Willis Allstead
10/1/18
CS 425

1. Software Engineering Clips

https://www.coursera.org/lecture/engineeringandroidapps/why-test-ZKMlZ
5 minutes, 40 seconds (might bother you to sign up, just ignore popups)

This video is called "Why Test", and it is in a course of android development, but it concisely talks about why you might want to do testing, and specifically unit testing when building an application. These same concepts can be translated to all software engineering. One key point that he touches on in this video is that you want to do testing so that while your application evolves, it doesn't start breaking in unknown ways. He also mentions that while it takes more time upfront, it will probably end up saving you time in the long run as your application/program changes. They show how a code reviewer can comment line-by-line on the code changes made by the developer and how they approve it for the next step in the review. These processes, while not requiring any specific software such as the ones provided by Atlassian, happen in the lives of developers around the world every day.

https://www.youtube.com/watch?v=OMLh-5O6Ub8
3 minutes, 20 seconds

This video is called "Atlassian Software Development Workflow", which is focusing on the workflow of a software developer using a specific group of web-based software provided by the company Atlassian. I linked this video because I use a similar workflow at my job and I think it is useful to understand what people actually use in their day-to-day work as a software engineer. In the video they go over how a non-developer would create a task, how a developer would prioritize their work depending on the tasks at hand, how the developer would take notes about their work

2. Code of Ethics

"PUBLIC — Software engineers shall act consistently with the public interest."

If a software engineer at a company chooses to spend a large amount of time making sure his program is bug-free even if he knows small bugs probably wouldn't be caught by QA and wouldn't come up as an issue for most users, he is following this clause. He is putting his private interests, such as finishing the project earlier, aside and doing the right thing in this case.

"SELF — Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession."

A software engineer who takes some of their free time, whether on weekends or after work throughout the week, to learn about new technologies so that he can bring them back into his job is following this clause. A software engineer with an emphasis on the web who decides to take a Saturday to learn a new front-end web framework so that he can offer that as a skill to his company's current and future clients is a great example of this.

"COLLEAGUES — Software engineers shall be fair to and supportive of their colleagues."

In software engineering it is commonplace to ask for help from colleagues, especially as a junior. If a senior software engineer is approached by a junior for help with a task, for example if the junior is supposed to alter a database that he has no experience working in, if the senior offers insight into the problem at hand he is following this clause. The senior doesn't have to solve the problem outright, but giving the junior the tools to learn things himself is key in a supportive software engineering workplace.

"PRODUCT — Software engineers shall ensure that their products and related modifications meet the highest professional standards possible."

For example, if a software engineer is given a deadline for a product release or deployment that he does not think he can reach reasonably within that time, he should let his superiors know as soon as he can. If an app developer is told that he needs to program and release a security update for his company's social networking app in a week, but he doesn't believe he could do so while making the app meet the highest professional standards possible, he should make that apparent to his superiors.

3. Software Processes

- A system to control anti-lock braking in a car

  The best software process model for this case would be the Waterfall model. Since the project is engineering a car, it can probably be assumed that there is not a lack of funds, which are required in this method. Also, this is a safety-critical system, which means the waterfall method would be appropriate.

- A virtual reality system to support software maintenance

  Since virtual reality systems are still relatively new to the market, especially for aiding in software maintenance, the most appropriate process model would be Incremental development. This is because issues will come up that couldn't have been planned for. Also, there is probably no concern of safety besides the usual issues that come with virtual reality, which are negligible.

- A university accounting system that replaces an existing system

An accounting system is extremely important for any entity, especially large ones. Caution must be taken in its replacement. Since there would be clear-cut needs in the replacement of a system like this, the Waterfall model could be used. Also, a large entity like a University probably has the funds to pay for a more expensive process model like that.

- An interactive travel planning system that helps users plan journeys with the lowest environmental impact

Depending on the amount of money the people have that are funding this, it could be best to go with either Incremental development model or Reuse-oriented software engineering. If I assume they have a good amount of money, I would choose Incremental development since that could cost more but end up with a better product. Also, the project seems like a new concept and new problems could arise along the way while working on it.

4. Intellectual Property

a. Patents "protect ideas and their embodiments". You must file a patent application and pay a large fee. Copyrights "protect expression". All you have to do for a copyright is create something of expression. It is the most flexible of the intellectual property protections. Trademarks protect the "designation of source of goods". They are used in commerce.
b. Patents have four criteria. These include:
    - Invention must be novel/new. – This means that an invention cannot be a recreation of something else.
    - Invention must be non-obvious. – This means that you cannot patent something that anybody could have thought of with little effort. For example, you cannot patent a certain color of water bottle.
    - Invention must have utility. – This means that an invention is only patentable if it is useful. You cannot patent something that nobody would use, as there would be no purpose.
    - Invention must be enabled. – This means that a person with an ordinary amount of skill in a certain profession should be able to create what you have laid out. For example, you cannot patent a time machine (yet).
c. Three questions that an open source software license should answer are:
    1. Can you create derivatives? – This means that someone should know if they can take your open source software, alter it, and call it their own.
    2. Can you sell software you create that includes open source code? – This means that someone should know if they can use your code in their commercial product. For example, Facebook cannot use code from open source projects that specify that the code cannot be used to make money.
    3. What about attribution? – This means that a programmer should know if they must give credit for any open-source code they used in their project.