Patrick Austin

CS 425 Individual Assignment 2  
9/25/2017

**1. Software Engineering Video Clips**

a. What is the ‘Daily Scrum’?

<https://www.youtube.com/watch?v=zxwEdbcrqD8>

This video reinforces and expands on what was covered in class pertaining to the importance of daily scrum meetings, how they should be structured, and what responsibilities are involved for the scrum master in the meeting. This video is significant for software engineering as scrum meetings are an important and widely-used tool in agile development, the dominant software engineering methodology. Using scrum meetings well can improve a team’s efficiency and productivity, which is clearly desirable to software engineers. While this video is partially a review of what has been presented in class, I found it worth watching because it focused concisely on a single topic, reminding me of the in-class material while also clarifying a few points that hadn’t been clear to me before, such as that the scrum meeting is not led by the scrum master- he merely enforces rules such as the time limit.

b. Therac 25 - Software Testing

<https://www.youtube.com/watch?v=izGSOsAGIVQ>

This video briefly introduces some background and takeaways from the Therac 25 case for why and how software testing is important. The Therac 25 was a radiation machine used to treat cancers in the 1980s; unfortunately, serious bugs discovered in its software led to several deaths and cases of severe injury. This is a famous case study in software engineering, and it reminds us of the importance of testing and safety analysis in the field. It also reminds us of a software engineer’s ethical responsibility to uphold the public welfare and do right by our clients. This video is worth watching because it is so succinct in explaining the basics of the Therac 25 case; it hits the main points in less than 3 minutes, where case studies of the incident can run into the many tens of pages. It also emphasizes the importance of software testing, a step in the software development process I hope we explore further in this class. As a refresher on this important case, which I had not reviewed in some time, I found it to be a valuable watch.

**2. Code of Ethics**

a. Public: Since software engineers should act to uphold the public interest- the well-being of the general public- they should not accept work which they know will prove harmful to the general populace of society. For example, if asked to develop a piece of software such as a malicious worm which would harvest the general population’s sensitive data such as bank or credit card information or social security numbers, a software engineer should refuse. In general, software engineers should not work on software that is likely to be harmful to public health and well-being.

b. Client and employer: Since software engineers should act as honest and faithful agents of their clients and employers, they should show the utmost discretion when sharing proprietary information belonging to their clients and employers. For example, if asked by colleagues for inside information about their work, code, or projects for an employer, a software engineer should refuse. By maintaining the confidentiality of proprietary information that has been entrusted to the engineer as an employee, the software engineer protects the best interests of employer.

c. Colleagues: Since software engineers should be fair to and supportive of their fellow software engineers, they should be serious, respectful, and unbiased in cases where disputes arise. For example, if asked to judge whether another software engineer has violated an ethical norm, the software engineer should approach the dispute in a serious, comprehensive, and unbiased way so as to maximize fairness and support. The software engineer should not prejudge a colleague one way or another in such a dispute; nor should the software engineer ignore any of the available evidence to try to guide the outcome towards one the engineer prefers.

d. Self: Software engineers should practice lifelong learning in order to maintain their expertise in the profession. For example, a software engineer should keep abreast of journals, news, professional organizations, and the like related to his area of expertise. To not do so but to claim expertise in a given area of software engineering regardless could be dangerous and unethical. In a field that changes as quickly as software engineering, relying on outdated experience could waste an employer’s money and time, or, worse, cause safety and security defects that might endanger the public interest.

**3. Software Processes**

a. The best software process model for approaching the design of a system to control anti-lock braking in a car is likely to be a waterfall method. Two criteria of waterfall development match closely to this situation. On the one hand, a system to control brakes likely has to interface directly with the hardware of the brake system, meaning that design and functionality decisions likely cannot change later on in the process as in the other methods.

Also, this system is clearly critical to safety, since it governs a vehicle’s brakes and therefore its ability to stop. Thorough specification and design has to happen at the outset in order for the best possible analysis of the safety of the system to take place, which tracks best with the waterfall method. As the text says, safety problems in specification and design are expensive to correct later in the process.

b. A virtual reality system to support software maintenance should probably use incremental development. The customer is likely to give lots of feedback on the interface and requirements might change a great deal during development. VR is a cutting edge-area and reacting to the user experience is important: one might find, say, a development feature gives users motion sickness and therefore needs significant redesign, or even exclusion from the project.

