CS253: Software Development

Welcome to Lecture 13!

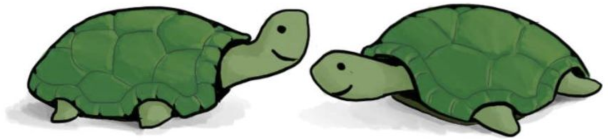
Daniel George

October 12, 2023

Announcements

- Great job with Assignment 2!
- Today: the software development process

Think-Pair-Share Terrapins!



How do we effectively create complex software?

How do we create complex software?

- Short “sprints”, set short-term goals, iterate
- Dividing work & delegating
- Communicating a lot!
- Testing frequently
- Git committing frequently (version control)
- Good code-level documentation

How do we create complex software?

- Step 0: “List the features” **(requirements gathering)**
- What are our concrete goals in terms of what the software should actually *do* when done?
- **User Stories:** a tool for gathering requirements
- **Design:** planning out the **software architecture** (the components and interfaces between them)

How do we create complex software?

- Coding & Implementation: Where to start?
  - Prioritization: start w/core functionality
  - The minimum (testable) needed to demonstrate anything
- Specification may need to change - that's OK!
- This is an **iterative** (adaptive) process: you can return to earlier steps if you need to
- Debugging: there exist tools (debuggers) to isolate bugs, parse error messages, and so on
- Testing: early & often (recall unit testing: testing small, individual functions. Build a testing suite and run often)

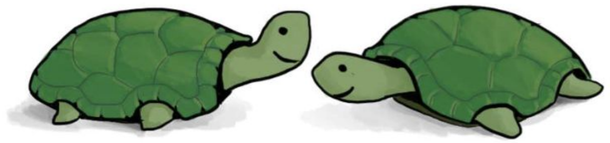
How do we create complex software?

- Recall: **test-driven development** - write a test for a piece of code *before* writing the code
- **Continuous integration (CI):** merge code often, and test integrated code often. Automate integration testing using a CI tool (Jenkins, CircleCI, TravisCI)
- Aiding understanding of code:
  - Documentation & comments in code (comment conventions)
  - Good commit messages
  - Pair programming
  - Code reviews (Github pull requests)
  - Coding standards (Linting)

How do we create complex software?

- Software Methodologies: **Agile** - incremental and iterative
  - Incremental: there are short “sprints” or “iterations”
  - Iterative: flexibility to return to earlier work
- Variants:
  - Scrum - invented in early 1990s by Jeff Sutherland and Ken Schwaber
  - Kanban - invented in 1940s by Taiichi Ohno of Toyota Motor Corporation (no concept of “sprints”; based on work capacity at any given time)

Think-Pair-Share Terrapins!



What are some of the pros and cons of the Agile software development approach?

How do we create complex software?

- Pros:
  - Flexible, can change & adapt
  - More testing
- Cons
  - More complex, harder to learn
  - Challenging for larger teams
  - Possibly too flexible

Project Requirements

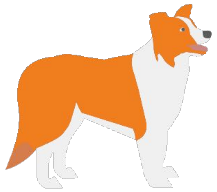
It's time to start into our final projects! To do this, we need project ideas.

For this mini-assignment, submit 1-2 ideas (or more!) for web applications that could be written by a project team for this class. Consider the following guidelines, but feel free to "think outside the box." (But not outside the flask. It has to stay within the flask. 😊) We can always discuss and adapt ideas to fit the course if there is enough interest. If you want to suggest something outside of these guidelines, though, be prepared to argue for your idea.

Project Requirements

- Projects will be web-based, using Python + Flask on the back end and Javascript on the front end. You can use additional existing libraries (in Python and/or Javascript) to handle specific functionality if you want.
- The back end must be dynamic. A static website, no matter how much Javascript runs in the browser, is not sufficient.
- Project teams will be somewhere around 3-4 students each.
- The project should be achievable by a team within ~9 weeks, and it should not be achievable by a team within substantially less time than that. (I'll help with this scoping, because it's always difficult to estimate something like this before you've done it.)

What are your questions?



Thank you!