Project Management

Lecture 10

Outline

- Q2 Sneak Peek
- Project Management

Next Quarter: Project Deliverables

Each project must have:

- A written report/paper.
- A visual or hands-on component (e.g. blog, website, app, usable software)
- A code artifact (versioned and buildable)
- Oral presentations (elevator pitch; longer slide presentation)

While each is required of every group, which is your main element of focus is up to you!

Next Quarter: Lecture/Methods

- Lecture will be lighter than 180A; focus on communication
- No Methodology HW.
- Focus on applying 180A methodology HW to your Capstone Project

Next Quarter: Section

- Before each Wednesday domain each team will update each other on their progress for the week (e.g. Zoom, Slack, etc).
- In each Wednesday domain, one (randomly chosen) person from each team will summarize:
    - The progress the team as a whole has made that week,
    - What problems were encountered in working the last week, and discussion about what's needed to solve those problems if they're impeding progress. We will spend most our time discussing these.
- What the team will be attempting for the next week.
- These updates are also submitted to Canvas (participation)

Next Quarter: Section

- Before each Wednesday domain each team will update each other on their progress for the week (e.g. Zoom, Slack, etc).
- In each Wednesday domain, one (randomly chosen) person from each team will summarize:
    - The progress the team as a whole has made that week,
    - What problems were encountered in working the last week, and discussion about what's needed to solve those problems if they're impeding progress. We will spend most our time discussing these.
- What the team will be attempting for the next week.
- These updates are also submitted to Canvas (participation)

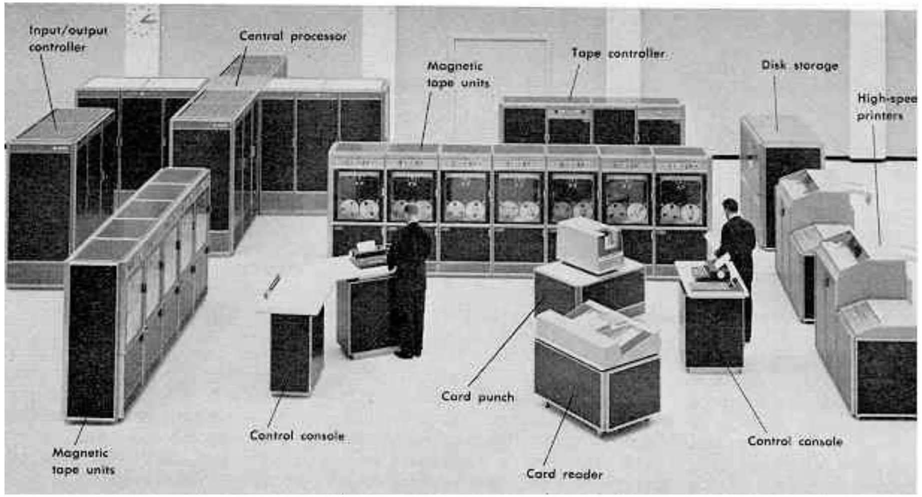
Developing Software on Teams

Section Outline

- A quick history of software development
- Agile software development methodology
- Agile in the context of the Capstone

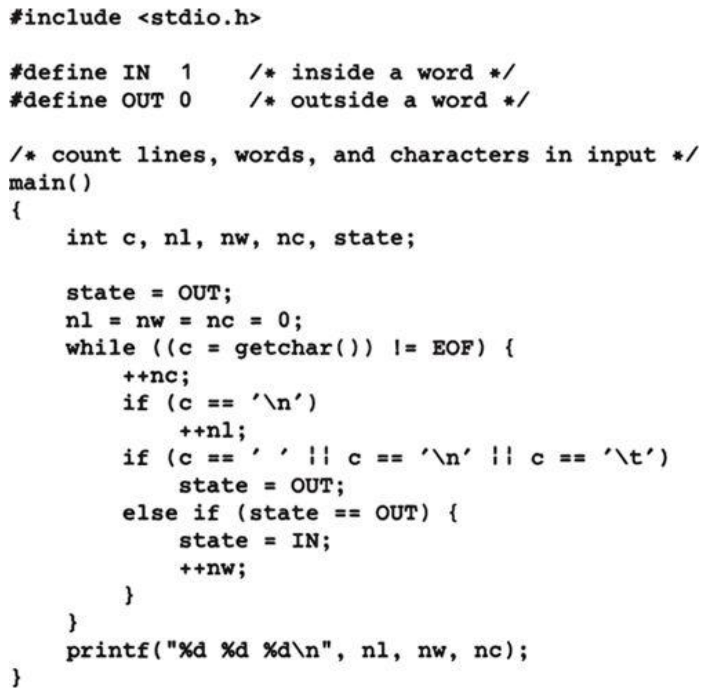
Software Development in the Dark Ages

- Before the 1960's, software didn't exist as we know it.
- Computer program inseparable from hardware; code developed for a single environment.
- Projects often didn't move beyond a computer or team.



