. Testing, documentation, maintenance, …

- Management activities: developers, teams, customers
- Guided by CS theory
Professional, ethical responsibility

- Above all, do no harm! (Hippocratic Oath)
  - NO VIRUSES or other malicious programs
  - Avoid inventing “the bomb” or a plague, or …
- Basically demonstrate *loyalty* to employer, clients, co-workers, country, humanity, …

- See “Software Engineering Code of Ethics and Professional Practice” by ACM/IEEE-CS
  - [https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct](https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct)

Extra credit exam question on this
Development process modeling

- The classic:
- Step after step, after step, …
- Never back up

The Waterfall Model

- Requirements Analysis
- System Design
- Program Design
- Coding
- Testing (several steps)
- Operation & Maintenance
Alternatives to waterfall model

- Okay, we all agree – this extreme doesn’t work either
- Is there a middle ground?

Software Development Reality
Considering risk

In a waterfall lifecycle, high risk issues such as integration and load test may be tackled late.

Research conclusion: it is wise to do some implementing and testing early in the process.
Engineering the risk factor

- Spiral Model
  - Includes frequent risk analyses
  - Frequent reevaluation during an extended planning stage
Spiral Model: Pros/Cons

Pros
- Risks are identified early on in the project
- Rapid prototyping shows progress and gives users an idea of the system
- Feedback can be obtained early from these prototypes

Cons
- Time is wasted on making too many prototypes
- Risk analysis is too time consuming for small projects
- It’s complex
- It can be hard to set milestones
- It can be never ending if hurdles are not overcome.
Testing and iterating

- Accounts for requirement changes and mistakes
- Key idea: plan to iterate
- But still a bit too rigid?

The V Model

- Requirements Analysis
  - System Design
    - Program Design
      - Coding
    - Unit & Integration Testing
      - System Testing
        - Acceptance Testing
          - Operation & Maintenance
            - Validate requirements
              - Verify design

Requirements Analysis
System Design
Program Design
Coding
Unit & Integration Testing
System Testing
Acceptance Testing
Operation & Maintenance
Incremental / iterative process

- Hmmm … a hybrid that makes sense!

Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.

The system grows incrementally.

Iterations are fixed in length, timeboxed.

The system grows incrementally.

4 weeks (for example)
Iterating reduces risk overall

- Especially if thorny issues are tackled early
Unified Process (UP)

- By Rumbaugh, Jacobson, Booch, others (~1999)
- Customizable to different settings/teams/projects
- Iterative and incremental through 4 phases
  - Release at end of each timeboxed iteration
- Use case driven
- Architecture-centric
- Risk-focused
- UML describes requirements & design
  - Standardized notation
    - Object oriented
  - Static models
  - Dynamic models
Agile Software Development

- **Agility** – observed to be a common feature of *successful* processes
- Different projects need different processes
- Generally better to focus on skills, communication, and community instead of processes
- Fruitful to consider it “a cooperative game of invention and communication” (Cockburn, 2002)
Agile Software Development

Manifesto for Agile Software Development (2001)
available at: http://agilemanifesto.org/

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

*Individuals and interactions* over *processes and tools*
*Working software* over *comprehensive documentation*
*Customer collaboration* over *contract negotiation*
*Responding to change* over *following a plan*

That is, while there is value in the items on the right, we value the items on the left more”
Extreme Programming (XP)

- A type of agile software development process proposed by Kent Beck (~late 90’s)
- XP follows the agile software development principles as
  - Software is built *iteratively*, with *frequent releases*
  - Each release implements the set of *most valuable features/use-cases/stories* that are chosen by the customer
  - Each release is implemented in a *series of iterations*, each iteration adds more features/use-cases/stories
  - Programmers turn the stories into *smaller-grained tasks*, which they individually accept responsibility for
  - The programmer turns a task into a set of *test cases* that will demonstrate that the task is finished
  - Working as *pairs*, the programmers make the test cases run, evolving the design in the meantime to maintain the simplest possible design for the system as a whole

See [www.extremeprogramming.org](http://www.extremeprogramming.org)
Scrum

- An evolutionary/iterative/incremental/agile software process implementation
  - See: Scrum and XP from the Trenches -- free online book by Kniberg
Scrum