Increments of the software can be delivered as features are polished to a stomach-able point, and then expanded on based on the desires of the customer. In general, this project seems well-suited to an incremental, and likely agile, approach.

c. The best software process model for creating a university accounting system to replace an existing system is likely the waterfall method. There is a preexisting, established system and so existing requirements and new ones are likely to be well understood. These requirements are also not likely to change much, or quickly.

This is likely to be a large system and a large project with multiple teams working on it, so having complete specifications may be important in order to coordinate the development of different subsystems.

Also, security is likely important to this system as it may handle bank and credit transactions, sensitive personal information, and etc. Waterfall is likely to be the best match in this situation so that a thorough security analysis can take place in the specification and design phase.

d. An interactive travel planning system which minimizes environmental impact is likely to be best suited to incremental development. Customer requirements could plausibly change a lot during development, and various features could be emphasized/deemphasized based on feedback as the project progresses. The project is likely to be developable and deliverable in increments as features, such as types of travel, visualization of travel, hook-in to other systems such as Google Maps and travel websites, etc., come online.

Likewise, customer feedback can be taken into account to scale up or down the emphasis and depth of these features as development progresses. For these reasons this project also seems suited to an incremental, and probably agile, approach.

**4. Intellectual Property**

a. Patents are a type of intellectual property designed to promote science and the useful arts. They do so by allowing an individual exclusive access to an innovation that he shares with society. For example, someone who creates a new design for a machine part gains the ability to prevent others from using that design for the term of the patent, in exchange with disclosing his design to society.

Copyright exists to protect works of individual, creative expression. An author (or the author’s employer, if the work was made for hire) automatically gains copyright over their expression when it is “saved” on a tangible medium, and can sue for damages if the copyright is violated by others reproducing their expression without permission.

Trademarks exist to protect the designation of a source of goods. That is, they exist to protect and distinguish the owner of a brand from others who might seek to imitate or masquerade as the original owner. For example, Lego has a trademark on their products and can sue for damages if a counterfeiter falsely claims to be selling Lego products.

b. There are four criteria for patents: the invention must be patentable, the invention must be novel, the invention must be non-obvious, and the invention must have utility. I will briefly discuss each of these criteria.

The first criteria reflects the reality that not all inventions are patentable in the first place. For example, many works of creative expression are better covered by the copyright system, since works of art and expression often lack the focus on science and the useful arts that typify patents. There are certain classes of patents, and something that falls outside these classes is likely to be unpatentable.

The second criteria, newness or novelty, reflects the intent of the patent system to reward the development of innovations in science and the useful arts. This is fairly intuitive- one cannot get a patent for something that has already been patented as a matter of course, since patents exist to protect new developments from exploitation by others for the period of the patent.

The third criteria, non-obviousness, also reflects the patent system’s emphasis on innovations in science and the useful arts. Something obvious is unlikely to represent an innovation or discovery in science, agriculture, manufacturing, etc., or it would have already been discovered and incorporated into science and industrial practice. Therefore such a claim should not be awarded a patent.

The final criteria, utility, is yet another reflection of the emphasis on science and the useful arts- with an emphasis on the useful. If a discovery is innovative but not useful to scientific or industrial processes, it is unlikely to be patentable. (If this discovery is of a more creative nature, it may be better served by the copyright system.) Ultimately, the patent system exists to reward an inventor for sharing a practical, useful discovery with society.

c. An open source license should answer a number of questions about how the software governed by the license can be used. Three of these are whether you can sell software that uses the open source software, what the legal jurisdiction will be should there be dispute about the use of the software, and whether you can create derivatives of the open source software. I will briefly discuss each of these.

The question of whether software that uses a piece of open source software in its operation can be sold is an important one. It may be desirable to leverage open source software in a commercial product due to it being free, but one must consult the license closely to see whether this is allowed or whether this decision could incur further hidden costs down the line, in terms of legal fees or licensing fees. It may be that software which appears to be free at first glance is not actually so.

The question of jurisdiction is likewise important. Knowing which nation or state the open source license is to be arbitrated under in the case of a dispute can be important in the decision of whether to include the open source software in a product. The laws of different countries may vary widely, and the costs of entering legal battle in a far-off country may be prohibitive. Therefore the license should clearly specify the ground rules, and those seeking to use the open source software should read carefully.

Finally, the question of selling derivatives of an open source software is an important one. For a business in search of projects and profit it may make sense to polish or innovate on an open-source base to create a commercial product. But all the effort of creating such a thing would likely be wasted if the license forbids the sale of such products. Therefore, once again, there are practical and economical reasons to heed (and create) open source licenses very carefully.