The Beginning of Software Development

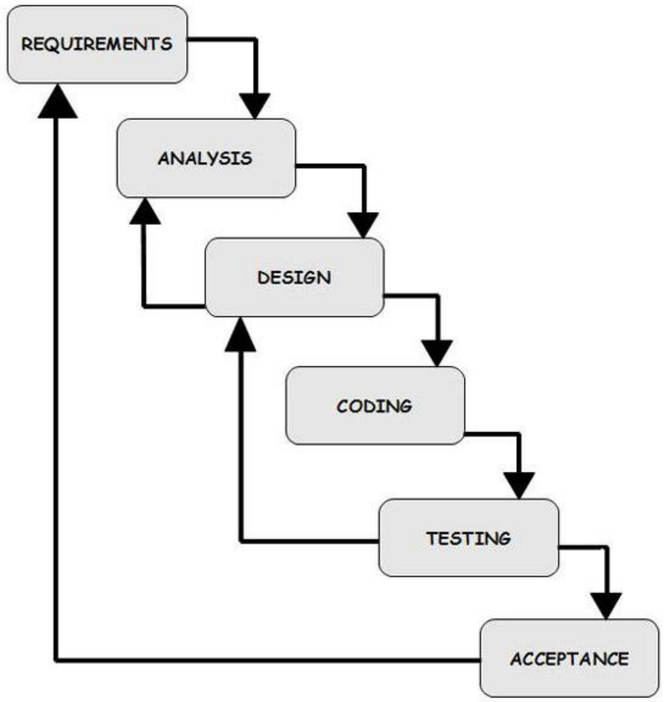
- The 1970s saw the appearance of programming languages decoupled from hardware.
- The size of software projects greatly increased, with little organization between developments (“cowboy coding”)
- During the 1970s-1980s, *most* code didn’t not work when shipped!



```
#include <stdio.h>

#define IN  1    /* inside a word */
#define OUT 0    /* outside a word */

/* count lines, words, and characters in input */
main()
{
    int c, nl, nw, nc, state;

    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF) {
        ++nc;
        if (c == '\n')
            ++nl;
        if (c == ' ' || c == '\n' || c == '\t')
            state = OUT;
        else if (state == OUT) {
            state = IN;
            ++nw;
        }
    }
    printf("%d %d %d\n", nl, nw, nc);
}
```

Enterprise Software: Waterfall Development

- Clearly delineate the steps required for a project beforehand.
- Specify the contract (inputs/outputs) for each step.
- Work on steps *serially* (wait for previous dev to finish, before starting your step).
- Disadvantage:
    - What if project requirements change?
    - What are others doing while you're working?



Agile Software Development

Agile Development adheres to the following values:

- **Individuals and Interactions** over processes and tools
- **Working Software** over comprehensive documentation
- **Customer Collaboration** over contract negotiation
- **Responding to Change** over following a plan.

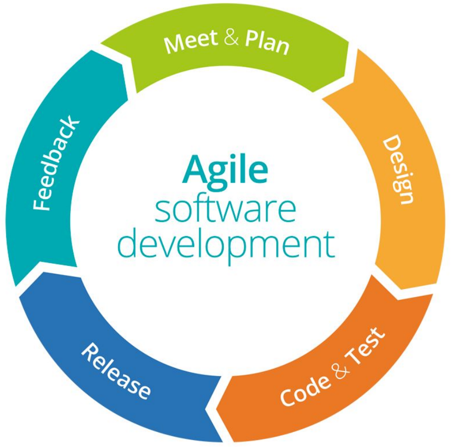
It encourages incremental progress by *doing* and works well in the face of changing requirements.

Agile Development Works in Tight Cycles

- Meet and plan needed tasks.
- Design potential solutions to smaller subtasks.
- Try to implement a solution (code)
- See if it works (release + feedback)
- Revisit your plans!

Requires a small team working incrementally.

Bonus: your software *always* works!



How is Agile used in 180B?

- Your project group is the small team.
- Your project proposal guides your work:
  - After every checkpoint, “meet and plan” to create a list of tasks needed to accomplish the proposal’s objective (and whether the objective needs changing!)
  - These tasks form your ‘backlog’ of tasks the team has to do.
- Every week, you go through the cycle:
  - Each Tues/Wed, review the work down in the previous week (feedback) and adjust the future plans (the backlog) as necessary.
  - Then plan the work each *individual* will do the upcoming week.
- Quick and frequent feedback keeps any one team member from flailing:
  - If a task was harder than expected, teammates can apportion time to help after feedback.
- Important: finish every week with a *finished* product!

Creating a Schedule

- Keep in mind:
  - Create an initial backlog of tasks ('to-do list') for 6 weeks; revise these as you work.
  - Estimate how many 'person-weeks' a task might take when creating the schedule.
  - At the beginning of each week, create individual-level tasks for the week (split up tasks from the backlog among group members)..
- When apportioning tasks among group members:
  - Try to split up work on the project so that everyone can work concurrently!
  - Approach 1: work on parallel tasks that don’t depend on each other.
  - Approach 2: work on serial tasks in parallel by using ‘stubs’ or ‘placeholder data’ to simulate the previous tasks completion. (Frequent communication is important here!).