- An evolutionary/iterative/incremental/agile software process impl.
  - See: *Scrum and XP from the Trenches* -- free online book by Kniberg

- The main roles in Scrum are:
  - Scrum team: Team of software developers
  - Scrum master: Project manager
  - Product owner: Client or business lead (accepts/rejects result)

- Characteristics of Scrum:
  - Self-organizing teams
  - Product development in two to four week *sprints (iterations)*
  - Requirements are captured as items in a list of *product backlog*
    - *Yours will come from your requirements analysis document*
    - *Sprint backlog* is subset of product backlog for this sprint
  - Retrospective after each iteration (identify ways to improve)
Sprint and Scrum: Implementation

- **Sprint planning**
  - Use cases or user stories broken down into tasks
    - Team members assign timings (how long each will take)
    - And pick tasks
  - Tasks = designing, implementing, testing, and demo'ing
    - Includes code review
  - Any new tasks identified enroute go onto backlog for inclusion next time

- **Daily standup**
  - What I did yesterday, what I'm doing today, + any blockers
    - Short/quick so done standing up!

- **Retrospective and end of each iteration (identify ways to improve)** — what worked and what didn't
  - Vote on what to work on next time
Next Up: Requirements Analysis & Specification
Requirements analysis

- What do *stakeholders* want from the system?
  - What should it do?
  - What should it look like? Sound like? Be like?

- Object-oriented analysis and design
  - From an *object* perspective
  - System composed of objects or concepts (UML helps here)
    - What must the system do?
    - How will system fulfill the requirements?
    - How do objects/concepts interact?

- Means not: function-oriented
  - System composed of processes, functions
    - What to do, and how to do it?
    - Mostly worry about “flow of control”

- Catalog    Library
- Book        Librarian

- Record loans
- Add resources
- Report fines
Requirements analysis

- **Analysis starts with a project description**
  - Usually written (or otherwise expressed) by major stakeholder
    - a.k.a. “Client” – might be a customer, another department in the company, management, professor, …
    - Or project team writes it for an anticipated market

- **Results in a series of RA artifacts:** 2 purposes
  - Shows the client what they will be getting
  - Used to kick-off and guide later development activities
  - **Vision**
    - Problem statement, system goals, features, constraints
  - **Use cases/user stories**
    - Used to guide development and estimate costs
    - Turn into product (and sprint/iteration) backlog
Start by not even thinking about programming

- *Try* to focus on domain concepts at first
  - Not software constructs (wait until design stage)
  - Avoids complexity overload
  - Design and eventual system will be better too!

- Create and maintain a steady stream of artifacts
  - Mostly pre-programming – diagrams, class specifications, glossary, …
    - Use online tools (UML, drawing tools)
  - Guides initial implementation, and aids subsequent modification, maintenance, and software reuse
  - Commit to github repo – counts as CS48 “progress”
CS 48 development schedule

- Overview: a planning phase, followed by at least 2 complete development iterations – each iteration produces a working system
  - Follow agile principles

- Now: Planning phase – Requirements Analysis
  - First be the client – describe the project
  - Then analyze the requirements
    - Identify/enumerate system features and characteristics
    - Write use cases and user stories
    - Prioritize and assign use cases/stories to development iterations
      - See next week lecture notes if you want to get started early
CS 48 schedule (cont.)

- Early iteration(s) – draft project (report and current system)
  - Analyze the domain pertinent to the iteration
    - Identify classes, class attributes, and associations
    - Identify system behavior (as a “black box”)
  - Design the current system
    - Specify the way objects will behave and interact
    - Tie to other systems/tools as necessary
  - Implement and test
- Complete at least 1 more iteration – final project
  - Analyze/design/implement/test and update documents
    - Demonstrate system to class during last week of quarter
Tools for Agile Specification, Design, & Development

- Requirements specification and design
  - PDF: Written document with drawings
  - UML – Unified Modeling Language
    - Standardized notation – now well accepted
    - Online UML drawing: [https://yuml.me/](https://yuml.me/)
      - Getting highly sophisticated now
        - Can generate code from modeling diagrams
        - Can do reverse engineering, ...

- Use cases and user stories: Trello, PivotalTracker, Jira
  - Helps with supporting your agile process (iterations, scrum)

- Free plans for small projects, trial period, students…